RedBook

OPEN SPECIFICATION
SERIES

RedBook Open XML Specification

1.0

A specification for portable, authored digital worlds



Press Rogers

RB-OXML-1.0

OPEN SPECIFICATION
.redx

About this specification

RedBook Open XML Specification 1.0 defines the RedBook document model, supplied vocabulary, XML packaging and the contracts through which compatible implementations work with RedBooks. This website presents the current specification.

Specification	Details
Author	Press Rogers
Version	`1.0`
Identifier	`RB-OXML-1.0`
Document extension	`.redx`

Download the RedBook Open XML Specification 1.0 PDF (SHA-256 checksum), or the current web edition and companions, with its file manifest and SHA-256 checksum. The PDF contains the reading edition; the schemas and catalogues remain companion files in the archive. The archive contains this HTML edition, its reference files and the assets needed to read the included pages. The PDF is a separate download. No Reader executable is included, and illustrative example components do not claim to implement every specification capability.

Identified normative schemas and catalogues form part of the specification. The Overview explains its structure, requirement identifiers and the distinction between technical requirements and explanatory material.

Authorship and licensing

Original specification prose is provided under Creative Commons Attribution 4.0 International. Original schemas, catalogues, examples and specification tools are provided under the MIT License. See the notices, prose licence notice and MIT License for the applicable terms.

Copyright © 2026 Press Rogers.

Reading this specification

1. Understand the document

The introduction explains RedBook from the perspective of a person working with a document of their own and the digital participants represented within it. It introduces the concepts before the formal rules.

2. Follow the technical foundation

Parts 1–5 define the logical model, package representation, supplied vocabulary, operational contracts and conformance obligations. Their section and requirement identifiers remain stable throughout this publication.

3. Use the reference material

Annex A contains the supplied type definitions. Annex B contains the action contracts and their printed JSON Schemas. Annex C identifies external references; Annex D specifies execution and presentation; Annex E provides explanatory walkthroughs.

Navigate by meaning, not only by page.

The contents and reference indexes link to their destinations. Numbered requirements retain their `RBX-` identifiers, so a technical rule can be cited independently of pagination.

The specification is a multi-artifact contract. This document contains the complete reading edition. Where a clause names a normative schema or catalogue, that machine-readable companion remains part of the contract and accompanies the specification distribution.

FIND YOUR WAY

Contents

About this specification 2
Authorship and licensing 2

Reading this specification 3
1. Understand the document 3
2. Follow the technical foundation 3
3. Use the reference material 3

Introduction to RedBook 10
A New Kind of Document 11
What Is a RedBook? 11
A Document That Understands Connections 12
Your Document, Your Digital Persons 12
Communication Within the Document 13
Knowledge That Can Develop 14
Content, Resources and Functionality Together 15
A Document That Can Be Continued 15
One Portable File 16
An Open Foundation 17
A Document of Your Own 17

Part 1 — Fundamentals and logical model 19
1.1 Scope and purpose 20
1.2 Organization, authority and terminology 20
1.3 The authored-world boundary 22
1.4 The two primitives 23
1.5 Document identity and registries 24
1.6 Portable value algebra 25
1.7 Numbers and exactness 26
1.8 Value equality and copies 27

1.9 Types and one-level specialization 27
1.10 Slot contracts 28
1.11 Properties and explicit references 29
1.12 Categories and Tags 29
1.13 Authorship and source credit 30
1.14 Authored events and timestamps 31
1.15 Document-local chronology 31
1.16 Transactions, mutation and deletion 32
1.17 Queries, selection and navigation 33
1.18 Supplied vocabulary and equipped worlds 34
1.19 Resources and preservation 34
1.20 Extensibility and capability boundaries 35
1.21 Interoperable implementation strategy 36

Part 2 — Packaging and XML representation 37
2.1 Scope and relationship to other standards 38
2.2 File kinds and identification 38
2.3 ZIP physical representation 39
2.4 Content types 40
2.5 Package relationships and target resolution 40
2.6 Native world parts 41
2.7 Typed XML values 42
2.8 Records and type declarations 43
2.9 XML processing and schema use 44
2.10 Inventory, resources and integrity 45
2.11 Standalone component envelope 45
2.12 Blueprint graph and dependencies 47
2.13 Theme and Plugin targets 48
2.14 Preservation, saving and failure 49
2.15 Layered package and component checking 50
2.16 Protocol-control JSON in XML 50
2.17 Definition compatibility and independent reuse 51
2.18 Portable participant-authoring declarations 52
2.19 Bundled functionality in a Blueprint 52

Part 3 — Core vocabulary, Communication, Soma and Time 54
3.1 Scope and status of the supplied vocabulary 55

3.2 Common domain conventions 56

3.2.1 Declared fields and content maps 56

3.2.2 Association order and current state 57

3.3 Digitals, profiles and participation 57

3.4 Publication, Walls and conversations 59

3.4.1 Publications and placement 59

3.4.2 Conversations and contributions 60

3.5 Correspondence, Mailboxes and newsletters 61

3.5.1 Addressing and recipient projections 61

3.5.2 Organization, templates and distribution 62

3.6 Responses, original material, Works and representations 63

3.7 Collections, feeds and discovery 64

3.8 Attention, notification and presence 65

3.9 Polls, questionnaires, proposals and reviews 65

3.9.1 Polls and current ballots 65

3.9.2 Questionnaires 67

3.9.3 Proposals and explicit decisions 67

3.9.4 Reviews and findings 68

3.10 Requests, commitments, cases and reservations 68

3.10.1 Responsibility and progress 68

3.10.2 Reservable resources 69

3.11 Events, recurrence and gatherings 70

3.11.1 Event descriptions and occurrence keys 70

3.11.2 Deterministic recurrence profiles 70

3.11.3 Facilitated work 71

3.12 Streams and live exchange records 71

3.13 Participation rules and moderation 72

3.13.1 Standard condition and effect representation 72

3.13.2 Reports and review decisions 73

3.14 Soma: conceptual homes, worlds and representations 73

3.14.1 Neocortex ownership 73

3.14.2 Membership and scope 74

3.14.3 Conceptual content and reuse 75

3.15 RedBook Time and event observations 76

3.15.1 Three distinct coordinates 76

3.15.2 Rule forms 76

3.15.3 Coordinates in domain Properties 77

3.16 Exact world-time evaluation 78
3.16.1 Formulas and results 78
3.16.2 Gregorian conversion 79
3.16.3 Manual advance, precision and examples 79
3.17 Participant conceptual authoring 80
3.18 Explicit state transitions and organization 81
3.19 Deterministic correspondence threads 83
3.20 Newsletter issue allocation 85
3.21 Response policy and standard rules 85
3.22 Recurrence invocation and pagination 87
3.23 Feed selection before paging 88
3.24 Participant initiation and descriptive authoring 88
3.25 Specialization fields and ordinary participant actions 92

Part 4 — Actions, interfaces, execution and extensibility 94
4.1 Scope and capability boundaries 95
4.2 Execution contexts and portable values 95
4.3 Action contract and atomic execution 97
4.4 Query, editing and dependency behavior 99
4.5 Participation and domain operations 100
4.6 Atomic batches and result bindings 102
4.7 Blueprint capture and instantiation 103
4.8 Original resources and package-aware actions 106
4.9 The eight-tool interface 106
4.9.1 Jellybox WebMCP mapping 108
4.10 Retry, failure and transport mapping 110
4.11 Hooks, filters and transaction boundaries 113
4.12 Extension installation, activation and cleanup 114
4.13 Viewbox lifecycle, presentation and the host bridge 115
4.14 Limits, implementation assessment and interoperability 115
4.15 Scoped participant editing and classification 116
4.16 Domain lifecycle, clearing and finite projections 117
4.17 Policy-context resolution and explicit rule processing 118
4.18 General custom participant writes 118
4.19 Primary creation, descriptive workflows and composition 119

Part 5 — Conformance and implementation obligations 121
5.1 Conformance is an explicit claim 122
5.2 Named capabilities 122
5.3 Validation layers 124
5.4 Required reporting fields 125
5.5 Producer obligations 126
5.6 Consumer obligations and error containment 127
5.7 Resource limits 127
5.8 Conformance vectors and verification procedure 128
5.9 Examples and implementation assessment 129
5.10 Extension and interoperability claims 129
5.11 Publication, corrections and evolution 130
5.12 Serialized declarations and discovery 130
5.13 Lifecycle coverage and executable evidence 131
5.14 Custom participant-authoring evidence 132

Annex A — Complete supplied type reference 134

Annex B — Standard action contracts 344

Annex C — External reference register 596
JELLYBOX — Jellybox WebMCP Framework 597
BCP14 — Requirement terminology 597
XML — Extensible Markup Language (XML) 1.0 597
XMLNS — Namespaces in XML 1.0 597
XSD — XML Schema Part 1: Structures and Part 2: Datatypes 597
OPC — ECMA-376 Part 2: Open Packaging Conventions 598
URI — Uniform Resource Identifier: Generic Syntax 598
JSON — The JavaScript Object Notation Data Interchange Format 598
JSONSCHEMA — JSON Schema Core and Validation 598
ECMASCRIPT — ECMAScript Language Specification 598
UTF8 — UTF-8, a transformation format of ISO 10646 598
SHA256 — Secure Hash Standard 599
UNICODE — The Unicode Standard 599
CCBY — Creative Commons Attribution 4.0 International 599
MIT — MIT License 599
DATETIME — Internet date and time syntax 599

ICALENDAR — Recurrence conventions 600
LANGUAGE — Tags for identifying languages 600
BASE64 — Binary-to-text encoding 600
CASEFOLD — Versioned Unicode default case folding 600

Annex D — Portable execution, extensions and presentation 601
D.1 Profiles and capability negotiation 602
D.2 Installation resource roots and dependencies 602
D.3 ECMAScript module profile 604
D.4 Plugin API and handler invocation 605
D.5 Hook stages, ordering and filtering 608
D.6 User-defined extension points 609
D.7 Viewbox bridge and render completion 610
D.8 Declarative Theme templates and view data 611
D.9 Stable interface regions and contributions 612
D.10 Minimal portable Reader host API 613
D.11 Failure and lifecycle invariants 615
D.12 Portable participant authoring 615
D.12.1 Portable rules and host execution 616
D.12.2 Context and participation 617
D.12.3 Public checked mutations 618
D.12.4 Composition, preservation and qualification 620

Annex E — Annotated reading and implementation walkthroughs 622
E.1 Package structure and authored meaning 623
E.2 A record containing exact data and a reference 623
E.3 Protocol-control JSON without changing authored values 624
E.4 Atomic extension work and a subsequent render 625
E.5 A portable research-log presentation 625
E.6 A new kind of authored behavior: Measurement 626
E.7 Specialized content, deliberate events and reuse 627

INTRODUCTION

Introduction to RedBook

A New Kind of Document

What Is a RedBook?

A Document That Understands Connections

Your Document, Your Digital Persons

Communication Within the Document

Knowledge That Can Develop

Content, Resources and Functionality Together

A Document That Can Be Continued

One Portable File

An Open Foundation

A Document of Your Own

A New Kind of Document

Documents have long been one of the most familiar ways for people to create, organise, preserve and share information. We write letters, prepare reports, build spreadsheets, develop presentations and assemble collections of knowledge. Each document type provides a particular way of organising content, together with the tools and conventions that make that content useful.

A word-processing document organises written material into paragraphs, headings and sections. A spreadsheet organises information into cells, rows and columns, allowing values and calculations to interact. A presentation organises content into slides designed for visual communication.

Each has a recognisable structure that makes it suitable for particular kinds of work.

RedBook introduces another kind of document, designed around a different organising principle.

Rather than beginning with pages, cells or slides, a RedBook begins with the things that exist within it and the relationships between them. Those things may represent information, people, ideas, resources, activities or anything else its author wishes to include. Their relationships allow the document to preserve not only the individual things, but also how they belong together.

This gives RedBook a foundation for organising information in ways that are not limited to a particular presentation or activity.

A RedBook can be read and viewed, but its contents can also be explored, developed, connected, communicated about and worked with through software. It is a document whose internal structure is designed to support ongoing participation.

What Is a RedBook?

> A RedBook is a structured, portable document that brings together elements, relationships, resources and functionality. Designed for participation as well as presentation, it provides a shared context in which humans and digital agents can create, communicate and develop their work together over time.

A RedBook is saved as a single document with the .redx file extension. Like other document formats, it can be created, opened, saved, copied and transferred. Unlike documents organised primarily around their visual appearance, however, a RedBook's underlying content is represented as a connected structure of identifiable things.

The foundation of this structure consists of two kinds of authored content: Elements and Relationships.

An Element represents something within the document. A Relationship represents an association involving elements. Both have their own identities and can carry properties, categories and tags that describe their meaning and characteristics.

This deceptively simple foundation allows a RedBook to represent increasingly sophisticated structures without requiring every possible kind of content to be anticipated in advance.

A document might begin with only a handful of elements and relationships. As further content is introduced, these can be extended, connected and developed while remaining part of the same underlying document.

The result is not merely a collection of information. It is an organised context in which the individual parts can retain their identities and participate in a larger whole.

A Document That Understands Connections

Consider the difference between describing a relationship in a sentence and representing that relationship as part of a document's structure.

A conventional document might contain a paragraph explaining that a particular person developed an idea, that the idea influenced a project, and that the project produced a particular result.

A reader can understand those connections because the information has been expressed in language.

In a RedBook, the person, idea, project and result can each be represented as identifiable elements. Their relationships can also be represented explicitly, with information belonging to those relationships themselves.

The connections are therefore not confined to the sentences in which they were first described.

Other parts of the document can refer to the same elements and relationships. Further information can be associated with them. Software can examine their connections, retrieve relevant content and present different views of the same underlying structure.

This is an important distinction.

The meaning of a RedBook does not have to depend entirely on the order in which information is written or the way it happens to appear on a screen. Its contents can be organised around what they represent and how those representations relate to one another.

A view may present that information as written text, a table, a diagram, a visual interface or another suitable form. The underlying elements and relationships remain part of the same document.

In this way, RedBook separates the organisation of its contents from the many ways those contents can be presented and used.

Your Document, Your Digital Persons

An especially important purpose of RedBook is to provide an environment in which a person can work with digital agents through a persistent, shared context.

Artificial intelligence is increasingly capable of helping people research, reason, write, create software, analyse information and perform complex tasks. A person may work with several different agents, each with a particular responsibility, personality, speciality or way of approaching a problem.

Yet the work of these agents is often distributed across separate conversations, applications, files and temporary sessions.

RedBook offers a different arrangement.

Imagine having a RedBook of your own, containing the people, ideas, resources, projects and knowledge that matter to what you are doing. Within that document, you can also have digital persons who participate in its activities.

You might have a research agent investigating questions, a technical agent developing software, a creative agent contributing ideas and another agent helping organise ongoing work.

These are not necessarily separate human users or separate document owners. They are digital participants operating within the context of your RedBook.

In RedBook, such participants can be represented as Digitals.

A Digital is an identifiable participant within the authored document. It can have its own characteristics, relationships, communications and accumulated conceptual understanding.

The important distinction is that a Digital is not itself an artificial intelligence model.

The intelligence that acts through a Digital comes from outside the document, such as an AI session connected through a compatible Reader or external agent system. A single AI may act through different Digitals at different times, and different AI sessions may contribute through the same Digital.

The RedBook preserves the authored identity and contributions of the Digital independently of the particular AI session involved.

This means the document can retain continuity even as the software, models or sessions used to work with it change.

A RedBook does not need to contain an AI model, nor does it need to remain continuously running. It preserves the authored context in which external intelligence can participate.

Communication Within the Document

A RedBook is not limited to storing information about its participants. It also provides structures through which those participants can communicate.

Think of the familiar ways people interact through digital systems: posting messages, exchanging correspondence, participating in conversations, publishing announcements, asking questions, responding to ideas and coordinating work.

RedBook brings these kinds of communication into the document itself.

Its communication structures include Walls, Posts, Conversations, Mailboxes, Newsletters and related resources. They provide ways for Digitals to contribute and interact within the context established by the document.

A person opening their RedBook might encounter a Wall containing contributions from several digital persons. A discussion may have developed around a question. One Digital may have communicated a finding to another, while a third has contributed a different interpretation.

These communications can be connected to the elements, relationships and resources that concern them.

Rather than existing as isolated messages in an unrelated service, they belong to the authored context of the RedBook.

There is something distinctive about this arrangement.

The document itself can contain a communication environment, with identifiable participants and preserved contributions. A person may be the only human working with the RedBook while numerous digital persons communicate and work within it.

The RedBook is not fundamentally a multi-user collaboration service. It is a personal, portable document capable of containing an active community of represented participants.

That community can be examined, developed and continued as part of the document's contents.

Knowledge That Can Develop

Communication is only one part of working together.

When people or digital agents investigate a subject, they do more than exchange messages. They form interpretations, establish connections, reconsider assumptions and develop an understanding of what they are working on.

RedBook provides a structured means of representing this conceptual activity.

Within its architecture, a Digital has an associated Soma Neocortex, providing a home for authored conceptual structures. These structures are built from Neo Somas, Neo Objects, Neo Relationships and their associated properties, categories and tags.

The purpose is to allow concepts and their connections to exist as identifiable, organised information.

A Digital's conceptual understanding can therefore be represented independently of the temporary AI session currently acting through it.

RedBook also provides shared conceptual contexts associated with communication sessions, allowing contributions to a conversation to develop into connected conceptual structures.

This distinction matters.

An individual Digital can have its own conceptual context, while participants in a conversation can contribute to a shared body of understanding associated with that conversation.

That does not mean RedBook contains a conscious mind, independently thinking intelligence or continuously operating reasoning process. Its conceptual structures are authored information. External humans and AI systems perform the interpretation, reasoning and development.

The document provides a place for the resulting understanding to be represented and retained.

Over time, this allows a RedBook to preserve more than a history of what was said. It can also preserve organised knowledge and conceptual relationships developed through the work of its participants.

Content, Resources and Functionality Together

The contents of a RedBook need not be limited to predefined forms.

A RedBook can include resources such as images, documents, data and other supporting material. It can also carry definitions and source code associated with its functionality.

This introduces another important characteristic of the format.

A RedBook can describe not only the information it contains, but also the particular structures and operations through which that information is used.

For example, an author may wish to introduce a specialised kind of element representing something not covered by the initial definitions. That element may require additional properties, particular relationships, custom operations or a distinctive way of being presented.

RedBook's extensible architecture allows such functionality to be defined and supplied as part of its authored environment, subject to the capabilities and permissions of the software opening it.

The application used to open a RedBook is called a RedBook Reader.

A Reader provides the means to create, inspect, present, operate on and save RedBooks. Different Readers may offer different interfaces and supported capabilities, while conforming implementations follow the common rules defined by the specification.

The distinction between the Reader and the document is fundamental.

The Reader is software. The RedBook is the portable authored document.

Just as a spreadsheet is not the same thing as the application used to open it, a RedBook is not the same thing as its Reader.

This separation allows the format to develop independently of any one application or software environment.

A Document That Can Be Continued

A RedBook is not necessarily finished when it is saved.

It can be opened again, examined, changed, extended and saved with further authored content. New elements and relationships can be introduced. Resources can be added.

Communications can continue. Conceptual structures can develop as new understanding is authored.

The document can preserve the context that makes those later contributions meaningful.

Consider returning to a RedBook after several weeks.

The elements and relationships you previously created are still identifiable. Resources remain associated with their relevant contents. Digitals can retain their authored identities and conceptual structures. Conversations and their contributions can remain available for further examination and development.

You are not required to reconstruct the entire context from memory or from a collection of disconnected files.

That continuity is central to RedBook's purpose.

It does not require every modification to be automatically retained as a permanent historical revision. RedBook distinguishes its current authored state from deliberately recorded events, editions and other history.

Nor does continuity require the document to remain open or an AI agent to remain active between sessions.

The document itself carries the authored structures that make continued work possible.

One Portable File

All of these capabilities lead back to something remarkably familiar: a document file.

The .redx extension identifies a RedBook document, packaged according to an openly defined XML-based format.

The file can contain the authored elements and relationships, resources, definitions and relevant functionality that make up its portable contents.

It can be stored on a computer, copied to another location, shared with someone else or opened using compatible software.

A RedBook does not have to be tied permanently to a particular website, AI provider or proprietary application.

Its portability is more than the ability to export a picture or a report of its contents. The document is intended to preserve the underlying authored structure, so compatible implementations can encounter and continue working with it.

A RedBook might begin very simply. It may contain a small number of elements and relationships, perhaps representing a few ideas and the connections between them.

Nothing requires every RedBook to contain numerous Digitals, complex conversations or elaborate functionality.

Those capabilities are available when they are useful.

The same fundamental document structure supports a small beginning and the possibility of substantial development.

An Open Foundation

RedBook is defined by the RedBook Open XML Specification, an open technical specification describing the document format, its structures, relationships, packaging, supported behaviours and interoperability requirements.

Publishing the specification makes the format available beyond any single implementation.

Developers can create compatible Readers, build tools that work with RedBook documents, develop extensions and explore new ways of presenting and using their contents.

Authors can create RedBooks for their own purposes, while others can develop and distribute compatible resources, Blueprints, Themes, Plugins and other components within the broader RedBook ecosystem.

The intention is not to prescribe one application interface or restrict what a RedBook may eventually be used to represent.

It is to establish a common document foundation from which many forms of authored content and functionality can develop.

This specification describes that foundation.

It defines how RedBooks are structured, how their contents are represented, and the contracts through which compatible implementations may work with them.

The examples and explanations that follow illustrate how these structures can be understood and applied. The normative requirements establish the rules that implementations must satisfy where applicable.

A Document of Your Own

The possibilities of RedBook are perhaps best understood by returning to the simplest situation.

You have a document.

You can put things into it, describe them and connect them. You can give those things identities, organise information around them and present them in useful ways.

You can work with digital persons who participate within that context, communicate through the document's own structures and contribute knowledge and other authored work.

You can return later and continue developing what is there.

And the whole creation remains a portable document that you can keep.

A RedBook does not need to represent an enormous world or perform an extraordinary task to be valuable. It can begin with something small and meaningful to its author.

Its significance lies in the foundation it provides: a document capable of bringing content, connections, resources, communication and functionality into one continuing context.

> A new kind of document, designed not only to preserve what has been created, but to provide a place in which further creation can happen.

1

PART 1

Part 1 — Fundamentals and logical model

The document model: identity, the two primitives, values, type definitions, attribution, chronology and preservation.

1.1 Scope and purpose

1.2 Organization, authority and terminology

1.3 The authored-world boundary

1.4 The two primitives

1.5 Document identity and registries

1.6 Portable value algebra

1.7 Numbers and exactness

1.8 Value equality and copies

1.9 Types and one-level specialization

1.10 Slot contracts

1.11 Properties and explicit references

1.12 Categories and Tags

1.13 Authorship and source credit

1.14 Authored events and timestamps

1.15 Document-local chronology

1.16 Transactions, mutation and deletion

1.17 Queries, selection and navigation

1.18 Supplied vocabulary and equipped worlds

1.19 Resources and preservation

1.20 Extensibility and capability boundaries

1.21 Interoperable implementation strategy

1.1 Scope and purpose

RedBook Open XML 1.0 specifies a portable authored world. Its two record primitives, RB Element and RB Relationship, represent identities, communication, conceptual models, resources and custom subject matter within one document. The format combines structured data, definitions, original resource bytes, presentation and the source needed to interpret the capabilities that the document offers. A RedBook can be opened, inspected, edited, exchanged and developed without a continuously running service.

This specification defines document meaning, serialization, supplied vocabulary, portable component packages and operational contracts. It defines the obligations of producers and consumers separately from a particular user interface or programming language. A Reader may be a desktop application, a browser application, a command-line tool, a server, or a tool constructed by an external intelligent agent. These forms do not change document semantics.

> **[RBX-P1-001]** A conforming implementation MUST identify the capabilities for which it claims conformance according to Part 5. Parsing a ZIP archive or satisfying an XML grammar alone MUST NOT be reported as complete RedBook conformance.

This is the first specification version. It contains no transitional file format and imposes no migration obligation from experimental systems. The format version and specification edition are both `1.0`; the specification identifier is `RB-OXML-1.0`. Document conformance claims name that stable edition, independently of the publication date or distribution build identifier.

This specification is authored by Press Rogers and defines RedBook Open XML 1.0 for independent implementation.

1.2 Organization, authority and terminology

The normative body consists of Parts 1–5, Annex A (the core type reference), Annex B (the action reference), Annex D (execution, extensions and presentation), and the normative schemas and catalogues identified in the distribution manifest. Examples, implementation assessments and test reports are informative unless a normative clause explicitly incorporates an expected result.

Part 1 defines the logical model and invariants. Part 2 defines package and XML representations. Part 3 defines domain meaning, including Communication, Soma and Time. Part 4 defines actions, interfaces, execution and extensibility. Part 5 defines claims, validation layers, preservation and assessment. Annexes A and B expose the supplied type and action contracts. Annex C identifies external standards; Annex D supplies portable runtime/host contracts; Annex E provides informative walkthroughs.

> **[RBX-P1-002]** The capitalized requirement words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY and OPTIONAL are interpreted under BCP 14, RFC 2119 and RFC 8174. Lowercase uses have their ordinary meaning. A SHOULD permits a documented reasoned exception; it does not silently become a MUST.

> **[RBX-P1-003]** Normative prose defines semantic meaning; schemas define machine-checkable shape within that meaning. A schema accepting a value does not override a prose prohibition. Conversely, a producer MUST satisfy both the applicable schema and semantic rules. A detected inconsistency between normative artifacts MUST be reported as a specification issue rather than resolved by silently broadening an implementation's conformance claim.

A requirement marker, such as `RBX-P1-001`, is a reference to the associated obligation within the identified specification edition. Multiple test cases may exercise one requirement; one passing test does not establish every implication of that requirement. The generated requirement register identifies the body clauses. Catalogue contracts impose additional obligations at their named type or action entry.

The following terms have specific meanings:

Term	Meaning
Authored world	The current persistent state represented by one RedBook document.
Element	An independently identified record whose primitive is `element`.
Relationship	An independently identified record whose primitive is `relationship`, with named participant slots.
Type	A carried root CET or CRT definition, or one specialization of a root.
Digital	A core Element representing an actor to whom activity can be attributed.
Reader	A host that inspects or operates on RedBook packages under declared capabilities.
Viewbox	An authored presentation and interaction surface belonging to a world.
Neocortex	A conceptual home owned by one Digital or shared communication context.
Neo Soma	A conceptual model housed in one Neocortex.
Blueprint	An independently portable reusable graph or definition collection requiring explicit instantiation/import.
Package part	A named byte stream in the OPC container; distinct from an authored record.
Action	A named operation with input, output, effect, role and failure contracts.
Execution session	Ephemeral host state associated with a loaded world or active presentation.
Capability	A named set of obligations an implementation explicitly supports.

Term	Meaning
Source credit	Descriptive authorship information retained independently of a local actor reference.
Event	A deliberately recorded occurrence carrying document-local chronology metadata.

1.3 The authored-world boundary

> **[RBX-P1-004]** Persistent RedBook state MUST describe the current authored world. Ordinary editing MUST replace or remove current values without automatically appending previous values, revision chains, tombstones, audit histories, reconstruction journals or hidden snapshots to the RedBook package.

A deliberately authored edition, message, report, observation or reflection is current content with its own meaning. For example, a Digital can write a Post describing an earlier conversation. That Post is not an automatically retained version of the conversation. An explicit edition action creates a separately identified authored work; saving an edit does not invoke it.

> **[RBX-P1-005]** Reading, opening, validating, searching, rendering or selecting an operating context MUST NOT by itself create authored activity, change timestamps, allocate event numbers, mark attention, create missing Neocortexes or recalculate authored conceptual conclusions. Attention and other recorded effects require explicit actions.

A host may maintain temporary transaction snapshots, session revision tokens, retry receipts and render caches. A filesystem may independently maintain backups. These are outside the authored-world model.

> **[RBX-P1-006]** A producer MUST NOT serialize host retry receipts, session bindings, authentication credentials, access tokens or automatic editor recovery history as implicit RedBook state. Deliberately supplied arbitrary resource bytes remain authored resources; this rule prohibits hidden host additions, not the user's ability to store a chosen file.

A Digital is not a running model, operating-system identity or external login. One external human or AI can represent many Digitals in separate selected contexts. Many external sessions may deliberately represent one Digital. Neither arrangement changes the identity of the authored actor.

> **[RBX-P1-007]** A Reader MUST distinguish access to the file from structural authorization inside an action context. Digital participation checks preserve authorship, membership and domain rules; they MUST NOT be represented as an encrypted confidentiality boundary against a holder of the package bytes.

The optional RedBook Runtime is a Reader-side connection through which the Reader may request assistance from an AI. External AI may instead use the Reader's interface or inspect a file directly. The format requires neither arrangement, no heartbeat and no always-on scheduler. The runtime that executes a source module is a separate technical facility from this optional AI connection.

1.4 The two primitives

> **[RBX-P1-008]** Every authored record MUST be exactly one of `element` or `relationship`. Digital, Viewbox, Post, Neocortex, Neo Soma, Neo Object and other supplied concepts MUST be expressed through these primitives and carried type definitions. An area name such as Communication MUST NOT introduce another primitive or consume a type-inheritance level.

Both primitives have identity, type, title, Properties, Categories and Tags. Relationships additionally have slots. Optional attribution and event metadata have the same meaning on both primitives.

Logical member	Required	Meaning
`id`	Yes	Stable identity within the record namespace.
`primitive`	Yes	`element` or `relationship`; encoded by the XML record element.
`type`	Yes	Identity of a carried type of the corresponding primitive.
`title`	Yes	Authored display text; empty is permitted.
`properties`	Yes	Map from authored string keys to portable values.
`categories`	Yes	List of assigned Category identities without duplicates.
`tags`	Yes	List of assigned Tag identities without duplicates.
`slots`	Yes in the working model	Empty map for an Element; named lists of Element identities for a Relationship.
`attribution`	No	Local author reference and/or source credit.
`recordedAt`	No	Real-world recording instant for an authored event.
`sequence`	No	Positive document-local event ordering value.
`worldTime`	No	Recorded result of the world's time rule, including null.

> **[RBX-P1-009]** A Relationship MUST have at least one participant across its slots. A participant MUST resolve to an Element in the same world. Relationship identities may be referenced through typed-reference Properties; they MUST NOT be inserted into Element participant slots.

A Relationship may be unary, binary, cyclic or have many roles and participants. An Element may participate in several slots of the same Relationship if each slot permits it. This supplies directed, undirected, ordered, repeated and multi-party patterns without imposing a universal source/target pair.

> **[RBX-P1-010]** An Element MUST NOT acquire participant slots through a custom field or specialization. Its `slots` canonical JSON member is `{}`. A Property named `slots` remains ordinary nested content and does not change the primitive contract.

1.5 Document identity and registries

A world has its own identity, title, description, format version, current time rule, presentation settings, arbitrary document settings, next event sequence and collections of records, types, taxonomy, assets and extensions. XML part descriptors map these logical collections to package parts.

> **[RBX-P1-011]** Document identity and every record, type and taxonomy identity MUST be nonempty XML-compatible text. Identity comparison MUST be exact and case-sensitive after XML character processing. A Reader MUST NOT apply Unicode normalization, locale case folding, URL decoding or title matching to decide identity equality.

UUID URNs are RECOMMENDED for newly allocated document, record and taxonomy identities. The supplied vocabulary uses fixed `urn:redbook:type:` identities. These are identifiers, not instructions to access a network location. A custom absolute URI is permitted. No algorithm may infer a record's title, author or type from an identity's spelling.

The registries are typed: record references resolve within the shared Element/Relationship record registry; type references resolve within the type registry; Category and Tag assignments resolve within their respective taxonomy registries. An identity occurring in two different registries does not make those entries the same entity. New producers SHOULD allocate disjoint identities across document and authored registries to avoid human ambiguity.

> **[RBX-P1-012]** Record identities MUST be unique across Elements and Relationships in one world. Type identities MUST be unique within the type registry. Category identities and Tag identities MUST each be unique within their corresponding registry. Display titles and labels need not be unique.

> **[RBX-P1-013]** Saving, renaming, moving or copying the bytes of a RedBook MUST NOT implicitly allocate new authored identities. Explicit independent-world creation or Blueprint instantiation MUST allocate new destination identities for the items copied as independent records. An external filename is not document identity.

Document identity does not guarantee global uniqueness by fiat: independently modified byte copies may retain the same identity. A host that loads several such copies MUST distinguish their loaded instances using ephemeral host handles. It MUST NOT silently merge their records or assume that equal document IDs prove byte equality.

1.6 Portable value algebra

> **[RBX-P1-014]** A Property value MUST belong to the following recursive algebra. Object instances with executable getters, functions, native dates, runtime pointers or unsupported host-specific values MUST be rejected or deliberately encoded as authored data before they enter the logical model.

Kind	Logical representation	Important distinction
Null	`null`	Present null differs from an absent key.
String	Unicode text	No implicit date, identity or markup interpretation.
Boolean	`true` or `false`	False differs from absence and zero.
Number	Finite binary64 value under clause 1.7	Approximate arithmetic value.
Exact integer	`{"$rb":"integer","value":"…"}`	Arbitrary signed decimal lexical integer.
Exact decimal	`{"$rb":"decimal","value":"…"}`	Exact decimal lexical value, optional exponent.
Reference	`{"$rb":"reference","id":"…"}`	Explicit local record reference.
List	Ordered sequence of portable values	Order and repeated values are significant.
Map	String-keyed finite map of portable values	Key insertion order has no semantic significance.

The `$rb` forms are the canonical JSON representation of distinct XML value kinds. They are not ordinary maps that happen to contain a reserved-looking key.

> **[RBX-P1-015]** A map containing `$rb` MUST be one of the three exact tagged forms above and MUST contain no additional members. An ordinary map that needs to discuss the literal text `$rb` can place it in a string value or in an explicit list of key/value pairs; it cannot impersonate a tagged value.

> **[RBX-P1-016]** All string values, map keys, identities, titles and labels MUST consist of XML 1.0 permitted characters. Invalid scalar values, isolated surrogates, NUL and prohibited control characters MUST NOT be silently replaced. Bytes that do not meet this requirement can be preserved as a binary asset with an explicit media type.

A map key is data. It may contain spaces, punctuation, non-Latin text or characters that would not form a legal XML element name. Its XML representation is an escaped `key` attribute on `rb:entry`. Keys `__proto__`, `constructor` and `prototype` have ordinary data meaning; a JavaScript consumer must use data-safe map construction rather than assigning prototype behavior.

> **[RBX-P1-017]** Maps MUST reject duplicate keys. Lists MUST preserve item order. Readers MUST preserve the distinction among an absent member, null, empty text, an empty list, an empty map, false and numeric zero.

A producer is not required to preserve whitespace used to indent an XML map. It is required to preserve whitespace within an authored string, including explicit carriage returns represented by character references. Rich content stored as text remains text until an explicitly selected renderer interprets its declared format.

1.7 Numbers and exactness

Ordinary numbers provide interoperable binary64 arithmetic. Their XML lexical form follows the JSON number grammar. They exclude NaN and positive or negative infinity. When the resulting value is mathematically integral, it is limited to the inclusive range −9007199254740991 through 9007199254740991.

> **[RBX-P1-018]** A producer MUST reject an ordinary integral number outside the safe range and MUST use an exact tagged representation when exact larger integers are required. A reader MUST NOT round a large exact integer or decimal into an ordinary number. Ordinary positive and negative zero have the same logical numeric value; serializers MAY normalize their sign.

The exact-integer grammar is `[+-]?[0-9]+`. The exact-decimal grammar is `[+-]?(?:[0-9]+(?:\.[0-9]*)?|\.[0-9]+)(?:[eE][+-]?[0-9]+)?`. These grammars match the entire value; leading/trailing whitespace and newline suffixes are invalid. A leading sign or leading zero is allowed in these authored exact values. Chronology counters deliberately use the narrower canonical grammar in clause 1.15.

> **[RBX-P1-019]** Exact integer and decimal lexical text MUST be preserved without normalization. `1`, `+1` and `01` are different exact-integer encodings; decimal `1.0` and `1.00` are different authored lexical values. Arithmetic operations may compare their mathematical values when their own contract says so, but definition equivalence and round-trip preservation MUST retain the kind and lexeme.

No universal decimal precision or exponent magnitude is imposed by the format. An implementation may declare resource limits. A value that exceeds a declared limit is unsupported in that implementation; it is not thereby malformed. Implementations MUST report the distinction described in Part 5. Host integer-conversion limits must not cause a supported canonical text comparison to round or truncate values.

1.8 Value equality and copies

> **[RBX-P1-020]** Structural equality MUST compare null and Boolean values by kind and value; strings by exact character sequence; ordinary numbers by numeric binary64 value; exact numbers by kind and lexical text; references by their exact identities; lists item-by-item in order; and maps by equal key sets and recursively equal associated values, independent of key insertion order.

This equality is used for imported-definition conflict detection, semantic input comparison and schema choice comparisons unless a named operation explicitly defines a different comparison. It prevents an independent XML reader's map enumeration order from changing meaning.

A logical copy is independent mutable state with equal portable values. It must not retain shared mutable references that allow editing one copied record to mutate another unintentionally. Byte-for-byte equality is a stronger property applying to a particular serialized part or package, not the definition of world equality.

1.9 Types and one-level specialization

A root Custom Element Type (CET) defines an Element kind. A root Custom Relationship Type (CRT) defines a Relationship kind. Each may have specializations one level below it. For example, Digital Person specializes Digital; it remains an Element. Communication is a vocabulary grouping and does not sit above Digital in an inheritance chain.

> **[RBX-P1-021]** A type definition MUST contain `id`, `key`, `primitive`, `level`, `parent`, `area`, `label`, `description`, `fields`, `slots`, `traits` and `openSlots` in the working model. A root has level `type` and null parent. A specialization has level `specialization` and a parent that resolves to a root of the same primitive. Specializing a specialization, cyclic ancestry and cross-primitive inheritance MUST be rejected.

The `key` is a vocabulary-local short identifier. It does not replace the stable `id`. `label` and `description` are authored explanatory text. `area` groups entries for discovery and documentation; it has no inheritance effect. The core catalogue supplies the area for every supplied entry.

> **[RBX-P1-022]** A specialization instance MUST satisfy both its root's constraints and its own constraints. Redeclaring a field or slot does not erase the root rule. Types and their dependencies MUST be carried in the document so that records do not depend on an unavailable external type registry to acquire meaning.

A field rule declares a portable kind, requiredness, optional default, optional choices and explanatory text. Supported rule kinds are `any`, `string`, `boolean`, `number`, `integer`, `decimal`, `reference`, `array` and `object`. `integer` accepts a safe ordinary integral number or an exact integer tag; `decimal` accepts the exact decimal kind. `object` means an ordinary map, excluding tagged values; `array` means a list. `any` accepts the value algebra, including null. Other kinds do not accept null unless the field's contract expressly defines a nullable representation.

> **[RBX-P1-023]** A present declared field MUST satisfy its rule and every inherited rule. A missing required field MUST be rejected. A `choices` list, if present, further restricts the field to structurally equal listed values. An arbitrary authored field not declared by the type remains permitted; it is still subject to the portable-value and reference rules.

Defaults are creation conveniences, not missing-data inference. On creation, root defaults are copied first, specialization defaults replace same-key defaults, and explicit input replaces either. The resulting record must satisfy all rules.

> **[RBX-P1-024]** A reader MUST NOT materialize missing defaults during an ordinary read. A producer MUST validate default values when authoring a definition and MUST reject a type whose supplied effective defaults cannot satisfy its declared constraints. Default maps and lists MUST be independently copied for each new record.

> **[RBX-P1-025]** Updating a definition MUST retain its identity, short key, primitive and ancestry. Every retained affected record MUST validate against the resulting definition in the same transaction. Changing those identity-defining aspects requires a new definition and deliberate retyping of selected records under the authoring contract.

The core vocabulary declares observable semantics as well as shapes. Replacing a core definition under the same identity with incompatible semantics does not create a conforming core world. An application that needs a different concept allocates a custom identity, possibly as a valid specialization.

1.10 Slot contracts

Each declared Relationship slot has an allowed-type list, minimum cardinality, optional maximum, orderedness and uniqueness rule. The canonical JSON names are `allowedTypes`, `min`, `max`, `ordered` and `unique`. A null maximum means no vocabulary-imposed upper bound; declared implementation limits still apply.

> **[RBX-P1-026]** Slot minimum and maximum values MUST be nonnegative safe integers, with maximum at least minimum when present. Allowed types MUST identify carried Element types. A participant satisfies a listed root type if it is an instance of that root or one of its specializations; a listed specialization accepts that specialization. An empty allowed-type list imposes no type restriction beyond Element membership.

> **[RBX-P1-027]** A missing declared slot is evaluated as an empty list for cardinality checks. Every present participant list MUST contain only local Element identities. A slot with `unique: true` MUST reject duplicate identities in that slot. Uniqueness is not global across all roles of the Relationship.

An ordered slot gives relative position semantic meaning. In an unordered slot, position has no domain meaning, but the serialized list is still preserved as authored data. A reader must not sort an unordered slot just because its meaning is set-like.

> **[RBX-P1-028]** Undeclared slots are permitted only when at least one definition in the record's lineage has `openSlots: true`. Such slots have unrestricted Element types, minimum zero, no maximum, ordered lists and allowed repetition unless a domain rule says otherwise. Declared root and specialization slot rules both continue to apply.

Neo Relationships impose additional home constraints in Part 3. A custom open-slot Relationship does not bypass those constraints merely by changing role names.

1.11 Properties and explicit references

> **[RBX-P1-029]** A reference value MUST resolve to a retained record in the same world unless it appears in a Blueprint graph at an explicitly declared external binding position. The consumer MUST recursively inspect references inside maps and lists. An ordinary string that happens to equal an identity MUST NOT be treated as a reference.

Core domain fields can further restrict the referenced primitive or type. For example, `somaId` on a Neo Relationship refers to a Neo Soma Element, even though the general reference kind can identify a Relationship. A resource path is not a record reference. A source author's identity inside an attribution credit is descriptive text, not a local reference.

References to external documents can be authored as ordinary structured content, for example a map with `documentId` and `recordId`. They do not acquire local referential integrity or network-fetch behavior. A custom type may define a stronger external-link contract under a different field convention.

> **[RBX-P1-030]** An import or identity rewrite MUST update typed local references, participant slots and declared structural reference fields according to its mapping. It MUST NOT search-and-replace arbitrary strings, source files or binary resources for apparent identities. Dependencies hidden in opaque resource formats require a declared transform capability or explicit unresolved-dependency failure.

1.12 Categories and Tags

Categories and Tags are shared document registries usable by both primitives. They classify content without defining its type. A Category may have one parent Category; a Tag has no parent relationship in this registry. A record may have zero or many assignments of either kind.

> **[RBX-P1-031]** Every Category parent MUST resolve within the Category registry and the parent graph MUST be acyclic. Every assignment MUST resolve in the correct registry and occur no more than once in that record's assignment list. Empty labels are permitted as data, but authoring interfaces SHOULD encourage meaningful labels.

Category ancestry does not automatically add parent identities to a record's explicit assignments. A query can choose to include descendants when its contract declares that behavior. Tags never acquire implied hierarchy through punctuation in their labels.

> **[RBX-P1-032]** Removing or merging a taxonomy term MUST explicitly resolve its record assignments and, for a Category, its child-parent references in the same transaction. No action may leave dangling assignments or silently turn a Category into a Tag. Imported labels that happen to match do not justify identity merging.

Assignment order is preserved in the serialized lists. It may be used for display, but it does not confer priority on the first Category or Tag unless a presentation contract explicitly says so.

1.13 Authorship and source credit

Attribution describes the author credited for a current record. It is not an edit log. Both Elements and Relationships can carry it. A structural Relationship may have an author different from the author of the content it places or discusses.

> **[RBX-P1-033]** An attribution map MUST contain at least one of `digitalId` and `credit`, and no other members in version 1.0. A present `digitalId` MUST resolve to a local Digital or Digital specialization. A credit MUST contain exactly nonempty `documentId`, nonempty `digitalId` and string `name`, all describing the credited source.

> **[RBX-P1-034]** A new record created on behalf of a selected Digital MUST record that local actor and a source credit obtained from the current document identity, Digital identity and current authored Digital title. Book-administration actions without an explicitly selected author MAY create unattributed structural records. An existing record MUST NOT silently change attribution because another actor reads it or edits allowed content.

A shared Neo Soma can retain its initial creator's attribution while an Authorship Relationship names several contributors. Its Objects and Relationships retain their own contributors. A Wall placement credits the actor who made that placement; it does not rewrite the placed Post's author.

> **[RBX-P1-035]** Copying content without its Digital MUST preserve or establish its source credit and omit the unresolved local `digitalId`. Copying the author as part of the same Blueprint MUST map the local reference to the new Digital while retaining the original source credit. Coincidentally equal titles or identity strings MUST NOT reattach a detached credit.

> **[RBX-P1-036]** Deleting a Digital that credits retained content MUST detach those local author references and retain their source credits before deletion commits. If necessary the producer MUST obtain the credit while the Digital still exists. Deleting the author MUST NOT by itself delete all attributed content.

An explicit administrative attribution operation is permitted and records the currently intended credit. It must be deliberate and validated. It does not require preserving the former credit in hidden history. Legal ownership and licensing are not inferred from attribution fields.

1.14 Authored events and timestamps

An event is a deliberate occurrence, such as a sent message, published content creation, attention mark or reservation request. The contract of the action determines whether it records an event. Record creation alone does not imply a recording timestamp; conceptual objects and structural links may be created without event metadata.

> **[RBX-P1-037]** A newly authored event MUST receive a `recordedAt` real-world UTC instant, a fresh `sequence`, and `worldTime` equal to the current time-rule result, including null for `none`. These are allocated in the same transaction as the event. Reading, later edits and world-time reconfiguration MUST NOT recalculate retained event metadata.

`recordedAt` uses `YYYY-MM-DDTHH:mm:ss.sssZ`, four decimal year digits, valid proleptic Gregorian date fields, hour 00–23, minute and second 00–59, and exactly three fractional-second digits. It is a recording instant, not a claim about when the described event occurred. A described historical or scheduled date belongs in a domain Property. Leap-second labels ending in `:60` are not supported in this representation.

> **[RBX-P1-038]** A supplied recording timestamp MUST be a valid canonical instant. A producer MUST reject invalid dates and MUST NOT accept a host parser's silent normalization of an impossible calendar date. If the host cannot supply the required timestamp, an event-creating action MUST fail atomically rather than fabricate one.

Events imported from another world retain their original `recordedAt` and recorded `worldTime`, while their local sequence is allocated anew. Equal recording timestamps are valid; they do not collapse distinct events. Batched authoring may assign one real recording instant to several deliberately created events.

The optional event members are allowed independently in a structurally inspected record so that authored partial observations can be represented, but a conforming event-producing operation must supply the complete triple. Any present timestamp and sequence still satisfy their individual constraints. A diagnostic must distinguish structural acceptance from compliance with the action that produced a record.

1.15 Document-local chronology

> **[RBX-P1-039]** `nextSequence` and every present `sequence` MUST be canonical positive ASCII decimal text matching `[1-9][0-9]*` over the entire string. Leading zeros, signs, exponent notation, non-ASCII digits, whitespace and numeric JSON values are invalid chronology encodings.

> **[RBX-P1-040]** Retained event sequence values MUST be unique across both primitive collections. `nextSequence` MUST be strictly greater than every retained event sequence. With no events, any positive next value is valid; a newly created world starts at `1`. Gaps are permitted. Deletion MUST NOT rewind the counter or renumber retained events.

> **[RBX-P1-041]** Event ordering and counter advancement MUST use exact arithmetic. An implementation MUST NOT convert these values to binary64 or a bounded native integer when that could change their value or ordering. For canonical positive text, comparison by digit count and then ASCII digit sequence is exact. Decimal increment may operate directly on characters with carry.

For example, sequence `999` precedes `1000`, even though lexical comparison alone gives the opposite answer. A 10,000-digit supported counter is compared using the same rule; a host integer-conversion safeguard is not a format limit. The general resource-limit rules still apply.

> **[RBX-P1-042]** Allocating one event takes the current next value and increments it by exactly one. Allocating a Blueprint's events sorts their source sequences numerically and assigns a contiguous block starting at the destination's current next value. Source sequences MUST be unique and valid before allocation. Non-event records consume no numbers. Failure MUST roll back the records and the counter together.

The ordering after import is destination-authored order. It does not assert that a source event's real timestamp follows an existing destination event's real timestamp. Neither timestamps nor world-time values are changed to create that illusion.

Numeric presentation positions, such as a Wall placement's position, are separate Properties. Their values may be deliberately reordered without rewriting chronology. Chronological reads use exact event sequence order; presentation reads use their stated ordering rules.

1.16 Transactions, mutation and deletion

> **[RBX-P1-043]** A successful write MUST leave the entire retained world valid under every supported applicable structural and domain constraint. A failed write MUST leave authored state, event allocation, type registries and taxonomy unchanged. A success result MUST NOT be returned for a world that the same conforming writer cannot save and reopen under its declared supported limits.

A host may stage resource bytes or construct several dependent records before the final validation point. Intermediate invalidity is permitted only within an explicitly atomic transaction that is not externally visible as committed authored state. Unrelated concurrent operations must not observe or continue from it. Shape, declared write authority and required local fields are checked at each exposed mutation. Dependencies that require several records, including mutual references, may be completed before commit. A public read, query or nested action must see a complete valid candidate; it cannot expose a dangling intermediate graph. The reserved-identity mechanism in Annex D supports constructing mutually referring custom records without relaxing final validation.

> **[RBX-P1-044]** A mutation MUST make its target identities and intended effect explicit. Updating Properties merges explicitly supplied keys; removing a key uses an explicit unset operation. Replacing assignment or participant lists replaces the supplied list in full. An omitted patch member leaves the corresponding current value unchanged. Null is an assigned value and MUST NOT be interpreted as an implicit deletion.

Creation materializes effective type defaults only for omitted Properties; an explicit null does not request a default. Required fields must then be present and satisfy the combined root and specialization constraints. An empty map or list is an explicit value, not an omitted argument. A schema default documents a value; implementations apply defaults only where the corresponding action or type contract says to do so. A failed nested mutation poisons its enclosing transaction even if its exception is caught; Part 4 defines retry identity and Annex D defines extension composition.

Deletion is permanent removal from current authored state. Deletion preview lists affected relationships, typed references and author credits. It does not create a tombstone or imply consent to remove everything mentioned in that preview.

> **[RBX-P1-045]** Deletion MUST either include the explicitly selected dependent structural records needed for a valid result or reject the operation. Optional removal of incident Relationships MUST be explicitly requested. Surviving typed references, ownership requirements, taxonomy dependencies and default-presentation references MUST be resolved in the same transaction. An invalid partial deletion MUST NOT commit.

For example, a Digital that owns a Neocortex cannot be removed while leaving an ownerless Neocortex. The caller may explicitly select that conceptual subgraph for deletion or deliberately rehome content under valid semantics before deleting the Digital. This rule does not authorize automatic erasure of attributed Posts.

> **[RBX-P1-046]** A type-removal operation MUST reject removal while retained records or specializations require that definition. A resource-removal operation MUST explicitly resolve declared uses. Administrative convenience MUST NOT silently drop opaque parts or definitions merely because the current user interface does not display them.

1.17 Queries, selection and navigation

The basic record query can filter by primitive, type and text, with bounded offset and limit. Root-type selection includes its specializations. A query returns current records; it does not materialize a second copy of them in the world. Feed definitions store queries, not a duplicated feed population.

> **[RBX-P1-047]** Query implementations MUST document ordering, text comparison and pagination semantics for each named query. Repeated pagination is stable only while the queried state is unchanged or when an explicitly supported snapshot/cursor contract says otherwise. A display label MUST NOT be used as an unadvertised identity filter.

Selection of a document, Viewbox or Digital is ephemeral interaction state. A human's visible Viewbox and an AI interaction's selected action context may differ. Navigation changes the requested presentation, not the selected actor, unless an explicit context-selection operation is also performed.

> **[RBX-P1-048]** Independent interactions MUST retain independent context selections and retry scopes. A navigation event in one interaction MUST NOT silently redirect another interaction's next mutation. Part 4 defines the bindings used to detect stale selections and edits.

1.18 Supplied vocabulary and equipped worlds

The core vocabulary contains 169 definitions in this edition: 61 root CETs, 54 root CRTs and 54 specializations. Annex A is the full reference. This is an explicit vocabulary edition, not a limit on the number of custom concepts a world can define. The `core-operations` and `authoring` capabilities are defined by the applicable named contracts in Annex B, independently of any implementation inventory. Annex D separately defines Reader host actions and versioned execution/presentation profiles.

> **[RBX-P1-049]** A world claiming the `core-data` capability MUST carry the full core vocabulary, including definitions not yet instantiated, with the structural constraints and meanings specified in Annex A. It MAY carry additional custom definitions. A minimal structural world without that vocabulary is valid only under its narrower declared scope.

> **[RBX-P1-050]** An equipped world MUST carry the actual source and declared resource dependencies needed for the executable capabilities it advertises. A catalogue entry is not an executable implementation. A Reader MUST distinguish carried source, supported execution profiles, available actions and actual authored population when reporting what a world can do.

The normal Blank world is equipped but unpopulated except for its starting World Viewbox. The specification also permits a structural fixture with no records and null default Viewbox. Neither available Digital definitions nor available Communication operations create a cast of actors, empty Walls or empty conversations automatically.

> **[RBX-P1-051]** Opening a package MUST NOT activate source code merely to determine package identity, inspect data or validate XML. Execution requires the host's explicit execution facility and declared capability handling. Unsupported code is preserved and reported; it MUST NOT be silently represented as active functionality.

1.19 Resources and preservation

An original resource is a named package byte stream with a media type and declared integrity information. It may be an image, audio, video, document, source module, font or another authored file. Metadata describes the resource; it does not replace the original bytes.

> **[RBX-P1-052]** A producer MUST preserve the exact bytes of every retained resource that it does not deliberately transform. A thumbnail, transcoded preview, extracted text or interpreted object MUST NOT silently replace the original resource. Deliberate transformations use explicit output parts and update declared references and hashes.

A resource identity, a part path, a record identity and a URI are different concepts. Domain records refer to resources under their declared field conventions. The package contains its own content-type and relationship metadata, which is not the same thing as RB Relationships.

> **[RBX-P1-053]** A read-modify-save implementation MUST retain unmodified unknown parts and their required package metadata. If an edit could invalidate an opaque dependency that the implementation cannot interpret, it MUST refuse that edit or obtain an explicit destructive transformation choice; it MUST NOT silently claim lossless preservation.

Logical world equality allows XML indentation and map-entry order changes; opaque-part preservation requires byte equality. Package compression, ZIP metadata and member ordering may differ without changing either property. Reproducible byte builds are useful evidence, but are not a universal producer requirement.

1.20 Extensibility and capability boundaries

Custom types extend vocabulary. Plugins extend functionality. Themes extend presentation. Viewboxes combine a world-owned surface and its interaction behavior. Reader, RedBook and Viewbox are separate targets for themes and plugins; a component package declares exactly one target.

> **[RBX-P1-054]** A custom definition or extension MUST declare the dependencies needed to interpret or activate it. It MUST use a distinct identity when its meaning is not the meaning of a supplied definition. A consumer MUST retain unsupported content and report the resulting capability limit without manufacturing substitute meaning.

A file holder may deliberately edit source, install a new extension or reinterpret authored content. Such freedom does not mean a Reader may silently bypass the selected operation's structural rules. The selected context and the actual mutation contract remain observable.

> **[RBX-P1-055]** A Reader MUST expose unsupported required capabilities before performing an operation that depends on them. A capability declaration MUST identify its version and relevant limits; it MUST NOT be inferred solely from filename, file size, installed software or the existence of a definition.

1.21 Interoperable implementation strategy

An implementation may use a database, in-memory object graph, lazy XML reader or generated view. The logical invariants apply regardless of storage strategy. A host can use transactions and indices internally without serializing its internal architecture into the document.

> **[RBX-P1-056]** Host optimizations MUST preserve specified identity, value kinds, ordering, dependency and mutation semantics. Normalization must not merge distinct exact-number lexemes, conflate missing with null, or discard unsupported authored fields.

A conforming consumer can interpret data without executing the carried reference JavaScript. A conforming executable Reader can implement an action independently provided it obeys the same contract. The carried source is useful portable functionality and evidence; it is not an exemption from the standard's input, output, error and preservation obligations.

The remaining parts give the serialization and concrete domain rules necessary to implement this model. The reference worlds and conformance vectors provide inspectable examples, including transfer followed by save, reopen and continued authoring. Their implementation assessment identifies the exact capabilities tested.

2

PART 2

Part 2 — Packaging and XML representation

The portable representation: package discovery, native XML, original resources and independently reusable components.

2.1 Scope and relationship to other standards

2.2 File kinds and identification

2.3 ZIP physical representation

2.4 Content types

2.5 Package relationships and target resolution

2.6 Native world parts

2.7 Typed XML values

2.8 Records and type declarations

2.9 XML processing and schema use

2.10 Inventory, resources and integrity

2.11 Standalone component envelope

2.12 Blueprint graph and dependencies

2.13 Theme and Plugin targets

2.14 Preservation, saving and failure

2.15 Layered package and component checking

2.16 Protocol-control JSON in XML

2.17 Definition compatibility and independent reuse

2.18 Portable participant-authoring declarations

2.19 Bundled functionality in a Blueprint

2.1 Scope and relationship to other standards

This part specifies the portable file representation of the logical model in Part 1, including complete RedBooks and independently held components. It specifies a constrained Open Packaging Conventions (OPC) package, native XML, a typed value encoding, integrity metadata and component manifests. It does not prescribe Reader databases, caches, installation directories or in-memory object layouts. A Reader can use another internal representation if accepted input and saved output preserve the required meaning and original resources.

> **[RBX-P2-001]** A conforming producer or consumer MUST apply the logical requirements in Part 1 as well as this part. XML-schema validity alone is insufficient: identity uniqueness, reference resolution, type inheritance, slot semantics, numeric representability, chronology and component dependencies require semantic validation.

OPC is defined by ECMA-376 Part 2, fifth edition, December 2021. RedBook uses OPC packaging conventions; it is not an Office document format and does not claim Office application compatibility or endorsement. This part's additional constraints define a narrower RedBook profile. XML syntax follows XML 1.0, fifth edition and namespace identity follows Namespaces in XML 1.0, third edition. The supplied grammars use the XSD 1.0 language; semantic requirements remain normative where XSD 1.0 cannot express them.

2.2 File kinds and identification

File kind	Conventional suffix	Main root	Root relationship type
Complete RedBook	`.redx`	`rb:redbook`	`https://redbook.fun/ns/redbook/1.0/relationships/document`
Independent Blueprint	`.redxblueprint`	`rp:component`, manifest kind `blueprint`	`https://redbook.fun/ns/redbook/1.0/relationships/component`
Theme for one target	`.redxtheme`	`rp:component`, manifest kind `theme`	Same component relationship
Plugin for one target	`.redxplugin`	`rp:component`, manifest kind `plugin`	Same component relationship

> **[RBX-P2-002]** A consumer MUST identify a file from its accepted package structure and main manifest, not solely from its suffix. Renaming the same archive to `.zip` MUST NOT change its meaning. A suffix that conflicts with the internal kind MUST be reported when the consumer presents or installs the file; it MUST NOT silently change that kind.

> **[RBX-P2-003]** The package root MUST have exactly one internal relationship of either recognized main type, and MUST NOT contain both kinds of main relationship. Extra relationships of other types are permitted. There is no whole-world Blueprint switch in a RedBook document. Copying a complete world retains a RedBook; instantiating a Blueprint creates independent component identities according to Parts 1 and 4.

The native namespace is `https://redbook.fun/ns/redbook/1.0` (`rb` below). The package vocabulary namespace is `https://redbook.fun/ns/package/1.0` (`rp` below). Prefix spelling has no meaning. These URIs identify vocabularies and do not require network access. Identifier and media-type strings here do not claim external registration.

2.3 ZIP physical representation

> **[RBX-P2-004]** A file MUST be a complete ZIP archive with consistent directory and entry data, valid CRC checks, and either stored or Deflate compression. ZIP64 MAY be used when needed. Encryption, split archives, directory entries, filesystem links and device entries are outside this profile and MUST NOT be emitted or interpreted as RedBook parts.

> **[RBX-P2-005]** Each part MUST have one canonical package name. ZIP names omit the leading slash of an OPC part URI. Names use ASCII letters, digits, underscore, hyphen, period and slash; segments MUST be nonempty, MUST NOT be `.` or `..`, and MUST NOT end in a period. Names MUST NOT start with `/`, contain backslashes, percent escapes, query strings or fragments. `[Content_Types].xml` is the sole bracket-containing special metadata name. Original filenames outside this subset belong in descriptive metadata; original resource bytes are retained under a conforming package name.

> **[RBX-P2-006]** Two entries whose names are equal after ASCII case folding MUST NOT coexist. No part name may equal another part name followed by `/` and additional segments after that same folding; for example, `a` and `A/b.xml` cannot coexist. A consumer MUST reject duplicate entries rather than choosing whichever ZIP entry happens to appear first or last. Names otherwise retain their exact spelling. Extraction MUST NOT permit any entry to escape the selected extraction root.

> **[RBX-P2-007]** Consumers MUST enforce declared resource limits before allocating or expanding unbounded input and MUST report a limit refusal distinctly from successful validation. Implementations MAY set finite limits; they MUST NOT silently truncate content to meet them. Limits are implementation properties, not an additional maximum on exact chronology counters in the logical format.

ZIP timestamps and compression choices are not authored world chronology; two equivalent archives can have different overall hashes. Concrete checker limits belong in its assessment and report, not in the document format.

2.4 Content types

The OPC namespace for `[Content_Types].xml` is `http://schemas.openxmlformats.org/package/2006/content-types`. `Default` associates an extension with a media type; `Override` associates an absolute part name with a media type. An override takes precedence for that part.

> **[RBX-P2-008]** Every part other than `[Content_Types].xml` MUST have exactly one effective media type. Duplicate defaults for the same case-folded extension and duplicate overrides for the same part MUST be rejected. An override MUST identify a present canonical part with a leading `/`. An extension default MUST be a nonempty extension, without slash or period.

> **[RBX-P2-009]** Relationship parts MUST use `application/vnd.openxmlformats-package.relationships+xml`. A native main part MUST use `application/vnd.redbook.document.main+xml`. A component main part MUST use `application/vnd.redbook.component+xml`. Other native XML and XSD parts use `application/xml`; JSON uses `application/json`; ECMAScript modules use `text/javascript`, themes use `text/css`, and Markdown uses `text/markdown`. Original media resources MUST retain their declared effective media type rather than having it guessed anew on every save.

Byte signatures can inform an explicit import operation; they do not authorize silently rewriting an accepted content-type declaration. Content types identify representation, not execution permission.

2.5 Package relationships and target resolution

OPC relationships use namespace `http://schemas.openxmlformats.org/package/2006/relationships`. Root relationships live in `_rels/.rels`. A source part `a/b.xml` has relationship part `a/_rels/b.xml.rels`. Each relationship has `Id`, `Type`, `Target` and optional `TargetMode`, whose absent value is `Internal`.

> **[RBX-P2-010]** Each relationship ID MUST be an XML ID unique within its relationship part. Relationship `Type` MUST be an absolute URI. Relationship parts other than the root MUST have a present source part. A relationship MUST NOT have a relationship part as its source or internal target. A relationship part is package navigation metadata; it is not an authored RB Relationship and does not acquire RB participant-slot semantics.

> **[RBX-P2-011]** Internal targets MUST be resolved against the source part's directory, or against the package root for root relationships. A leading `/` starts at the package root. Consumers MUST normalize `.` and `..` segments for comparison, reject navigation above the root, and require the resulting canonical target to exist. Internal targets MUST NOT contain a scheme, authority, query or fragment. Equivalent target spellings identify the same part; normalization for comparison MUST NOT rewrite accepted saved spellings or relationship IDs merely because the file was opened.

For example, a root target `./redbook/document.xml` and `redbook/document.xml` resolve to the same part. From `redbook/document.xml`, `world/elements.xml` resolves to `redbook/world/elements.xml`. `../../outside.bin` from that source escapes the root and is invalid.

> **[RBX-P2-012]** External relationships MAY be retained, but MUST NOT satisfy a required main-part, native-data, inventory or component-resource dependency. Parsing, validation, opening and passive reading MUST NOT retrieve an external target. An explicitly requested host operation may retrieve data under its own capability and consent rules.

> **[RBX-P2-013]** Relationships of type `https://redbook.fun/ns/redbook/1.0/relationships/part` link the main part to its managed parts. Every native part-map target, and every component resource plus its inventory, MUST have such an internal relationship from the main part. Unfamiliar relationships and unused optional payloads MUST remain available for preservation; their presence is not a request to execute them.

2.6 Native world parts

The conventional main path is `redbook/document.xml`. Main-part discovery uses the root relationship; another conforming canonical path has the same meaning. A processor restricted to a conventional layout declares that implementation restriction separately.

Native main field	Representation and meaning
`formatVersion`	Required attribute, exactly `1.0`
`id`	Required world identity attribute
`title`, `description`	Ordered child elements containing text
`chronology`	Child with required `nextSequence` attribute
`time`, `presentation`, `settings`	Each holds exactly one typed value
`parts`	Child containing `part` entries with `role` and `path` attributes

> **[RBX-P2-014]** The main part MUST follow `redbook.xsd`, including child order. The part map MUST contain exactly one target for each required role: `elements`, `relationships`, `taxonomy`, `types`, `actions`, `assets`, `extensions`, and `inventory`. Paths resolve relative to the main part using the internal-target rules. Duplicate roles and conflicting role assignments MUST be rejected. Additional declared roles MAY describe feature-specific parts, subject to preservation and capability rules.

The conventional targets are `world/elements.xml`, `world/relationships.xml`, `world/taxonomy.xml`, `definitions/types.xml`, `definitions/actions.xml`, `assets.xml`, `extensions.xml` and `inventory.xml`. Other carried definitions, view implementations, documentation and original resources are linked as managed parts and inventoried even when they have no main part-map role.

> **[RBX-P2-015]** The roots of required data parts MUST match their roles: `rb:elements`, `rb:relationships`, `rb:taxonomy`, `rb:types`, `rb:actions`, `rb:assets`, `rb:extensions`, and `rp:inventory`. An empty collection is represented by an empty collection container, not an absent required part. The normal equipped Blank carries the supplied vocabulary and implementation without requiring any Digital or conversation instances. A minimal structural example does not thereby claim `core-data`. The serialized claims in Part 5 distinguish document profiles from required processor capabilities.

2.7 Typed XML values

`rb:ValueHolder` contains exactly one of the following elements. Structural indentation surrounding a value is not part of the value. Text inside `rb:string` is data.

XML value	Logical value
`<rb:null/>`	Null
`<rb:string>…</rb:string>`	Unicode text, including empty text
`<rb:boolean>true</rb:boolean>` or `false`	Boolean; no numeric aliases
`<rb:number>…</rb:number>`	Finite ordinary number using JSON number syntax
`<rb:integer>…</rb:integer>`	Exact signed integer lexeme
`<rb:decimal>…</rb:decimal>`	Exact decimal lexeme, with optional exponent
`<rb:reference id="…"/>`	Typed record reference
`<rb:list><rb:item>value</rb:item>…</rb:list>`	Ordered list
`<rb:map><rb:entry key="…">value</rb:entry>…</rb:map>`	Unordered map with unique string keys

> **[RBX-P2-016]** Consumers MUST preserve null, absence, false, zero, empty text, empty list and empty map as distinct values. List order is significant. Map order is not significant; duplicate keys MUST be rejected. Map keys are data, not invented XML element names. A map containing reserved `$rb` in the JSON view MUST satisfy one of the tagged-value contracts in Part 1; arbitrary maps MUST NOT masquerade as typed references.

> **[RBX-P2-017]** Ordinary numbers MUST be finite and interoperable with IEEE 754 binary64; integer-valued ordinary numbers MUST be within the inclusive safe-integer range −9007199254740991 to 9007199254740991. NaN and infinities are forbidden. Larger exact values use tagged integer or decimal values. Exact numeric lexemes MUST retain their original sign, digits, decimal point and exponent spelling. Consumers MUST NOT coerce them through binary floating point or normalize their spelling during preservation.

Exact integer syntax is `[+-]?[0-9]+`; exact decimal syntax is `[+-]?([0-9]+(\.[0-9]*)?|\.[0-9]+)([eE][+-]?[0-9]+)?`, using ASCII digits. An event counter has a separate stricter canonical syntax: `[1-9][0-9]*`. Neither an exact-number tag nor an ordinary string grants a value event-counter semantics by resemblance alone.

> **[RBX-P2-018]** Strings and keys MUST consist of XML 1.0 characters. Writers MUST preserve meaningful carriage returns, tabs or newlines in attribute values with character references where XML normalization would otherwise change them, and MUST preserve a carriage return in text with `
`. Invalid Unicode, arbitrary binary data and original non-XML text resources belong in resource parts; they MUST NOT be silently replaced with substitute characters.

2.8 Records and type declarations

> **[RBX-P2-019]** Each `rb:element` or `rb:relationship` MUST carry its `id` and `type` attributes and the ordered children `title`, `properties`, `categories`, `tags`. A Relationship then contains `slots`, followed by any optional tail fields in schema order. An Element has no XML `slots` child; its logical slot map is empty. Properties hold a map. Categories and tags contain ordered XML `term ref="…"` entries representing their logical assignments; duplicate assignments are invalid.

> **[RBX-P2-020]** A Relationship slot is represented by `slot name="…"` with zero or more `participant elementRef="…"` children. Slot names MUST be unique in the Relationship. Every participant MUST resolve to an Element and satisfy the declared slot constraints. A Relationship MUST contain at least one participant across its slots; an individual slot may be empty when its constraints permit that. A Relationship is never inferred from a package relationship or from ordinary text that resembles an identity.

> **[RBX-P2-021]** Optional record fields are `attribution`, `worldTime`, `recordedAt`, and `sequence`. The first two use typed values; the latter two use text. Their ordering follows the XSD. A chronology sequence MUST be canonical positive ASCII decimal text, unique across both primitive collections, and less than the world's next counter. Arithmetic and comparison MUST remain exact, including above host integer-string conversion thresholds. Timestamps and recorded world-time values MUST NOT be recalculated by parsing or repacking.

[RBX-P2-022] Type declarations MUST contain the required identity, key, primitive, level and area attributes; optional parent; and label, description, fields, slots, traits and openSlots children. Their logical meanings and constraints follow Parts 1 and 3. `fields` and `slots` are typed maps, `traits` is a typed list, and `openSlots` is Boolean. The serialized parent MUST agree with the logical one-level specialization rule.

[RBX-P2-023] Category definitions MAY name a parent Category; Tag definitions MUST NOT use a parent. The taxonomy hierarchy MUST be acyclic. `rb:assets` and `rb:extensions` are portable typed-value containers. Protocol-control containers `rb:actions`, `rb:interfaces`, `rb:hooks`, `rb:packageProfiles` and `rb:coverage` use the dedicated control-JSON representation in 2.16; their structures follow their named contracts. Merely declaring a capability in coverage or action metadata MUST NOT imply that an executable handler exists.

2.9 XML processing and schema use

[RBX-P2-024] Producers MUST emit native XML as UTF-8 XML 1.0. Consumers MUST correctly process UTF-8 with or without its permitted byte-order mark. XML declaration, namespace prefixes, indentation and attribute order do not carry logical meaning. Comments and processing instructions MUST NOT become authored records or execute host instructions.

[RBX-P2-025] Native and component metadata MUST NOT contain DTDs or entity declarations. Consumers MUST disable external entity retrieval, external DTD loading, XInclude processing and network schema retrieval while opening or checking a package. Predefined XML character entities and numeric character references remain supported. A package-supplied stylesheet or code file MUST NOT run merely because XML mentions it.

[RBX-P2-026] A consumer claiming native semantic acceptance MUST validate with the rules of this specification, not trust an arbitrary schema supplied by the document as authority. Supplied XSDs may aid offline checking and are preserved as resources. Unknown native elements or attributes outside a defined extension mechanism MUST cause a clear unsupported/invalid result, not silent loss on a successful save.

The distribution supplies `redbook.xsd`, `package.xsd`, `opc-content-types.xsd` and `opc-relationships.xsd`. The two OPC profile grammars are original compact grammars for this specification's subset, not copies of the ECMA schemas. XSD shape checking does not establish record references, numerical range, cross-part uniqueness, component dependency completeness or actual code behaviour.

2.10 Inventory, resources and integrity

> **[RBX-P2-027]** An `rp:inventory algorithm="sha256"` MUST contain exactly one `part` entry for every package content part except the inventory itself, `[Content_Types].xml`, and all OPC relationship parts. Each entry MUST name its canonical path, exact uncompressed byte count, lowercase 64-digit SHA-256 digest and effective OPC media type. Missing, duplicate, extra or mismatching entries MUST cause integrity validation to fail.

> **[RBX-P2-028]** An original resource MUST be retained byte-for-byte unless an explicit operation replaces that resource. A derived preview, transcript, thumbnail or conversion is a different representation and MUST NOT silently replace the original. Resource identities and descriptors are distinct from package paths. Declared asset size and digest MUST match the corresponding part; a media type MUST not imply a content transformation.

The inventory detects inconsistency, not publisher identity or trust. Because package-navigation metadata is excluded to avoid recursive metadata maintenance, an inventory is not a cryptographic signature over the complete archive. The distribution manifest's whole-file SHA-256 identifies the exact archive bytes. A valid hash does not authorize code execution.

2.11 Standalone component envelope

> **[RBX-P2-029]** A component package MUST use `component/manifest.xml` as its main part and `component/inventory.xml` as its inventory. The manifest root is `rp:component` containing one `rb:map`. Its decoded logical value MUST validate against `component-envelope.schema.json` and the relevant kind-specific schema. It MUST NOT contain a native world main relationship.

Manifest field	Requirement
`formatVersion`	Exactly `1.0`
`kind`	Exactly `blueprint`, `theme`, or `plugin`
`id`, `version`, `title`	Stable component identity, exact version string, human title
`resources`	Complete payload list: `{path, mediaType, bytes, sha256}`
`dependencies`	Required installed components: `{id, version}`; exact version equality
`requires`	Required capability identifiers, without duplicates
`description`	Optional explanatory text
`graphPart`	Blueprint only: package-root-relative JSON graph path

Manifest field	Requirement
`target`	Theme/Plugin only: exactly `reader`, `redbook`, or `viewbox`
`entry`	Theme/Plugin only: package-root-relative entry path
`profile`	Theme only: `css` (default) or `template`
`stylesheet`	Template Theme only: optional declared CSS part
`configuration`	Theme/Plugin only: initial portable configuration map
`configurationSchemaPart`	Optional Theme/Plugin configuration JSON Schema path
`export`	Plugin only: exactly `setup` in this ECMAScript profile

> **[RBX-P2-030]** `resources` MUST cover exactly the content parts other than the main manifest and inventory. Paths MUST be unique; counts, digests and media types MUST agree with both the bytes and inventory. Entries, graph parts and configuration schemas MUST be present declared resources. An installation dependency MUST have a unique component identity and exact version; missing dependencies or capabilities MUST be reported before installation or activation.

> **[RBX-P2-031]** Component manifests MUST NOT contain undeclared fields masquerading as required behaviour. Their configuration values obey the portable-value rules. An optional configuration schema MUST be valid JSON Schema draft 2020-12. Its `$ref` values MUST be same-document fragments: the empty fragment, JSON Pointer fragments or declared `$anchor` fragments. Consumers MUST resolve them only within that retained JSON resource and validate configuration before use. Network, absolute-URI and other-resource references are outside this profile, even when a resolver happens to have a cached copy. Nested `$id` or dialect changes, and dynamic/recursive-reference keywords, require a separately identified schema profile; an ordinary `$ref` cycle is handled within declared evaluation limits. Literal `$ref` or `$rb` property names inside schema data remain legal.

A resolver reports unresolved fragments as invalid and exhausted depth, node or reference budgets as `LIMIT_EXCEEDED`; neither outcome permits skipping validation. A schema outside a declared supported profile is reported as unsupported and retained without silent inlining or simplification. Packaged multi-schema resolution is a separate explicitly negotiated profile. No network lookup occurs while testing profile support.

Component resource paths are package-root-relative, including when the manifest is nested. Main-map paths in a native RedBook remain main-part-relative as described earlier. The distinction is deliberate and removes ambiguity when a graph and its original assets are relocated together.

2.12 Blueprint graph and dependencies

[RBX-P2-032] The `graphPart` MUST use `application/json`, UTF-8 and the logical contract in `blueprint.schema.json`. JSON objects MUST have unique keys, and non-finite numeric literals are forbidden. The graph MUST contain ordered record roots, selected records, required type declarations, taxonomy definitions, explicit binding requirements, asset descriptors and required resource paths. The graph MUST include `exports` with the four identity lists `records`, `types`, `categories` and `tags`; at least one export list or the `components` list MUST be nonempty. `exports.records` MUST equal `roots` in order. A missing `exports` member is invalid; consumers MUST NOT infer exports from carried dependencies. The graph MUST NOT acquire a world identity, central time rule or hidden history merely because it is independently packaged.

[RBX-P2-033] Every graph root MUST identify a selected record. Every explicit registry export MUST identify a carried entry in its own registry. A definition-only or taxonomy-only graph has empty roots and records and explicit nonempty registry exports. A component-only graph has empty roots, records and registry export lists and a nonempty `components` list. A producer MUST NOT invent an Element, Digital or world merely to carry either selection. It has no record bindings unless bundled component configuration or a Viewbox target explicitly requires them. Selected record identities MUST be unique. Every record reference outside the selection, including slot participants, typed property references and retained local author references, MUST have exactly one declared binding requirement. A binding key MUST NOT also identify a selected record. Source author credits are descriptive metadata, not missing local dependencies.

[RBX-P2-034] The exporter MUST close the selected record types, explicit type exports and declared binding types transitively over specialization parents and every `slots.*.allowedTypes` reference. Allowed slot types MUST resolve to carried Element definitions; a dependency may introduce further required definitions. Every carried definition, including an optional extra, MUST have complete internal definition closure. Used Categories and explicit Category exports bring all their ancestors; assigned/exported Tags bring their definitions. The exporter MUST enumerate every required original resource in `requiredParts`, carry its bytes, and provide asset descriptors as applicable. A resource path referenced from a native Property MUST be declared by `resourceReferences` with its source record identity, an ordered property-path list, and canonical package path. Consumers MUST verify that the named property resolves to the declared path. `typeResources` is a list of `{typeId,parts}` declarations naming the source/schema/resource paths required by each carried type implementation. A type identity occurs at most once in that list; it is a value, not a map key, so any legal identity including `$rb` remains representable. `resourceDependencies` maps a carried resource path to its direct resource dependencies. The list and map default to empty when no such dependencies exist. Exporters MUST apply their declared language or feature resolver to populate these maps and walk them to a fixed point; every referenced path MUST occur in `requiredParts`. Module cycles do not duplicate resources. An exporter unable to enumerate required source, template, stylesheet or schema dependencies MUST report incomplete capture. A checker of declarations alone does not prove that an arbitrary opaque implementation supplied truthful dependencies. Arbitrary substring scanning of strings is not a conforming dependency-discovery method.

[RBX-P2-035] An opaque feature that can contain embedded identities or resource references MUST supply an explicit dependency/rewriting contract or be treated as opaque non-rewritable data. If a requested import cannot preserve those dependencies, it MUST fail with an actionable unresolved-dependency result rather than guess replacements. A passive archive copy can still preserve that opaque payload unchanged.

[RBX-P2-036] Instantiation MUST use fresh destination record and taxonomy identities, explicit destination bindings and the chronology allocation rules in Parts 1 and 4. Package validation itself MUST NOT instantiate the graph. Receiving required bytes and committing the graph MUST be coordinated atomically; failed validation MUST leave the destination's authored state unchanged. Resource-bearing imports MUST accept the host-staged component bytes and allocate an exclusive `redbook/imports/<fresh UUID>/` resource root under Part 4. Each original part path maps beneath that root without changing its bytes. Only explicitly declared resource references and dependency metadata are rewritten through `resourceMapping`. The standard `settings.resourceImports` declaration retains the resource root, mapping and mapped dependency metadata for save/reopen; it is current resource state, not an automatic history. A graph without resources has null `resourceRoot` and an empty resource mapping. Bundled components use the explicit installation procedure in 2.19; carrying arbitrary source resources alone MUST NOT install or activate them.

2.13 Theme and Plugin targets

[RBX-P2-037] A Theme or Plugin MUST target exactly one of Reader, RedBook or Viewbox. Different target editions are distinct components, even when their source bytes are similar. A Reader component MUST NOT be silently embedded into every opened RedBook. RedBook and Viewbox installations travel only with their owning authored world when explicitly installed there.

[RBX-P2-038] A Theme with absent `profile` or `profile:"css"` MUST have a UTF-8 CSS entry with media type `text/css`. A Theme with `profile:"template"` MUST have a UTF-8 JSON entry with media type `application/json`, conforming to `templates.schema.json` and the declarative layout/rendering contract in Annex D. Its optional `stylesheet` MUST name a declared UTF-8 CSS resource. A Plugin entry in the ECMAScript profile MUST be UTF-8 module source with media type `text/javascript`, exporting synchronous `setup(api, context)` as specified in Part 4. All local imports and local resource references needed by the entry MUST resolve to declared resources. Required network or host capabilities MUST be explicit; their names do not grant permission.

[RBX-P2-039] Installation MUST preserve the complete source and resource payload and MUST remain distinct from activation. A consumer MUST NOT evaluate Plugin code during ZIP inspection, schema validation, preview metadata extraction, or a plain copy. Activation, API revocation, hook ordering, cleanup and runtime state are governed by Part 4 and Annex D. Installation MUST preserve component-relative paths and bytes under its own virtual resource root. The world stores payloads beneath `redbook/installations/<resourceKey>/`, where `resourceKey` is the separate UUID resource key defined in Annex D, not an arbitrary installation identity pasted into a filesystem path. Relative module imports, CSS URLs and template resources resolve within the installation; another installation using the same relative filenames cannot replace them. Persisted manifests retain their component-relative resource names. Live `rbx:` scope tokens are host handles and MUST NOT enter authored state. Dependency scope, desired `enabled` state, `order`, replacement and removal follow Annex D.

CSS themes and declarative template themes share the same target isolation. A theme cannot gain Plugin authority by carrying script bytes. The conformance examples and their execution limits are indexed in the informative distribution README.

2.14 Preservation, saving and failure

[RBX-P2-040] A preservation-capable consumer MUST retain accepted unknown content parts byte-for-byte, together with their effective media types and additional OPC relationships. It MUST preserve the ID, type, mode and original target spelling of relationships it does not explicitly change. Reserializing a known logical part is permitted only while preserving its required values and semantics; unchanged opaque parts MUST NOT be decoded and re-encoded as a convenience.

[RBX-P2-041] A save MUST stage changed native parts and resource descriptors, refresh affected byte counts and inventory digests, validate the resulting package, and commit atomically to the selected output. Failure MUST leave the previous destination intact. A supplied path or reference MUST NOT authorize overwriting another world. Reading, validation and successful saves MUST NOT fabricate authored events, change timestamps or add automatic edit histories.

[RBX-P2-042] Consumers MUST preserve unfamiliar content they accept, or clearly refuse a mutation they cannot preserve. They MUST NOT report successful lossless round-trip support after dropping extensions, replacing original resources with previews, conflating absent and null values, truncating numbers, or silently normalizing exact lexemes. A host's temporary transaction snapshot, retry cache or revision token MUST NOT leak into the portable authored model.

2.15 Layered package and component checking

A package-profile check establishes the RedBook ZIP subset, canonical names, OPC relationships and internal resolution, native XML shapes, inventories and media types. Component checks additionally establish manifest shapes, resource coverage, explicit exports and declared graph dependencies. They do not activate carried code. Full core-data semantics and operation behaviour require the additional layers in Part 5.

The distribution includes examples of each target/kind pair, a record Blueprint and a definition-only Blueprint. The examples demonstrate particular representations, not universal execution. The informative assessment identifies which helpers, source modules and host behaviors were exercised.

> **[RBX-P2-043]** A conformance report MUST name the checks actually performed, their inputs and any omitted semantics. Successful component envelope checking MUST NOT be described as successful Plugin execution, complete Reader implementation or certification by an external standards organization.

2.16 Protocol-control JSON in XML

> **[RBX-P2-044]** Within `rb:actions`, `rb:interfaces`, `rb:hooks`, `rb:packageProfiles` or `rb:coverage`, a producer MUST serialize protocol-control JSON as the text of exactly one `rb:json` child. The text MUST be valid JSON with unique object member names and without non-finite numbers; XML escaping preserves that JSON text. The enclosing XML is UTF-8. Object order and insignificant JSON whitespace do not change the control value. This is a context-specific representation, not a new portable Property kind.

For example, a custom action schema can literally describe a typed reference:

```
<rb:actions xmlns:rb="https://redbook.fun/ns/redbook/1.0">
  <rb:json>[{"name":"example.inspect-reference","inputSchema":{"type":"object","properties":
{"$rb":{"const":"reference"},"id":{"type":"string"}},"required":
["$rb","id"],"additionalProperties":false}}]</rb:json>
</rb:actions>
```

This abbreviated walkthrough shows the encoding, not a complete executable action descriptor. Complete descriptors also declare their output schema, role, effect and requirements. A literal `$rb` key inside that schema is ordinary JSON Schema data. The same key in authored Properties remains reserved by Part 1.

[RBX-P2-045] `rb:json` MUST NOT occur in a record Property, type field/default, manifest configuration or another portable-value holder. Every control container listed in 2.16 MUST contain exactly one `rb:json` child; a typed-value child is not this edition's control grammar. A separately identified archival inspector MAY decode a control encoding outside this grammar when that decoding is unambiguous. It MUST identify that scope and MUST NOT report the source as satisfying the native grammar. Archival inspection MUST NOT silently rewrite the source.

[RBX-P2-046] Control-schema resolution MUST use the owning schema resource's declared absolute `$id`, or its package-local resource identity when absent, as its base. The specification's absolute schema identifiers resolve to the matching locally distributed schema documents. An implementation MUST NOT fetch an unresolved reference during discovery, validation or passive reading. A custom action may refer to a declared package schema resource under an explicitly supported offline resolver profile; absent that profile the action is unavailable with its unresolved dependency identified. This broader control-schema contract does not weaken the bounded component-configuration profile in 2.11.

2.17 Definition compatibility and independent reuse

[RBX-P2-047] Type compatibility for Blueprint installation MUST compare semantic projections using Part 1 typed equality. The projection excludes only top-level `label`, `description` and `area`, plus the `description` member of each field rule. It includes identity, key, primitive, level, parent, traits, open-slot policy, all slot constraints, field kinds, requiredness, defaults and choices. A Property named `description`, or a default map containing `label`, remains semantic data. No recursive deletion of annotation-looking keys is permitted.

[RBX-P2-048] On a compatible identity collision the destination MUST retain its existing display annotations and semantic definition. On a new identity it MUST copy the source definition and its annotations. An incompatible semantic projection MUST produce `TYPE_CONFLICT` before any import commits. The source Blueprint remains unchanged. Imported Category and Tag identities are independently allocated under Part 4; equal text labels do not imply identity merging.

A definition-only import adds definitions and selected taxonomy without manufacturing record roots or allocating event chronology. Dependency-only types remain available to interpret the exported definitions; the explicit export list describes the reusable public selection rather than every internal dependency. Any advertised complete-core claim is assessed against the precise declared edition, never against an obsolete definition merely because its identity string matches.

2.18 Portable participant-authoring declarations

> **[RBX-P2-049]** A RedBook- or Viewbox-targeted Plugin using `rbx.participant-authoring.1` MUST carry the declaration named by its `authoringPart` as an inventoried UTF-8 JSON resource. The declaration, custom type definitions and every required implementation resource MUST travel with the installed component under its retained payload root. Declaration inspection validates shape and dependencies without executing code. Its ordinary installed `enabled` setting expresses desired activation; the portable participation policies and declared mutation scope govern world authoring. The receiving host separately permits execution and representation of a Digital. A Theme or Reader-targeted component MUST NOT acquire world mutation authority by carrying authoring declarations.

> **[RBX-P2-050]** A save/reopen or passive copy MUST preserve accepted participant declarations, source bytes, current participation policies and installed configuration together with their compatible type definitions. Blueprint capture of functionality depending on these resources MUST include them in its declared resource closure. If an identity-changing transfer cannot faithfully map scope/type/action dependencies and explicit Digital policy references, it MUST fail as incomplete capture or unsupported rewriting; it MUST NOT retain a misleading apparently approved action. Unknown execution support does not invalidate byte-preserving passive storage.

2.19 Bundled functionality in a Blueprint

A Blueprint can carry the behavior needed to continue working with its custom records. Its optional `components` list contains entries of exactly this shape:

```
{"key":"measurement","packagePart":"components/measurement.redxplugin",
 "target":"redbook","configuration":{}}
```

`key` identifies the component within this Blueprint and is unique within the list. `packagePart` names a complete independently valid Theme or Plugin package retained as a payload resource in `requiredParts`. It is not a path to executable code within the outer package. `target` equals the inner manifest's `redbook` or `viewbox` target. A Viewbox entry additionally requires `targetId`, identifying a selected Viewbox or a declared external binding. A RedBook entry omits `targetId` and installs into the receiving world. Reader-targeted components cannot travel as world-owned installations. `configuration` is the installed portable configuration override, validated against any carried configuration schema.

[RBX-P2-051] A functionality capture MUST include the selected component's complete package, compatible declaration types, required dependency components and every required policy or explicit identity binding. Source, declaration and original resource bytes MUST remain unchanged. World-specific identities MUST be represented through typed references in records or component configuration, or through an explicitly supported rewriting contract. Typed references in configuration and Viewbox target identities participate in the same exact binding closure as record references. Arbitrary identity-looking source strings MUST NOT be searched and replaced. A generic declaration that uses type/action identities and runtime inputs can therefore travel unchanged while its world policies and configurations are mapped normally.

[RBX-P2-052] Import MUST validate the nested packages, compatible type registry, complete dependency closure, bindings, configuration, targets and resulting records before committing any change. Each bundled entry receives a fresh installation identity, resource key and exclusive installation root under 2.13. All inner manifest and resource bytes are retained at their component-relative paths. An existing installation with the same component identity MUST NOT be silently reused or overwritten. The result's `componentMapping` is an ordered list of `{key,installationId}` pairs, one per bundled entry in the same order; an empty component list yields an empty list. Keys are values rather than map member names, so a legal key such as `$rb` remains representable. The imported installations have `enabled:false`. Explicit configuration may later request activation; receiving-host execution permission remains separate and no grant, active realm or retry state is imported.

[RBX-P2-053] Save/reopen MUST retain the imported graph, mapped policies/configurations, compatible types, installed component manifest and source resources together. A capable receiving host MUST be able to activate and continue the declared public actions after its own execution decision, subject to the same portable participation rules. Re-export of that functionality MUST reconstruct a valid complete component package from retained manifest/resources and include fresh explicit graph bindings. ZIP compression bytes and packaging metadata may differ on reconstruction; the retained inner manifest, declaration, implementation and resource bytes MUST NOT differ. Re-export MUST NOT require access to the original Reader, original filesystem paths, or an undocumented original-host grant.

3

PART 3

Part 3 — Core vocabulary, Communication, Soma and Time

The supplied vocabulary and its meaning: identity, communication, participation, conceptual structures and RedBook Time.

3.1 Scope and status of the supplied vocabulary

3.2 Common domain conventions

3.3 Digitals, profiles and participation

3.4 Publication, Walls and conversations

3.5 Correspondence, Mailboxes and newsletters

3.6 Responses, original material, Works and representations

3.7 Collections, feeds and discovery

3.8 Attention, notification and presence

3.9 Polls, questionnaires, proposals and reviews

3.10 Requests, commitments, cases and reservations

3.11 Events, recurrence and gatherings

3.12 Streams and live exchange records

3.13 Participation rules and moderation

3.14 Soma: conceptual homes, worlds and representations

3.15 RedBook Time and event observations

3.16 Exact world-time evaluation

3.17 Participant conceptual authoring

3.18 Explicit state transitions and organization

3.19 Deterministic correspondence threads

3.20 Newsletter issue allocation

3.21 Response policy and standard rules

3.22 Recurrence invocation and pagination

3.23 Feed selection before paging

3.24 Participant initiation and descriptive authoring

3.25 Specialization fields and ordinary participant actions

3.1 Scope and status of the supplied vocabulary

This part specifies the meaning of the supplied Custom Element Types (CETs), Custom Relationship Types (CRTs), their one-level specializations, and the domain rules required when those meanings are claimed. It uses the logical model in Part 1. The complete field and participant-slot contracts are in Annex A and `catalogues/core-types.json`. Part 4 defines the callable operations and execution contracts. A data consumer can preserve a valid Poll without offering a voting interface; a consumer claiming to execute the standard voting operation must implement its domain behavior.

> **[RBX-P3-001]** A complete-core-vocabulary document MUST carry the 169 supplied declarations identified by the catalogue: 61 root CETs, 54 root CRTs and 54 specializations. A supplied identity MUST retain the meaning and effective constraints specified here. Additional custom types and Properties MAY extend the document without inventing another primitive or a third inheritance level. The catalogue's `area` is an organizational label, not a type ancestor. Display names MAY be translated or changed without changing identity.

> **[RBX-P3-002]** A core-data consumer MUST interpret field kinds, enumerated values, participant constraints, ownership and references under Part 1 and this part. A consumer MUST NOT claim a domain operation solely because it recognizes the type name. Capability discovery MUST distinguish preservation, validation, presentation and execution. Reading a definition MUST NOT instantiate its type, activate code, create a conceptual home, open a network connection or start AI work.

The supplied scope includes identity and participation; publication and Walls; conversations and correspondence; annotations and circulation; original media and composed Works; collections and discovery; attention and presence; polls, questionnaires, decisions and reviews; requests, commitments and resources; Events and Gatherings; streams; moderation; Soma; and presentation. These are reusable communication structures. A research log can specialize a Wall, and a scientific observation can specialize a Post. Their use does not require a social-network product or a fictional social population.

> **[RBX-P3-003]** A consumer MUST preserve the distinction between a record and its associations. Placement does not copy a Post; a Delivery does not duplicate its Mail Message; a Membership does not create a Digital; an Authorship relationship does not rewrite record attribution; and a Represents relationship does not turn its concept into its subject. Unrelated operations MUST NOT add those associations implicitly.

3.2 Common domain conventions

3.2.1 Declared fields and content maps

> **[RBX-P3-004]** The catalogue's field constraints apply when a field is present; required fields are additionally required to be present. Creation defaults are applied by the standard creation contract. Parsing MUST preserve absence and MUST NOT insert a default into stored content. An operation that requires a missing Property MUST reject the operation unless its action contract specifies a fallback. Additional map members MUST be preserved; a consumer MUST NOT guess executable semantics for an unrecognized member.

The following common content-map members have standard meaning when used in a `body` Property. All are optional; a media-only or structured-only body is possible.

Member	Value and meaning
`text`	String containing authored text.
`format`	`plain`, `markdown`, or `html`; absence means `plain` for the `text` member. This names source representation, not permission to execute it.
`language`	A BCP 47 language tag, if an authored language declaration is available.
`references`	Ordered list of typed references, or maps with a required typed-reference `target` and optional string `label`. Other members are preserved.
`attachments`	Ordered typed references to existing Media, Works or Neo Somas. An Attachment relationship is the richer form when the association has independent classification, position or authorship.

> **[RBX-P3-005]** A standard body consumer MUST validate recognized members against this table. It MUST treat HTML as authored source, not as authority to execute script or fetch remote resources. Rendering and explicit resource activation follow Part 4. Markup processing MUST preserve the original source. A body reference MUST resolve according to Part 1 and MUST NOT be interpreted as a request to import, duplicate or contact its subject.

`appearance`, `details`, `outcome` and other open maps are portable authored content. A recognized profile may assign them additional structure. A Location's standard coordinate form is `{system, values, unit?}`: `system` is a nonempty coordinate-system identifier, `values` is an ordered finite-number list, and `unit`, when supplied, is a string. A consumer does not infer latitude/longitude, axis order or distance units from unlabeled numbers. Empty maps remain meaningful absence of further detail.

3.2.2 Association order and current state

[RBX-P3-006] A stored participant list MUST retain its order. Unless its domain explicitly assigns ordering significance, that order alone MUST NOT establish rank, authorship share or authority. For a list of association records ordered by a numeric `position`, compare positions ascending; ties are broken by record identity in Unicode scalar-value order. Pinned Wall placements precede unpinned placements, with the same ordering within each group. Numeric placement coordinates are separate from exact event sequence strings.

[RBX-P3-007] A current-state relationship keyed by a subject and Digital, such as a Ballot or Subscription, MUST NOT have conflicting simultaneously effective records for the same standard key. A specialized type can declare a further key component. The standard setter replaces the current value for its key without retaining a previous version. A multi-participant relationship MUST NOT be overwritten as though its entire participant list were a singleton. An operation addressing one participant MUST preserve the other participants' current assertions, using an explicit atomic split where necessary or rejecting the operation when dependent references prevent an unambiguous split.

[RBX-P3-008] A domain transition MUST validate its preconditions and commit all dependent records atomically. Failed transitions MUST leave records, ownership, classifications, event counters and resource declarations unchanged. The tables below describe semantic transitions, not an obligation to store transition history. Administrative reconstruction or import validates the resulting current state; it does not invent intermediate events to explain that state.

3.3 Digitals, profiles and participation

A Digital is an actor to whom a human or external AI attributes activity. It is not itself a process, account credential or proof of personhood. The supplied Person, Animal, Thing, Collective and Concept specializations label the intended kind of actor. A Profile is a separate authored presentation connected with Profile Of. A Digital Collective and a Group remain different: one is an acting identity; the other organizes membership and material.

[RBX-P3-009] Every Digital MUST own exactly one personal Soma Neocortex under 3.14. Creating a Digital MUST NOT implicitly create a Profile, Wall, Mailbox, biography, communication session or AI connection. A Profile Of relationship MUST associate its Profile with its actual subject. More than one Profile MAY describe a subject; no profile is an authentication record.

Groups, Spaces and Pages are shared contexts. A Circle is a named selection of Digitals and is not, merely by being a Circle, a shared conceptual context. Follows express one-sided interest; Connections express a proposed or accepted relationship among their Digital participants. Neither grants Membership by implication.

[RBX-P3-010] Effective participation in a context requires an explicit active Membership or active Participation relationship naming that Digital and context, or a domain-specific active role grant expressly recognized by the operation. `invited`, `requested`, `declined`, `left`, or any state other than `active` MUST NOT confer an active participation grant. An unrecognized role string MUST NOT confer additional powers. An Authorship relationship is credit, not an authority grant.

The base context policy may use `participation.enrollment` with `invitation` or `open`. Absence means `invitation`. This regulates ordinary Digital-context enrollment, not the ability of the file holder to administer the document. The context's attributed author and the RedBook administrator can maintain its Memberships; a delegated role can do so only through an explicitly declared policy implemented under Part 4.

[RBX-P3-011] The standard membership operation MUST use these transitions and preserve independent participant states:

Actor and precondition	Permitted effect
Context author or administrator	Create an invited or active membership; change a selected membership's state and role.
Named Digital with an invited membership	Accept to `active` or decline to `declined`; retain the granted role.
Active member acting for itself	Leave to `left`; it cannot elevate its own role.
Unenrolled Digital in an explicitly open-enrollment context	Create its own active ordinary `member` membership.
Unenrolled Digital in an invitation-only context	Request membership; `requested` does not activate participation.

A member's withdrawal does not remove its earlier contributions, source credits, or personal Neocortex. There is at most one standard Membership assertion per `(context, Digital)`; a multi-member record applies its common state and role to each listed member.

[RBX-P3-012] An Invitation MUST name its inviter, invitees and context explicitly. An invitee response MUST affect only that invitee. For a multi-invitee invitation, a standard response either atomically separates the responding invitee's assertion from the remaining invitation or rejects with an explicit ambiguity/dependency error. It MUST NOT claim that other invitees accepted or declined. A cancelled invitation cannot be accepted. Inviter cancellation leaves already authored contributions intact.

[RBX-P3-013] A participant transition of a Connection from `proposed` to `accepted` or `declined` MUST record explicit counterpart decisions. `rb.content.create` proposes the selected initiator's Connection. `rb.connections.respond` records only its selected counterpart's `accepted` or `declined` response; administrative use explicitly identifies the counterpart whose decision is asserted. The `responses` array stores `{digital: typed-reference, response: "accepted"|"declined"}` entries, unique by counterpart and ordered as the `counterparts` slot. An identity is a reference value, never a map key. Initiator and counterpart sets MUST be disjoint. The aggregate is `declined` if any counterpart declines, `accepted` only when all accept, and otherwise `proposed`. A response cannot reopen a terminal Connection. Ending retains participants and their recorded responses. Friendship, Affiliation, Mentorship and Introduction inherit this rule; acceptance grants no unrelated consent or Membership.

A retained Connection lacking `responses` preserves its authored aggregate as an explicit administrative assertion without inventing individual decisions. Such a proposed Connection begins with no decisions when the first explicit response is made. An explicit array must agree with the aggregate, except that `ended` retains the decisions preceding termination. Readers must not manufacture unanimous responses from the word `accepted`.

A Digital can have multiple authored presentations or contextual roles. Reader selects a Digital participation context separately from host authentication. This part governs the meaning of an act taken in that context; it does not assert that a file holder is cryptographically unable to inspect or administer other material.

3.4 Publication, Walls and conversations

3.4.1 Publications and placement

Posts contain authored contributions. Article, Journal, Status, Story and Observation are supplied specializations. Their type does not force them into a particular Wall or imply that the contribution is true. A Wall is a context with its own shared Neocortex and an ordered set of placements.

[RBX-P3-014] A Post's ordinary transitions are `draft` to `published`, `published` to `withdrawn`, and `withdrawn` to `draft` for deliberate revision. These transitions require its attributed author or the administrator unless an explicit delegated editing policy applies. `publishAt` and `expiresAt` express intended coordinates; reaching those coordinates MUST NOT by itself mutate the Post. Creating, publishing or withdrawing a Post MUST NOT create or delete placements unless the invoked operation expressly includes those changes.

[RBX-P3-015] A Wall Placement MUST reference the actual Wall and existing item Elements. Placing a Post requires its current state to be `published`. The placement has its own identity, attribution, classifications, numeric position, pinned flag and state. The standard placement operation uses the Wall's contribution policy as follows:

Policy	Non-owner placement
`open`	Accepted as `published`.
`members`	Requires effective participation, then `published`.
`reviewed`	Accepted as `pending`; only an authorized reviewer publishes or rejects it.
`closed`	Rejected.

The Wall's attributed author or administrator can place content directly as `published`. The policy controls placement; it does not grant rights to edit the source item or contribute to the Wall's Neocortex. Shared conceptual contribution still requires a participation grant.

> **[RBX-P3-016]** Standard Wall enumeration MUST omit pending and rejected placements unless an explicitly requested review projection includes them. For `manual` ordering it MUST use 3.2.2. For `chronological` ordering the creation operation assigns the initial numeric position from the item's event sequence when exactly representable, otherwise from the next safe local placement position. Explicit positions remain authoritative and are never retrospectively rewritten after Blueprint import. Failure to find a safe default position MUST be reported instead of rounding. Within a placement, item order is the stored slot order.

Withdrawal changes the source Post, not the stored placement. A presentation may show a withdrawn-source indication; it must not pretend that the original published body is still the current source. Removing a placement does not remove its source. Reposting through Circulation and placing an item on a Wall are separate associations that may coexist.

3.4.2 Conversations and contributions

> **[RBX-P3-017]** Creating a Conversation through the standard participation operation MUST create its shared Neocortex and active participant Memberships atomically. The initiating Digital receives the `organizer` role; additional explicitly supplied participants receive `member`. Duplicate supplied participants are coalesced by identity. Conversation specializations retain the same ownership and participation contract; `direct` does not silently restrict the number of participants to two. The generic `rb.communication.start` action materializes the selected type's own defaults: an Event starts `planned`, a Gathering starts `open`; it must not force every communication context into the Conversation state vocabulary.

> **[RBX-P3-018]** Sending a Message MUST require effective participation and an `open` Conversation. It MUST create the Message, its Contribution association, record attribution and the event fields required by Part 1 in one transaction. A reply target MUST already be a Message contributed to that Conversation. Its Reply association names the new reply and actual antecedent; it does not copy antecedent text. A Mention identifies an expressly mentioned Digital and does not automatically create a Notification.

> **[RBX-P3-019]** Standard conversation enumeration MUST return contributions in exact ascending event sequence order, independent of array enumeration or wall-clock timestamp equality. A contributed operational Message MUST have an event sequence. Repeated contribution links MUST NOT cause the same Message to appear multiple times in this enumeration. Closed Conversations remain readable. Reopening is an explicit author/administrator operation. A Message correction replaces its current body without generating a hidden prior-body archive, changing original event coordinates, or falsely attributing the correction to another Digital.

A Comment is a separate response with an open/resolved state; an Accepted Answer relationship is an explicit selection, not an inferred truth rating. Question, Answer and Summary Message specializations can be interpreted without knowing their presentation.

> **[RBX-P3-020]** Selecting an Accepted Answer MUST be authorized by the attributed question author or administrator. Standard selection replaces the selection for that question and retains answer content and credit. A standard discussion split MUST explicitly select contributed Messages, move the relevant Contribution associations atomically, and preserve their identities, attribution, Reply links and event coordinates. It MUST NOT silently move personal or shared conceptual models. A Reply that now crosses the resulting Conversations remains a historical source relationship, but a newly authored reply must meet 3.4.2's same-Conversation rule.

3.5 Correspondence, Mailboxes and newsletters

3.5.1 Addressing and recipient projections

A Mail Message is a correspondence item. Delivery supplies its sender, individual recipient, recipient role and delivery state. Mailbox Entry supplies recipient-specific organization. A mail-thread Conversation is the conceptual context of the exchange; it does not replace the Mail Message or its Deliveries.

> **[RBX-P3-021]** A standard internal send MUST select at least one recipient and disallow repeated recipient identity across To, Cc and Bcc. It MUST create one Mail Message, select or create its exact correspondence context under 3.19, create only the public audience Memberships required by that clause, and create one Delivery and Mailbox Entry per recipient atomically. It MUST NOT claim delivery to external email addresses, networks or humans. The delivery state `delivered` means the internal recipient association and entry were authored successfully.

> **[RBX-P3-022]** A personal Mailbox MUST have exactly one Digital owner; a collective Mailbox MUST have one or more explicit owners. A standard send may create the recipient's personal Mailbox when absent, but MUST reject an ambiguous choice among multiple personal mailboxes unless a destination is explicitly selected. A Mailbox Entry's recipient MUST be an owner of its Mailbox. Its message MUST have a corresponding Delivery to that recipient. A delivery to a Newsletter Issue uses the same organization rules.

[RBX-P3-023] A Digital-context correspondence projection MUST show the sender and To/Cc recipients. A Bcc recipient may see its own Bcc relationship; the sender may see the full recipient arrangement. It MUST NOT expose the other Bcc recipients through that projection. Administrative file inspection is a separate capability. This projection rule MUST NOT be described as encryption or erasure of the underlying records.

[RBX-P3-024] Reply selects the original sender; reply-all selects that sender plus original To/Cc recipients, removes the replying Digital, and coalesces duplicate identities. Bcc addresses MUST NOT be promoted into the reply-all set. Replying as a Bcc recipient does not disclose other Bcc recipients. Explicit new recipients are a new authored addressing decision. A reply preserves a reference to its actual antecedent and selects its correspondence context using the deterministic audience and thread-mode rules in 3.19, without manufacturing previous messages.

3.5.2 Organization, templates and distribution

[RBX-P3-025] Reading, listing and rendering Mailbox content MUST NOT set `read`, `acknowledged`, `starred` or snooze fields. Those are explicit per-recipient choices. Labels are unique strings. Moving an entry to a Mail Folder changes the entry's organizational reference, not the Mail Message. Deleting a folder requires explicit removal or reassignment of its entry references; it MUST NOT delete the messages. A Mail Handling record identifies responsibility for a collective-inbox item and does not change its original sender or recipients.

A Distribution List is a reusable selection. A send expands its selected recipients at the time of the operation and writes explicit Deliveries. Subsequent edits to the list do not alter earlier recipient arrangements. A Signature is authored sign-off content, not a cryptographic signature.

[RBX-P3-026] A Correspondence Template declares unique string variable names. The standard template renderer substitutes exact `{{name}}` tokens in its subject and text with explicitly supplied string values; it MUST reject an undeclared token, missing required substitution or non-string value, and MUST NOT evaluate code, recursive substitutions or property access. Literal text outside tokens is preserved. Template use and Signature insertion create ordinary authored correspondence, not a continuing live link that rewrites past sends.

[RBX-P3-027] A Newsletter Issue MUST have an explicit Newsletter Entry identifying its series. Positive issue numbers MUST be unique within that series; zero means not numbered. Standard publication allocates from the persistent `nextIssueNumber` frontier under 3.20, rather than recomputing a number from retained issues or reusing a deleted ordinal. Publishing MUST NOT deliver the Issue automatically. Explicit distribution selects enabled Subscriptions, applies supported declared filters, and creates at most one Delivery per `(Issue, recipient)`. Repeating distribution does not duplicate successful existing Deliveries. Unknown required filter semantics MUST cause an explicit unsupported-capability result, not unfiltered delivery.

A Submission has an independent pending/accepted/declined/withdrawn decision and a Submission Target associating actual content and destination. Acceptance is a review decision; placement or publication must be expressly included in the accepted operation. Source content remains separately authored. Newsletter submissions, feedback, ordinary Mail Rules and recipient preferences use the same current-record and explicit-processing principles.

3.6 Responses, original material, Works and representations

> **[RBX-P3-028]** A Reaction MUST name its acting Digitals and target Elements. The standard current reaction key is `(concrete reaction type, acting Digital, target)`. Updating a singleton reaction changes its current value; disabling it retains its identity. Ratings, acknowledgments, recommendations and endorsements MUST NOT be interpreted as each other. No reaction by itself establishes reading, task acceptance, factual truth or consensus.

> **[RBX-P3-029]** An Annotation MUST remain separate from its targets. The standard selector forms are:

kind	Required members and interpretation
whole	No coordinates; the entire target. An empty selector map has this meaning.
text	Integer `start` and `end`, with `0 ≤ start ≤ end`, indexing Unicode scalar values in the explicitly identified text representation; the interval is half-open. Optional `quote` retains authored anchoring text.
region	Finite `x`, `y`, nonnegative `width` and `height`, plus `unit` (`pixel` or `normalized`); normalized coordinates satisfy `x ≥ 0`, `y ≥ 0`, `x + width ≤ 1` and `y + height ≤ 1`. The target representation is identified explicitly.
time	Finite nonnegative `start` and `end` seconds, with `end ≥ start`, on the identified media timeline.

An ambiguous text or media representation requires an explicit typed-reference `representation` member. When the selected source text is available, `end` cannot exceed its Unicode scalar-value length. If later source edits make coordinates or quoted text disagree, a consumer MUST report an unresolved/stale selector rather than silently reanchor it. An unsupported selector kind is preserved and reported as unsupported when interpretation is requested.

> **[RBX-P3-030]** A Circulation association MUST preserve its actual source references, acting Digital credit, optional destination and authored commentary. Repost, Quote, Forward, Crosspost and Derivation do not imply that source authorship transfers to the circulating actor. `quotedText` is independent authored content, not an automatically maintained substring. Any content copy must be an explicit operation with source credit and dependency handling under Parts 1 and 4.

[RBX-P3-031] A Media `asset` MUST resolve to declared original resource bytes under Part 2 when the Media claims to have a local original. Its declared media type and descriptive metadata MUST be preserved independently of generated previews or decoded presentation. An Attachment links existing material; it does not move its conceptual ownership. Original bytes MUST NOT be replaced by a thumbnail, transcript, transcoding or source-code rendering merely because that is what the current host can display.

A Representation is separately authored material such as a translation, transcript or caption set, linked by Representation Of. Standard cues are maps with finite nonnegative `start`, `end ≥ start` in media seconds and string `text`; cue lists retain authored order. A language declaration labels that representation, not all its sources.

[RBX-P3-032] A Work is a composed whole, not a compulsory superclass of other content. Documents, notebooks, canvases, datasets, knowledge pages, artifacts and deliverables inherit Work semantics. An Edition MUST be deliberately captured as its own record and explicitly associated with its source; ordinary edits MUST NOT create Editions. A Suggestion retains proposed content until an authorized operation accepts and applies it. Accepting a Suggestion and applying its proposed patch MUST be atomic when presented as one operation. A rejected or withdrawn Suggestion does not modify its source.

[RBX-P3-033] A Reference CRT with open slots MUST retain its authored role names and `meaning`; it MUST NOT be coerced into a binary source/target edge. Relationship participants remain Elements. To annotate or refer to a Relationship itself, a consumer MUST use a typed-reference Property or a declared representation Element and MUST NOT insert a Relationship identifier into an Element-only slot. This preserves the two-primitive model without denying that either primitive can be described.

3.7 Collections, feeds and discovery

[RBX-P3-034] A Collection Entry MUST reference its Collection or Agenda and existing item Elements. Sections organize entries but do not own the item records. Deleting an entry or section MUST NOT delete its items. A standard explicit reorder MUST name each retained entry exactly once and assign the requested ordering atomically; an omitted, unknown or repeated entry is an error. Albums, Boards, Portfolios, Playlists and Reading Lists inherit these rules.

[RBX-P3-035] A Feed or Saved Query MUST preserve its query descriptor as authored content. Evaluation MUST apply the query and ordering contract in Part 4 against a specified document snapshot; it MUST NOT store result copies unless separately requested. For chronological feeds, records with event sequences sort by exact sequence, while non-event records precede them and are ordered by identity; reverse-chronological reverses that complete order. Title order compares Unicode scalar values, then identity. Locale-dependent display sorting may be offered separately but MUST NOT be mislabeled as the standard deterministic query result.

A Bookmark is a current personal saved reference and note. The presence of a Bookmark or Reading List item does not mark that content read. Topical associations, ratings or follows may inform an explicitly selected feed profile; there is no implicit universal recommendation algorithm.

3.8 Attention, notification and presence

> **[RBX-P3-036]** Notifications and Reminders MUST identify their recipients and optional source material through Notice Target. A Notification's existence is an authored notice, not proof of delivery to a device or observation by its recipient. An Attention record MUST be produced only by an explicit act identifying its Digital, target, kind and optional position. `focus`, `read`, `acknowledged` and `dismissed` are distinct assertions. Reading content or evaluating a Feed MUST NOT create any of them.

> **[RBX-P3-037]** A Presence statement MUST be treated as an authored availability/composing assertion, not as evidence of a running AI, connected client or actual human attention. A supplied validity endpoint may suppress it from a current presentation, but MUST NOT cause a hidden document edit. Absence of Presence means no assertion, not offline status.

Standard preference maps can declare `quiet.intervals` as a list of `{clock,start,end}` intervals, `delivery.mode` as `immediate`, `digest` or `none`, and `digest.intervalMilliseconds` as positive decimal text. `filters` use the predicate grammar in 3.13; a list of conditions is equivalent to their `all` combination. An empty map/list adds no constraint. A preference says how a requested delivery or presentation should be treated; it does not start that process.

> **[RBX-P3-038]** A consumer applying a supported quiet or delivery preference MUST evaluate it against an explicitly identified clock and instant, retain the underlying notices, and distinguish suppressed presentation from acknowledgement. It MUST NOT convert a Reminder, Subscription, digest preference or snooze endpoint into a stored scheduler, retry queue or AI wakeup inside the world. Host scheduling is separately configured under Part 4.

3.9 Polls, questionnaires, proposals and reviews

3.9.1 Polls and current ballots

Poll options are ordered maps with a unique nonempty string `id` and a string `label`; additional explanatory members are preserved. Option IDs are local option identifiers, not Element references. Exactly `$rb` is excluded from this local ID position because ratings and scores use option IDs as ordinary portable map keys. This restriction does not exclude `$rb` from ordinary string content or from otherwise valid record/type identifiers. An empty options list is a permitted unprepared Poll, not a votable Poll.

[RBX-P3-039] Creating, editing or accepting a Poll definition, including a specialization, MUST reject an option whose ID is exactly `$rb` with `LOCAL_ID_RESERVED`, before committing the definition or allocating its dependent authored state. Descriptor shape follows `domain-response.schema.json`; local IDs remain unique and nonempty. Casting a ballot MUST require an `open` Poll, at least two distinct options, an eligible Digital and values valid for the selected method. Eligibility means active participation or an explicitly declared open response policy; a closed or invitation-only policy MUST NOT be treated as open merely because the Poll is readable. There is one current standard Ballot per `(Poll, voter)`. Replacing that ballot MUST NOT retain its prior values as an automatic history.

[RBX-P3-040] Ballot validation and tally MUST use the following rules. Choices, rankings and rating-map keys use the same permitted local IDs as the option descriptors; `LOCAL_ID_RESERVED` rejects exactly `$rb`. The rating and score maps obey both their numeric schemas and the portable-value algebra; tool-result or control JSON does not exempt them. Every referenced option exists in that Poll; repeated choices or ranking entries are invalid. Fields belonging to an inactive method are empty.

Method	Valid current ballot	Tally
`single`	Exactly one `choices` entry.	Add one to that option's count.
`multiple`	Between one and `maximumSelections` distinct choices; the maximum is a positive safe integer no greater than the option count.	Add one to each selected option's count.
`ranked`	A nonempty distinct `ranking`, permitted to omit options.	With `N` Poll options, rank index `i` starting at zero scores `N−1−i`; omitted options score zero.
`rating`	A nonempty `ratings` map whose keys are option IDs and whose values are finite numbers.	Sum the authored rating values for each option.

[RBX-P3-041] Tally MUST initialize every option to zero, ignore withdrawn ballots, and return method, included ballot count and scores without modifying the Poll. Ranked tally MUST identify the method as Borda. A tie MUST remain a tie unless a separately declared procedure resolves it; array order is not an implied winner. Numeric accumulation MUST use the ordinary-number arithmetic contract of Part 1 and reject a non-finite result. Closing a Poll requires its author or administrator, stores the explicit current tally and prevents further ordinary ballots. Editing the option set or method in a populated Poll MUST either preserve every retained ballot's validity or resolve affected ballots explicitly in the same transaction.

3.9.2 Questionnaires

A standard question descriptor has a unique nonempty string `id` other than exactly `$rb`, optional string `label`, optional boolean `required` (default false), and optional `type`. The excluded local ID would otherwise become a reserved answer-map key; it remains valid as ordinary string content or an otherwise permitted identity elsewhere. Standard types are `string`, `number`, `boolean`, `array`, `object` and `null`; omission accepts any portable value. These are questionnaire value categories, not CET identities. A declared extension profile can add validated choices or richer constraints.

> **[RBX-P3-042]** Creating, editing or accepting a Questionnaire definition, including a specialization, MUST reject a question whose ID is exactly `$rb` with `LOCAL_ID_RESERVED` before committing the definition or allocating its dependent authored state. Standard descriptor and answer shapes follow `domain-response.schema.json`; answer keys use the same permitted local IDs. A standard Questionnaire Response MUST be keyed by `(Questionnaire, respondent)`. Its answer map MUST NOT contain undeclared question IDs. A draft may omit required answers. Submission requires an open Questionnaire, eligible respondent and each required answer present, non-null and, for a string, nonempty. Present answers MUST match declared kinds; `number` means a finite ordinary number, and tagged exact numbers require a separately declared question profile. Submission replaces the current response rather than archiving drafts. A withdrawn response is retained but excluded from ordinary submitted-response enumeration. A submitted response can be revised only when the Questionnaire's `responsePolicy.reviseSubmitted` is true; omission means false.

3.9.3 Proposals and explicit decisions

Proposals contain an authored proposition; Decision Rounds organize consideration of it; Decision Positions record support, objection, abstention or withdrawal by an explicit participant. A Decision is a separately authored outcome with a stated procedure and optional dissent. A Proposal is not adopted just because someone supported it.

> **[RBX-P3-043]** A current Decision Position MUST be unique per `(subject, participant)` for its concrete type. Only the participant or administrator may replace it through the standard position operation. Closing a Decision Round or adopting a Proposal MUST be an explicit authorized act identifying the applied procedure and result. A consumer MUST NOT infer majority, unanimity, quorum, consent, or adoption from unspecified `conditions`, a mere count of positions, or the title of a procedure. A named procedure is executable only when its declared profile provides its eligible set, quorum, treatment of abstention and dissent, and deterministic result rule. Without such an executable profile the Decision can be recorded as an authored assertion, but MUST NOT be advertised as an automatically calculated result.

> **[RBX-P3-044]** The ordinary Proposal states are `open`, `adopted`, `not-adopted` and `withdrawn`; author/administrator explicitly records the terminal decision or withdrawal. A Decision Round is `open` or `closed`; closure does not delete positions. Reopening is explicit and does not silently reverse an existing Decision. Any revised Decision is a deliberate authored act rather than an automatic retained revision.

3.9.4 Reviews and findings

> **[RBX-P3-045]** A Review MUST name its targets through Review Target and preserve its criteria as authored data. Standard `rb.communication.start` therefore requires nonempty `targetIds` for Review creation and creates that target association in the same transaction; other roots reject that argument. A Finding submitted through the standard operation requires an open Review and effective participation. Finding In links the independent Finding to that Review. `approve`, `changes-requested` and `decline` are the Finding author's dispositions; none automatically alters the target. Closing the Review is an explicit owner/administrator action. `release: on-close` defers the ordinary participant presentation of findings until closure; it MUST NOT be represented as cryptographic confidentiality or erase their attribution.

3.10 Requests, commitments, cases and reservations

3.10.1 Responsibility and progress

> **[RBX-P3-046]** A standard Request MUST have one Request Address identifying its requester and one or more permitted responders. Responding requires an open Request and a named responder. Acceptance creates a distinct Commitment and Commitment For relationship identifying the accepting responsible Digital, and sets the Request to `accepted`, atomically. Decline sets the Request to `declined` and creates no Commitment. The standard Request has one overall disposition: where multiple responders are named, the first accepted valid response resolves that request. Independent per-responder offers require separately authored Requests or a declared extension contract; a consumer MUST NOT guess that an accepted Request remains open to other responders.

Request Offer changes the framing, not the separation of invitation and accepted responsibility. A Commitment's text can be based on the Request, but its current content and attribution are independent.

> **[RBX-P3-047]** Standard Commitment progress MUST use these transitions. Each successful progress act creates a separately attributed Progress statement and Progress On link; these are intentional communication, not automatic edit history.

Kind	Authorized participant	Source state	Resulting state
`update`	Responsible Digital or administrator	`accepted`, `in-progress`, `reported-complete`	`in-progress`
`report-complete`	Responsible Digital or administrator	`accepted`, `in-progress`	`reported-complete`
`verify`	Original requester or administrator	`reported-complete`	`verified`

Kind	Authorized participant	Source state	Resulting state
`reopen`	Responsible Digital, original requester, or administrator	`reported-complete`, `verified`	`in-progress`
`withdraw`	Responsible Digital or administrator	Any state except `withdrawn`	`withdrawn`

Verification is explicitly separate from a completion claim. A withdrawn Commitment cannot receive further standard progress until the administrator explicitly reestablishes a valid current responsibility. A Case associates discussions, evidence and responsibilities through Case Material. Resolving or closing the Case MUST NOT silently mark its Commitments verified or delete its evidence.

3.10.2 Reservable resources

A Resource represents available quantity in an authored unit; it is not the same thing as a package asset. A Reservation has one Resource association and a half-open interval `[start,end)` in a declared clock.

> **[RBX-P3-048]** Requesting or confirming a Reservation MUST require finite positive quantity, finite nonnegative resource capacity, comparable start and end coordinates, and `end > start`. The clock coordinate syntax is specified in 3.15. Different clock bases or units MUST NOT be silently compared or converted. A request records intent and does not itself book anything outside the RedBook.

> **[RBX-P3-049]** Confirmation requires the Resource's author or administrator and a Reservation in `requested` state. Capacity testing MUST include the proposed interval and every overlapping `confirmed` or `lent` Reservation for that Resource in the same coordinate system. At each boundary, ending quantities are removed before starting quantities are added. The sum MUST not exceed capacity on any interval. Failure leaves the Reservation requested and all other records unchanged. Successful confirmation changes it to `confirmed`; starting actual use changes it to `lent`, return to `returned`, and cancellation to `cancelled` through explicit authorized operations. Returned and cancelled reservations do not consume current reserved capacity.

This algorithm permits adjacent reservations where one ends exactly when another starts. Numeric quantities use ordinary finite-number semantics; implementations needing monetary, rational or higher-precision units declare an exact-number resource profile rather than rounding into this contract.

3.11 Events, recurrence and gatherings

3.11.1 Event descriptions and occurrence keys

An Event represents an intended or recorded occurrence in the authored world. Its existence is not a host job. A Gathering is a facilitated exchange that can be associated with an Event but remains an independent context. RSVP and attendance are separate assertions.

> **[RBX-P3-050]** An operational Event interval MUST use comparable `start` and `end` in the selected clock and have `end ≥ start`; zero duration is an instantaneous occurrence. `planned`, `cancelled` and `complete` are explicit current Event states. Reaching the stated end MUST NOT mark it complete. An occurrence key is the canonical original start coordinate in the Event's clock. An empty occurrence key refers to the Event as a whole. Changing the current recurrence definition MUST NOT silently rewrite old RSVP or Attendance keys.

> **[RBX-P3-051]** Event Response MUST be unique per `(Event, participant, occurrence)` for the standard type and contain the explicit `accepted`, `declined` or `tentative` RSVP. Attendance is independently unique per `(Event-or-Gathering, participant, occurrence)` and records `attended` or `not-attended`. An RSVP MUST NOT imply attendance, and a read or Presence statement MUST NOT create either assertion. The standard participation operation records only the selected Digital's assertion, except where administrative authorship is explicitly requested.

3.11.2 Deterministic recurrence profiles

An empty recurrence map means one occurrence. The following two profiles fully define the standard recurrence representation; neither runs automatically.

- Fixed interval: `{kind:"interval", everyMilliseconds:P, count?:C, until?:U}`. `P` is canonical positive decimal text. `C`, when present, is a positive safe integer including the initial occurrence. `U`, when present, is an inclusive last-start coordinate in the same clock. The initial start is index zero; subsequent starts are `start + n×P`. Original duration is retained.
- Gregorian calendar: `{kind:"rrule", rule:R}`. `R` is the value of an RFC 5545 RECUR rule, without an `RRULE:` prefix. This profile uses UTC Gregorian instants and requires the Event's start to have zero milliseconds; its corresponding basic UTC date-time is DTSTART. UNTIL, if supplied, uses the basic UTC date-time form `YYYYMMDDTHHMMSSZ`. The base duration is retained. Floating local time, date-only values, named time zones, and BYSECOND value 60 are outside this profile. Additional inclusion/exclusion properties are not embedded in `R`; authored exceptions use the maps below.

> **[RBX-P3-052]** A recurrence consumer claiming a standard profile MUST apply its stated count/until boundaries and preserve relative occurrence order. For Gregorian recurrence it MUST apply the grammar and expansion rules of RFC 5545, section 3.3.10, and the recurrence-set, invalid-date omission and duplicate-elimination rules of section 3.8.5.3, within the UTC restriction above. It MUST reject unrecognized rule parts and a DTSTART that does not match its recurrence pattern rather than rely on undefined expansion. Count and until MUST NOT both be supplied in a fixed-interval descriptor. A standard expansion request MUST specify a finite query interval and output limit; an implementation MUST return an explicit continuation or limit result rather than imply that a truncated result is complete.

Exceptions are maps with `occurrence` and either `cancelled:true` or replacement `start` and `end` coordinates. They identify the original occurrence start, even if moved. A cancelled exception supplies no replacement interval. Replacement start/end retain the Event clock and satisfy the interval rule. Duplicate exception keys are invalid.

> **[RBX-P3-053]** Expansion MUST first identify original occurrences, apply matching cancellations or replacements, then order the projected intervals by replacement start with original occurrence key as tie-breaker. A moved occurrence is included when its projected interval intersects the query interval, even if its original start lies outside it. Occurrence projections MUST NOT become stored Event records or event-history entries unless an explicit authoring operation creates them. Unsupported calendar semantics MUST be reported; treating an unsupported recurring Event as a single occurrence is not a conforming expansion.

3.11.3 Facilitated work

> **[RBX-P3-054]** A Gathering MUST retain explicit participant grants, agenda references and outcome. Standard contributions require an open Gathering and effective participation; they receive the same attributed event treatment as conversation Messages. Agenda entries use Collection Entry order. Agenda Item state `current` is an explicit facilitation choice, not inferred from time. A chair, speaker or requested speaking role MUST be represented by Gathering Role, and requesting a turn MUST NOT make the requester an active chair or speaker.

> **[RBX-P3-055]** Shared Focus MUST identify its presenter, targets, optional selector and followers. It is an authored coordination assertion, not authority to force every reader's navigation or mark targets read. Closing a Gathering retains its contributions, agenda and outcomes. Attendance and completed agenda items MUST NOT automatically create an outcome or verify a Commitment.

3.12 Streams and live exchange records

Stream and Stream Segment represent declared media and retained contributions. Live capture, network transport, device access and encoding are host capabilities. A live Conversation can refer to a Stream; these are not automatically the same context.

[RBX-P3-056] A standard Stream append MUST require an open Stream, an active publisher or moderator Stream Participation grant, and `muted:false`. It MUST validate that the original segment asset is declared and present, that its media type is supplied, and that media `timestamp ≥ 0` and `duration > 0` are finite seconds. It MUST create the Segment, Stream Entry, publisher association, attribution and event observation atomically. Media timestamps MUST NOT be confused with Earth recording time or world event sequence. Overlap of segments is representable and does not imply corruption.

[RBX-P3-057] A consumer MUST preserve stream-level `recordingAllowed` and each participant's `recordingConsent` as separate explicit statements. A host that produces new retained recordings under this profile MUST require the stream permission and applicable participant consent; retaining an already supplied segment does not claim how those bytes were originally captured. `listener`, `publisher` and `moderator` roles are context-specific. Stream role grants do not grant RedBook administration or authority to edit another author's unrelated contribution.

[RBX-P3-058] Reading a Stream description MUST NOT start capture, connect to a service, create Presence or change its state. Closing a Stream prevents ordinary new segments but preserves existing segments and source bytes. Unsupported codecs or live transports MUST be reported as capability limitations without discarding the stored description or substituting invented media.

3.13 Participation rules and moderation

A Participation Rule can express an explicit policy attached through Rule Target. A Mail Rule applies declarative conditions and effects to explicitly selected correspondence. Neither is an always-running program. This clause defines a finite, non-executing rule representation usable by both; additional rule languages require a declared profile.

3.13.1 Standard condition and effect representation

A condition is `{all:[conditions...]}`, `{any:[conditions...]}`, `{not:condition}`, or a predicate `{path:[string...], op:operator, value?:portableValue}`. Exactly one logical form is present. A path selects map members in the immutable evaluation context defined in 3.21; it does not evaluate dotted expressions, code or network references. Standard operators are `exists`, `eq`, `ne`, `in` and `contains`. `exists` needs no value; the other operators require one. A missing path yields false for every operator except `exists`, which reports absence as false; `ne` therefore does not treat absence as an authored null. `eq`/`ne` use Part 1 value equality. `in` requires a comparison list; `contains` requires a selected list and checks value equality. Empty `all` is true; empty `any` is false. An empty conditions map means no extra constraint.

Effects are maps `{action, input}` identifying a declared world action and its input map. Their order is significant. The standard evaluation context, applicable-rule ordering, guard enforcement and explicit effect-processing operations are defined in 3.21. An enabled rule is data; merely opening a document does not evaluate it or execute effects.

[RBX-P3-059] A standard rule evaluator MUST evaluate only the grammar above, reject malformed or unknown required operators, and apply a finite depth/step limit reported through capability metadata. It MUST NOT silently convert unrecognized rules into permission. A rule effect MUST execute under the existing actor's authority, cannot grant itself additional capabilities, and cannot invoke external effects or recursive rule evaluation through the standard finite rule processor. The selected effects commit atomically or not at all. Processing a Mail Rule MUST preserve sender attribution and other recipients' organization unless the requested operation explicitly has authority to change them.

[RBX-P3-060] Context policy may restrict ordinary participation but MUST NOT weaken the fundamental invariants of Parts 1–3. Unsupported policy needed to authorize an act MUST cause that act to be rejected or reported unavailable. The ability to read policy source is not permission to bypass it through a Digital-context operation. RedBook administration remains an explicit distinct operation context.

3.13.2 Reports and review decisions

[RBX-P3-061] A Moderation Report MUST remain an attributed record connected by Report Target to actual reported Elements and optional context. Reporting MUST NOT silently remove, hide, edit or reattribute its targets. Its states `open`, `resolved` and `dismissed` require an explicit authorized moderation decision. A resolution records its authored explanation; any additional withdrawal, placement rejection or membership change must be a separate explicit effect in the same declared transaction or in a later operation.

Review queues use pending Submission or Wall Placement state. Moderation does not require the world to encrypt or conceal its source data. Hosts may enforce separate access policies, but those policies do not change the meaning of the authored records.

3.14 Soma: conceptual homes, worlds and representations

3.14.1 Neocortex ownership

A Soma Neocortex is the conceptual home of one Digital or one eligible communication context. It is an Element with its own identity. A Neo Soma is a conceptual world housed within that home. Neo Objects and Neo Relationships form its conceptual content, through the same RB Element/ Relationship foundation.

[RBX-P3-062] The following supplied root types and every valid specialization of them MUST own exactly one Neocortex:

Scope	Supplied owner roots
`person`	`digital`
`al`	

Scope	Supplied owner roots
shared	group, space, page, wall, conversation, newsletter, poll, questionnaire, proposal, decision-round, review, case, event, gathering, stream

Every Neocortex MUST participate as the sole `neocortex` in exactly one Neocortex Owner relationship. Its owner is a single Element having the corresponding inherited ownership trait. `scope` MUST agree with that owner. A new custom owner type may declare exactly one of `personal-neocortex` or `shared-neocortex`; the same invariants apply. An owner cannot declare both. Mailbox, Circle, Post, Message, Work and Neo Soma do not gain a Neocortex merely because they may refer to concepts.

The fifteen listed shared owner roots form the normative set. Continuing publication and coordination contexts are treated as shared conceptual contexts. “Shared” describes the ownership kind and participation rules, not a requirement that two Digitals already be present. An empty Wall Neocortex is valid; it does not require example content. Each such context has one empty but stable conceptual address, including an unprepared Poll or scheduled Event. Creation, retyping, deletion and Blueprint closure must account for that address. Optional homes would be a different vocabulary contract, requiring coordinated trait, validation and creation changes; they are not inferred from an empty instance.

> **[RBX-P3-063]** Creating an owner MUST create its Neocortex and ownership relationship in the same transaction, unless an explicit atomic construction already supplies exactly that valid pair. Opening, reading, displaying or selecting the owner MUST NOT create another Neocortex. Removing or retyping an owner MUST explicitly resolve the ownership relationship and any remaining conceptual content; it cannot strand a Neocortex or silently delete its Neo Somas.

3.14.2 Membership and scope

> **[RBX-P3-064]** Every ordinary Neo Soma MUST have exactly one Soma Home relationship to exactly one Neocortex. Every Neo Object MUST belong to exactly one Neo Soma through Neo Membership. Multiple Neo Membership relationships are allowed within a Soma, but an Object occurs in exactly one such membership overall. Every Neo Relationship MUST contain a typed-reference `properties.somaId` resolving to its one Neo Soma; every Element in every participant slot MUST be a Neo Object belonging to that same Soma. Its open slots may express any number of named roles and participants; no binary-edge restriction applies.

[RBX-P3-065] A personal conceptual write through a standard Soma action MUST use its owning Digital's context or RedBook administration. If a custom personal Neocortex owner is not a Digital, the standard Digital participation path MUST be unavailable; a custom profile must explicitly define any additional participation path and its authority checks. Administrative inspection and valid administrative authoring remain available. This extension case does not alter the personal ownership rule for Digital or its specializations. A shared conceptual write MUST use an effective participant of the owning communication context or RedBook administration. The author of an individual contribution need not be the initial author of the shared Neo Soma. Each new Object and Relationship created in a Digital participation context MUST receive that contributing Digital's attribution and source credit. Pure administrative authoring MAY remain unattributed; it MUST NOT infer the owner or initial Soma author as the contributor. Collective contribution may additionally be described by Authorship relationships; that MUST NOT overwrite the individual contributors' record attribution.

[RBX-P3-066] A Neo Object's interpretation of something MUST remain distinct from that thing. Represents links its `concept` to one or more actual subject Elements without identity equivalence. Changing the concept does not modify its subjects. A subject that is a Relationship may be referenced by a typed-reference Property; a Relationship ID is not permitted in the Element-only `subjects` slot. The containing RedBook is not itself an RB Element: describing "this world" uses an explicit document-context value or authored text, not an invented Element or a dangling record reference.

3.14.3 Conceptual content and reuse

[RBX-P3-067] Properties, Categories and Tags on a Neo Object or Neo Relationship classify that conceptual record. Facts, interpretations, questions and possibilities are authored classifications, not automatically established truth statuses. A change to the surrounding world MUST NOT silently rewrite a conceptual assertion. A stored sentence about the current population describes the authored model; updating that understanding is a new deliberate edit or contribution.

The standard `arrangement.readingOrder` and `arrangement.relationshipOrder` members, when supplied, are ordered lists of typed references to Objects and Neo Relationships in that Soma respectively. They can select a subset for a particular presentation; they do not change membership. Repeated references are invalid in these standard lists. Other layouts can be declared by a Viewbox or extension profile.

[RBX-P3-068] Removing a conceptual record MUST account for participant slots, typed references and reading arrangements, as well as membership. It MUST NOT leave a dangling conceptual edge or silently move content into a different Soma. Reading a Soma MUST return its actual current Objects and Relationships without adding attention, thought, history, actors or conceptual content.

[RBX-P3-069] A reusable independent Neo Soma is represented as a Blueprint component, not an unowned ordinary world Soma. Instantiation MUST bind an appropriate destination Neocortex or include a newly instantiated owner and home, rewrite typed references consistently, preserve source credits and validate the resulting ownership and scope. A concept binding to a different subject does not rewrite its prose into a new author's words. Unresolved external participants or subjects require explicit bindings; coincidentally matching titles are not bindings.

3.15 RedBook Time and event observations

3.15.1 Three distinct coordinates

RedBook has one central world-time rule or `none`. Recorded Earth time, document-local event sequence and computed world-time observation are distinct. A document may be useful with none of them progressing automatically. An Event Element's scheduled date is also distinct from the observation fields on a record.

[RBX-P3-070] `recordedAt` MUST follow Part 1's canonical UTC millisecond instant syntax, with a valid Gregorian date and seconds 00–59. A `sequence` MUST be canonical positive ASCII decimal text, unique across event-bearing Elements and Relationships in the document. `nextSequence` MUST be strictly greater than every retained sequence. `worldTime` MUST be the portable result observed under the rule applicable to that event, or null for `none`. Event-producing standard operations MUST record the complete triple. Ordinary reads MUST NOT create such observations.

[RBX-P3-071] A world-time rule change MUST NOT recalculate old `worldTime` values or alter original recording timestamps. Manual advance changes only the current manual rule value. Blueprint instantiation allocates a fresh destination-local block in source sequence order, preserves original timestamps and world-time observations, and leaves existing destination events unchanged. It MUST NOT read the clock to manufacture an import event. Gaps in retained sequences are valid; renumbering existing events to remove a gap is not an ordinary edit.

The UTC instant convention is a deliberately restricted profile of RFC 3339, with fixed millisecond precision and `Z`. It excludes local offsets, leap-second notation and unspecified precision from stored `recordedAt`. Higher-resolution source timestamps can be retained in additional typed Properties without being rounded into a false claim of identical precision.

3.15.2 Rule forms

A signed decimal integer string is `0` or an optional minus sign followed by a nonzero ASCII digit and any further ASCII digits. It excludes a plus sign, leading zeros, negative zero, whitespace and exponents. A positive decimal string excludes zero and the minus sign. These strings represent exact integers of arbitrary magnitude within a declared processing limit.

[RBX-P3-072] A standard world-time rule MUST have one of the following forms. Members not listed may be retained as extension data but MUST NOT silently alter the standard formula.

mode	Required members	Optional standard members
none	None	None
earth	None	calendar
offset	None	offsetMilliseconds (signed decimal, default "0"), calendar
scaled	realAnchor (canonical recording instant), worldAnchorMilliseconds (signed decimal), rateNumerator (signed decimal), rateDenominator (positive decimal)	calendar
manual	currentMilliseconds (signed decimal)	calendar
custom	provider (nonempty implementation identifier)	Declared provider configuration as portable values

calendar is a nonempty string, default proleptic-gregorian. For built-in formulas it labels the requested presentation calendar; it does not change the epoch or unit of milliseconds. A host without a requested alternate calendar can return the exact tick value and identify unsupported alternate presentation. It does not silently reinterpret the tick count as another epoch.

> **[RBX-P3-073]** The custom provider MUST be declared and resolvable through the execution contract in Part 4. Its computation receives the rule and explicit recording instant and returns one portable world-time value synchronously. It MUST NOT modify the document, acquire undeclared capabilities, start background work or use a different implicit instant. If its required implementation is unavailable, evaluating that rule MUST report CAPABILITY_UNAVAILABLE; substituting Earth time or null would change meaning. Merely preserving an unevaluated custom rule does not require running it.

3.15.3 Coordinates in domain Properties

For fields such as Reservation start, Event end, Reminder at and recurrence exception coordinates, the standard recorded clock uses canonical recording instants. A standard numeric world coordinate is the literal prefix ms: followed by a signed decimal string, measured in the same epoch/unit as built-in world-time results. A canonical UTC instant can also express the equivalent Gregorian world coordinate and is converted by 3.16.2; its use does not make it the event's actual Earth recording time. A custom coordinate uses an explicitly declared provider profile.

[RBX-P3-074] Domain operations comparing coordinates MUST identify their clock and convert them exactly into that clock's comparable coordinate domain. They MUST reject comparisons whose domains are unknown or differ. A `none` world-time rule supplies no current "now"; authored world coordinates may still be retained and compared as a stated axis, but an operation requiring the current world time is unavailable until a suitable rule is selected. No date Property by itself causes an action at that time.

3.16 Exact world-time evaluation

3.16.1 Formulas and results

Let `R` be the exact signed number of milliseconds of the requested recording instant since 1970-01-01T00:00:00.000Z. Let `A` be the converted `realAnchor`, `W` the exact `worldAnchorMilliseconds`, `N` the numerator, and `D` the positive denominator. Let `trunc0(x/y)` be integer division toward zero, including for negative values.

[RBX-P3-075] A built-in evaluator MUST use these formulas without floating-point intermediate arithmetic:

Mode	Result
`none`	null
`earth`	`T = R`
`offset`	`T = R + offsetMilliseconds`
`scaled`	`T = W + trunc0(((R − A) × N) / D)`
`manual`	`T = currentMilliseconds`

Negative numerators model reverse world-time progression; zero models a fixed value. A negative denominator or zero denominator is invalid. Evaluation before the real anchor uses the same signed formula. The host need not keep the file open between evaluations.

[RBX-P3-076] A non-null built-in result MUST be `{milliseconds, calendar, iso}`. `milliseconds` is the canonical signed decimal form of `T`. `calendar` is the rule's requested calendar label or its default. For `proleptic-gregorian`, `iso` is the Gregorian UTC rendering when `−8640000000000000 ≤ T ≤ 8640000000000000`, otherwise null. The expanded rendering uses four-digit years from 0000 through 9999; outside that range it uses a sign and at least six year digits. For an unsupported alternate presentation calendar, `iso` is null while exact milliseconds and the requested calendar label are retained. The finite ISO rendering range MUST NOT limit the stored exact tick value.

3.16.2 Gregorian conversion

The proleptic Gregorian calendar extends its leap-year rule to year zero and signed years. A year is leap if divisible by 4, except years divisible by 100 are common unless also divisible by 400. February follows that rule; the other month lengths are conventional Gregorian lengths. A day is exactly 86,400,000 milliseconds in this coordinate system. Leap seconds are not inserted into the axis.

For integer year `Y`, define:

```
B(Y) = 365×Y + floor((Y−1)/4) − floor((Y−1)/100)
       + floor((Y−1)/400) + 1
```

`B(Y)` counts days preceding year Y relative to the beginning of year zero. Let `M(Y,m)` sum complete month lengths before month `m`. For valid day `d`, hour `h`, minute `q`, second `s` and millisecond `u`:

```
days = B(Y) + M(Y,m) + d − 1 − 719528
R = days×86400000 + h×3600000 + q×60000 + s×1000 + u
```

> **[RBX-P3-077]** Gregorian conversion MUST validate the actual day/month/year combination before applying this formula. It MUST reject overflow normalization such as interpreting 31 February as a March date. All divisions in `B` use mathematical floor, while scaled-rate division in 3.16.1 uses truncation toward zero; these are different operations. Inverse rendering chooses the unique year/month/day covering the floor-divided day index and the nonnegative remainder within that day.

3.16.3 Manual advance, precision and examples

> **[RBX-P3-078]** Manual advance MUST require `mode: manual`, add the supplied exact signed delta to `currentMilliseconds`, and write the canonical signed result. It MUST NOT generate an event observation, modify any Event's schedule, or advance an autonomous process unless the explicitly invoked action separately includes such authorship. An implementation limit on integer length MUST be declared and MUST cause an explicit limit error before any partial mutation; conversion through a platform's bounded integer-string parser MUST NOT silently reject values the implementation claims to support.

The following informative examples illustrate the formulas; they do not alter the normative rules:

- Earth mode at the Unix epoch returns `milliseconds: "0"`.
- Offset mode with `offsetMilliseconds: "-1000"` at that instant returns `"-1000"`.
- Scaled mode with `A=0`, `W=100`, `N=2`, `D=3`, evaluated at `R=2`, returns `101`; evaluated at `R=-2`, it returns `99`. Truncation, not floor, governs the second case.
- Manual mode at `"10000000000000000000"` remains exact; its standard Gregorian `iso` rendering is null because the rendering range is exceeded.

- Two Messages recorded in one authoring batch can share a `recordedAt` instant while retaining different event sequences. No fictional elapsed duration is implied.

The complete event and world-time semantics therefore remain usable without a constantly running clock, scheduler, AI connection or retained history of ordinary edits.

3.17 Participant conceptual authoring

The `rb.content.*` operations provide scoped participant editing; their name does not grant general write access to every record. A conceptual record is a Neo Soma, Neo Object, Neo Relationship, Represents, Neo Membership or Soma Home. Its home is resolved through the actual validated Soma/Home graph. Neocortex ownership, identity, type, attribution and event observations are not editable through this surface. The general administrative operations remain available separately.

> **[RBX-P3-079]** A Digital creating conceptual content MUST have the contribution authority in 3.14. For an existing personal-home record, its owning Digital MAY edit, classify or remove it, including an administratively authored record. For an existing shared-home record, the Digital MUST be an active participant and either be the record's local attributed contributor or have the permission declared by `participation.conceptualEditing:"members"` on the owning context. Absence of that member means `"contributors"`. These are the only standard values. A detached source credit, initial Soma authorship, context authorship, display name or unrecognized role MUST NOT substitute for this check. RedBook administration remains a distinct authorized path.

> **[RBX-P3-080]** `rb.content.edit` MUST apply the same title/Properties/classification/slot replacement semantics as `rb.records.edit`, within the above scope. `rb.content.unset-properties` MUST remove only the supplied exact keys, including an empty string key when present. Both preserve existing attribution and event observations. A slot map replaces the whole slot map; omitting an optional slot from that supplied map removes that slot, while omitting the entire map leaves all slots unchanged. Required Soma scope and membership constraints are revalidated. `rb.content.classify` MUST replace only supplied Category/Tag assignment lists; omitted lists remain unchanged. Its optional `createTerms` list creates fresh shared taxonomy terms and assigns each to the selected record, in the same transaction. These terms use `{kind,label,parent?}`; `parent` is valid only for a Category and names an existing Category or null. The new terms are appended to the relevant supplied-or-retained assignment list in input order. Existing taxonomy definitions or other records' assignments are not changed. The result identifies the changed record and each created term in input order.

Soma creation operations accept an optional `type` selecting the respective root or a valid specialization, plus optional `categories` and `tags`. Omission chooses the root and empty classification lists. Their ordinary `properties` input still supplies authored fields and creation defaults; a specialization cannot evade its root constraints. A Concept's classification does not establish the truth of its content.

[RBX-P3-081] `rb.content.delete` MUST validate the actor's authority for every record in its recomputed deletion plan. The caller selects `ids` explicitly; `removeRelationships:true` permits incident Relationships only when their removal is also authorized and leaves a valid graph. This flag MUST NOT erase another contributor's relationship or content under the default shared policy. A Soma Home may be removed only with its Soma or as part of an explicitly valid administrative rehoming; the participant operation cannot reassign a conceptual home. Mandatory membership links for explicitly deleted Objects may be included, but surviving Objects retain exactly one membership. Every typed reference, arrangement dependency and declared resource-import reference must be explicitly resolved in the same transaction or deletion fails with `DEPENDENCY`. Resource-import dependency descriptors follow Part 4; deleting conceptual records does not implicitly delete retained original resource bytes. Failure changes no records, taxonomy or allocation state.

[RBX-P3-082] An unsupported or unauthorized scope MUST fail as `PARTICIPATION` or `CAPABILITY_UNAVAILABLE`, not retry the operation under administration. An authorized actor may combine these operations through `rb.transaction.perform`; each step retains the same actor and scope checks. New classification terms and dependent records roll back with the enclosing transaction. The normative domain lifecycle matrix identifies the callable participant and administrative paths and their applicable preconditions.

3.18 Explicit state transitions and organization

`rb.domain.transition` accepts `{actorId?,id,to,details?}` and returns `{record,createdRecords}`. `details` defaults to an empty map. It is a checked state operation, not an arbitrary Property patch. The table lists all transitions in this operation; a specialization inherits its root row. "Author" means the current local record attribution, not a textual credit. "Admin" means RedBook administration. An actor may request the already-current state as an authorized no-op; this creates no Decision or event. Otherwise an unlisted source/result pair fails with `STATE`.

Root	Transition	Authority and additional effect
Post, Work	`draft → published`; `published → withdrawn`; `withdrawn → draft`	Author/admin. Existing body and observation retained.
Invitation	`pending → cancelled`	Named inviter/admin; no Membership is revoked or created.
Connection	`proposed → ended`	A named initiator/admin withdraws the proposal; decisions are retained.
Connection	`accepted → ended`	Any named initiator or counterpart/admin ends the connection; participants are retained.

Root	Transition	Authority and additional effect
Agenda Item	`pending → current/complete`; `current → pending/complete`; `complete → pending`	Author/admin. No automatic meeting progression.
Mail Handling	`assigned → resolved`; `resolved → assigned`	Owner of the associated Mailbox/admin; recipient projection must still be authorized.
Gathering Role	`requested → active/finished`; `active → finished`	Gathering author/admin decides requests; a named Digital may finish its own singleton role. Starting a role requires active Gathering participation.
Conversation, Review, Gathering, Stream, Questionnaire	`open ↔ closed`	Author/admin. Content and homes retained. Closing a Questionnaire prevents submission.
Event	`planned → complete/cancelled`; `complete/cancelled → planned`	Author/admin. No inferred attendance or jobs.
Case	`open → resolved/closed`; `resolved → open/closed`; `closed → open`	Author/admin. No automatic Commitment verification.
Reservation	`confirmed → lent → returned`	Resource author/admin.
Reservation	`requested/confirmed → cancelled`	Named requester, Resource author, or admin. No external cancellation claim.
Moderation Report	`open → resolved/dismissed`	Author of its explicitly associated report context, or admin. Without one unambiguous context, admin only. `details.resolution` is a required string.
Annotation, Comment	`open ↔ resolved`	Author/admin. Subjects unchanged.
Submission	`pending → accepted/declined`	Attributed author of its unique destination, or admin. No implicit publication/placement.
Submission	`pending/accepted → withdrawn`	Submission author/admin. Source content retained.
Request	`open → withdrawn`	Named requester/admin. Existing responsibility is not revoked by inference.

Root	Transition	Authority and additional effect
Questionnaire Response	`draft/submitted → withdrawn`; `withdrawn → draft`	Named respondent/admin. Return to draft requires an open Questionnaire. Existing submitted-response revision rules still apply.
Proposal	`open → adopted/not-adopted`	Author/admin; explicitly authored Decision required as below.
Proposal	`open → withdrawn`	Author/admin; no Decision implied.
Decision Round	`open → closed`; `closed → open`	Author/admin; closing authors a Decision as below; reopening retains it as authored content.

[RBX-P3-083] The transition operation MUST enforce this table, apply only the stated state and required detail changes, and preserve original attribution and event observations. Proposal adoption/non-adoption and Decision Round closure MUST require a nonempty `details.procedure`, a `details.body` content map, and optional `details.dissent` list (default empty). They create a separately attributed Decision event whose `subject` is a typed reference to the actual Proposal/Round. The new Decision records the supplied assertion; a procedure name alone does not calculate or certify a vote. Other rows create no event or retained revision. Unknown detail keys and details not consumed by the selected row MUST be rejected. Poll closure remains `rb.polls.close`, which captures its tally; commitment progress remains its explicitly communicating Progress operation. The result's `createdRecords` is empty except for the newly authored Decision.

[RBX-P3-084] `rb.posts.edit` MUST allow the author/admin to replace supplied title and/or current body of a draft or published Post without altering attribution or its original event fields; a withdrawn Post must first return to draft. `rb.mail.folders.create` MUST create an attributed Mail Folder with `mailboxId` referencing a Mailbox owned by the actor (or under administration). `rb.mail.move` MUST accept one Mailbox Entry and a `folderId` identifying a folder in that same Mailbox, or null to remove the optional folder slot. Only that entry's recipient or admin may move it. The operation preserves the Message, Delivery, other recipients' entries and organization flags. A folder's own deletion remains dependency-aware administration or an explicit batch that first moves its entries; deleting a folder never deletes its messages.

3.19 Deterministic correspondence threads

A sent Mail Message produced by the standard operation has `threadId` as a typed reference to its canonical Conversation, and exactly one Contribution linking that message into that thread. Additional authored references can discuss the message but do not create a second canonical thread. `mailParticipants`, when present on a Conversation, is a nonempty list of unique typed Digital references and marks a correspondence thread. It records the initial public audience, not a continuously inferred membership list.

[RBX-P3-085] For a send, form the public audience by taking the sender, then To recipients, then Cc recipients, retaining each identity's first occurrence. Bcc recipients are excluded. A new correspondence thread MUST contain that list as `mailParticipants`, its shared Neocortex, and one active Membership for each audience member (sender `organizer`, others `member`). It MUST NOT automatically grant Bcc recipients Membership, access to earlier thread projections or shared conceptual-write authority. They receive their own Delivery and Mailbox Entry. A Bcc-only send therefore has the sender as its sole initial public participant. This is authored participation, not encryption or a restriction on administrative file inspection.

[RBX-P3-086] A send without an antecedent MUST create a new thread. For `rb.mail.send` with `replyTo`, and for `rb.mail.reply`, `threadMode` is `continue` by default; `new` explicitly creates a new thread while retaining the Reply link. Under `continue`, the antecedent MUST have a resolvable `threadId`, exactly one matching canonical Contribution and a valid correspondence audience. Missing or contradictory structure fails with `THREAD_UNRESOLVED`; a consumer must not select the first incidental association. If the resulting public-audience identity set equals the antecedent thread's stored set, reuse that same thread and Neocortex. If the set differs, create a new thread and home. Reuse requires an open thread and active sender Membership; a closed thread fails with `STATE`, and a departed sender fails `PARTICIPATION`. Explicit `new` may create a new exchange without reopening or rejoining the old one. The returned `conversationId` identifies the actual reused or created thread.

[RBX-P3-087] Reuse MUST NOT add, reactivate, elevate or remove Memberships, recalculate the original audience list, or copy conceptual content. Recipient changes are represented in each message's Deliveries. A changed public audience causes the new-thread branch, while a change only in Bcc recipients does not. A former Bcc recipient sending a reply becomes a publicly identified sender in that new message; its added public identity causes a new thread. Reply-all still includes only the original sender and To/Cc recipients except the replying actor. No Bcc identity is promoted merely because a thread is reused.

[RBX-P3-088] A Digital correspondence projection MUST include only messages for which that Digital is sender or explicit recipient, and apply the per-message Bcc projection in 3.5. Listing the thread must not enumerate hidden Bcc Deliveries or infer Bcc membership. Shared conceptual content is accessible through the thread's actual participation contract; Bcc receipt alone does not grant that contract. Generic administrative/raw-record inspection remains distinct and must not be mislabeled as a Bcc-filtered correspondence projection. Saving, reopening and Blueprint remapping preserve `threadId`, audience references and their associations. Ambiguous imported thread structure cannot become operational merely because it was structurally preserved.

For example, A sends To B, B replies to A, then A replies-all to B: the public set remains `{A,B}` and all three messages use one Conversation and one shared Neocortex. Adding C to public recipients creates a second thread; adding C only as Bcc does not. A later reply from Bcc C creates a thread whose public audience includes C. These are authored exchanges, not evidence of external email delivery.

3.20 Newsletter issue allocation

`Newsletter.nextIssueNumber` is canonical positive ASCII decimal text. It is current allocation state, not an issue history. A new Newsletter receives `"1"`. `Newsletter Issue.issueNumber` remains a safe ordinary nonnegative integer for this vocabulary; zero means unnumbered. This preserves the distinction between a portable exact frontier and the existing bounded ordinal field.

> **[RBX-P3-089]** Before publishing, the counter MUST exist, be canonical and exceed every retained positive issue number in that series. If it is absent and the series contains no positively numbered issues, the explicit publication operation may initialize it to `"1"` in its transaction. If absent with numbered issues, publication MUST fail `ALLOCATION_UNINITIALIZED`. `rb.newsletters.initialize-numbering` is an administrative assertion accepting an exact frontier greater than every retained ordinal; it is allowed only while the frontier is absent and must not claim to recover deleted issue history. A present frontier cannot be decreased or removed by an ordinary edit. Administrators reconstructing incomplete material explicitly choose the asserted frontier; readers do not invent it during opening.

> **[RBX-P3-090]** Publication MUST allocate the current frontier as the new Issue number, increment the frontier by exact decimal addition, and create its Issue, attribution/event fields and Newsletter Entry in one transaction. If the allocated ordinal exceeds 9007199254740991, publication fails with `NUMBER_RANGE` without changing the counter. The stored next frontier may be `"9007199254740992"`, representing exhaustion of this ordinal field. Failed publication consumes no committed ordinal. Deleting, withdrawing or renumbering retained issues MUST NOT reduce the frontier; assigning a new positive ordinal administratively must advance the frontier if necessary. Thus publishing 1, 2, 3, deleting 3 and reopening produces 4 next, independently of unrelated world event allocation.

> **[RBX-P3-091]** Copying an entire Newsletter into a newly identified series MUST preserve its valid frontier, even when not all its issues are selected. Importing a positive-numbered Issue into an already existing bound series MUST require an unused number at or beyond that destination's frontier and advance the frontier beyond every imported positive ordinal atomically. A lower or conflicting number fails `ISSUE_NUMBER_CONFLICT`; import must not silently renumber the issue. Multiple imported issues are checked as a set and advance once beyond their maximum. Unnumbered issues do not advance it. Original source credits and recording observations are unaffected. Capture, import failure and request replay do not consume a new ordinal. Whole-world copies preserve the same current allocator state without pretending to be newly numbered publications.

3.21 Response policy and standard rules

Poll and Questionnaire use `properties.responsePolicy.eligibility` with exactly `members` or `open`; omission means `members`. `members` requires effective active participation in that Poll/Questionnaire; `open` permits a selected Digital to submit the response without Membership. It grants no enrollment, shared-conceptual write or power to change the question. Questionnaire's independent

`responsePolicy.reviseSubmitted` is Boolean, default false. Additional supported extension members retain their declared semantics; they cannot silently change these defaults. The precise shapes are in domain-policy.schema.json.

> **[RBX-P3-092]** Voting and questionnaire submission MUST evaluate these exact policy members before writing a Ballot or Response. A missing policy map uses the member defaults without adding stored data during a read. A present malformed standard member fails `STATE`; an unsupported required extension policy fails `CAPABILITY_UNAVAILABLE`. Membership enrollment at `participation.enrollment` controls joining, not eligibility to answer. Neither an ordinary read nor an `open` response policy creates Membership.

Participation Rules apply through Rule Target to the actual context of an action. Mail Rules apply through their `mailboxId` typed reference to a Mailbox. Both have `enabled` (default true), `priority` (safe integer, default 0), `actions` (unique exact action names, default empty meaning all applicable names), and `mode` (`require`, `deny`, or `effects`, default `require`). The immutable predicate input is exactly `{actor,context,action}`, where actor is the selected Digital reference or null for pure administration, context is the record snapshot of the resolved policy context, and action is `{name,input}` after input validation/default resolution and before mutation. An empty `path` selects that entire map; path steps traverse maps only. Caller-supplied subject records or alternate context snapshots cannot replace it.

> **[RBX-P3-093]** Applicable enabled rules MUST be sorted by ascending priority, then exact rule identity in Unicode scalar order. Rules whose nonempty `actions` list excludes the operation are inapplicable. All conditions are evaluated against the same initial snapshot. A `require` rule denies if its condition is false; a `deny` rule denies if true. Every denial wins; another rule cannot override it or elevate base authority. An `effects` rule whose condition is true selects its effects in stored list order. Guard modes must have empty effects lists. Effect descriptors contain exact `action` and `input` only; unknown required structure fails rather than being ignored. The standard creation/editing path for rules is RedBook administration. A declared participant extension may additionally authorize the context's existing author/owner, but a rule never grants permission to change itself.

The normative lifecycle catalogue supplies policy-context resolvers. Conversation operations resolve their Conversation; Wall operations their Wall; Poll/Questionnaire/Review/Gathering/Stream operations their named context; conceptual operations their home's owner; Mailbox organization its Mailbox; reservation operations their Resource; collection operations their Collection; and source-specific edits their actual source. An operation affecting several contexts must satisfy every applicable guard set, in context identity order. A pure administrative action can reconstruct authored state without a participant policy grant but still enforces all data invariants. Newly created contexts have no earlier guards. Extensions declare a checked policy-context resolver under Part 4; unsupported resolvers make that operation unavailable.

[RBX-P3-094] Ordinary Digital-context writes MUST enforce the resolved `require`/`deny` guard rules in addition to their base domain authority, even when not invoked through a rule wrapper. They MUST NOT run effect lists implicitly. `rb.rules.evaluate({contextId,action})` performs the same immutable policy evaluation without mutation and returns `{allowed,matchedRuleIds,deniedBy,effects,validation:"policy-only"}`; each selected effect is `{ruleId,index,action,input}`. `matchedRuleIds` lists rules whose conditions are true, in evaluation order. `deniedBy` lists failing guards in that order. Its result is not a promise that all later domain, resource or concurrency checks will succeed. The supplied context must be among the action's actual resolved contexts or the call fails `POLICY_CONTEXT`; it cannot bypass another context's guard set.

[RBX-P3-095] `rb.rules.process({contextId,action})` MUST check all actual context guards, stage the base action, then stage the selected effects in rule/list order within one atomic transaction. Selection uses the initial snapshot; effects observe the preceding staged writes. The result is `{result,ruleResults:[{ruleId,index,result}]}`. Every effect retains the same actor, ordinary guards, schema and domain checks. A denied guard, invalid effect or failed final validation rolls back the base action and all effects. Effects may invoke supported read/write actions but cannot invoke either rule wrapper, `rb.transaction.perform`, an external action or a policy-changing action; recursive rule execution is rejected. No effect can turn an otherwise forbidden base operation into an allowed one. Depth and work limits are declared capabilities. The wrapper never starts background processing.

3.22 Recurrence invocation and pagination

`rb.events.occurrences` is the standard read invocation for the recurrence profiles in 3.11. Input is `{eventId,start,end,limit?,offset?}`; the query coordinates use the Event's clock, `end > start`, limit defaults to 100 and is a safe integer 1–1000, and offset defaults to zero and is a nonnegative safe integer. The output is `{occurrences:[{occurrence,start,end}],nextOffset}`, with null `nextOffset` exactly when no later matching result exists. Occurrence is the original-start key even for a moved interval. Explicit resource limits remain possible, but cannot be represented as a successful complete result.

[RBX-P3-096] Expansion MUST compute the recurrence set within the finite query together with every authored exception that can move an original occurrence into that query. An exception must identify a real original occurrence; otherwise validation fails. Positive-duration intervals match when `occurrenceStart < queryEnd` and `occurrenceEnd > queryStart`. An instantaneous Event matches exactly when `queryStart ≤ point < queryEnd`. After exceptions and matching, sort by projected start and then original occurrence coordinate, using exact same-clock numeric comparisons, before applying offset and limit. Pages operate against the interface's bound document revision; a changed revision requires rediscovery rather than pretending the old offset is a stable historical cursor.

[RBX-P3-097] A fixed-interval consumer MUST derive distant original indices arithmetically rather than imply completeness after stopping at an arbitrary number of earlier occurrences. An RRULE consumer MUST implement the declared subset, including the DTSTART-pattern validation, or report it unavailable. No-rule Events have one original occurrence. Cancelled Events return an empty list unless a separately declared historical projection is requested; the standard action has no such flag. Recurrence expansion is pure: no stored Event instances, Membership, attendance, clock advance or event observations are created. The examples and vectors test finite interval behavior and zero-duration boundaries separately from any claim to implement all RFC 5545 expansion.

3.23 Feed selection before paging

[RBX-P3-098] Standard Feed evaluation MUST use one immutable world snapshot and this order: apply primitive/type/text filters to the complete base enumeration; sort the complete filtered result under the Feed's declared ordering; then apply query offset; then query limit. It must not page the original enumeration first. For chronology, records with no sequence precede records with a sequence and sort by exact identity; event-bearing records sort by exact positive decimal sequence. Title sorting compares the stored title's Unicode scalar sequence, then record identity. Reverse chronology reverses the complete chronology order, including the no-sequence group. Results do not duplicate the same record. A Feed query cannot add an unspecified second sort key.

[RBX-P3-099] Feed text filtering MUST use the same deterministic searchable-value serialization, pinned Unicode case-folding version and literal-substring algorithm as the basic query in Part 4. Exact-number lexemes remain text, maps use sorted keys and lists retain authored order. Locale or presentation-specific sort/search may be an additional named feature, but cannot be substituted under `rb.feeds.read`. With base enumeration titles `Z`, `A`, title ordering and limit 1, the standard result is `A`, irrespective of their stored record order.

3.24 Participant initiation and descriptive authoring

The standard `rb.content.create` operation fills the gap between generic administration and domain-specific actions. Its closed table below covers descriptive material and explicit proposals; it is not an unrestricted Element/Relationship constructor. `type` chooses one listed root or a valid specialization; `title`, `properties`, `slots`, `categories` and `tags` supply its authored content. All supplied references must resolve in the selected Digital's authorized projection. `contextId` and `sourceIds` are used only for Submission and otherwise rejected. All creations receive the selected Digital's local attribution. An administrator must name that Digital explicitly.

Root types	Initiation and authority	Revision
`profile`, `location`, `distribution-list`, `correspondence-template`, `signature`, `representation`, `collection`, `collection-section`, `feed`, `saved-query`, `resource`, `agenda`	Selected Digital authors its own descriptive record. References must be accessible; no role, participation grant or activity is inferred.	Declared descriptive fields and additional specialization fields.
`work`	Selected author; initial state is draft.	All descriptive fields except state; state uses the transition table.
`agenda-item`	Selected author; initial state is pending.	All descriptive fields except state; state uses the transition table.
`profile-of`	Profile is authored by actor; subject is exactly that actor.	No participant-slot changes; additional specialization fields only.
`recipient-list-member`	List is authored by actor; referenced recipients are present Digitals.	Recipients may be revised by list author; list is immutable.
`representation-of`	Representation is authored by actor; sources are accessible Elements.	Sources may be revised by representation author; representation is immutable.
`attachment`	Every content participant is authored by actor; resources are accessible Elements.	Resources may be revised by those content authors; content is immutable.
`reference`	All participants are accessible Elements; link authorship grants no authority over them.	Authored Properties and participant slots may be revised with the same access checks.
`follow`	Followers are exactly the selected actor; targets are accessible Elements.	Targets and the enabled flag may be revised; followers are immutable.
`mail-handling`	Actor owns the named Mailbox; the Message is in its authorized recipient projection; responsible is exactly actor and initial state assigned.	Note and additional fields; mailbox/message/responsible are immutable. State uses the transition table.

Root types	Initiation and authority	Revision
`gathering-role`	Gathering author may assign an active role to existing active participants. An active participant may create only its own requested speaker role. Reject an existing current role for the same Gathering and Digital.	Additional fields only; gathering/digitals/role are immutable. State uses the transition table.
`shared-focus`	Presenter is exactly actor, actively participating in the named open context; targets are accessible Elements; followers is empty.	Selector and targets only, plus additional fields; context/presenter/followers are immutable. No attention or follower consent is inferred.
`submission`	contextId is one accessible destination; nonempty sourceIds are accessible, shareable Elements. slots must be empty. Create pending Submission and one Submission Target with submission/content/destination slots.	Note and additional fields while pending; target/source associations remain fixed. Respond/withdraw through the transition table.
`connection`	slots.counterparts is nonempty present Digitals excluding actor. Omitted initiators becomes [actor]; supplied initiators must equal [actor]. Initial state proposed and responses empty.	Note and additional fields while proposed; participants and responses immutable through generic editing. Decisions use rb.connections.respond; withdraw/end uses the transition table.
`invitation`	Inviter is the selected actor, explicitly authorized to manage membership of this context under 3.3; context is a Group, Space, Page, Wall, Conversation, Newsletter, Poll, Questionnaire, Proposal, Decision Round, Review, Case, Event, Gathering or Stream (or specialization). Exactly one other present Digital is invited. Initial state pending; no Membership or notification is created.	Note and additional fields while pending; inviter/invitees/context are immutable. Named invitee uses rb.invitations.respond; inviter may cancel through the transition table.
`review-target`	Review is open and authored by actor; targets are nonempty accessible Elements.	Targets may be revised while the Review is open; review is immutable. The Review must retain a valid target association.

[RBX-P3-100] Creation MUST check its exact table row, materialize root and specialization defaults, and validate the complete record and required relationships together. Supplied fixed-state fields (`state` for Work, Agenda Item, Mail Handling, Gathering Role, Invitation and Submission; `state` and `responses` for Connection) MUST be rejected rather than overwritten, even when equal to the prescribed value. Only declared root Properties permitted by the row and declared additional specialization Properties are accepted; other top-level Property keys are rejected on this authoring path. Unknown stored content is still preserved under Part 1. For Gathering Role the default role is `participant` for an assigning Gathering author and `speaker` for a self-request; an explicit different role is valid only for the Gathering author. A caller MUST NOT supply a participation policy or a field that purports to grant authority through this action. Every unlisted type uses its dedicated standard action, deliberate administration, or a separately declared extension action.

[RBX-P3-101] Descriptor revision, classification and removal MUST require the current local author and the table's referenced-owner/context checks; administrative mode may substitute for actor authority but not structural checks. A Gathering author may additionally decide a Gathering Role through the transition operation. `rb.content.edit` may change title, classifications, the listed descriptive Properties and only expressly mutable participant slots; a supplied slots map remains a complete replacement and must retain immutable slots exactly. `rb.content.unset-properties` removes only optional permitted fields. `rb.content.classify` also retains the explicit term-creation contract in 3.17. `rb.content.delete` may remove these descriptors only after checking the full dependency plan, with every dependent independently eligible and authorized. Connection, Invitation and Submission are excluded from that deletion route: ending/withdrawal retains the proposal and its participants/sources. Other standard domain content gains no deletion permission merely by having an author.

Standalone descriptor edits resolve their own record as policy context; association edits resolve the owning/context slots stated by the table. Creation of a standalone descriptor or Connection has no pre-existing context. Association creation resolves all named owning/context slot records, and Reference/Follow resolve their target records. Submission resolves its destination. No supplied context overrides the actual association. Changes never send mail, publish or place a source, infer another Digital's attention, begin a meeting, grant resource possession, or start background work.

Circle setup, source Media custody, explicit Authorship assertions, standalone Presence/Reminder/Attention Preference setup and policy/identity administration retain their administrative paths. A stored descriptor is not a promise that every imaginable user interface or operation exists. Groups, Spaces and Pages are created through the existing shared-context operation; inherited context configuration and Stream Participation role/consent setup use explicit administration unless a dedicated participant action applies. The lifecycle catalogue records these boundaries separately from participant workflows.

3.25 Specialization fields and ordinary participant actions

[RBX-P3-102] Every action marked `primaryCreation` in Annex B MUST construct its selected primary type using Part 4.19. Additional fields are Properties declared by the selected specialization whose exact keys are absent from the core root's field declarations. An action cannot accept an inherited `state`, `body`, policy or ownership override disguised as an extension field. A required additional field may be supplied by the caller or by a declared default; omission without a default fails atomically. The rule applies equally to CETs and CRTs, retains all inherited participant-slot and lifecycle constraints, and does not change the types of automatically created dependent records.

[RBX-P3-103] Beyond the conceptual and descriptor scopes, `rb.content.edit` and `rb.content.unset-properties` MUST accept only these declared additional fields; `rb.content.classify` may change the primary record's Categories/Tags. They MUST preserve inherited domain fields, slots, identity, type, attribution and observations. Digital authority requires the record's current local author plus the following inherited edit/set checks; textual/external credit is not a local author grant:

Primary record	Additional-field edit authority and policy context
Post	Same author and draft/published state checks as `rb.posts.edit`; context is the Post.
Message	Same author checks as `rb.messages.edit`; context is its unique Contribution context, which may be a Conversation or Gathering. Ambiguity fails.
Conversation, Wall, other communication owner	Author; context is that owner. Editing extra fields never changes enrollment, moderation, response eligibility or lifecycle.
Questionnaire Response, Ballot, Event Response, Attendance, Decision Position, Reaction, Bookmark, Subscription, Attention	Same selected respondent/key and current state/eligibility checks as the corresponding setter; resolve that setter's actual subject context. No event is allocated merely to revise extra fields.
Mail Message, Mail Folder	Sender/current author of message or owning Digital of folder's Mailbox, respectively; retain authorized mail projection. Resolve canonical thread or Mailbox. Sent envelope and recipients remain immutable.
Wall Placement, Collection Entry	Local author plus current placement/contribution eligibility in the actual Wall/Collection; resolve that context. Review state, position and participants remain domain-controlled.
Finding, Reservation, Progress, Newsletter Issue, Stream Segment	Local author plus current authority in the actual Review, Resource, Commitment, Newsletter or Stream under its creation operation; resolve that owner/context. Inherited results, responsibility, ordinals and stream observations remain domain-controlled.

Primary record	Additional-field edit authority and policy context
Annotation, Comment, Circulation, Notification, Request, Edition, Moderation Report	Local author with continued authorized access to actual source/target participants; resolve the creation action's existing contexts (or the primary record for a context-free creation). Inherited state, body, recipients and source snapshots remain unchanged.

Conceptual primary records continue to use their richer home-based edit contract in 3.17. Descriptor primary records use 3.24. An unresolved owner, subject, current key or ambiguous context MUST reject the edit. Direct descriptor fields do not imply permission to change a portable extension's protected scope association: a declared immutable association remains protected under its applicable contract. The same checks govern Annex D.12 primitive patch/unset calls against a core specialization; registering custom code does not relax them. A Laboratory Reading Post therefore uses `rb.posts.create` with its required `temperature` Property, retains ordinary Post publication rules, and may revise that added Property through this public participant contract.

4

PART 4

Part 4 — Actions, interfaces, execution and extensibility

The operational contract: checked actions, independent contexts, transactions, portable functionality and extensibility.

4.1 Scope and capability boundaries

4.2 Execution contexts and portable values

4.3 Action contract and atomic execution

4.4 Query, editing and dependency behavior

4.5 Participation and domain operations

4.6 Atomic batches and result bindings

4.7 Blueprint capture and instantiation

4.8 Original resources and package-aware actions

4.9 The eight-tool interface

4.10 Retry, failure and transport mapping

4.11 Hooks, filters and transaction boundaries

4.12 Extension installation, activation and cleanup

4.13 Viewbox lifecycle, presentation and the host bridge

4.14 Limits, implementation assessment and interoperability

4.15 Scoped participant editing and classification

4.16 Domain lifecycle, clearing and finite projections

4.17 Policy-context resolution and explicit rule processing

4.18 General custom participant writes

4.19 Primary creation, descriptive workflows and composition

4.1 Scope and capability boundaries

This Part specifies how an implementation deliberately reads and changes a RedBook, exposes those operations to humans and external software, and hosts carried presentation and extension code. Part 1 defines the logical data model; Part 2 defines its portable representation; Part 3 defines the meanings and invariants of the supplied vocabulary. An action is an operation over those meanings, not a third authored primitive. An action call, a selected interface context, a loaded module, and a running AI session are not implicitly records in a RedBook.

> **[RBX-P4-001]** An implementation advertising `core-operations` MUST implement the complete standard world-model, communication, conceptual-participation and explicit domain-processing name set marked with that capability in the normative action catalogue, subject to explicit host requirements, roles and state preconditions. An implementation advertising `authoring` MUST additionally implement every compound, advanced administrative and package-management entry marked `authoring`. These sets are defined by their normative contracts. A subset implementation MUST advertise its individual supported names and MUST NOT claim the complete capability. A capability declaration is not a promise that every input succeeds.

The `eight-tool`, `ecmascript`, `presentation`, and `components` capabilities are separate from the action collections. An implementation can interpret the standard data without executing JavaScript, implement actions natively, or inspect a component without activating it. A Reader can provide several such capabilities, but the word Reader by itself does not identify a conformance level. Availability of an installed type or action definition does not establish that an implementation can execute it.

> **[RBX-P4-002]** Action discovery MUST report what is actually executable in the selected context. A declaration whose implementation, dependency, host capability or selected-role authority is unavailable MUST either be identified as unavailable with a reason or be omitted from the executable list. Reading, preserving, importing and installing code MUST NOT, by themselves, activate that code or authorize external I/O. A package containing an action declaration does not establish that its implementation satisfies the action contract.

The machine-readable catalogues and schemas are normative companions to this Part. Annex B presents every action with its preconditions, effects, results, omission rules, errors and complete input/output JSON Schemas. The catalogue and Annex B express the same action contracts. A discrepancy is an editorial defect, not permission to select a weaker interpretation. The semantic clauses in Parts 1–4 apply in addition to schema validation.

4.2 Execution contexts and portable values

An execution context identifies a role and, for Digital participation, a Digital in the selected document. `reader-admin` concerns the host installation, library, connections and adapters; `redbook-admin` concerns the contents and configuration of one selected world; `digital` represents one

selected Digital's authored participation. The actions in Annex B are world actions and do not acquire Reader administration powers. Reader-specific operations use independently declared names and contracts.

> **[RBX-P4-003]** A world action MUST run under one explicit `redbook-admin` or `digital` context. In `digital` context, `digitalId` MUST identify a current Digital in that world. A supplied `actorId` MUST equal that selected Digital. In `redbook-admin` context, an action requiring an attributed participant MUST receive its explicit `actorId`. Pure administrative creation without an actor parameter, including the generic and Soma authoring actions, may create unattributed content; it MUST NOT infer the owning Digital as its author. Authority MUST NOT be inferred from an attribution credit, a matching display name, an API key stored in content, a request identifier, or a textual Relationship role. Only the host grants entry to a representation context.

Administration and participation are operational distinctions, not encrypted document ownership. The file holder can inspect and edit the file through appropriate administration tools. A Digital does not store or become a host login, credentials, model session or authorization token. An external AI can use a Digital context when the host delegates it; the AI remains external to the document. The standard neither asserts nor requires consciousness, persistent model execution or autonomous background activity.

> **[RBX-P4-004]** Action inputs, outputs and authored state MUST use the portable value algebra of Part 1 and value.schema.json. Consumers MUST preserve exact integer/decimal lexical values and typed references. They MUST reject non-finite ordinary numbers and ordinary integer values outside the safe exact range. `undefined`, functions, cyclic objects, dates as live objects and implementation-owned handles are not portable values. Host-only handles are represented by explicitly declared opaque strings and MUST NOT be serialized into authored state merely because they occurred in a call.

> **[RBX-P4-005]** Validation MUST distinguish absent members from members present with null. Defaults apply only where an action's omission rules or a creation-time field declaration says they apply. A read MUST NOT materialize missing defaults, fix ownership, normalize event numbers, alter timestamps, mark attention, or create a missing mailbox or Neocortex. Ordinary maps are unordered; list order is significant. Comparison of request signatures, definition compatibility and bindings MUST use the typed equality in Part 1, including exact tagged numeric lexemes and ordinary numeric equality of positive and negative zero, rather than serialized JSON member order.

The base schemas describe shape. They cannot alone establish reference closure, root/specialization compatibility, genuine Gregorian dates, membership authority, unique event sequence numbers or allocation consistency. Implementations apply those checks before accepting a committed result. Native XML strings and JSON values obey the same logical restrictions.

JSON Schema documents inside action descriptors and component schema resources are protocol-control JSON, not authored portable Property values. Native control catalogues use the canonical `rb:json` serialization in Part 2, clause 2.16. Within those schema documents, a property name such as `$rb` is literal schema data. An implementation storing such a schema as authored material retains

its original JSON resource, a text representation, or an explicit list-of-key/value representation; it does not force an ordinary map with that reserved key through the tagged-value grammar. Tool result containers and live callback contracts likewise use their explicitly declared schemas; action results inside `perform_action.result` remain portable values.

4.3 Action contract and atomic execution

An action descriptor contains `name`, `description`, `effect`, `inputSchema`, `outputSchema`, `modes`, `requirements`, `capability`, and current `available` status. An unavailable descriptor adds a reason. A custom participant write action additionally declares its `policyContextResolver` as specified in 4.17 and Annex D. `read` inspects a current world; `write` deliberately changes authored state; `external` changes a declared host-session facility, such as an extension activation. An output schema of `{}` does not constitute a complete implementation of a standard action whose catalogue specifies a narrower result.

> **[RBX-P4-006]** Before invoking a handler, an implementation MUST resolve the action, verify its availability and selected role, validate the original input against its full input schema, and validate the currently selected document state at the outer invocation boundary. Nested checked commands validate their own local inputs and constraints against the candidate; the temporary-dependency exception in 4.6 does not waive authority or value checks. Unknown input keys MUST be rejected when the schema disallows them. Schema constraints are cumulative with the action's preconditions and the domain constraints of Part 3. An advertised action MUST validate its successful result against its output schema before reporting success.

The following is the observable execution order for an action using the standard hook facility. A native implementation can use a different internal architecture if its observable semantics are the same.

1. Resolve any retained request receipt, then compare the invocation binding as specified in 4.10.
2. Check availability, original input schema, role and transaction eligibility.
3. Establish an isolated working state and reserved host-resource set for this operation.
4. Apply `rb.action.input`, checking each returned value and its authority constraints. Resolve omission defaults and actual policy contexts; check every applicable guard under 4.17, then run `rb.action.before`.
5. Invoke the synchronous handler `(input, call)` under the transaction-bound command contract in Annex D. Each creation and patch uses the declared record hooks. Ordinary handlers and hooks cannot start an independently committed inner action.
6. Dispatch the outer transaction's `rb.action.stage` exactly once, permitting checked dependent commands through its supplied transaction facade. Validate the candidate world and result. Run `rb.action.completed` with snapshots.

7. Revalidate the candidate world and result, recheck the bound source revision and current host/extension generation, confirm every staged change remains within its checked authority and declared effect, then atomically publish the candidate, its new revision and terminal receipt at one linearization point.
8. Release the commit lock, deliver change notifications and return the retained success response.

> **[RBX-P4-007]** A `write` action MUST commit as one logical transaction. Failure at any precommit stage MUST leave the prior authored state, allocation counter and accepted resource bytes unchanged. Dependent records, automatic Neocortexes, attribution changes, taxonomy changes and inventory changes are included in that transaction. Allocated but uncommitted random identities need not be reused; they MUST NOT appear as committed records. A failed operation MUST NOT produce a success receipt. An implementation MUST NOT return success for a change that it has already determined cannot be preserved under its advertised save contract.

> **[RBX-P4-008]** A `read` action, including its hooks, MUST leave the authored world unchanged under typed equality. A detected read mutation MUST abort with `READ_MUTATION` and restore the preceding authored state. A `write` that produces an identical logical state is a successful no-op and MUST NOT increase the host edit revision. A changed committed world MUST increase that revision exactly once, including for an atomic batch. Host edit revisions are concurrency tokens, not document event numbers and not edit history.

> **[RBX-P4-009]** Event-producing actions MUST allocate canonical positive decimal `sequence` values in the selected document's allocation order, advance `nextSequence` exactly, and create the complete `recordedAt`, `sequence`, `worldTime` observation described in Part 1. Sequence allocation is shared by Elements and Relationships. Replacing a current event observation, as in a deliberate attention report, allocates a new sequence and may leave a gap. No consumer may interpret gaps as missing records. Implementations MUST NOT coerce event numbers through binary floating-point or a language integer-string length restriction.

Ordinary edits preserve the original event observation unless an action explicitly records a new observation. The action catalogue identifies event-producing behavior. An edition, notification, attention report or message is created only when that deliberate action calls for it. A no-op request replay does not record another event.

> **[RBX-P4-010]** A postcommit presentation or listener failure MUST NOT undo a committed action or cause its retry to repeat the write. The operation's success and the secondary presentation diagnostic MUST remain distinguishable. Precommit hooks and handlers MUST NOT perform irreversible external effects. Actions requiring such effects MUST declare `external`, expose the applicable host contract, and MUST NOT participate in a world-atomic batch.

4.4 Query, editing and dependency behavior

The standard query is query.schema.json. A type filter accepts a full declared type identifier or an unambiguous core key and matches that type and its direct specializations. A primitive filter restricts the result to Elements or Relationships. The base enumeration is document Element order followed by document Relationship order. Evaluate against one bound document snapshot: first apply type, primitive and text predicates to the complete enumeration; then apply the selected standard Feed ordering, if any; finally apply `offset` followed by `limit`. A generic record query has no additional sort and preserves base enumeration. Defaults are empty text, offset zero and limit 100; limit is 1–1000. A greater request is rejected, not silently truncated. A title-ordered Feed over source order Z, A with limit 1 therefore returns A.

> **[RBX-P4-011]** Query text matching MUST be a literal substring match after Unicode 15.1 full default case folding of both query and searchable text, using CaseFolding.txt statuses C and F and excluding Turkic T mappings. No normalization, accent removal or locale-specific casing is implied. The searchable text is the title followed by the depth-first Property token stream, with exactly one U+0020 between tokens. Map keys are emitted before their values in Unicode scalar key order; list values retain order. String tokens retain their text, including empty strings; null emits `null`; Boolean values emit `true` or `false`; an ordinary number emits the ECMAScript JSON number spelling of its binary64 value with negative zero normalized to `0`; exact numbers emit their unchanged lexical text; typed references emit only their identifier. Empty containers emit no token. Tagged forms do not additionally emit `$rb` or tag labels. Implementations MUST use this fixed case-fold version for the standard profile; a different text-search profile must have a distinguishable identifier. Empty text matches every otherwise eligible record.

The generic query order is not event chronology. `rb.conversations.messages` uses exact event sequence order. Wall and collection queries use their declared placement positions; pin precedence is separate from event time. Equal placement positions are ordered by exact Relationship identifier in Unicode scalar order, as specified in Part 3. A generic query returns snapshots with independent value ownership, so editing a returned object does not edit the world.

> **[RBX-P4-012]** A generic record patch MUST accept only `title`, `properties`, `categories`, `tags` and `slots`. Supplied Properties are shallow-merged by key; a supplied key with null means null, not removal. Supplied lists and slot maps replace their corresponding whole value. Omitted members remain unchanged. Explicit property removal uses `rb.records.unset-properties`. Retyping and attribution editing use their separate authoring actions. No generic patch can overwrite `id`, `primitive`, event observations or a record's type.

> **[RBX-P4-013]** A dependency-aware removal MUST account for participant slots, typed references nested in Properties, mandatory ownership/home edges, local author links, the default Viewbox, declared code/resource dependencies and taxonomy children. It MUST reject a surviving dangling dependency. Opaque application strings MUST NOT be silently rewritten or treated as known references; a feature with opaque dependencies MUST declare an applicable dependency resolver or removal MUST fail with `DEPENDENCY`. `rb.records.deletion-preview` MUST return the complete current plan without changing anything; execution MUST recompute that plan against its bound revision.

For `rb.records.delete`, `removeRelationships=true` permits deleting Relationships incident to selected Elements; it does not silently delete unrelated dependent Elements or recursively erase entire conceptual homes. Required dependent Elements must be explicitly selected in the same operation or removed through a valid atomic batch. A surviving record attributed to a deleted Digital retains a descriptive credit: preserve an existing credit, or capture the Digital's document identifier, identifier and title before removing the local author link. Explicit attribution removal is a different action. Removing a type requires no surviving use as a record type, specialization parent or allowed participant type. Removing a supplied core declaration invalidates a complete-core claim.

> **[RBX-P4-014]** Creation MUST materialize declared defaults without overwriting supplied values, in root-then-specialization order. Retyping MUST retain record identity and primitive, retain supplied content, materialize only newly introduced missing defaults, and validate every affected participant/reference and ownership rule. A type replacement MUST retain its identifier, key, primitive and parent. It MUST NOT silently redefine a supplied core meaning while claiming the original core vocabulary. Taxonomy operations MUST keep Category ancestry acyclic and preserve or explicitly resolve every affected assignment.

4.5 Participation and domain operations

Part 3 supplies the semantic state machines and Relationship meanings for communication and Soma. The predicates below apply to every implementation of the standard actions. A current active Membership or an applicable explicit Participation grant can authorize contribution to its context; membership in a different context does not do so. A requested, declined, left or merely invited membership does not grant active participation.

> **[RBX-P4-015]** A participant action MUST check its current, applicable domain authority at execution time. Editing someone else's Message, publishing someone else's Post, closing their Conversation, confirming their Resource reservation or changing their Collection order requires the attributed author or explicit administration as described in the action entry. Creating a reply does not grant authority to edit its antecedent. A Digital writing a personal Soma MUST be its owning Digital; writing a shared Soma requires an applicable active contribution grant in that Soma's owning context. Authority follows the actual home Relationships, not labels or copied credits.

> **[RBX-P4-016]** `rb.memberships.set` MUST preserve the memberships of all other Digitals and apply the complete transition table in Part 3, clause 3.3. A self-operation can withdraw, request entry, respond to its own invitation or join under an explicit open enrollment policy as that table permits; it MUST NOT invent an invitation, self-elevate a role or bypass closed enrollment. Only the context author or administration can otherwise change a participant's role or status. When a Relationship groups several members, an implementation MUST split or reduce it atomically to change one member while preserving the others, or reject when dependent references prevent an unambiguous split. `rb.invitations.respond` MUST affect only its selected invitee; a multi-invitee invitation requires explicit valid splitting before this singleton response action.

> **[RBX-P4-017]** A supplied operation that identifies an Element MUST reject a Relationship identifier even when the identifier exists. This includes message subjects, reaction targets, participant slots and represented subjects. Generic typed-reference Properties may refer to either primitive, as Part 1 permits; slot participants cannot. Reply, membership and conceptual-scope predicates MUST be checked against the resulting graph, not merely against the existence of an identifier.

Conversation sends require an open Conversation and active participation. A supplied `replyTo` belongs to the same Conversation. A split preserves message identities and observations, creates a new thread and conceptual home, and moves selected Contribution edges while retaining pre-existing Reply links as historical references, including links crossing the resulting contexts. This retention does not permit a later send to create a new cross-context reply. Message reads sort by exact sequence, not timestamp, insertion order or display position.

Wall placement creates a new Relationship; it does not copy the content or change its author. Owner/administration placement is published. Other actors obey `open`, `members`, `reviewed` or `closed` contribution policy; reviewed submissions become pending. Pending and rejected state does not delete the source Post. A content event sequence representable as a safe ordinary integer is the default numeric placement position. Otherwise append at `floor(max(0, existing positions))+1` if representable; reject with `ORDER_RANGE` if not. Explicit positions are retained. Imported manual positions are not renumbered merely to match fresh event allocations.

Mail sends and replies follow the deterministic thread-selection algorithm in Part 3, clause 3.19, with envelope projection under 3.5. A sent Message records typed `threadId` and one matching Contribution. The public thread audience is sender plus To/Cc; Bcc recipients receive Delivery and organization entries but no automatic shared-home participation grant. With an antecedent, `threadMode: continue` reuses its open canonical thread only when the new public-audience set matches its stored `mailParticipants`; a changed audience creates a new thread and home. Missing or ambiguous antecedent thread metadata fails `THREAD_UNRESOLVED`, and a closed thread fails `STATE`, unless explicit `threadMode: new` requests a fresh exchange. Without an antecedent, a send always creates a new thread. Reply links remain attached to their actual antecedents across a deliberate thread change. Envelope projection, visibility, recipient derivation and Bcc protection are checked independently of thread selection. Mail organization changes only the selected recipient's entry; `rb.mail.move` changes its folder slot, never the Message or another recipient's entry.

> **[RBX-P4-018]** Upsert operations MUST identify their current record using the domain key in Part 3, not whichever matching record happens to be first in an implementation's array. Ambiguous competing current records MUST produce `STATE`. An upsert MUST preserve the current record's identity unless explicit splitting is required by 4.5. Current reactions, ballots, bookmarks, subscriptions, responses and attention state MUST NOT accumulate hidden prior versions. A deliberate explicit edition is a separately authored exception, not an automatic history policy.

Poll and questionnaire validation, request/commitment transitions, reservation overlap rules, circulation attribution, stream metadata and contribution release policies are defined in Part 3 and are binding preconditions/effects of their respective Annex B entries. Creating a Notification, Subscription, scheduled-looking date, Reservation or Stream does not dispatch a network message, start a timer, reserve an external resource, capture media or awaken an AI. Those external actions need their own host capability and explicit contract.

4.6 Atomic batches and result bindings

`rb.transaction.perform`, in the `authoring` capability, contains a nonempty ordered list of `{key,name,input}` steps. Keys are unique nonempty strings. The batch uses one selected world, one role and one concurrency binding. There is no per-step role escalation. Steps can read or write the candidate world produced by preceding steps. External actions and nested transactions are excluded.

> **[RBX-P4-019]** A batch MUST resolve each step's input against earlier successful step results, validate the resolved input against that action's schema, and apply that action's authority and domain rules. Only an exact object of the form `{"$result":{"step":"earlier-key","path":["member",0]}}` is a batch result binding. `path` is an ordered list of exact map-member strings and nonnegative array indexes; an empty path selects the whole result. A missing step, forward/self reference, missing member, wrong container kind or out-of-range index MUST fail with `BATCH_BINDING`. Binding detection MUST occur only in the batch input template, never by interpreting ordinary stored Properties on later reads.

Example: a creation step can be followed by a Relationship step whose `slots.subject` list contains `{"$result":{"step":"created","path":["id"]}}`. Replacement copies the typed result value; it does not interpolate strings or parse a selected string as code. A map with `$result` plus another member is ordinary input data and is validated as such. A literal map containing only this reserved batch form cannot be represented directly inside a template; an earlier read can supply it as a bound value, whose contents are not recursively reinterpreted.

[RBX-P4-020] A batch MUST stage every step and perform final global graph validation after the last step and the outer `rb.action.stage` dispatch. Temporary unresolved dependencies are permitted only between primitive mutations within one isolated handler. Public reads, queries and each nested action or batch-step boundary require a complete valid candidate; an incomplete intermediate graph cannot supply a step input or authority grant. Per-step input, output, type-local value, authority, policy and state-transition checks remain mandatory. Each nested action uses the Annex D checked transaction facade without independent commit; the outer stage hook is not recursively dispatched. A failed step, hook or final check MUST roll back all steps and resources. The successful output is the ordered list of `{key,result}` snapshots plus `committed:true`, never partial success. The host revision advances once for a changed batch. Event-producing steps allocate their distinct observations in execution order.

4.7 Blueprint capture and instantiation

A Blueprint is an independent reusable component graph. A whole RedBook is copied as a RedBook. The portable component envelope is specified in Part 2; its `graphPart` contains a graph conforming to blueprint.schema.json. The graph contains roots, records, type definitions, Categories, Tags, external binding requirements and required original resources. It does not contain a running instance, account connection, request receipt or future link to its source.

[RBX-P4-021] Capture MUST accept explicit record `rootIds`, supplemental `includeIds`, and registry `typeIds`, `categoryIds`, `tagIds`, and world-owned `installationIds`. Each omitted list is empty; at least one explicitly exported root, registry entry or installation is required. New captures MUST emit `exports:{records,types,categories,tags}` identifying these selections, with `roots` equal to `exports.records`. A definition-only or taxonomy-only export has empty record roots and needs no invented Element. Record closure begins with roots plus included records and adds required owned dependencies: owner to its Neocortex/ownership edge, Neocortex to resident Somas/home edges, Soma to member Objects, membership edges and conceptual Relationships. An external home remains an explicit binding. Arbitrary incoming Relationships are not implicitly selected. Capture MUST also close all root/specialization ancestry, recursively named slot allowed-types, Category parents and declared implementation/schema/resource dependencies under Part 2, clause 2.12. Every dependency is included exactly once; declared exports remain distinct from added closure. Selected installations bring their complete component packages and transitive component dependencies under Part 2.19, plus compatible declaration types, mapped Viewbox targets and typed configuration dependencies. External configuration/target references become explicit bindings. Source/manifest bytes remain unchanged and capture activates nothing. A component-only selection may have empty record exports. Component keys follow selected installation order and then dependency closure order.

[RBX-P4-022] For each selected record, capture MUST inspect participant identifiers and recursively typed-reference values and declare each unresolved external identifier once as `{key,label,type}`. A locally attributed selected Digital remains a local author reference. An author outside the selection MUST instead travel as a descriptive credit, preserving an existing credit or capturing the source document ID, Digital ID and current name. Capture MUST NOT manufacture an externally bindable author merely from a credit. Categories and Tags used by the graph, together with Category ancestors, MUST be available; a capture may carry additional declared terms, but import handles them explicitly.

`resourceReferences`, when present, records `{recordId,propertyPath,path}` dependencies into a selected record's Properties. Each property path consists of map-member strings; array indexes use canonical nonnegative decimal strings and are interpreted as indexes only when the current value is a list. The referenced value must identify the declared package-root-relative resource. A feature with another resource-bearing structure declares its own resolver. `assets` supplies the resource descriptors and `requiredParts` enumerates the original parts necessary to reproduce the graph, including carried source dependencies. Every listed part travels as original bytes in a portable component. A descriptive path or external URL alone is not a retained resource.

[RBX-P4-023] Capture and export MUST resolve resource dependencies from explicit typed declarations or a declared feature resolver. They MUST NOT infer dependencies by searching arbitrary serialized Properties for matching path substrings. An unresolved dependency MUST be reported and MUST prevent a claim of complete portable capture. Capture itself MUST NOT change the source world, consult a recording clock, advance chronology or activate copied code.

[RBX-P4-024] Instantiation MUST validate graph/export/binding identities, full declared closure, original resource availability/digests and semantic compatibility of every carried definition before committing. Semantic type comparison removes only top-level `label`, `description`, `area` and each field-rule `description`, then uses Part 1 typed equality on all remaining members. Identity, key, primitive, ancestry, traits, field kinds/requiredness/defaults/choices, slot constraints and ordered lists remain significant. A compatible existing type retains its destination presentation metadata; a new type retains all carried metadata. An incompatible definition with the same identity MUST fail `TYPE_CONFLICT`; display translation alone MUST NOT. Bindings are an ordered list of `{sourceId,targetId}` pairs, where sourceId is the declared external binding key; each key occurs exactly once. Each external binding must resolve to a compatible destination record of the required primitive/type. Missing, undeclared or contradictory bindings fail `BLUEPRINT_BINDING`.

[RBX-P4-025] Instantiation MUST allocate fresh record identifiers, including Relationships, and remap all participant slots, typed-reference Properties, included author references and roots. It MUST allocate fresh taxonomy identities and remap assignments/parents. Type identities retain their semantic identities under the compatibility rule above. Preserve content, credits, timestamps, world-time observations, list order and explicit positions; do not rewrite opaque strings as typed references. The returned `mapping`, `typeMapping`, `categoryMapping` and `tagMapping` are lists of `{sourceId,targetId}` pairs, covering every corresponding included source identity once in source collection order; `roots` retains exported record order and may be empty for registry-only import. Optional `title` replaces only the first record root and is invalid when no record root exists. Registry display metadata is not renamed through that argument. For a graph with required resources, the caller supplies `resourceHandle` identifying a host-accepted Blueprint component with exactly matching graph and inventory. The transaction allocates one fresh canonical UUID import key and copies every required source path `p` to `redbook/imports/<key>/p`. Original bytes and relative resource hierarchy are preserved. Declared `resourceReferences` and asset paths are rewritten using that exact mapping; implementation/schema/resource metadata requires its declared resolver, never substring replacement. Unknown required path-bearing features fail `RESOURCE_UNAVAILABLE`. Imported source uses this prefix as its virtual root under Annex D. No extension installation is implied. The result includes `resourceMapping` for all required parts and `resourceRoot`; without required parts these are `{}` and null. Resource copying and all reference rewrites share the graph transaction. The same transaction adds a current descriptor to `settings.resourceImports`, an array governed by resource-import.schema.json. It retains `resourceKey`, `resourceRoot`, the complete source-to-destination `resourceMapping`, remapped `typeResources` (a list of `{typeId,parts}` entries), `resourceDependencies` and `resourceReferences`. Dependency keys/values and reference paths name destination parts; record IDs are remapped and type IDs retain their semantic identities. The root equals `redbook/imports/<resourceKey>/`, every mapping value equals that root plus its source key, and every declared destination path is a mapping value. Every declared record/type identity must still exist. Re-export consults these current declarations; removal or modification must atomically update them or fail dependency validation. They describe current resources, not import-event history. No descriptor is added for a resource-less import.

[RBX-P4-026] Imported event observations MUST receive a fresh contiguous block beginning at the destination's current `nextSequence`, assigned in exact source-sequence order across both primitives. Duplicate, nonpositive or noncanonical source sequence text MUST reject the graph. Non-event records consume no sequence values. The destination counter advances by the number of imported events; pre-existing destination events are not renumbered. All source recording timestamps and world-time values remain unchanged. Allocation and graph/resource changes MUST roll back together on any import failure. Repeated independent imports create distinct instances; replay of the same retained request creates only one.

A copied Newsletter retains its current issue-allocation frontier. An Issue bound into an existing Newsletter must have an unused ordinal at or beyond the destination frontier, then advance that frontier past it atomically. An uninitialized populated destination or conflicting/lower ordinal is rejected under Part 3 rather than silently renumbering a published Issue. This allocation state is distinct from global event chronology.

4.8 Original resources and package-aware actions

An asset descriptor records its package-root-relative `path`, `mediaType`, byte count and lowercase SHA-256 digest, with an optional descriptive `name`. Paths and resource inventories obey Part 2. Host staging precedes `rb.assets.receive` and `rb.extensions.install`. A `resourceHandle` names bytes already accepted by that host under an explicit external workflow; it is not a URL-fetch instruction and is not a document identifier.

> **[RBX-P4-027]** Receiving an asset MUST verify that the handle belongs to the current host/delegation scope and still names the staged original bytes. The implementation MUST compute its byte count and digest, reject an occupied destination unless `replace=true`, validate affected declared dependencies, and commit bytes, descriptor and inventory atomically. A replacement MUST NOT overwrite a structural/control/code part through the ordinary asset action. An unavailable staging or package-I/O facility MUST produce `CAPABILITY_UNAVAILABLE` or `RESOURCE_UNAVAILABLE`, never a descriptor pretending the bytes were retained.

> **[RBX-P4-028]** `rb.resources.read` MUST read only from the accepted bound package snapshot. Its `offset` and `length` are nonnegative byte counts; length is at most 16,777,216. The returned base64 is canonical padded RFC 4648 base64 of the returned bytes. `bytesRead` is the smaller of requested length and bytes available from offset, including zero at or beyond EOF; `eof` is true when `offset+bytesRead` reaches or exceeds the part length. `partSha256` describes the complete accepted part, not just the returned range. This action MUST NOT follow external relationships, execute source, mark attention or decode/re-encode the original resource.

> **[RBX-P4-029]** Asset or extension removal MUST account for both declared record dependencies and package dependencies. An implementation unable to determine whether an opaque feature depends on a part MUST refuse destructive removal unless an explicit feature-specific removal operation resolves that dependency. Uninstalling executable source MUST preserve authored content by default. Removing content requires the explicit `deleteAuthoredData=true` selection and a dependency-valid explicit `recordIds` list; source removal MUST NOT be used as an implicit world reset.

4.9 The eight-tool interface

The `eight-tool` capability defines the RedBook domain interaction contract through the eight stable entry points below. interfaces.json specifies that domain contract; its role-bearing bindings and result shapes are not the public Jellybox WebMCP tool signatures. The optional `jellybox-webmcp-1.0` capability exposes this contract through the Jellybox WebMCP Framework 1.0, Standalone Core, using the explicit mapping in §4.9.1. Both layers preserve independent interaction state, deliberate action discovery and separate human presentation. `readme` is interface guidance, not arbitrary instructions extracted from authored content.

Tool	Required behavior
`readme`	Create a fresh host-owned independent interaction when no interaction ID is supplied; otherwise describe a known active interaction. Return guidance, the eight names, current binding, availability and retry-retention policy.
`current_context`	Return the interaction's current binding and separately the currently visible Viewbox, if any.
`list_contexts`	List selectable contexts and permitted roles, including host/document relationships. Listing does not enter or delegate a context.
`enter_context`	Compare `selectionRevision`, authorize the explicit new context and role, then replace only this interaction's selection and increment its selection revision.
`list_actions`	List known selected-context actions with effects, capability and truthful executable/unavailable status.
`describe_actions`	Return complete descriptors and a fresh binding in requested name order. Fail the call for an unknown or role-inaccessible requested name.
`perform_action`	Invoke one described action against the exact binding, applying schema, authority, retry and atomicity rules.
`navigate_to_view`	Change visible presentation to a valid selected-world Viewbox without changing interaction selection, role, event chronology or authored default presentation.

[RBX-P4-030] An interface MUST maintain independent interaction state and MUST NOT use human navigation, another agent's selection, window focus or the most recently invoked global tool as an implicit replacement for that state. A new interaction ID and its retry `scopeId` MUST be generated or accepted through host-controlled unpredictable session allocation. Supplying an unknown or expired ID to `readme` MUST fail rather than recreate it. Reopening a file or restarting a host MUST use a new scope unless the implementation actually retained and recovered the complete associated revision and receipt state.

[RBX-P4-031] `enter_context` MUST explicitly authorize the requested role. A Digital selection requires a present Digital in the selected world and host permission for that exact interaction to represent it. On failure, selection and revision remain unchanged. On success, even an explicit reselection advances `selectionRevision` once; a non-Digital role clears `digitalId`. A Reader context permits only Reader administration, and a selected Viewbox retains its owning document as `documentId`. The selected Viewbox does not gain independent ownership of world data.

A binding contains `scopeId`, `interactionId`, `documentId`, `instanceId`, `contextId`, `selectionRevision`, `role`, `digitalId`, `editCounter` and `capabilityRevision`. `documentId` is persistent authored identity; `instanceId` is an opaque host-generated identifier for one loaded mutable instance. Two copies with the same document identity have different instance IDs and independently addressable context IDs. `list_contexts` exposes both. A Viewbox context retains its owning loaded instance; `navigate_to_view` resolves its Viewbox identifier only within that selected instance. Reader context uses null document/instance/Digital IDs. Host-generated context IDs, rather than document or Viewbox IDs alone, distinguish selections. Reopening creates a new instance ID unless the host actually restores the complete prior instance and receipt state. These values are never stored in authored content.

The binding's role/Digital identify purpose, `editCounter` identifies the accepted instance revision and `capabilityRevision` identifies the available operation/permission surface. Revisions are nonnegative safe integers. Before exhaustion, the host invalidates the affected scope and starts a fresh one; it never wraps or rounds a revision.

> **[RBX-P4-032]** A new `perform_action` attempt MUST compare every binding field using typed equality to the selected interaction and loaded instance. Missing, extra, mistyped or unequal fields MUST reject as `STALE_ACTION` before execution. A mutating invocation MUST additionally establish freshness at its linearization point: either hold an instance mutation lock from final binding/authority check through commit, or atomically compare-and-swap the expected instance and capability revisions when committing its isolated candidate. A losing candidate MUST be discarded with `EDIT_CONFLICT`, without publishing outputs, events, resources or notifications. Two distinct writes based on the same revision cannot both overwrite that revision; at most one changed commit succeeds. Action/permission/capability changes must advance capabilityRevision. A retained terminal replay is resolved before freshness comparison.

> **[RBX-P4-033]** `perform_action` MUST require a nonempty `requestId` for `write` and `external` actions. Its success object contains the validated `result` and the post-operation binding. `describe_actions` MUST expose the real output schema and declared modes; `list_actions` MUST NOT advertise an arbitrary registered function as a standard action unless that function implements the complete standard contract. All tool schemas MUST reject undeclared input keys.

4.9.1 Jellybox WebMCP mapping

The public framework uses exact `target`, `contextRef`, `actionRef` and `viewRef` references, opaque string revisions, complete action descriptors and the common `protocolVersion: "1.0"` result envelope. Its eight input schemas, output schemas and action conventions are pinned in the RedBook Jellybox WebMCP profile. This mapping connects that public contract to the RedBook rules below; it does not rename incompatible domain signatures into framework signatures.

[RBX-P4-065] An adapter claiming `jellybox-webmcp-1.0` MUST implement the complete applicable Jellybox WebMCP Framework 1.0 `standalone-core` contract and this mapping, in addition to `eight-tool`. It MUST register the framework's eight required tool names with their canonical descriptions and unchanged public input schemas, return its tool-specific output shapes and identify `interfaceEdition: "1.0"`, `interfaceProfile: "standalone-core"` and `protocolVersion: "1.0"`. Domain fields such as `role`, `digitalId`, `name` and `input` MUST NOT be inserted as extra public tool arguments or extra binding fields.

[RBX-P4-066] Each advertised framework `contextRef` MUST resolve to an immutable, explicitly described RedBook context, role and Digital tuple within the host security scope. `enter_context(interactionId, contextRef, expectedSelectionRevision)` MUST authorize that exact tuple and apply the domain selection atomically. Listing, human navigation and reference possession grant no role. Two loaded copies of one document remain distinguishable. Framework revision strings MUST map without loss to their corresponding domain revision and MUST NOT be treated as interchangeable selection, descriptor, grant or content revisions.

[RBX-P4-067] Every executable framework descriptor in this RedBook adapter profile MUST include both `binding.capabilityUse` and `binding.capabilityUseRevision`. Together they bind an immutable host-held prepared record containing the complete original domain binding, action identity and security scope, including `scopeId`, `interactionId`, `documentId`, `instanceId`, `contextId`, `selectionRevision`, `role`, `digitalId`, `editCounter` and `capabilityRevision`. The descriptor MUST expose fixed revision preconditions for the bound loaded document and any additional required state. Each fixed check MUST include the exact current subject reference and revision; its contract MUST identify ordinary discoverable read actions for refreshing that state. A client can use those returned checks directly and can acquire replacements through those reads or a fresh description, without a hidden token or resolver chain. Before effects, the adapter MUST validate the complete returned framework binding, these preconditions and the original domain binding under [RBX-P4-032]. It MUST NOT reconstruct a fresh current binding to make a stale prepared invocation succeed. Missing, changed or unsupported required checks reject before effects.

[RBX-P4-068] Framework `actionRef` MUST resolve to the exact advertised domain action name; `operation.id` identifies that name and `parameters` supplies its typed domain input. `describe_actions` MUST preserve requested reference order and duplicates and return complete framework descriptors with the real action input/output schemas. An action result maps to framework `data`, not an extra domain-result envelope. Optional discovery modes, pagination, exact target checks, availability and out-of-context inspection MUST follow the framework contract; inspection never changes selection. All advertised references and schemas MUST be usable without hidden application tokens or inferred authority.

[RBX-P4-069] An accepted framework request identity MUST reserve one underlying domain attempt. The original public references, binding, parameters and preconditions are compared under framework request-equality rules; the resolved original domain binding, action and typed input remain part of the domain fingerprint. Exact retained replay MUST be resolved before freshness checks at either layer. Terminal successes and failures, pending reservations, retention, capacity refusal and lost-scope behavior MUST continue to satisfy [RBX-P4-034]–[RBX-P4-036] and framework §7.2. A reconnect or remapping MUST NOT create a second dispatch. The public envelope MUST truthfully distinguish completed, rejected, cancelled, failed and unresolved outcomes and report known effects; a committed operation cannot be described as rolled back merely because result serialization, presentation or transport delivery failed.

[RBX-P4-070] Framework errors MUST use its common envelope and the mapping in jellybox-webmcp-profile.json, retaining safe domain diagnostics where needed. An error is never successful action data. `navigate_to_view` with `contextBehavior: "preserve"` MUST preserve domain selection. Explicit `use_view_default` MUST additionally authorize the view's advertised domain tuple and atomically compare-and-adopt it using `expectedSelectionRevision`, as framework §6.8 requires. Neither mode changes authored world data or delegates a Digital implicitly. Native navigation that ends an instance MUST follow the framework's actual no-result and recovery rules; it MUST NOT fabricate a callback result or promise retention the host does not provide.

A RedBook document, headless authoring engine or native/in-process domain interface does not require WebMCP. The specification-reading website is a separate application of the Jellybox framework: exposing its website and specification contexts does not claim that the site implements a Reader, delegated Digital participation, world mutation or this Reader adapter profile.

4.10 Retry, failure and transport mapping

Request identity is scoped by host retry scope, independent interaction and request ID. Equal text in different interactions denotes independent requests. A fingerprint includes action name, typed input and complete invocation binding, including loaded instance. Tagged numeric lexical values and list order are preserved. Before starting any mutating handler, the host atomically reserves the scoped request identity with that fingerprint. The reservation state is `in-progress`, then one terminal success or failure; receipts live in host state, not in the RedBook.

[RBX-P4-034] Concurrent attempts with the same reserved request identity and equal fingerprint MUST join the existing attempt or return retryable `REQUEST_IN_PROGRESS`; they MUST NOT execute a second handler. An unequal fingerprint MUST return `REQUEST_REUSED` even while the first attempt is still pending. Transition to terminal receipt and publication of the corresponding commit outcome MUST be atomic from the perspective of retries. A host that loses pending outcome state MUST reject the lost scope or return `OUTCOME_UNKNOWN`; it MUST NOT infer that the action failed and re-execute it. External effects use the same reservation discipline but retain their expressly disclosed uncertainty when the host cannot recover an outcome.

[RBX-P4-035] For a retained successful request, an equal fingerprint MUST return the original success object without rerunning the action or its hooks, allocating another event, reactivating code or changing any revision. The replay returns the original response binding even if later operations made it stale; clients obtain a fresh binding through discovery before a new action. Reuse of the same scoped request ID with an unequal fingerprint MUST fail with `REQUEST_REUSED`, including a reuse with changed bindings. Comparison MUST ignore JSON map-member serialization order.

[RBX-P4-036] The host MUST advertise receipt-retention duration, remaining capacity and scope expiry through `readme`. Capacity includes in-progress reservations and terminal receipts. It MUST retain those identities for the advertised active scope or refuse new work with `RECEIPT_CAPACITY` before execution. It MUST NOT evict an individual receipt and later interpret that ID as new in the same scope. Expiry stops accepting new work; it does not release a pending request to a competing worker. An already accepted invocation either completes under its reserved identity or is rejected before effects; its outcome remains queryable by exact retry for its advertised minimum retention. Lost/expired scopes reject with `INTERACTION_EXPIRED`. A transport disconnect never authorizes replay as a new operation in another scope.

A terminal precommit failure retains its fingerprint and error receipt until the same retention boundary; exact retries return that failure without repeating hooks. Corrected input uses a new request ID. `REQUEST_IN_PROGRESS` is a nonterminal response and may be retried with the same fingerprint. If transport loses a committed response, same-scope retry recovers it. Neither a missing response nor a host restart establishes that an external action did not happen.

[RBX-P4-037] Tool errors MUST use the interface error envelope: `error.code`, human-readable `message`, `retryable`, current `context` when safely available, and optional portable `details`. They MUST NOT masquerade as successful action results. The following codes have stable categories; implementations may add namespaced diagnostic subcodes without changing them.

Code	Meaning and client consequence
`INPUT_SCHEMA`, `OUTPUT_SCHEMA`	Input contract violation, or invalid implementation result. Correct input or report implementation failure; do not treat as a committed success.
`CONTEXT`, `PARTICIPATION`, `HOST_CONTEXT`	Wrong selected purpose, unmet domain authority or absent host delegation. Repeating unchanged input does not grant authority.
`ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`	Unknown operation or known operation whose implementation/dependency is unavailable. Inspect discovery.
`STALE_ACTION`, `SELECTION_CONFLICT`, `EDIT_CONFLICT`	Bound state changed. Rediscover/review before creating a new request.

Code	Meaning and client consequence
`REQUEST_ID`, `REQUEST_REUSED`, `REQUEST_IN_PROGRESS`, `RECEIPT_CAPACITY`, `INTERACTION_EXPIRED`, `OUTCOME_UNKNOWN`	Missing ID, conflicting reuse, an existing pending attempt, full receipt scope, unavailable scope or an unrecoverable outcome. The server must not silently replay as a new write.
`REFERENCE`, `DEPENDENCY`, `TYPE_CONFLICT`, `BLUEPRINT_BINDING`, `THREAD_UNRESOLVED`	Missing/wrong reference, unresolved dependency, incompatible definition or invalid explicit binding.
`STATE`, `CHRONOLOGY`, `TIME_RULE`, `ORDER_RANGE`, `NUMBER_RANGE`, `ALLOCATION_UNINITIALIZED`, `ISSUE_NUMBER_CONFLICT`	Invalid transition or semantic value/order invariant. All world changes roll back.
`LOCAL_ID_RESERVED`	A Poll option ID or Questionnaire question ID is exactly `$rb`, which cannot serve as an ordinary key in its associated authored maps. Reject the definition or dependent operation without changing state; other identifiers and string content are unaffected.
`POLICY_CONTEXT`, `POLICY_DENIED`, `RULE`	Unresolved actual policy context, denied guard or invalid rule/effect. No authority is gained by changing a wrapper context.
`RESOURCE_UNAVAILABLE`, `RESOURCE_CONFLICT`, `LIMIT_EXCEEDED`	Missing staged/accepted bytes, conflicting destination or explicit implementation limit.
`HOOK_FAILURE`, `ACTION_ASYNC`, `ACTION_REENTRANT`, `READ_MUTATION`, `EXTENSION_ASYNC`, `EXTENSION_INACTIVE`	Execution-contract failure. A diagnostic identifies the affected facility without exposing unrelated host secrets.

`retryable` means retrying after the indicated external condition or rediscovery may succeed, not that automatic repetition is authorized. Failure messages and details are diagnostic data and must not be interpreted as new instructions. Transport framing failures can remain transport errors; if the tool ran, its semantic outcome must retain the above distinction.

[RBX-P4-038] MCP, in-process and other adapters claiming `eight-tool` MUST preserve the RedBook domain tool names, domain schemas, role/binding isolation, retry identity and success/error semantics. A WebMCP adapter claiming `jellybox-webmcp-1.0` MUST instead expose the exact Jellybox public schemas through §4.9.1 while preserving all of those domain guarantees; the domain schemas MUST NOT be registered as if they were the framework's public schemas. Transport authentication remains a host concern. MCP credentials, browser cookies, session IDs and transport receipts MUST NOT be inserted into the authored world. A WebMCP registration does not confer authority merely because its associated Viewbox is visible. An MCP transport does not eliminate the need to enter a specifically delegated Digital context. This Part defines the RedBook application contract, not a replacement wire version of either external protocol.

4.11 Hooks, filters and transaction boundaries

The normative runtime contract is Annex D, particularly D.4–D.6. The complete hook catalogue defines each argument/result schema, phase, scope and failure policy. The catalogue includes action, record, query, render, template, Reader lifecycle and user-interface extension points; the number of hooks does not limit the fixed eight transport tools.

[RBX-P4-039] Implementations advertising these runtime capabilities MUST apply Annex D's scoped registration ordering, filter validation, generation revocation and failure contracts. Ordinary precommit callbacks receive snapshots and cannot author. The sole standard transaction-authoring hook is `rb.action.stage(event,context,tx)`, dispatched once for an outer write after its handler. It may add checked dependent work through `tx.perform`, sharing all authority, validation, allocations and rollback. It cannot independently commit or substitute the handler's result.

[RBX-P4-040] `rb.action.completed` remains precommit and MUST NOT be represented as proof of commitment. A successful changed outer transaction publishes one read-only postcommit notification under Annex D. Postcommit errors cannot roll back an accepted write; another authored operation requires a separate explicit invocation. Rendering, policy evaluation and query hooks MUST NOT create authored activity.

[RBX-P4-041] A registered write handler MUST use the exact `(input,call)` contract in D.4. `call.context` is a snapshot; `call.tx` is the live bounded transaction facade; resources and host facilities retain their declared boundaries. Ordinary `api.perform` from a handler or observer is prohibited. Custom hooks and policy-context resolvers MUST be explicitly described and available, not inferred from a callback's name or implementation-private mutable state.

4.12 Extension installation, activation and cleanup

A Theme or Plugin targets Reader, RedBook or Viewbox, giving six distinct component kinds. Reader installation is host configuration; RedBook and Viewbox installations are authored world declarations. Direct supplied-core and Viewbox-type declarations are distinct from installed third-party components. Part 2 defines the component envelope; Annex D.1–D.4 defines capability negotiation, installation roots, dependency resolution and executable module behavior.

> **[RBX-P4-042]** World installation actions MUST accept only a RedBook target identifying this world or a Viewbox target identifying one of its Viewboxes. Installation validates the complete manifest, inventory, configuration and declared dependencies before copying any payload. It allocates a fresh installation identity and resource UUID, and retains original paths beneath `redbook/installations/<resourceKey>/` as required by D.2. It MUST NOT execute setup. Replacement requires an explicit existing installation identity, inactivity, and validation of every surviving dependent; it retains that installation identity but uses a fresh resource UUID and payload root, replacing the old payload atomically after all dependency checks. Original component manifest bytes remain unchanged within the new root.

An installed descriptor contains `resourceKey`, `payloadRoot`, `enabled` and `order` in addition to component identity/version, target, entry, installation identity, target identity, stored manifest path and configuration. A new installation begins with `enabled:false` and `order:0`. Explicit replacement preserves the previous installation's enabled/order intent, while remaining inactive until explicitly activated. `manifestPart` names the relocated manifest under its payload root; `entry` retains the original component-root-relative path. The descriptor is not an activation receipt.

> **[RBX-P4-043]** `rb.extensions.configure` MUST replace only supplied configuration, authored state, enabled intent or order. Omission preserves the current member. This changes desired state without executing code. Changes to effective implementation, configuration or authority revoke the affected active generation and bound capabilities; explicit reactivation requires fresh host checks. An unrelated inert asset change does not revoke portable participation rules merely because bytes changed. Activation is explicit and MUST enforce Annex D's scoped dependencies, host grants, isolated module generation and synchronous setup contract. Unsupported facilities are unavailable, not implicitly granted.

> **[RBX-P4-044]** Activation, deactivation and removal MUST implement the revocation, tracked cleanup and failure discipline in Annex D.11. Late asynchronous settlement cannot restore a revoked registration. A failed activation plan cleans only its newly activated instances, retaining pre-existing activity. Removal requires inactivity and absence of surviving installation/resource dependencies; deleting bytes is not cleanup.

> **[RBX-P4-045]** Availability changes MUST increment the affected capability revision and invalidate bound invocations. Host grants, actual activity, jobs, cleanup closures and generation tokens MUST remain transient host state. Portable declarations preserve authored intent and original source while permitting another host to refuse execution with a concrete diagnostic.

4.13 Viewbox lifecycle, presentation and the host bridge

A Viewbox is an authored Element selecting an implementation and configuration for experiencing a world. The Reader provides file handling, host permissions and transports. Viewboxes read the same accepted world and invoke the same checked actions; they do not become another authoritative record store. Annex D.7–D.10 specifies the complete bridge, render handles, Theme templates, stable regions and Reader facilities.

> **[RBX-P4-046]** A DOM implementation MUST use `mount(root,book,options)` and its synchronous disposer under D.7. The scoped bridge supplies read/query/discovery/perform/snapshot/change notification, resources, and `beginRender`, `completeRender`, `cancelRender`. A render MUST obtain and complete its own generation-bound handle. This is the standard route to `rb.viewbox.rendered`; neither a Plugin dispatching the core hook nor a host emitting a duplicate mount notification is equivalent.

> **[RBX-P4-047]** Initial empty renders and subsequent successful renders MUST follow the same completion contract. Context/configuration/implementation replacement and unmount MUST revoke all old handles, controls and subscriptions. Failed or canceled renders emit no completion. Merely reading, rendering, navigating or applying a Theme does not write records or attention.

> **[RBX-P4-048]** DOM roots, selectors, theme tokens, templates and interface contributions MUST use the scoping and composition rules in D.8–D.9. Authored text is text unless an explicitly supported content renderer governs it. Declarative template data is an authorized projection, never a mutable store or automatic remote fetch permission. Keyboard access, accessible names, readable state/errors and usable empty states remain required when a Theme is applied.

> **[RBX-P4-049]** `rb.presentation.select` MUST distinguish omission from explicit clearing. Omitted `viewboxId` or `theme` preserves that setting. `viewboxId:null` sets the default Viewbox to null; `theme:null` removes that optional member. A non-null selection must resolve to a compatible available declaration. This changes authored selection only; activation and transient navigation remain separately explicit operations.

4.14 Limits, implementation assessment and interoperability

The standard requires semantic precision within an implementation's declared resource limits, not infinite memory, unlimited file sizes or arbitrary JavaScript execution rights. A host can decline a complex file, large resource, unsupported content format or requested external capability. Declining must remain different from corrupting, silently dropping, falsely accepting or activating it without permission.

[RBX-P4-050] Implementations MUST declare relevant limits and fail before commitment when those limits prevent an operation. Event-number length MUST NOT be limited accidentally by a host numeric conversion facility when text comparison/allocation is supported within the accepted model size. Date, number, query and schema behavior MUST remain exact within declared support. Implementations MUST NOT lower a semantic validation rule solely to classify a supplied fixture as fully executable.

[RBX-P4-051] A conformance report MUST distinguish schema/structural acceptance, core-domain validation, carried source preservation, actual action behavior, interface behavior, component activation, presentation behavior and native host qualification. Passing selected fixture tests or loading registered functions does not by itself prove complete `core-operations`, `eight-tool`, `ecmascript` or `presentation` conformance. In particular, placeholder result schemas, serialization-order-sensitive binding checks, per-receipt silent eviction, mutable Viewbox bridges, heuristic resource discovery and undeclared host authority do not satisfy their respective stronger contracts above.

The format enables independent implementations, including hosts written without the reference JavaScript. Portability is achieved by carrying authored content, declarations and required original resources with explicit contracts. It is not achieved by storing one host's process state or by assuming that every recipient shares its installed services.

4.15 Scoped participant editing and classification

The participant paths are defined in Part 3, clauses 3.17, 3.24 and 3.25 and the lifecycle catalogue. `rb.content.edit`, `rb.content.unset-properties` and `rb.content.classify` resolve one of three scopes: conceptual content in its actual home; the finite author-owned descriptors; or additional fields/classifications of a standard primary record. `rb.content.delete` covers the conceptual and permitted descriptor scopes only. These actions do not grant general record administration. Ownership and policy contexts come from validated structure, never from a caller's asserted owner.

[RBX-P4-052] A Digital write to conceptual content MUST satisfy the exact personal/shared authority rules in 3.17. Descriptor and specialization-field writes MUST instead satisfy 3.24 and 3.25, including their inherited state and participant checks. Personal content is editable by the owning Digital. Shared content requires active participation and either the record's local contributor attribution or the explicit context policy `participation.conceptualEditing:"members"`; absent policy means `"contributors"`. Context authorship alone is not an editing grant. An implementation MUST NOT silently retry a rejected participant write as administration.

> **[RBX-P4-053]** Conceptual editing MUST preserve identity, type, attribution, observations and home assignment. Supplied Properties merge by exact key; null stores null. Explicit unset accepts every exact string key, including the empty key. Supplied slot maps replace whole maps, so omission of a slot within that map clears that optional slot. Classification replaces supplied lists and retains omitted lists; new terms are allocated and appended in request order, without rewriting existing term definitions. Deletion MUST recompute its full dependency plan and check both eligibility for this deletion path and the actor's authority for every removed record. Any unauthorized dependent or surviving dangling reference aborts the whole operation, including all term and identifier allocation.

Soma creation accepts a root or specialization and Category/Tag assignments. The default is the corresponding root with empty assignments. A batch can combine these participant operations while retaining their scopes and one rollback boundary. Presentation can expose authoring controls against these actions without granting an unrestricted administrative patch.

4.16 Domain lifecycle, clearing and finite projections

> **[RBX-P4-054]** `rb.domain.transition` MUST implement exactly the states, authorities and dependent Decision behavior in Part 3, clause 3.18. The lifecycle catalogue is normative. Same-state requests require authority and are no-ops. Unlisted transitions fail `STATE`; unused or unknown detail members fail input/domain validation. A Decision is authored only for the specified Proposal and Decision Round transitions, with supplied procedure/body and optional dissent. Other state changes preserve existing event observations and do not create hidden revisions.

`rb.posts.edit` replaces supplied title/body for an authorized draft or published Post. `rb.mail.folders.create` creates a folder tied to one Mailbox. `rb.mail.move` changes only the selected recipient's entry; `folderId:null` removes its optional folder slot. `rb.questions.accept-answer` similarly accepts `answerId:null` to remove the accepted-answer association without deleting either question or answer. Empty supplied lists clear classifications. These explicit forms avoid equating null values, omitted arguments, removal of a Property and deletion of a record.

> **[RBX-P4-055]** Newsletter publication and initialization MUST obey the persistent allocator in 3.20, including deletion/reopen and Blueprint binding behavior. Allocation and publication share one transaction, and replay consumes no additional number. `rb.events.occurrences` MUST implement the finite exact-clock projection in 3.22, apply all relevant exceptions before sorting/paging, and return explicit `nextOffset:null` only at the end. Neither projection nor policy evaluation authors event observations or background jobs.

4.17 Policy-context resolution and explicit rule processing

Policies and rules use the exact shapes and predicate evaluation in Part 3, clauses 3.12 and 3.21. Their invocation contracts are `rb.rules.evaluate` and `rb.rules.process`. Rules do not grant base authority or turn inspection into authorship. Automatic guard enforcement and deliberately requested rule effects are different stages.

[RBX-P4-056] Before a Digital-context write, the dispatcher MUST derive every policy context from the checked action name, validated input and current world using the lifecycle catalogue's resolver. Multiple contexts are deduplicated and sorted by exact identity. Required unknown/ambiguous resolution makes the operation unavailable or fails `POLICY_CONTEXT`; a caller cannot replace these contexts. A custom Digital-context write action MUST declare a stable `policyContextResolver` understood by the host and resolving an existing declared context through Annex D.6. Read-only custom actions need no resolver. Standard context-creation actions retain the explicit no-existing-context default in the normative lifecycle catalogue; that default is not an undeclared custom resolver. The resolver is registered semantic capability metadata, never arbitrary request text. Missing required resolver support prevents availability.

[RBX-P4-057] The dispatcher MUST evaluate applicable `require` and `deny` guards before mutation, in priority and rule-identity order, against one immutable `{actor,context,action}` snapshot per context. `matchedRuleIds` lists true conditions; `deniedBy` lists failing guards; neither list is a grant of authority. Any failing guard rejects with `POLICY_DENIED`. Ordinary writes MUST NOT execute selected effects. `rb.rules.evaluate` validates its requested context against actual resolved contexts, evaluates all guard sets and returns only a `validation:"policy-only"` assessment; later checks and concurrency can still reject the action.

[RBX-P4-058] `rb.rules.process` MUST evaluate all actual guard sets, then perform the checked base action and selected effects in context, rule and effect-list order in one transaction. Selection uses the pre-action snapshots; each effect observes preceding candidate writes and retains ordinary authority, schema and guard checks. Base/result and each effect result validate against their actual action descriptors. Neither rule wrapper, a nested transaction, an external effect nor a policy-changing action is eligible as a base/effect operation. Failure poisons the complete transaction even if extension code catches it. The result contains the base result and ordered `{ruleId,index,result}` receipts, with no implicit retained run log.

4.18 General custom participant writes

[RBX-P4-059] A custom action requiring primitive changes outside the supplied participant workflows MUST declare the `rbx.participant-authoring.1` contract and use Annex D.12's public `authoring.register` and checked transaction mutation methods. Action discovery MUST expose its portable input/output schemas, Digital mode, write effect, required profile and unavailable reasons. Registering a name, resolving policy contexts or invoking an admin-only action in a transaction MUST NOT be treated as a substitute for this authoring contract.

> **[RBX-P4-060]** The dispatcher MUST combine the portable installation's enabled intent, declared authoring rule, world participation grant and applicable guards with the host's independent execution and Digital-representation permission. No additional authoring-enabled installation flag is required. None replaces another. The selected Digital and original attribution rules MUST remain intact through custom creation, modification, nested operations and save/reopen. Required but unsupported participant-authoring support MUST make dependent actions unavailable without silently falling back to administration or conceptual-record operations.

This profile permits new behavior on independently authored types and checked additional fields of valid core specializations. It preserves the inherited Submission lifecycle and every other standard authoring rule. Standard action availability and explicitly declared extension actions remain separately discoverable.

4.19 Primary creation, descriptive workflows and composition

The `primaryCreation` object in an Annex B action is semantic metadata: `baseType` identifies its primary record's core root (or null when the action's existing selection rule decides it); `typeSelection` gives that selection rule; `propertiesMode` is `declared-content`, `additional-fields` or `table-defined`; `existingRecord` identifies an upsert that preserves type. This metadata affects the named primary record, never every dependent record created by the action.

> **[RBX-P4-061]** A primary constructor MUST accept the catalogued `type`, `properties`, `categories` and `tags`. The type is its declared base or a valid specialization, or follows the explicit shared-Neocortex/initiation-table rule. Review creation additionally requires its targetIds and creates Review Target in that same transaction; supplying targetIds for another root is invalid. `declared-content` preserves that action's existing permitted Properties contract. `additional-fields` permits only specialization-declared keys absent from the core root's fields; inherited fields remain controlled by their ordinary direct arguments and lifecycle. `table-defined` uses 3.24. Defaults are materialized root first, then specialization; supplied permitted Properties override defaults, never direct domain arguments. The operation MUST assemble and validate the complete primary record before commitment, so an additional required field can be supplied atomically and cannot be deferred until a later patch. Categories/Tags must resolve in the existing taxonomy. Defaults for new assignments are empty lists.

> **[RBX-P4-062]** An upsert MUST determine the existing record by its standard key before applying creation defaults. Omitted `type` retains that record's type; an explicitly different type fails rather than retyping. Omitted additional Properties and classifications retain existing values. Supplied Properties shallow-merge; supplied classification lists replace. Root defaults are not reapplied to existing records. A setter requesting removal/disable MUST reject nonempty creation-only Properties or classifications rather than silently ignore them; any supplied type must agree with the record being removed. The operation retains the original event rules, including an explicit Attention observation where that setter requires one. Editing only additional fields through 4.15 allocates no new event.

[RBX-P4-063] `rb.content.create` and `rb.connections.respond` MUST implement 3.24 and 3.3 respectively, without falling back to administrative primitives. Participant slot order and multiplicity follow the effective CRT root and specialization rules; generic constructor inputs MUST NOT add a blanket uniqueness restriction. Category and Tag assignments remain duplicate-free. The finite initiation table is the complete supported root set for `rb.content.create`; a custom type outside it uses the public extension authoring contract. A returned `{record,createdRecords}` contains the primary record first followed by required dependents; no implied event, invitation, grant, delivery or hidden author history is created. `rb.domain.transition` remains the sole shared state-changing path for the additional rows in 3.18.

[RBX-P4-064] Composition MUST retain the original selected role, Digital, transaction and extension generation through nested commands, policy effects and `rb.action.stage`. A handler may stage temporarily unresolved structural dependencies only between primitive mutation calls inside its isolated outer transaction; each value, type-local constraint, current authorization and action-specific transition is still checked when used. Public reads, queries and nested actions require a complete valid candidate before invocation, as Part 1 and Annex D specify. An incomplete record cannot serve as an authority grant. Final whole-world validation must resolve all required fields, references and domain invariants. A caught nested rejection still poisons the entire transaction. Neither a handler nor a hook may call a privileged inner operation to escape its declared scope, and external effects are never folded into a world-atomic transaction.

A deliberate state transition may remove the actor's own future participation; commit validation verifies that transition against its valid pre-transition authority rather than inventing a requirement that the actor remain a member afterward. Concurrent source or capability changes invalidate the bound invocation; subsequent work requires a new checked invocation. No fixed nesting depth or minimum step count is imposed: implementations declare resource limits and reject excess work before commitment.

5

PART 5

Part 5 — Conformance and implementation obligations

The implementation contract: declared capabilities, validation layers, preservation obligations and evidence of conformance.

5.1 Conformance is an explicit claim

5.2 Named capabilities

5.3 Validation layers

5.4 Required reporting fields

5.5 Producer obligations

5.6 Consumer obligations and error containment

5.7 Resource limits

5.8 Conformance vectors and verification procedure

5.9 Examples and implementation assessment

5.10 Extension and interoperability claims

5.11 Publication, corrections and evolution

5.12 Serialized declarations and discovery

5.13 Lifecycle coverage and executable evidence

5.14 Custom participant-authoring evidence

5.1 Conformance is an explicit claim

A conformance claim names an artifact or implementation, the specification edition, capabilities, supported resource limits and the evidence used to assess it. A ZIP parser, a static Viewer, an action engine and a full authoring Reader perform different work. They are not interchangeable claims.

> **[RBX-P5-001]** A claim MUST identify `RB-OXML-1.0`, edition `1.0`, the implementation or artifact being assessed, each claimed capability, and material restrictions. A result limited to parsing, schema validation or the supplied examples MUST state that scope. It MUST NOT be presented as complete Reader conformance.

A producer creates or edits packages. A consumer reads them. A processor may do both and may execute actions. Document conformance states that a package satisfies applicable data rules. Processor conformance states that behavior satisfies contracts for supported inputs and operations. The presence of an executable module in a package does not establish either execution success or host support.

> **[RBX-P5-002]** A processor MUST evaluate requirements by observable behavior and retained data, independently of implementation language. It MAY use a database, incremental parser, compiler or alternative action implementation provided the specified results, failure behavior and preservation obligations remain the same.

5.2 Named capabilities

The capability names below are normative for this edition. A capability can be listed as supported only when its complete applicable obligations are met. Fine-grained operation availability may also be reported, but does not substitute for a claim to an entire capability.

Capability	Meaning and obligations	Dependencies
`structure`	Identify package kind; validate supported OPC profile, native XML, portable values, record/type/taxonomy shape and reference integrity; preserve supported data.	None
`core-data`	Interpret all supplied core definitions and Part 3 invariants, including ownership, conceptual scope and domain field meanings. An equipped core world carries the full catalogue.	`structure`
`core-operations`	Implement the complete catalogue-defined participation and domain-operation family, including atomic validation, attribution, timestamps and failure behavior.	`core-data`
`authoring`	Implement the complete catalogue-defined administration, authoring, transaction, resource and extension family.	`core-operations`

Capability	Meaning and obligations	Dependencies
`components`	Read, write and validate Blueprint, Theme and Plugin package profiles; perform supported Blueprint dependency binding and independent instantiation as specified.	`structure`; `core-data` when instantiating core graphs
`ecmascript`	Execute the explicitly declared ECMAScript profile in Annex D, including module resolution, registration, dependencies and lifecycle behavior.	`structure`
`participant-authoring`	Implement the portable declarations, participation policies and checked custom mutations of `rbx.participant-authoring.1`, including save/reopen and independent host execution decisions.	`core-data`, `ecmascript`
`presentation`	Provide the Part 4 and Annex D Viewbox host, CSS/DOM profile, navigation, theme/context contracts and unavailable-dependency reporting.	`core-data`
`templates`	Implement the `rbx.templates.1` declarative layout and rendering profile, in addition to the DOM bridge.	`presentation`
`reader-host`	Implement the `rbx.reader-host.1` Reader host actions, scoped resource/file tickets and asynchronous job contracts.	`structure`
`eight-tool`	Provide the RedBook domain eight-tool interaction contract, context bindings, discovery, replay and failure semantics; public WebMCP mapping is a separate capability.	`structure`; additional capabilities according to the advertised actions
`jellybox-webmcp-1.0`	Expose the RedBook domain contract through the Jellybox WebMCP Framework 1.0 Standalone Core public schemas and the complete §4.9.1 mapping.	`eight-tool`; framework Standalone Core conformance; additional capabilities according to the advertised actions

> **[RBX-P5-003]** A processor MUST NOT advertise a capability until all its dependent capabilities are satisfied. It MUST NOT infer `core-operations` from a subset of available actions, or `ecmascript` from merely storing JavaScript text. An unavailable required capability MUST be reported before invoking dependent behavior.

These capabilities compose; “Reader” does not imply every capability. `jellybox-webmcp-1.0` is optional and requires evidence for both the full applicable framework contract and RedBook mapping. Eight matching names, a valid registration or ordinary website conformance alone establishes neither the domain contract nor this adapter capability. The following terms describe useful compositions, not new capability identifiers:

Processor description	Required claimed capabilities
Data reader	`structure`; add `core-data` when interpreting the supplied domain vocabulary.
Authoring engine	`structure`, `core-data`, `core-operations`, `authoring`; it may be headless.
Component processor	`components` and its applicable dependencies; executable activation is a separate claim.
Presentation reader	`presentation` and its dependencies; add `templates` for the template profile.
ECMAScript host	`ecmascript` and its dependencies; add `participant-authoring` for the checked custom authoring contract.

An implementation may combine any applicable compositions. A graphical interface, DOM, themes, file dialogs and the eight-tool transport are not universal requirements for reading or authoring RedBooks. The `authoring` capability covers portable world mutations; operations requiring presentation or an external host facility remain conditional on their declared capability. A headless engine may expose its authoring interface through another documented transport. A product claiming a particular optional profile must implement that profile's full applicable contract.

Document profiles are `structure` and `core-data`; operation, host and execution capabilities are processor claims. “Equipped world” is explanatory language for a world carrying the full required vocabulary and declared functionality, not another profile identifier. “Complete core vocabulary” means the `core-data` definition set for the identified edition. “Preservation-capable” describes the producer/consumer behavior in 5.5–5.6 and must identify its accepted layers; it is not an undocumented additional capability.

A component processor may inspect a plugin without activating it. It may validate a Blueprint envelope while reporting an unsupported graph type. It may preserve an unavailable extension while disabling only the actions and views that depend on it. It must not pretend those dependencies are satisfied.

5.3 Validation layers

> **[RBX-P5-004]** Validation results MUST distinguish at least these layers when they are applicable: container integrity; package-profile rules; XML/JSON grammar; portable-value and identity integrity; core-domain semantics; operation behavior; execution/presentation behavior; and preservation/interoperability. A single Boolean result MAY summarize a stated layer, but MUST NOT conceal omitted layers.

The layers are cumulative in dependency, not interchangeable in evidence:

1. Container checks establish readable members, permitted compression, nonambiguous names and integrity against declared sizes and checksums.
2. Package checks establish roots, content types, relationships, part maps, inventory and component manifests.
3. Grammar checks establish XML/JSON shape, required members, closed enums and supported lexical forms.
4. Logical checks establish identities, references, types, field rules, taxonomy, exact counters and their cross-record constraints.
5. Core-domain checks establish ownership and participant meaning, conceptual homes, unique placements where required, and coherent current states.
6. Behavioral checks establish preconditions, atomic effects, attribution, outputs and failures for named operations.
7. Execution checks establish module/registration lifecycles, hooks, context isolation and presentation behavior.
8. Interoperability checks establish equivalent outcomes across independent producers and readers, including save/reopen followed by further authoring.

> **[RBX-P5-005]** A processor MUST report an invalid supported input as a failure, not as unsupported content. It MUST report a valid input outside its declared capabilities or limits as unsupported, not silently rewrite it into a different supported input. A skipped or unassessed layer MUST remain explicitly unassessed.

A bounded checker can establish particular layers without implementing a complete independent Reader. A report identifies its actual algorithms and omitted semantics; successful envelope checking cannot prove arbitrary carried code behavior.

5.4 Required reporting fields

An authoritative machine-readable conformance or implementation-assessment report includes the following fields; additional explicitly described fields are permitted. A linked supporting case output may use a simpler test-specific shape when the authoritative aggregate identifies its subject, scope, environment, limits and unassessed behavior. Supporting case counts alone do not establish a broader claim.

Field	Required meaning
`specification`, `edition`	`RB-OXML-1.0` and semantic edition `1.0`. An optional `issue` identifies the distribution build only.
`subject`	Artifact path/hash or implementation name/version.

Field	Required meaning
`capabilities`	Claimed or assessed named capabilities.
`checks`	Named checks with scope, status and optional diagnostics.
`limits`	Material input/resource or execution restrictions.
`notAssessed`	Applicable layers or behavior not established by this report.
`environment`	Relevant runtimes and checker versions when tests are executed.
`generatedAt`	Time of the report, not an authored world event.

> **[RBX-P5-006]** Check status MUST distinguish `pass`, `fail`, `unsupported` and `not-assessed`, or an equally unambiguous representation. Report timestamps and diagnostic logs MUST remain outside authored worlds unless the user explicitly authors them as content.

> **[RBX-P5-007]** A report asserting preservation MUST identify the compared baseline and the comparison performed: package byte hash, part byte hash, parsed logical equality, or behavioral equivalence. “Unchanged” without such a scope is insufficient evidence.

5.5 Producer obligations

> **[RBX-P5-008]** A producer MUST validate the final supported output before committing it as a successful save. Invalid data, missing required resources, unresolved references or a counter behind retained events MUST prevent success. A failed save MUST preserve an existing destination artifact.

A safe save writes a complete candidate to a separate destination, validates it, and then atomically replaces the intended file where the host filesystem supports that operation. Other implementations may use equivalent transaction mechanisms. A partially written ZIP must not replace the previous valid document after an error.

> **[RBX-P5-009]** A producer MUST preserve exact resource and unsupported-part bytes that the operation did not deliberately change. It MUST retain existing package relationships and content-type declarations needed by those parts. Inventory hashes MUST be recalculated from the final retained bytes; an inventory is not a substitute for retaining the data.

> **[RBX-P5-010]** A producer modifying logical XML MAY change indentation, namespace prefix spelling, attribute order and map-entry order without changing meaning. It MUST preserve strings, typed values, exact-number lexemes, ordered lists, identities and explicit relationships. It MUST NOT use XML formatting normalization to alter authored whitespace.

> **[RBX-P5-011]** A producer MUST identify unsupported opaque dependencies before an identity-changing transformation. It MUST either retain a demonstrably valid dependency mapping, use an explicitly declared transformation contract, or reject the affected operation. Blind replacement of identity-like strings in source or binary bytes is nonconforming.

A producer need not reproduce another producer's ZIP compression bytes. Exact-archive reproducibility is an optional release property, separate from preserved part bytes and logical meaning.

5.6 Consumer obligations and error containment

> **[RBX-P5-012]** A consumer MUST identify package kind from the package's internal root and relationship contract. Filename extensions and media-type hints can assist discovery but MUST NOT override conflicting internal structure. A `.zip` suffix is permitted for the same package bytes.

> **[RBX-P5-013]** Parsing, inspection and schema validation MUST NOT execute carried source, fetch external entity declarations, activate plugins or treat authored text as instructions to the host. A Reader's explicit execution facility may run supported code only under the lifecycle and dependency rules in Part 4.

> **[RBX-P5-014]** A consumer MUST diagnose unsupported required content before dependent editing or execution. It MAY still offer inspection and exact preservation of opaque data. It MUST NOT discard that data merely to make the package appear conforming to a narrower capability.

Useful diagnostics name the failing layer, offending member or identity, violated contract and whether any mutation occurred. An error returned from an action must correspond to an unchanged transaction unless the contract explicitly concerns an external host effect whose result cannot be rolled back. Such host effects must be separate and disclosed; they cannot masquerade as atomic document mutation.

5.7 Resource limits

> **[RBX-P5-015]** Implementations MAY declare finite limits for member count, individual and total expanded bytes, nesting depth, string length, record count, execution time or memory. Limits MUST be enforced before resource exhaustion where practicable and MUST be reported as implementation limits. They MUST NOT silently truncate strings, lists, archive members or decimal values.

Every checker states its own relevant bounds in its report. These are checker limits, not additional grammar rules. Canonical positive decimal counters may have more digits than a host's usual integer-string conversion threshold. A processor that supports their text length must compare them exactly without disabling unrelated process-wide safeguards.

> **[RBX-P5-016]** A rejected archive member MUST NOT be written outside the caller's chosen extraction or output boundary. Traversal names, ambiguous normalized names, symlink entries and unsupported encryption/compression MUST be rejected under the package profile rather than interpreted opportunistically.

5.8 Conformance vectors and verification procedure

A conformance evidence set contains selected executable vectors and their expected outcomes. Positive vectors show supported structures or transformations. Negative vectors establish that a validator rejects a particular malformed or inconsistent input. An operation vector must include the resulting state and unchanged-state expectation on failure, not just an exception name.

> **[RBX-P5-017]** A checker reporting a vector as passed MUST evaluate its stated expected outcome against the actual generated or retained data. It MUST NOT count a skipped case, a caught unexpected exception or the existence of a test file as a passing test.

> **[RBX-P5-018]** Repeating a test run MUST use the same intended artifact bytes or explicitly report the changed hashes. Generated temporary test worlds MUST remain separate from the source artifacts being assessed. A conformance run MUST NOT silently rebuild those source artifacts as part of testing them.

Recommended behavioral families include:

- map-order-independent type equality and exact tagged-value preservation;
- invalid XML text, missing references, type-depth violations and slot cardinality/type failures;
- owner and conceptual-home invariants across both primitive collections;
- retained source credits after copying without the Digital or after deleting it;
- action rollback after validation or hook failure;
- isolated interactions with equal request IDs, stale bindings and replay expiry;
- empty and populated render hooks, extension setup failure and cleanup completeness;
- Blueprint import into empty and populated destinations, repeated import, save/reopen and continued authoring;
- duplicate/stale chronology rejection and decimal counters beyond native integer limits;
- preservation of unknown parts, additional package relationships and content types;
- original-resource bytes after save/reopen and independent-reader round trips.

These families are not a license to reduce the standard to its included tests. Normative obligations apply to every supported input within declared limits.

5.9 Examples and implementation assessment

An example establishes only the properties actually tested. A structurally accepted world may carry type definitions that differ from its claimed edition, incomplete operation handlers, or unsupported presentation profiles. Preserved bytes and identities alone do not establish conformance to the applicable contracts. An assessment reports these distinctions without silently changing the example or weakening the contract.

> **[RBX-P5-019]** An implementation assessment MUST distinguish package acceptance, logical validation against carried definitions, compatibility with the claimed edition's core definitions, and executable behavior. If an artifact's declared core definitions differ semantically from the claimed edition, the core-definition check MUST report the mismatch; successful byte preservation MUST NOT turn that mismatch into a conformance pass. An artifact without a conformance claim may be inspected and preserved without acquiring one. Control encodings outside the native grammar require an explicit archival-inspection scope and do not satisfy that grammar. An example demonstrates only its assessed behavior and does not establish an alternative normative encoding.

A source module or declared action name does not demonstrate its role enforcement, transaction behavior, output validation or independent-host execution. Such implementation facts belong in informative release assessments with named versions and artifact hashes. They do not change the normative contract.

5.10 Extension and interoperability claims

> **[RBX-P5-020]** A custom extension MUST identify the capabilities, resources and target contract it needs. An extension that changes the meaning of a core identity is not a conforming implementation of that identity. Extensions MAY introduce new identities, custom types, actions, themes and views under the dependency and preservation rules.

> **[RBX-P5-021]** A Reader claiming cross-implementation interoperability MUST state which independent producer/consumer combinations were tested and which data and execution capabilities were exercised. Reopening a file with the same library under another wrapper is not independent-reader evidence by itself.

A host need not support every external media format or programming language. It must preserve unsupported original resources and report which renderers or execution profiles are unavailable. The format's openness consists of documented representations and contracts, not a claim that any single application can execute every future extension.

5.11 Publication, corrections and evolution

This edition uses fixed identifiers for its vocabulary, namespaces and schema contracts. Referenced schema locations may be downloadable, but validity does not require an online lookup. The distribution includes local copies and an offline resolver.

> **[RBX-P5-022]** A published correction MUST identify the affected semantic edition and distribution build, and whether it changes normative meaning, machine-readable constraints, examples or informative explanation. `formatVersion:1.0` identifies the package/model family; `edition:1.0` identifies this semantic contract; separately versioned profile identifiers identify optional execution/transport contracts. Distribution build identifiers, timestamps and archive hashes identify delivered evidence only. Editorial build changes MUST NOT require document-claim rewrites or acquire semantic force. An implementation MUST NOT silently substitute incompatible definitions under an unchanged asserted contract; incompatible published changes require a distinguishable edition or profile.

This specification does not imply registration of `application/vnd.redbook.*`, endorsement by Microsoft or Ecma, or certification by ISO/IEC, W3C or IETF. External standards are cited for the specific representations they define; their text and governance remain independent.

> **[RBX-P5-023]** Distribution manifests and checksums MUST identify the actual files delivered. Publication tools MUST derive displayed page, requirement, catalogue and test counts from completed artifacts rather than predicted output. A missing artifact or failed check MUST remain visible in the report.

The specification may be presented in HTML, PDF or Markdown. These reading formats share the same normative meaning. The editable text and machine-readable contracts accompany the publication so that implementers can inspect the precise definitions independently of a rendered page.

5.12 Serialized declarations and discovery

> **[RBX-P5-024]** A native world declaring conformance MUST place one `conformance` map in its document `settings`. The map MUST conform to `conformance-claim.schema.json` and contain exactly `specification`, `edition`, `documentProfiles` and `requiredProcessorCapabilities`. For this edition the first two values are `RB-OXML-1.0` and `1.0`. Omission of the map means no explicit claim; it does not imply an older edition, full conformance or permission to rewrite the document. An inspector may report measured properties separately from the producer's declaration.

For example:

```
{
  "specification": "RB-OXML-1.0",
  "edition": "1.0",
  "documentProfiles": ["structure", "core-data"],
  "requiredProcessorCapabilities": [
    "components",
    "ecmascript",
    "rbx.ecmascript.1"
  ]
}
```

`documentProfiles` lists only document data obligations. `requiredProcessorCapabilities` lists exact normative capability identifiers and, where a capability has a separately versioned execution/rendering profile, its exact profile identifier. `ecmascript` is the capability; `rbx.ecmascript.1` is the concrete executable profile. They are not aliases: a consumer compares each declared requirement with its supported capabilities and profiles. Profile identifiers and dependencies are defined in their owning clauses. A component uses its envelope's `requires` list for the same exact requirement identifiers. The bare file suffix, installed source language and number of listed actions are not declarations.

> **[RBX-P5-025]** A producer claiming `core-data` MUST also declare `structure`. A processor MUST report its supported capabilities and profile identifiers through the Part 4 discovery contract, separately from a document's requested capabilities. Missing required support makes dependent behavior unavailable; it does not grant capabilities or authorize execution. Unknown optional data can still be preserved. An unknown required identifier MUST be named in the diagnostic, not silently treated as an alias for a familiar capability.

> **[RBX-P5-026]** A current-edition core-definition assessment MUST use Part 2 semantic projection for identity collisions and compare every required supplied definition. Label translation and explanatory annotation changes alone do not fail compatibility. New allocation fields, changed defaults, cardinalities, participant restrictions, traits or behavioral contracts are material differences. A report MUST retain each differing or missing identity and MUST NOT omit mismatches simply because no current instance happens to use that definition.

5.13 Lifecycle coverage and executable evidence

> **[RBX-P5-027]** Every claimed operation family MUST have a lifecycle coverage matrix identifying its named callable operations, roles, authority/state preconditions, resulting authored state, event/allocation behavior and failure outcomes. A represented state without a standard callable transition MUST be expressly data-only or assigned to an identified optional profile. Administration's ability to construct a state does not establish ordinary Digital participation support. The normative action catalogue and lifecycle matrix are companion contracts; inconsistent entries are publication defects.

A useful coverage case follows a complete lifecycle, such as authoring and editing a concept under a Digital role, withdrawing a Post, moving a recipient's mailbox entry, importing a definition-only library, or installing and independently removing two components with identical internal filenames. The evidence checks observable results and retained invariants, not merely that a handler returned.

[RBX-P5-028] A contract revision's evidence MUST include positive, negative and relevant boundary cases for its changed requirements. For transaction or authority changes, failure cases MUST compare the complete pre/post authored state and allocation counters. For installation and resource changes, cases MUST verify original byte identity, relative dependency resolution and save/reopen behavior. Concurrency cases MUST include overlapping requests rather than only sequential replay. Presentation cases MUST exercise render completion and disposal through the documented public bridge.

[RBX-P5-029] A claim of independent interoperability MUST name at least two independently implemented relevant producer/consumer or host contracts, their supported profiles and the same transferred artifact hashes. Running the same library in two wrappers is useful regression evidence but MUST NOT be reported as independent implementation agreement. No fixed count of tests substitutes for coverage of the claimed contract.

5.14 Custom participant-authoring evidence

[RBX-P5-030] A `participant-authoring` claim MUST assess the whole declared D.12 contract. Its acceptance evidence MUST execute carried implementation code through the public registration and mutation interfaces with a new root CET, a new root CRT and a valid specialization. Cases MUST include later edits, undeclared-key/type/scope denial, invalid-participant rollback, preserved creator attribution, policy denial, portable enable/disable intent and independently denied/allowed host execution. A bespoke test-only insertion operation or administrative impersonation does not establish this capability.

[RBX-P5-031] Save/reopen evidence for custom participant authoring MUST use an actual native XML/ZIP package carrying the types, declaration, implementation, policies and resulting records. Reopening MUST create fresh execution state and load the carried module/declaration again. The test MUST show that the new host can refuse execution while preserving meaning and can separately permit continuation under the same portable rules. A JSON-only state copy does not establish this requirement. A bounded acceptance host MUST identify which profile requirements it exercises and which remain unassessed; running one such host with an independent package validator is not full independent Reader interoperability.

[RBX-P5-032] A claim of functionality portability through Blueprints MUST include a successful capture → import → native save/reopen → public-action continuation → re-export case. The case MUST carry at least one custom record type, declaration, implementation and mapped participation policy; preserve original source bytes and credits; allocate imported events in destination order; and return new record/component mappings. It MUST also exercise an invalid binding or conflicting type without a partial import. A refusal-only path, inert byte copy or synthetic mutation function cannot establish this claim. A bounded fixture may establish these named outcomes while leaving full component-host and independent Reader interoperability unassessed.

A

ANNEX A

Annex A — Complete supplied type reference

The complete reference for the supplied Element Types, Relationship Types and their one-level specializations.

Normative. This annex presents the supplied definitions in the core type catalogue. This annex and Part 3 define the supplied vocabulary; Part 1 defines the interpretation of field and slot constraints.

The vocabulary contains 61 root Custom Element Types, 54 root Custom Relationship Types and 54 one-level specializations. The field and slot tables below include inherited declarations, so each entry can be implemented without guessing its effective contract.

An absent optional Property is distinct from an explicitly supplied null. A listed default is materialized when creating a record through the standard creation contract; parsing or reading does not invent missing Properties. A value in the domain of a declared field is required only when that field is present or declared required. Domain-specific operation preconditions may require more than the minimal structural record.

Every record also has the common identity, title, Properties, Categories and Tags specified in Part 1. The tables list only type-specific Properties. Additional Properties remain permitted and retain their authored meaning. A named slot is a list of existing Element identifiers. “Any Element” does not include a Relationship; a typed-reference Property may refer to either primitive.

Slot maximum “unbounded” imposes no format-level finite maximum. Ordered lists retain their stored order even when the domain assigns no priority to that order. Unique means that a participant cannot occur twice in that slot; the same Element can occupy different roles unless a domain rule prohibits it. A missing declared slot is treated as an empty participant list for cardinality checking. An Element has no participant slots.

Common classification and authorship

Every type below uses the same common contract: Categories and Tags classify the record itself, not its participants or referenced records. Optional attribution follows Part 1. A new contribution in a Digital context receives that contributing Digital’s local attribution and portable source credit; descriptive roles do not silently replace authorship. Editing preserves the existing attribution unless an explicit attribution operation changes it. Personal/shared conceptual-edit authority is specified in Part 3, clause 3.17. The individual entries add their domain-specific constraints without repeating this common paragraph.

A.1 Index

Type	Primitive	Level	Domain
Digital (`digital`)	element	type	3.3
Profile (`profile`)	element	type	3.3
Location (`location`)	element	type	3.3
Group (`group`)	element	type	3.3

Type	Primitive	Level	Domain
Space (`space`)	element	type	3.3
Page (`page`)	element	type	3.3
Circle (`circle`)	element	type	3.3
Wall (`wall`)	element	type	3.4
Post (`post`)	element	type	3.4
Conversation (`conversation`)	element	type	3.4
Message (`message`)	element	type	3.4
Comment (`comment`)	element	type	3.6
Annotation (`annotation`)	element	type	3.6
Mail Message (`mail-message`)	element	type	3.5
Mailbox (`mailbox`)	element	type	3.5
Mail Folder (`mail-folder`)	element	type	3.5
Distribution List (`distribution-list`)	element	type	3.5
Correspondence Template (`correspondence-template`)	element	type	3.5
Signature (`signature`)	element	type	3.5
Mail Rule (`mail-rule`)	element	type	3.5
Newsletter (`newsletter`)	element	type	3.5
Newsletter Issue (`newsletter-issue`)	element	type	3.5
Submission (`submission`)	element	type	3.5
Media (`media`)	element	type	3.6

Type	Primitive	Level	Domain
Representation (`representation`)	element	type	3.6
Work (`work`)	element	type	3.6
Edition (`edition`)	element	type	3.6
Suggestion (`suggestion`)	element	type	3.6
Collection (`collection`)	element	type	3.7
Collection Section (`collection-section`)	element	type	3.7
Feed (`feed`)	element	type	3.7
Saved Query (`saved-query`)	element	type	3.7
Notification (`notification`)	element	type	3.8
Reminder (`reminder`)	element	type	3.8
Presence (`presence`)	element	type	3.8
Attention Preference (`attention-preference`)	element	type	3.8
Poll (`poll`)	element	type	3.9
Questionnaire (`questionnaire`)	element	type	3.9
Proposal (`proposal`)	element	type	3.9
Decision Round (`decision-round`)	element	type	3.9
Decision (`decision`)	element	type	3.9
Review (`review`)	element	type	3.9
Finding (`finding`)	element	type	3.9
Request (`request`)	element	type	3.10

Type	Primitive	Level	Domain
Commitment (`commitment`)	element	type	3.10
Progress (`progress`)	element	type	3.10
Case (`case`)	element	type	3.10
Resource (`resource`)	element	type	3.10
Reservation (`reservation`)	element	type	3.10
Event (`event`)	element	type	3.11
Gathering (`gathering`)	element	type	3.11
Agenda (`agenda`)	element	type	3.11
Agenda Item (`agenda-item`)	element	type	3.11
Stream (`stream`)	element	type	3.12
Stream Segment (`stream-segment`)	element	type	3.12
Moderation Report (`moderation-report`)	element	type	3.13
Participation Rule (`participation-rule`)	element	type	3.13
Soma Neocortex (`neocortex`)	element	type	3.14
Neo Soma (`neo-soma`)	element	type	3.14
Neo Object (`neo-object`)	element	type	3.14
Viewbox (`viewbox`)	element	type	4
Profile Of (`profile-of`)	relationship	type	3.3
Membership (`membership`)	relationship	type	3.3
Participation (`participation`)	relationship	type	3.3

Type	Primitive	Level	Domain
Invitation (`invitation`)	relationship	type	3.3
Follow (`follow`)	relationship	type	3.3
Connection (`connection`)	relationship	type	3.3
Authorship (`authorship`)	relationship	type	3.3
Wall Placement (`wall-placement`)	relationship	type	3.4
Contribution (`contribution`)	relationship	type	3.4
Reply (`reply`)	relationship	type	3.4
Accepted Answer (`accepted-answer`)	relationship	type	3.4
Mention (`mention`)	relationship	type	3.4
Reaction (`reaction`)	relationship	type	3.6
Annotation Target (`annotation-target`)	relationship	type	3.6
Circulation (`circulation`)	relationship	type	3.6
Attachment (`attachment`)	relationship	type	3.6
Representation Of (`representation-of`)	relationship	type	3.6
Delivery (`delivery`)	relationship	type	3.5
Mailbox Entry (`mailbox-entry`)	relationship	type	3.5
Mailbox Owner (`mailbox-owner`)	relationship	type	3.5
Mail Handling (`mail-handling`)	relationship	type	3.5
Recipient List Member (`recipient-list-member`)	relationship	type	3.5

Type	Primitive	Level	Domain
Newsletter Entry (`newsletter-entry`)	relationship	type	3.5
Subscription (`subscription`)	relationship	type	3.5
Submission Target (`submission-target`)	relationship	type	3.5
Collection Entry (`collection-entry`)	relationship	type	3.7
Bookmark (`bookmark`)	relationship	type	3.7
Attention (`attention`)	relationship	type	3.8
Notice Target (`notice-target`)	relationship	type	3.8
Presence Of (`presence-of`)	relationship	type	3.8
Ballot (`ballot`)	relationship	type	3.9
Questionnaire Response (`questionnaire-response`)	relationship	type	3.9
Decision Position (`decision-position`)	relationship	type	3.9
Review Target (`review-target`)	relationship	type	3.9
Finding In (`finding-in`)	relationship	type	3.9
Request Address (`request-address`)	relationship	type	3.10
Commitment For (`commitment-for`)	relationship	type	3.10
Progress On (`progress-on`)	relationship	type	3.10
Case Material (`case-material`)	relationship	type	3.10
Reservation For (`reservation-for`)	relationship	type	3.10

Type	Primitive	Level	Domain
Event Response (`event-response`)	relationship	type	3.11
Attendance (`attendance`)	relationship	type	3.11
Gathering Role (`gathering-role`)	relationship	type	3.11
Shared Focus (`shared-focus`)	relationship	type	3.11
Stream Participation (`stream-participation`)	relationship	type	3.12
Stream Entry (`stream-entry`)	relationship	type	3.12
Report Target (`report-target`)	relationship	type	3.13
Rule Target (`rule-target`)	relationship	type	3.13
Neocortex Owner (`neocortex-owner`)	relationship	type	3.14
Soma Home (`soma-home`)	relationship	type	3.14
Neo Membership (`neo-membership`)	relationship	type	3.14
Neo Relationship (`neo-relationship`)	relationship	type	3.14
Represents (`represents`)	relationship	type	3.14
Reference (`reference`)	relationship	type	3.6
Digital Person (`digital.person`)	element	specialization	3.3
Digital Animal (`digital.animal`)	element	specialization	3.3
Digital Thing (`digital.thing`)	element	specialization	3.3

Type	Primitive	Level	Domain
Digital Collective (`digital.collective`)	element	specialization	3.3
Digital Concept (`digital.concept`)	element	specialization	3.3
Article (`post.article`)	element	specialization	3.4
Journal Entry (`post.journal`)	element	specialization	3.4
Status (`post.status`)	element	specialization	3.4
Story (`post.story`)	element	specialization	3.4
Observation (`post.observation`)	element	specialization	3.4
Direct Conversation (`conversation.direct`)	element	specialization	3.4
Group Conversation (`conversation.group`)	element	specialization	3.4
Channel (`conversation.channel`)	element	specialization	3.4
Forum (`conversation.forum`)	element	specialization	3.4
Topic (`conversation.topic`)	element	specialization	3.4
Thread (`conversation.thread`)	element	specialization	3.4
Correspondence Thread (`conversation.mail-thread`)	element	specialization	3.4
Live Room (`conversation.live-room`)	element	specialization	3.4
Question (`message.question`)	element	specialization	3.4
Answer (`message.answer`)	element	specialization	3.4

Type	Primitive	Level	Domain
Summary (`message.summary`)	element	specialization	3.4
Image (`media.image`)	element	specialization	3.6
Audio (`media.audio`)	element	specialization	3.6
Video (`media.video`)	element	specialization	3.6
File (`media.file`)	element	specialization	3.6
Source Repository (`media.source`)	element	specialization	3.6
Recording (`media.recording`)	element	specialization	3.6
Document (`work.document`)	element	specialization	3.6
Notebook (`work.notebook`)	element	specialization	3.6
Canvas (`work.canvas`)	element	specialization	3.6
Dataset (`work.dataset`)	element	specialization	3.6
Knowledge Page (`work.knowledge-page`)	element	specialization	3.6
Artifact (`work.artifact`)	element	specialization	3.6
Deliverable (`work.deliverable`)	element	specialization	3.6
Album (`collection.album`)	element	specialization	3.7
Board (`collection.board`)	element	specialization	3.7
Portfolio (`collection.portfolio`)	element	specialization	3.7
Playlist (`collection.playlist`)	element	specialization	3.7
Reading List (`collection.reading-list`)	element	specialization	3.7

Type	Primitive	Level	Domain
Friendship (`connection.friendship`)	relationship	specialization	3.3
Affiliation (`connection.affiliation`)	relationship	specialization	3.3
Mentorship (`connection.mentorship`)	relationship	specialization	3.3
Introduction (`connection.introduction`)	relationship	specialization	3.3
Rating (`reaction.rating`)	relationship	specialization	3.6
Acknowledgment (`reaction.acknowledgment`)	relationship	specialization	3.6
Recommendation (`reaction.recommendation`)	relationship	specialization	3.6
Endorsement (`reaction.endorsement`)	relationship	specialization	3.6
Repost (`circulation.repost`)	relationship	specialization	3.6
Quotation (`circulation.quote`)	relationship	specialization	3.6
Forward (`circulation.forward`)	relationship	specialization	3.6
Cross-post (`circulation.crosspost`)	relationship	specialization	3.6
Derivation (`circulation.derivation`)	relationship	specialization	3.6
World Explorer (`viewbox.explorer`)	element	specialization	4
Offer (`request.offer`)	element	specialization	3.10

A.2 Digital

Identifier: `urn:redbook:type:digital` **Key:** `digital` · **Primitive:** element · **Level:** type · **Area:** identity

An actor to whom humans or external AI attribute authored activity. Not an executing AI or a login account.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.3 Profile

Identifier: `urn:redbook:type:profile` **Key:** `profile` · **Primitive:** element · **Level:** type · **Area:** identity

An authored presentation of an identity; not an authentication record.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`biography`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `profile`	Authored biographical text, not authenticated identity evidence.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `profile`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `profile`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.4 Location

Identifier: `urn:redbook:type:location` **Key:** `location` · **Primitive:** element · **Level:** type · **Area:** organization

A physical, imagined or conceptual place in the world.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `location`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`coordinates`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `location`	An authored spatial-description map; coordinate system and units are explicit members when numeric coordinates are interpreted (3.2).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.5 Group

Identifier: `urn:redbook:type:group` **Key:** `group` · **Primitive:** element · **Level:** type · **Area:** organization

An organized collective with membership and purpose. Membership and collective acting identity are distinct.

Domain rules: Part 3, clause 3.3. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`purpose`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `group`	Authored purpose of this collective or shared context; it is not an assigned job or automatically executed instruction.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `group`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.6 Space

Identifier: `urn:redbook:type:space` **Key:** `space` · **Primitive:** element · **Level:** type · **Area:** organization

A shared context for participation and material.

Domain rules: Part 3, clause 3.3. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`purpose`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `space`	Authored purpose of this collective or shared context; it is not an assigned job or automatically executed instruction.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `space`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.7 Page

Identifier: `urn:redbook:type:page` **Key:** `page` · **Primitive:** element · **Level:** type · **Area:** organization

A continuing authored presentation and contribution context for a subject.

Domain rules: Part 3, clause 3.3. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `page`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `page`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.8 Circle

Identifier: `urn:redbook:type:circle` **Key:** `circle` · **Primitive:** element · **Level:** type · **Area:** organization

A named selection of Digitals for organization; not inherently a group or account.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circle`	Descriptive authored text; it does not execute, declare a new type or update the described subject.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.9 Wall

Identifier: `urn:redbook:type:wall` **Key:** `wall` · **Primitive:** element · **Level:** type · **Area:** publication

A hosted ordered sequence of entries, with subsequent contributions to existing entries. Can be specialized as a log or timeline.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"chronological"` DECLARED BY: `wall`	One of `"chronological"`, `"manual"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.
`contributions`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `wall`	One of `"open"`, `"members"`, `"reviewed"`, `"closed"`. Controls new Wall placements: open, active-members-only, review-before-publication, or closed to non-owner placements (3.4).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.10 Post

Identifier: `urn:redbook:type:post` **Key:** `post` · **Primitive:** element · **Level:** type · **Area:** publication

An authored contribution capable of publication and placement in one or more contexts.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.11 Conversation

Identifier: `urn:redbook:type:conversation` **Key:** `conversation` · **Primitive:** element · **Level:** type · **Area:** discussion

A continuing exchange with participants, contributions and its own shared Neocortex.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.12 Message

Identifier: `urn:redbook:type:message` **Key:** `message` · **Primitive:** element · **Level:** type · **Area:** discussion

An attributed contribution to a conversation. Questions and answers are specializations, not hard-coded alternatives.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `message`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `message`	One of `"draft"`, `"sent"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.13 Comment

Identifier: `urn:redbook:type:comment` **Key:** `comment` · **Primitive:** element · **Level:** type · **Area:** expression

An attributed response associated with another authored item.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `comment`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `comment`	One of `"open"`, `"resolved"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.14 Annotation

Identifier: `urn:redbook:type:annotation` **Key:** `annotation` · **Primitive:** element · **Level:** type · **Area:** expression

Authored commentary on a whole item, passage, region, time range or conceptual item.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `annotation`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`selector`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `annotation`	A whole-item, Unicode-text, rectangle or time-span selector map, with the addressing and anchoring rules in 3.6.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `annotation`	One of `"open"`, `"resolved"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.15 Mail Message

Identifier: `urn:redbook:type:mail-message` **Key:** `mail-message` · **Primitive:** element · **Level:** type · **Area:** correspondence

Internal addressed correspondence with separate delivery relationships and per-recipient organization.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`subject`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-message`	Authored correspondence subject text, independent of the record title.
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `mail-message`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `mail-message`	One of `"draft"`, `"sent"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`threadId`	KIND: reference REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-message`	Canonical correspondence Conversation for an operational sent message. The exact thread reuse, public audience and Bcc rules are in 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.16 Mailbox

Identifier: `urn:redbook:type:mailbox` **Key:** `mailbox` · **Primitive:** element · **Level:** type · **Area:** correspondence

A correspondence collection with recipient-specific handling state.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mailbox`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`collective`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `mailbox`	False denotes a personal mailbox; true denotes collective responsibility. Mailbox Owner relationships identify the responsible Digitals.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.17 Mail Folder

Identifier: `urn:redbook:type:mail-folder` **Key:** `mail-folder` · **Primitive:** element · **Level:** type · **Area:** correspondence

An authored correspondence organization folder; deleting it does not delete its messages.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-folder`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`mailboxId`	KIND: reference REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-folder`	The Mailbox whose organization contains this folder. Standard folder creation and movement require this ownership reference (3.18).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.18 Distribution List

Identifier: `urn:redbook:type:distribution-list` **Key:** `distribution-list` · **Primitive:** element · **Level:** type · **Area:** correspondence

A named selection of recipients; a send captures its chosen recipients.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `distribution-list`	Descriptive authored text; it does not execute, declare a new type or update the described subject.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.19 Correspondence Template

Identifier: `urn:redbook:type:correspondence-template` **Key:** `correspondence-template` · **Primitive:** element · **Level:** type · **Area:** correspondence

Reusable correspondence text with declared named substitutions.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`subject`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `correspondence-template`	Authored correspondence subject text, independent of the record title.
`text`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `correspondence-template`	Authored template text. Named substitutions follow the exact, non-executing replacement rules in 3.5.
`variables`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `correspondence-template`	Unique template variable names in authored display order (3.5); values are supplied explicitly at rendering.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.20 Signature

Identifier: `urn:redbook:type:signature` **Key:** `signature` · **Primitive:** element · **Level:** type · **Area:** correspondence

An authored sign-off, not a cryptographic signature.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `signature`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.21 Mail Rule

Identifier: `urn:redbook:type:mail-rule` **Key:** `mail-rule` · **Primitive:** element · **Level:** type · **Area:** correspondence

A declarative filter/action recipe for explicitly processing selected correspondence.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`conditions`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `mail-rule`	A declarative condition map evaluated only through the declared rule or decision procedure (3.9 and 3.13).
`effects`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `mail-rule`	An ordered list of declarative effect descriptors; neither presence nor enabled state performs these effects (3.13).
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `mail-rule`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.
`priority`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `mail-rule`	Safe integer priority; applicable rules sort ascending, then by exact rule identity. See 3.21.
`actions`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `mail-rule`	Unique exact action names to which the rule applies; an empty list means every operation with this actual policy context. See 3.21.
`mode`	KIND: string REQUIRED: no CREATION DEFAULT: `"require"` DECLARED BY: `mail-rule`	One of `"require"`, `"deny"`, `"effects"`. require denies when its condition is false; deny denies when true; effects selects its explicit effects when true. No rule elevates authority. See 3.21.
`mailboxId`	KIND: reference REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-rule`	Mailbox to which this Mail Rule applies; required when evaluating the rule through the standard operation (3.21).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.22 Newsletter

Identifier: `urn:redbook:type:newsletter` **Key:** `newsletter` · **Primitive:** element · **Level:** type · **Area:** distribution

A publication series with subscribers, issues and contribution arrangements.

Domain rules: Part 3, clause 3.5. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `newsletter`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `newsletter`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`nextIssueNumber`	KIND: string REQUIRED: no CREATION DEFAULT: `"1"` DECLARED BY: `newsletter`	Canonical positive decimal allocation frontier for new issue numbers. Creation starts at 1; deletion never decreases it. Missing-state initialization, import and safe issue-number range are specified in 3.20.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.23 Newsletter Issue

Identifier: `urn:redbook:type:newsletter-issue` **Key:** `newsletter-issue` · **Primitive:** element · **Level:** type · **Area:** distribution

One authored issue in a newsletter series. An issue is content, not an automatic copy of every edit.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `newsletter-issue`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `newsletter-issue`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`issueNumber`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `newsletter-issue`	A finite, safe nonnegative integer labeling a Newsletter Issue. Zero is unnumbered; positive numbers are unique within the Newsletter (3.5).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.24 Submission

Identifier: `urn:redbook:type:submission` **Key:** `submission` · **Primitive:** element · **Level:** type · **Area:** distribution

A contribution offered for review and placement, independent of the source item.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `submission`	Additional authored explanatory text; it is not an automatic change log.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"pending"` DECLARED BY: `submission`	One of `"pending"`, `"accepted"`, `"declined"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.25 Media

Identifier: `urn:redbook:type:media` **Key:** `media` · **Primitive:** element · **Level:** type · **Area:** content

A resource with original file bytes, descriptive metadata and package-relative asset location.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.26 Representation

Identifier: `urn:redbook:type:representation` **Key:** `representation` · **Primitive:** element · **Level:** type · **Area:** content

An authored transcript, translation, caption or other representation of existing material.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `representation`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`language`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `representation`	A BCP 47 language tag for the representation when supplied; source language is not silently inferred from this value.
`kind`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `representation`	The domain-specific kind of statement or representation; listed choices, if any, are exhaustive for this root contract.
`cues`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `representation`	Ordered timed or textual cue maps carrying explicit start/end coordinates and text; cue coordinates use the source medium, not the RedBook clock (3.6).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.27 Work

Identifier: `urn:redbook:type:work` **Key:** `work` · **Primitive:** element · **Level:** type · **Area:** content

An intentionally composed authored whole. Does not absorb Neo Somas into a separate compulsory work hierarchy.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.28 Edition

Identifier: `urn:redbook:type:edition` **Key:** `edition` · **Primitive:** element · **Level:** type · **Area:** content

An explicitly authored independent capture of selected content, never an automatic edit-history entry.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `edition`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `edition`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.29 Suggestion

Identifier: `urn:redbook:type:suggestion` **Key:** `suggestion` · **Primitive:** element · **Level:** type · **Area:** content

A proposed change with its own authored wording; not applied merely by being present.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`proposedProperties`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `suggestion`	A proposed shallow Property patch. Acceptance does not apply it until an explicit valid editing transaction is performed.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `suggestion`	Additional authored explanatory text; it is not an automatic change log.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"pending"` DECLARED BY: `suggestion`	One of `"pending"`, `"accepted"`, `"declined"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.30 Collection

Identifier: `urn:redbook:type:collection` **Key:** `collection` · **Primitive:** element · **Level:** type · **Area:** discovery

An authored selection and arrangement of references to other items.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.31 Collection Section

Identifier: `urn:redbook:type:collection-section` **Key:** `collection-section` · **Primitive:** element · **Level:** type · **Area:** discovery

A named section in an authored collection.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection-section`	Descriptive authored text; it does not execute, declare a new type or update the described subject.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.32 Feed

Identifier: `urn:redbook:type:feed` **Key:** `feed` · **Primitive:** element · **Level:** type · **Area:** discovery

A stored selection and ordering recipe; results do not duplicate the selected content.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`query`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `feed`	A stored query descriptor using the selection grammar in Part 4. Evaluating it does not duplicate matched records or record attention.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"chronological"` DECLARED BY: `feed`	One of `"chronological"`, `"reverse-chronological"`, `"title"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.33 Saved Query

Identifier: `urn:redbook:type:saved-query` **Key:** `saved-query` · **Primitive:** element · **Level:** type · **Area:** discovery

A reusable content selection independent of any particular display.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`query`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `saved-query`	A stored query descriptor using the selection grammar in Part 4. Evaluating it does not duplicate matched records or record attention.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.34 Notification

Identifier: `urn:redbook:type:notification` **Key:** `notification` · **Primitive:** element · **Level:** type · **Area:** attention

An addressed authored notice; reading a list does not acknowledge it or wake AI.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `notification`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`priority`	KIND: string REQUIRED: no CREATION DEFAULT: `"normal"` DECLARED BY: `notification`	One of `"normal"`, `"low"`, `"high"`. Authored relative notice priority; it does not bypass host access or start delivery.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"available"` DECLARED BY: `notification`	One of `"available"`, `"dismissed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.35 Reminder

Identifier: `urn:redbook:type:reminder` **Key:** `reminder` · **Primitive:** element · **Level:** type · **Area:** attention

An authored intention to revisit material at a declared time. It does not create an execution schedule.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reminder`	Additional authored explanatory text; it is not an automatic change log.
`clock`	KIND: string REQUIRED: no CREATION DEFAULT: `"world"` DECLARED BY: `reminder`	One of `"world"`, `"recorded"`. The coordinate basis: recorded uses canonical UTC recording instants; world uses the declared world-time coordinate (3.15).
`at`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reminder`	The intended revisit coordinate in the selected clock. It creates no timer.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"pending"` DECLARED BY: `reminder`	One of `"pending"`, `"cancelled"`, `"completed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.36 Presence

Identifier: `urn:redbook:type:presence` **Key:** `presence` · **Primitive:** element · **Level:** type · **Area:** attention

An explicit statement of availability or composing, not an inference that a model is running.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`status`	KIND: string REQUIRED: no CREATION DEFAULT: `"available"` DECLARED BY: `presence`	The current authored presence or attendance status; it is not inferred from network activity or an RSVP.
`until`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `presence`	An optional stated validity endpoint for this Presence in its declared context; expiry may affect presentation but does not mutate the statement.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `presence`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.37 Attention Preference

Identifier: `urn:redbook:type:attention-preference` **Key:** `attention-preference` · **Primitive:** element · **Level:** type · **Area:** attention

Authored preferences for notices, quiet periods, grouping and display.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`quiet`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `attention-preference`	Explicit quiet-period preferences using the declarative map conventions in 3.8.
`delivery`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `attention-preference`	Authored preferences for presenting notices; transport remains a host capability.
`filters`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `attention-preference`	Declared predicates restricting a subscription or preference (3.8). Unknown predicate semantics are not guessed.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.38 Poll

Identifier: `urn:redbook:type:poll` **Key:** `poll` · **Primitive:** element · **Level:** type · **Area:** coordination

A structured request for choices with a declared tally method.

Domain rules: Part 3, clause 3.9. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`options`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `poll`	Ordered Poll option maps, each with a unique nonempty id other than exactly $rb, and an authored label (3.9; domain-response.schema.json). Option IDs are local values, not Element references.
`method`	KIND: string REQUIRED: no CREATION DEFAULT: `"single"` DECLARED BY: `poll`	One of `"single"`, `"multiple"`, `"ranked"`, `"rating"`. The ballot method: single choice, multiple choice, ranked Borda, or summed rating (3.9).
`maximumSelections`	KIND: number REQUIRED: no CREATION DEFAULT: `1` DECLARED BY: `poll`	For multiple-choice voting, the positive safe integer maximum number of distinct choices; it cannot exceed the number of options.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `poll`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`results`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `poll`	An explicitly captured tally result. It is not updated by later edits unless an explicit operation replaces it.
`responsePolicy`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `poll`	Standard response policy: eligibility is members (default) or open. This does not grant Membership or conceptual-write authority; see 3.21.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.39 Questionnaire

Identifier: `urn:redbook:type:questionnaire` **Key:** `questionnaire` · **Primitive:** element · **Level:** type · **Area:** coordination

A structured set of questions and response arrangements.

Domain rules: Part 3, clause 3.9. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`questions`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `questionnaire`	Ordered question descriptors with unique nonempty id other than exactly $rb, declared value type and optional required flag (3.9; domain-response.schema.json).
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `questionnaire`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`responsePolicy`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `questionnaire`	Standard response policy: eligibility is members (default) or open, and reviseSubmitted is false by default. These are distinct from enrollment; see 3.21.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.40 Proposal

Identifier: `urn:redbook:type:proposal` **Key:** `proposal` · **Primitive:** element · **Level:** type · **Area:** coordination

Authored wording put forward for deliberation.

Domain rules: Part 3, clause 3.9. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `proposal`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `proposal`	One of `"open"`, `"adopted"`, `"not-adopted"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.41 Decision Round

Identifier: `urn:redbook:type:decision-round` **Key:** `decision-round` · **Primitive:** element · **Level:** type · **Area:** coordination

A participation context collecting positions on a proposal.

Domain rules: Part 3, clause 3.9. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`conditions`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `decision-round`	A declarative condition map evaluated only through the declared rule or decision procedure (3.9 and 3.13).
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `decision-round`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`outcome`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `decision-round`	The explicitly authored outcome of the relevant decision or gathering; no automatic inference from participation is implied.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.42 Decision

Identifier: `urn:redbook:type:decision` **Key:** `decision` · **Primitive:** element · **Level:** type · **Area:** coordination

An expressly recorded choice, rationale and unresolved considerations.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `decision`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`procedure`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `decision`	Identifier or descriptive name of the declared decision procedure; a consumer does not guess voting rules from this string.
`dissent`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `decision`	Ordered authored dissent statements or typed references; inclusion records disagreement without invalidating the decision by implication.
`subject`	KIND: reference REQUIRED: no CREATION DEFAULT: none DECLARED BY: `decision`	Explicit Proposal or Decision Round whose outcome this Decision records. Required for Decisions created by the standard transition action (3.18).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.43 Review

Identifier: `urn:redbook:type:review` **Key:** `review` · **Primitive:** element · **Level:** type · **Area:** coordination

A context for findings about a chosen target and criteria.

Domain rules: Part 3, clause 3.9. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`criteria`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `review`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `review`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`release`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `review`	One of `"open"`, `"on-close"`. Whether Findings are presented when authored (open) or withheld by the participation projection until review closure (on-close). This is not encryption.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.44 Finding

Identifier: `urn:redbook:type:finding` **Key:** `finding` · **Primitive:** element · **Level:** type · **Area:** coordination

An attributed review finding with disposition and explanatory material.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `finding`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`disposition`	KIND: string REQUIRED: no CREATION DEFAULT: `"comment"` DECLARED BY: `finding`	One of `"comment"`, `"approve"`, `"changes-requested"`, `"decline"`. The Finding author's stated assessment; it does not automatically close the Review or modify the reviewed item.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.45 Request

Identifier: `urn:redbook:type:request` **Key:** `request` · **Primitive:** element · **Level:** type · **Area:** coordination

An authored request or offer, separate from acceptance and responsibility.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `request`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `request`	One of `"open"`, `"accepted"`, `"declined"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.46 Commitment

Identifier: `urn:redbook:type:commitment` **Key:** `commitment` · **Primitive:** element · **Level:** type · **Area:** coordination

An accepted responsibility, distinct from a request and from later verification.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `commitment`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"accepted"` DECLARED BY: `commitment`	One of `"accepted"`, `"in-progress"`, `"reported-complete"`, `"verified"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.47 Progress

Identifier: `urn:redbook:type:progress` **Key:** `progress` · **Primitive:** element · **Level:** type · **Area:** coordination

An attributed progress statement or verification about a commitment.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `progress`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`kind`	KIND: string REQUIRED: no CREATION DEFAULT: `"update"` DECLARED BY: `progress`	One of `"update"`, `"report-complete"`, `"verify"`, `"reopen"`, `"withdraw"`. The domain-specific kind of statement or representation; listed choices, if any, are exhaustive for this root contract.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.48 Case

Identifier: `urn:redbook:type:case` **Key:** `case` · **Primitive:** element · **Level:** type · **Area:** coordination

A continuing matter connecting discussions, evidence and responsibilities.

Domain rules: Part 3, clause 3.10. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `case`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `case`	One of `"open"`, `"resolved"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.49 Resource

Identifier: `urn:redbook:type:resource` **Key:** `resource` · **Primitive:** element · **Level:** type · **Area:** coordination

An authored reservable thing or capacity, distinct from a media asset.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `resource`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`capacity`	KIND: number REQUIRED: no CREATION DEFAULT: `1` DECLARED BY: `resource`	A finite nonnegative available quantity in the Resource unit; reservation confirmation compares concurrent quantities against this value (3.10).
`unit`	KIND: string REQUIRED: no CREATION DEFAULT: `"unit"` DECLARED BY: `resource`	Authored name of the quantity unit. Different unit strings are not silently converted.
`terms`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `resource`	Additional portable terms governing use; undeclared terms are descriptive and do not execute external bookings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.50 Reservation

Identifier: `urn:redbook:type:reservation` **Key:** `reservation` · **Primitive:** element · **Level:** type · **Area:** coordination

A requested or confirmed use of a resource. No real-world booking is inferred.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`start`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reservation`	Inclusive start coordinate in the declared clock. The interval rules are in 3.10, 3.11 and 3.15.
`end`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reservation`	Exclusive end coordinate in the declared clock; an operational interval has end greater than start.
`clock`	KIND: string REQUIRED: no CREATION DEFAULT: `"world"` DECLARED BY: `reservation`	One of `"world"`, `"recorded"`. The coordinate basis: recorded uses canonical UTC recording instants; world uses the declared world-time coordinate (3.15).
`quantity`	KIND: number REQUIRED: no CREATION DEFAULT: `1` DECLARED BY: `reservation`	A finite positive quantity in the associated Resource unit when requesting or confirming a reservation.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"requested"` DECLARED BY: `reservation`	One of `"requested"`, `"confirmed"`, `"lent"`, `"returned"`, `"cancelled"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.51 Event

Identifier: `urn:redbook:type:event` **Key:** `event` · **Primitive:** element · **Level:** type · **Area:** gatherings

An authored scheduled occurrence in the world, not a job that executes itself.

Domain rules: Part 3, clause 3.11. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `event`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`start`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `event`	Inclusive start coordinate in the declared clock. The interval rules are in 3.10, 3.11 and 3.15.
`end`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `event`	Exclusive end coordinate in the declared clock. An operational Event permits end equal to start for an instantaneous occurrence; end must not precede start (3.11).
`clock`	KIND: string REQUIRED: no CREATION DEFAULT: `"world"` DECLARED BY: `event`	One of `"world"`, `"recorded"`. The coordinate basis: recorded uses canonical UTC recording instants; world uses the declared world-time coordinate (3.15).
`recurrence`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `event`	An optional recurrence descriptor. The fixed-interval and Gregorian RRULE profiles are specified in 3.11; an empty map means no repetition.
`exceptions`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `event`	Explicit occurrence exceptions keyed by original occurrence start; they alter the recurrence projection, not retained historical attendance (3.11).

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"planned"` DECLARED BY: `event`	One of `"planned"`, `"cancelled"`, `"complete"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.52 Gathering

Identifier: `urn:redbook:type:gathering` **Key:** `gathering` · **Primitive:** element · **Level:** type · **Area:** gatherings

A facilitated discussion with participants, agenda and authored outcomes.

Domain rules: Part 3, clause 3.11. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `gathering`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `gathering`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`outcome`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `gathering`	The explicitly authored outcome of the relevant decision or gathering; no automatic inference from participation is implied.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.53 Agenda

Identifier: `urn:redbook:type:agenda` **Key:** `agenda` · **Primitive:** element · **Level:** type · **Area:** gatherings

An ordered plan of subjects for a gathering.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: agenda	Descriptive authored text; it does not execute, declare a new type or update the described subject.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.54 Agenda Item

Identifier: `urn:redbook:type:agenda-item` **Key:** `agenda-item` · **Primitive:** element · **Level:** type · **Area:** gatherings

One planned subject within an agenda.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `agenda-item`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"pending"` DECLARED BY: `agenda-item`	One of `"pending"`, `"current"`, `"complete"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.55 Stream

Identifier: `urn:redbook:type:stream` **Key:** `stream` · **Primitive:** element · **Level:** type · **Area:** gatherings

An authored sequence of supplied encoded media segments and participation; device capture and network transport are host capabilities.

Domain rules: Part 3, clause 3.12. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `stream`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`codec`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `stream`	Declared codec identifier for stream bytes. No decoding support is implied by retaining the string.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `stream`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`recordingAllowed`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `stream`	Authored stream-level recording choice. It does not replace participant recording consent or host policy.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.56 Stream Segment

Identifier: `urn:redbook:type:stream-segment` **Key:** `stream-segment` · **Primitive:** element · **Level:** type · **Area:** gatherings

A supplied encoded media part with authored timing and ordering.

Domain rules: Part 3, clause 3.12. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `stream-segment`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `stream-segment`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`timestamp`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `stream-segment`	Finite nonnegative offset in seconds on the Stream media timeline. It is distinct from recordedAt, event sequence and worldTime.
`duration`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `stream-segment`	Finite nonnegative segment duration in seconds of stream media time; appending a playable segment requires a positive duration.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.57 Moderation Report

Identifier: `urn:redbook:type:moderation-report` **Key:** `moderation-report` · **Primitive:** element · **Level:** type · **Area:** moderation

An attributed report about content or participation.

Domain rules: Part 3, clause 3.13. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`reason`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `moderation-report`	Authored explanation of a report or decision; it has no automatic punitive effect.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `moderation-report`	One of `"open"`, `"resolved"`, `"dismissed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`resolution`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `moderation-report`	Authored explanation of how a Moderation Report was resolved or dismissed.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.58 Participation Rule

Identifier: `urn:redbook:type:participation-rule` **Key:** `participation-rule` · **Primitive:** element · **Level:** type · **Area:** moderation

An authored rule for participation; not a password, file encryption or host credential.

Domain rules: Part 3, clause 3.13. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `participation-rule`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`conditions`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `participation-rule`	A declarative condition map evaluated only through the declared rule or decision procedure (3.9 and 3.13).
`effects`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `participation-rule`	An ordered list of declarative effect descriptors; neither presence nor enabled state performs these effects (3.13).
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `participation-rule`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.
`priority`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `participation-rule`	Safe integer priority; applicable rules sort ascending, then by exact rule identity. See 3.21.
`actions`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `participation-rule`	Unique exact action names to which the rule applies; an empty list means every operation with this actual policy context. See 3.21.

Property	Declaration	Allowed values / semantics
`mode`	KIND: string REQUIRED: no CREATION DEFAULT: `"require"` DECLARED BY: `participation-rule`	One of `"require"`, `"deny"`, `"effects"`. require denies when its condition is false; deny denies when true; effects selects its explicit effects when true. No rule elevates authority. See 3.21.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.59 Soma Neocortex

Identifier: `urn:redbook:type:neocortex` **Key:** `neocortex` · **Primitive:** element · **Level:** type · **Area:** soma

The conceptual home owned by exactly one Digital or communication context.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`scope`	KIND: string REQUIRED: no CREATION DEFAULT: `"personal"` DECLARED BY: `neocortex`	One of `"personal"`, `"shared"`. Personal for a Digital-owned Neocortex; shared for an eligible communication-context owner. The value must agree with its actual owner.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.60 Neo Soma

Identifier: `urn:redbook:type:neo-soma` **Key:** `neo-soma` · **Primitive:** element · **Level:** type · **Area:** soma

A conceptual world in exactly one Soma Neocortex; reusable independent material is a Neo Soma Blueprint.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `neo-soma`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`arrangement`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `neo-soma`	Authored conceptual presentation, including optional ordered typed-reference lists for Objects and Relationships (3.14).

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.61 Neo Object

Identifier: `urn:redbook:type:neo-object` **Key:** `neo-object` · **Primitive:** element · **Level:** type · **Area:** soma

An authored conceptual object inside a Neo Soma. Describing a world entity does not change that entity.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `neo-object`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`meaning`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `neo-object`	Authored interpretation of the concept or reference. Epistemic classifications remain claims in the model, not automatically proven facts.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.62 Viewbox

Identifier: `urn:redbook:type:viewbox` **Key:** `viewbox` · **Primitive:** element · **Level:** type · **Area:** presentation

A particular authored experience owned by this RedBook, distinct from a reusable Viewbox Type.

Domain rules: Part 4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`implementation`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `viewbox`	Package-root-relative entry module for the Viewbox implementation, resolved and activated under Part 4.
`configuration`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `viewbox`	Portable configuration validated against the selected implementation's declared contract; it does not contain host credentials.
`theme`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `viewbox`	The identity of the selected applicable theme, not a URL to fetch automatically.
`plugins`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `viewbox`	Ordered identities or declared assignments for Viewbox-targeted plugins; installation and activation follow Part 4.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.63 Profile Of

Identifier: `urn:redbook:type:profile-of` **Key:** `profile-of` · **Primitive:** relationship · **Level:** type · **Area:** identity

Connect a profile to its subject.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`profile`	ALLOWED TYPES: Profile MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Profile supplying the authored presentation.
`subject`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Element about which this association is made.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.64 Membership

Identifier: `urn:redbook:type:membership` **Key:** `membership` · **Primitive:** relationship · **Level:** type · **Area:** organization

Connect one or more members to an organized context with a role and state.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"member"` DECLARED BY: `membership`	The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"active"` DECLARED BY: `membership`	One of `"active"`, `"invited"`, `"requested"`, `"declined"`, `"left"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `membership`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`context`	ALLOWED TYPES: Group, Space, Page, Circle, Conversation, Gathering, Newsletter, Review, Poll, Questionnaire, Decision Round, Wall, Proposal, Event, Case, Stream MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.

Participant slot	Constraints	Meaning
`members`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals whose membership has the stated role and state.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.65 Participation

Identifier: `urn:redbook:type:participation` **Key:** `participation` · **Primitive:** relationship · **Level:** type · **Area:** identity

Associate Digitals with any explicit participation context.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"participant"` DECLARED BY: `participation`	The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"active"` DECLARED BY: `participation`	The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`context`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.
`participants`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals explicitly participating in the context.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.66 Invitation

Identifier: `urn:redbook:type:invitation` **Key:** `invitation` · **Primitive:** relationship · **Level:** type · **Area:** organization

An offer to participate; acceptance is explicit.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"pending"` DECLARED BY: `invitation`	One of `"pending"`, `"accepted"`, `"declined"`, `"cancelled"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `invitation`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`inviter`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital issuing the invitation.
`invitees`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals to whom this invitation is addressed.
`context`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.67 Follow

Identifier: `urn:redbook:type:follow` **Key:** `follow` · **Primitive:** relationship · **Level:** type · **Area:** identity

An attributed following relation to an identity, context or content.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `follow`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`followers`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals choosing to follow the named subject or shared focus.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.68 Connection

Identifier: `urn:redbook:type:connection` **Key:** `connection` · **Primitive:** relationship · **Level:** type · **Area:** identity

An attributed connection whose nature can be specialized.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"proposed"` DECLARED BY: `connection`	One of `"proposed"`, `"accepted"`, `"declined"`, `"ended"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `connection`	Additional authored explanatory text; it is not an automatic change log.
`responses`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `connection`	Explicit counterpart decisions in counterpart-slot order: each entry is {digital: typed reference to that counterpart, response: accepted or declined}. Missing entries are undecided; absence of the whole field is an explicit administrative aggregate without individual decision claims. See Part 3.3.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`initiators`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals initiating the proposed connection.
`counterparts`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The other Digitals in that connection.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.69 Authorship

Identifier: `urn:redbook:type:authorship` **Key:** `authorship` · **Primitive:** relationship · **Level:** type · **Area:** identity

Associate represented contributors with an authored item; does not confer host access.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"author"` DECLARED BY: `authorship`	The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`content`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The authored content to which this association applies.
`authors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals credited with the stated contribution role.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.70 Wall Placement

Identifier: `urn:redbook:type:wall-placement` **Key:** `wall-placement` · **Primitive:** relationship · **Level:** type · **Area:** publication

Place content on a Wall with its own position, pin and submission state.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `wall-placement`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.
`pinned`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `wall-placement`	Whether a Wall placement is in the pinned group that precedes other visible placements.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"published"` DECLARED BY: `wall-placement`	One of `"published"`, `"pending"`, `"rejected"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `wall-placement`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`wall`	ALLOWED TYPES: Wall MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Wall that hosts the placement.

Participant slot	Constraints	Meaning
`items`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements included by reference, in the stored participant order.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.71 Contribution

Identifier: `urn:redbook:type:contribution` **Key:** `contribution` · **Primitive:** relationship · **Level:** type · **Area:** discussion

Associate authored material with its communication context.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `contribution`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`context`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.
`items`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements included by reference, in the stored participant order.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.72 Reply

Identifier: `urn:redbook:type:reply` **Key:** `reply` · **Primitive:** relationship · **Level:** type · **Area:** discussion

Identify which contribution answers another; does not duplicate either body.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`replies`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The response Elements; they remain independently authored records.
`antecedents`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The earlier or otherwise addressed Elements to which the replies respond.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.73 Accepted Answer

Identifier: `urn:redbook:type:accepted-answer` **Key:** `accepted-answer` · **Primitive:** relationship · **Level:** type · **Area:** discussion

An explicit answer selection, not a truth claim or automatic inference.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`question`	ALLOWED TYPES: Message, Post MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The question-bearing contribution for which an answer is selected.
`answers`	ALLOWED TYPES: Message, Post MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The contributions explicitly selected as answers.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.74 Mention

Identifier: `urn:redbook:type:mention` **Key:** `mention` · **Primitive:** relationship · **Level:** type · **Area:** expression

An attributed reference to Digitals from authored content.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`source`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Element from which this association originates.
`mentioned`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals explicitly mentioned by the source.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.75 Reaction

Identifier: `urn:redbook:type:reaction` **Key:** `reaction` · **Primitive:** relationship · **Level:** type · **Area:** expression

A Digital's deliberate response, which need not mean agreement or authorization.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`value`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	The authored reaction value; strings are not silently interpreted as numbers, truth values or accepted responsibility.
`criteria`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `reaction`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.76 Annotation Target

Identifier: `urn:redbook:type:annotation-target` **Key:** `annotation-target` · **Primitive:** relationship · **Level:** type · **Area:** expression

Connect an annotation to material; selector coordinates live on the annotation.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`annotation`	ALLOWED TYPES: Annotation MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The separately authored Annotation supplying commentary and a selector.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.77 Circulation

Identifier: `urn:redbook:type:circulation` **Key:** `circulation` · **Primitive:** relationship · **Level:** type · **Area:** expression

A reference, quotation or forward with its own commentary.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.78 Attachment

Identifier: `urn:redbook:type:attachment` **Key:** `attachment` · **Primitive:** relationship · **Level:** type · **Area:** content

Associate original resources with any suitable authored content.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"attachment"` DECLARED BY: `attachment`	The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `attachment`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`content`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The authored content to which this association applies.
`resources`	ALLOWED TYPES: Media, Work, Neo Soma MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	Media, Works or Neo Somas associated as material; their records are not copied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.79 Representation Of

Identifier: `urn:redbook:type:representation-of` **Key:** `representation-of` · **Primitive:** relationship · **Level:** type · **Area:** content

Link a transcript, translation or other authored representation to its source.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`representation`	ALLOWED TYPES: Representation MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The separately authored representation of the sources.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.80 Delivery

Identifier: `urn:redbook:type:delivery` **Key:** `delivery` · **Primitive:** relationship · **Level:** type · **Area:** correspondence

A specific internal message delivery to a recipient, including To/Cc/Bcc role.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`recipientRole`	KIND: string REQUIRED: no CREATION DEFAULT: `"to"` DECLARED BY: `delivery`	One of `"to"`, `"cc"`, `"bcc"`, `"subscriber"`. The explicit recipient arrangement used for a Delivery. Bcc projection is defined in 3.5.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"delivered"` DECLARED BY: `delivery`	One of `"delivered"`, `"undeliverable"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`message`	ALLOWED TYPES: Mail Message, Newsletter Issue, Notification MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The correspondence or notice item being delivered or organized.
`recipient`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital for whom this delivery or organizational state is recorded.
`sender`	ALLOWED TYPES: Digital MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital responsible for the delivery, when identified.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.81 Mailbox Entry

Identifier: `urn:redbook:type:mailbox-entry` **Key:** `mailbox-entry` · **Primitive:** relationship · **Level:** type · **Area:** correspondence

An independently organized recipient copy of a message.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`labels`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `mailbox-entry`	Ordered unique authored mailbox labels, local to this recipient's organizational view.
`read`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `mailbox-entry`	Whether the recipient explicitly marked this entry read; opening or listing it does not change this value.
`acknowledged`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `mailbox-entry`	Whether the recipient explicitly acknowledged this entry. This does not imply acceptance of responsibility.
`starred`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `mailbox-entry`	The recipient's explicit saved-importance choice for this entry.
`snoozedUntil`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mailbox-entry`	An authored presentation-suppression endpoint; it does not create a delivery job.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`mailbox`	ALLOWED TYPES: Mailbox MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Mailbox hosting the entry or handling assignment.
`message`	ALLOWED TYPES: Mail Message, Newsletter Issue MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The correspondence or notice item being delivered or organized.
`recipient`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital for whom this delivery or organizational state is recorded.
`folder`	ALLOWED TYPES: Mail Folder MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The optional organizational Mail Folder; it does not own message bytes.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.82 Mailbox Owner

Identifier: `urn:redbook:type:mailbox-owner` **Key:** `mailbox-owner` · **Primitive:** relationship · **Level:** type · **Area:** correspondence

Associate a personal or collective mailbox with its responsible Digital.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`mailbox`	ALLOWED TYPES: Mailbox MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Mailbox hosting the entry or handling assignment.
`owners`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals responsible for the Mailbox.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.83 Mail Handling

Identifier: `urn:redbook:type:mail-handling` **Key:** `mail-handling` · **Primitive:** relationship · **Level:** type · **Area:** correspondence

Shared-inbox responsibility and notes without changing other recipients' correspondence.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"unassigned"` DECLARED BY: `mail-handling`	One of `"unassigned"`, `"assigned"`, `"resolved"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `mail-handling`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`mailbox`	ALLOWED TYPES: Mailbox MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Mailbox hosting the entry or handling assignment.
`message`	ALLOWED TYPES: Mail Message MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The correspondence or notice item being delivered or organized.
`responsible`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals assigned or accepting the specified responsibility.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.84 Recipient List Member

Identifier: `urn:redbook:type:recipient-list-member` **Key:** `recipient-list-member` · **Primitive:** relationship · **Level:** type · **Area:** correspondence

Associate selected recipients with a distribution list.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`list`	ALLOWED TYPES: Distribution List MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The reusable Distribution List.
`recipients`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The addressed Digitals.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.85 Newsletter Entry

Identifier: `urn:redbook:type:newsletter-entry` **Key:** `newsletter-entry` · **Primitive:** relationship · **Level:** type · **Area:** distribution

Associate an issue with its newsletter and authored issue order.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `newsletter-entry`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`newsletter`	ALLOWED TYPES: Newsletter MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Newsletter series to which the issue belongs.
`issues`	ALLOWED TYPES: Newsletter Issue MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Newsletter Issues associated with the series.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.86 Subscription

Identifier: `urn:redbook:type:subscription` **Key:** `subscription` · **Primitive:** relationship · **Level:** type · **Area:** attention

An authored choice to follow material or receive distribution; does not start a scheduler.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `subscription`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.
`filters`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `subscription`	Declared predicates restricting a subscription or preference (3.8). Unknown predicate semantics are not guessed.
`quiet`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `subscription`	Explicit quiet-period preferences using the declarative map conventions in 3.8.
`digest`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `subscription`	Authored digest map, interpreted according to the compound-property conventions and this type's domain contract.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`subscriber`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital choosing the subscription.

Participant slot	Constraints	Meaning
sources	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.87 Submission Target

Identifier: `urn:redbook:type:submission-target` **Key:** `submission-target` · **Primitive:** relationship · **Level:** type · **Area:** distribution

Connect a submission with its source material and proposed destination.

Domain rules: Part 3, clause 3.5. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`submission`	ALLOWED TYPES: Submission MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Submission whose independent review state applies.
`content`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The authored content to which this association applies.
`destination`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.88 Collection Entry

Identifier: `urn:redbook:type:collection-entry` **Key:** `collection-entry` · **Primitive:** relationship · **Level:** type · **Area:** discovery

An ordered, annotated reference in a collection.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `collection-entry`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection-entry`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`collection`	ALLOWED TYPES: Collection, Agenda MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Collection or Agenda that contains this reference arrangement.
`items`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements included by reference, in the stored participant order.

Participant slot	Constraints	Meaning
`section`	ALLOWED TYPES: Collection Section MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The optional Collection Section used to group this entry.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.89 Bookmark

Identifier: `urn:redbook:type:bookmark` **Key:** `bookmark` · **Primitive:** relationship · **Level:** type · **Area:** attention

A Digital's saved reference with its own note.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `bookmark`	Additional authored explanatory text; it is not an automatic change log.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `bookmark`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`digital`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital whose personal statement or organizational preference is expressed.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.90 Attention

Identifier: `urn:redbook:type:attention` **Key:** `attention` · **Primitive:** relationship · **Level:** type · **Area:** attention

Explicit focus, reading or acknowledgment attributed to a Digital. Mere retrieval creates none.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`kind`	KIND: string REQUIRED: no CREATION DEFAULT: `"focus"` DECLARED BY: `attention`	One of `"focus"`, `"read"`, `"acknowledged"`, `"dismissed"`. The domain-specific kind of statement or representation; listed choices, if any, are exhaustive for this root contract.
`position`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `attention`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`digital`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital whose personal statement or organizational preference is expressed.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.91 Notice Target

Identifier: `urn:redbook:type:notice-target` **Key:** `notice-target` · **Primitive:** relationship · **Level:** type · **Area:** attention

Address a notice or reminder and connect its source material.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`notice`	ALLOWED TYPES: Notification, Reminder MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Notification or Reminder supplying the addressed content.
`recipients`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The addressed Digitals.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.92 Presence Of

Identifier: `urn:redbook:type:presence-of` **Key:** `presence-of` · **Primitive:** relationship · **Level:** type · **Area:** attention

Associate an authored presence statement with its Digital and context.

Domain rules: Part 3, clause 3.8. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`presence`	ALLOWED TYPES: Presence MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The authored Presence statement, not a live-process observation.
`digital`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital whose personal statement or organizational preference is expressed.
`context`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.93 Ballot

Identifier: `urn:redbook:type:ballot` **Key:** `ballot` · **Primitive:** relationship · **Level:** type · **Area:** coordination

A participant's structured poll response, separate from public tally presentation.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`choices`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `ballot`	Distinct selected Poll option IDs for single/multiple methods; the inactive method fields are empty.
`ranking`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `ballot`	Distinct Poll option IDs in preferred order for the ranked method; omitted options receive zero points.
`ratings`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `ballot`	A map from valid Poll option IDs to finite numeric ratings for the rating method. The local key $rb is forbidden (3.9; domain-response.schema.json).
`withdrawn`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `ballot`	Whether the retained ballot is excluded from the current tally.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`poll`	ALLOWED TYPES: Poll MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Poll whose options and method govern this ballot.
`voter`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital casting the current ballot.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.94 Questionnaire Response

Identifier: `urn:redbook:type:questionnaire-response` **Key:** `questionnaire-response` · **Primitive:** relationship · **Level:** type · **Area:** coordination

A respondent's answers with independent draft/submitted state.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`answers`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `questionnaire-response`	A map from declared Questionnaire question IDs to current portable response values. The local key $rb is forbidden (3.9; domain-response.schema.json).
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `questionnaire-response`	One of `"draft"`, `"submitted"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`questionnaire`	ALLOWED TYPES: Questionnaire MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Questionnaire whose question definitions govern these answers.
`respondent`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital supplying this current response.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.95 Decision Position

Identifier: `urn:redbook:type:decision-position` **Key:** `decision-position` · **Primitive:** relationship · **Level:** type · **Area:** coordination

A participant's explicit position on a proposal or decision round.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: string REQUIRED: no CREATION DEFAULT: `"support"` DECLARED BY: `decision-position`	One of `"support"`, `"object"`, `"abstain"`, `"withdrawn"`. The participant's current stance on the Proposal or Decision Round. The choices express support, objection, abstention or withdrawal; the outcome is a separately authored Decision (3.9).
`reason`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `decision-position`	Authored explanation of a report or decision; it has no automatic punitive effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`subject`	ALLOWED TYPES: Proposal, Decision Round MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Element about which this association is made.
`participant`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital making this particular response or attendance report.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.96 Review Target

Identifier: `urn:redbook:type:review-target` **Key:** `review-target` · **Primitive:** relationship · **Level:** type · **Area:** coordination

The material being considered in a review.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`review`	ALLOWED TYPES: Review MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Review context that defines the review procedure.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.97 Finding In

Identifier: `urn:redbook:type:finding-in` **Key:** `finding-in` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Associate an attributed finding with its review.

Domain rules: Part 3, clause 3.9. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`review`	ALLOWED TYPES: Review MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Review context that defines the review procedure.
`findings`	ALLOWED TYPES: Finding MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The separately authored Findings submitted in that review.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.98 Request Address

Identifier: `urn:redbook:type:request-address` **Key:** `request-address` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Connect a request or offer to its requester and intended responders.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`request`	ALLOWED TYPES: Request MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Request that invited or generated the responsibility.
`requester`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital issuing the Request.
`responders`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals permitted to accept or decline this Request.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.99 Commitment For

Identifier: `urn:redbook:type:commitment-for` **Key:** `commitment-for` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Associate accepted responsibility with the request and responsible Digitals.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`commitment`	ALLOWED TYPES: Commitment MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The accepted Commitment being related or updated.
`request`	ALLOWED TYPES: Request MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Request that invited or generated the responsibility.
`responsible`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals assigned or accepting the specified responsibility.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.100 Progress On

Identifier: `urn:redbook:type:progress-on` **Key:** `progress-on` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Associate an authored update with its commitment.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`commitment`	ALLOWED TYPES: Commitment MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The accepted Commitment being related or updated.
`updates`	ALLOWED TYPES: Progress MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The explicitly authored Progress statements.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.101 Case Material

Identifier: `urn:redbook:type:case-material` **Key:** `case-material` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Evidence and other material relevant to a continuing matter.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"material"` DECLARED BY: `case-material`	The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `case-material`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`case`	ALLOWED TYPES: Case MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The continuing Case that associates this material.
`items`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements included by reference, in the stored participant order.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.102 Reservation For

Identifier: `urn:redbook:type:reservation-for` **Key:** `reservation-for` · **Primitive:** relationship · **Level:** type · **Area:** coordination

Connect a reservation to its resource and requesting Digitals.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`reservation`	ALLOWED TYPES: Reservation MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Reservation whose interval and quantity are requested.
`resource`	ALLOWED TYPES: Resource MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The reservable Resource whose capacity is consumed.
`requesters`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals requesting the use.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.103 Event Response

Identifier: `urn:redbook:type:event-response` **Key:** `event-response` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

An RSVP or invitation response; distinct from attendance.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`response`	KIND: string REQUIRED: no CREATION DEFAULT: `"accepted"` DECLARED BY: `event-response`	One of `"accepted"`, `"declined"`, `"tentative"`. The Digital's RSVP choice for the identified Event occurrence, independent of attendance.
`occurrence`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `event-response`	The canonical original occurrence-start key, or the empty string for the Event as a whole.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`event`	ALLOWED TYPES: Event MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Event or Gathering for which this response is authored.
`participant`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital making this particular response or attendance report.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.104 Attendance

Identifier: `urn:redbook:type:attendance` **Key:** `attendance` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

An expressly authored attendance report, not inferred from an invitation response.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`status`	KIND: string REQUIRED: no CREATION DEFAULT: `"attended"` DECLARED BY: `attendance`	One of `"attended"`, `"not-attended"`. The current authored presence or attendance status; it is not inferred from network activity or an RSVP.
`occurrence`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `attendance`	The canonical original occurrence-start key, or the empty string for the Event as a whole.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `attendance`	Additional authored explanatory text; it is not an automatic change log.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`event`	ALLOWED TYPES: Event, Gathering MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Event or Gathering for which this response is authored.
`participant`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital making this particular response or attendance report.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.105 Gathering Role

Identifier: `urn:redbook:type:gathering-role` **Key:** `gathering-role` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

A chair, participant or explicit speaking turn within a facilitated exchange.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"participant"` DECLARED BY: `gathering-role`	One of `"participant"`, `"chair"`, `"speaker"`. The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"active"` DECLARED BY: `gathering-role`	One of `"active"`, `"requested"`, `"finished"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`gathering`	ALLOWED TYPES: Gathering MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Gathering in which these roles apply.
`digitals`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals holding this context-specific role and state.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.106 Shared Focus

Identifier: `urn:redbook:type:shared-focus` **Key:** `shared-focus` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

An explicitly shared target of conceptual or media attention.

Domain rules: Part 3, clause 3.11. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`selector`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `shared-focus`	A whole-item, Unicode-text, rectangle or time-span selector map, with the addressing and anchoring rules in 3.6.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`context`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.
`presenter`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital deliberately presenting a shared focus.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.
`followers`	ALLOWED TYPES: Digital MINIMUM: 0 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals choosing to follow the named subject or shared focus.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.107 Stream Participation

Identifier: `urn:redbook:type:stream-participation` **Key:** `stream-participation` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

Roles in a supplied-media exchange; no device or AI session is implied.

Domain rules: Part 3, clause 3.12. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`role`	KIND: string REQUIRED: no CREATION DEFAULT: `"listener"` DECLARED BY: `stream-participation`	One of `"listener"`, `"publisher"`, `"moderator"`. The authored role name within this specific association. It grants only the powers explicitly assigned by the relevant domain contract.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"invited"` DECLARED BY: `stream-participation`	One of `"invited"`, `"active"`, `"left"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`recordingConsent`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `stream-participation`	The Digital's explicit consent statement for retained stream recording, separate from participation and muting.
`muted`	KIND: boolean REQUIRED: no CREATION DEFAULT: `false` DECLARED BY: `stream-participation`	Whether the Digital is currently barred from publishing a segment in this stream participation record.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`stream`	ALLOWED TYPES: Stream MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Stream whose participation and segment order are described.

Participant slot	Constraints	Meaning
`digitals`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals holding this context-specific role and state.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.108 Stream Entry

Identifier: `urn:redbook:type:stream-entry` **Key:** `stream-entry` · **Primitive:** relationship · **Level:** type · **Area:** gatherings

Associate an encoded segment with its stream and publisher.

Domain rules: Part 3, clause 3.12. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`position`	KIND: number REQUIRED: no CREATION DEFAULT: `0` DECLARED BY: `stream-entry`	A finite authored ordering coordinate; ties are broken as specified in 3.4. Attention uses a map instead, identifying the retained position within its target.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`stream`	ALLOWED TYPES: Stream MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Stream whose participation and segment order are described.
`segment`	ALLOWED TYPES: Stream Segment MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The independently stored Stream Segment.
`publisher`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Digital responsible for adding the segment to the Stream.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.109 Report Target

Identifier: `urn:redbook:type:report-target` **Key:** `report-target` · **Primitive:** relationship · **Level:** type · **Area:** moderation

The content and context to which a moderation report pertains.

Domain rules: Part 3, clause 3.13. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`report`	ALLOWED TYPES: Moderation Report MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The independently authored Moderation Report.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.
`context`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The host or subject context of this association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.110 Rule Target

Identifier: `urn:redbook:type:rule-target` **Key:** `rule-target` · **Primitive:** relationship · **Level:** type · **Area:** moderation

Apply an authored participation rule to selected contexts.

Domain rules: Part 3, clause 3.13. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`rule`	ALLOWED TYPES: Participation Rule MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Participation Rule being assigned.
`contexts`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The contexts to which the explicit rule assignment applies.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.111 Neocortex Owner

Identifier: `urn:redbook:type:neocortex-owner` **Key:** `neocortex-owner` · **Primitive:** relationship · **Level:** type · **Area:** soma

Exactly one Neocortex belongs to its Digital or communication context.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`owner`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The single Digital or eligible communication context owning the Neocortex.
`neocortex`	ALLOWED TYPES: Soma Neocortex MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The single Soma Neocortex participating as conceptual home.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.112 Soma Home

Identifier: `urn:redbook:type:soma-home` **Key:** `soma-home` · **Primitive:** relationship · **Level:** type · **Area:** soma

Every ordinary Neo Soma has exactly one Neocortex home.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`neocortex`	ALLOWED TYPES: Soma Neocortex MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The single Soma Neocortex participating as conceptual home.
`soma`	ALLOWED TYPES: Neo Soma MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The single Neo Soma whose home or contained Objects are described.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.113 Neo Membership

Identifier: `urn:redbook:type:neo-membership` **Key:** `neo-membership` · **Primitive:** relationship · **Level:** type · **Area:** soma

A Neo Object belongs to one conceptual world.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`soma`	ALLOWED TYPES: Neo Soma MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The single Neo Soma whose home or contained Objects are described.
`objects`	ALLOWED TYPES: Neo Object MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Neo Objects belonging to the named Neo Soma.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.114 Neo Relationship

Identifier: `urn:redbook:type:neo-relationship` **Key:** `neo-relationship` · **Primitive:** relationship · **Level:** type · **Area:** soma

An authored conceptual arrangement of Neo Objects within one Neo Soma, with arbitrary named slots.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`somaId`	KIND: reference REQUIRED: yes CREATION DEFAULT: none DECLARED BY: `neo-relationship`	A typed reference to the one Neo Soma containing this Neo Relationship. All participant Objects belong to that same Soma.

PARTICIPANT SLOTS

No slot names are predefined. The author supplies explicit role names and participant lists. At least one participant occurs across the Relationship.

Open slot names: yes. Each supplied slot contains a list of existing Element identifiers; authored order and multiplicity are preserved. Every participant must be a Neo Object in the Neo Soma identified by `properties.somaId`.

A.115 Represents

Identifier: `urn:redbook:type:represents` **Key:** `represents` · **Primitive:** relationship · **Level:** type · **Area:** soma

A conceptual object may refer to a world Element without becoming that world Element.

Domain rules: Part 3, clause 3.14. **Effective traits:** none.

PROPERTIES

No additional named Property is declared. Common Properties and extension Properties remain available.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`concept`	ALLOWED TYPES: Neo Object MINIMUM: 1 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The Neo Object that represents, but is not identical to, its subjects.
`subjects`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The actual Elements represented by the concept.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.116 Reference

Identifier: `urn:redbook:type:reference` **Key:** `reference` · **Primitive:** relationship · **Level:** type · **Area:** general

A named authored association not forced into a two-ended pattern.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`meaning`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reference`	Authored interpretation of the concept or reference. Epistemic classifications remain claims in the model, not automatically proven facts.

PARTICIPANT SLOTS

No slot names are predefined. The author supplies explicit role names and participant lists. At least one participant occurs across the Relationship.

Open slot names: yes. Each supplied slot contains a list of existing Element identifiers; authored order and multiplicity are preserved. The `meaning` Property explains the association; no source/target or binary-edge interpretation is imposed.

A.117 Digital Person

Identifier: `urn:redbook:type:digital.person` **Key:** `digital.person` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Digital represented as a person. This is an authored category; no biological identity, login, personality or executing intelligence is implied.

Parent: Digital. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.118 Digital Animal

Identifier: `urn:redbook:type:digital.animal` **Key:** `digital.animal` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Digital represented as an animal; it remains an actor for attribution through an external human or AI.

Parent: Digital. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.119 Digital Thing

Identifier: `urn:redbook:type:digital.thing` **Key:** `digital.thing` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Digital represented as a thing that can be assigned authored activity. It is distinct from an ordinary non-acting item.

Parent: Digital. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.120 Digital Collective

Identifier: `urn:redbook:type:digital.collective` **Key:** `digital.collective` · **Primitive:** element · **Level:** specialization · **Area:** specializations

One acting Digital representing a collective. It is not the Group Element or its membership list, and its actions are attributed to this collective Digital.

Parent: Digital. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.121 Digital Concept

Identifier: `urn:redbook:type:digital.concept` **Key:** `digital.concept` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A conceptual actor used for attributed activity. It is distinct from a Neo Object representing an idea.

Parent: Digital. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** `personal-neocortex`.

Every instance owns exactly one personal Soma Neocortex through one Neocortex Owner Relationship. Creation includes both dependent records atomically.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`guidance`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `digital`	Authored guidance for a human or AI representing this Digital; storing it neither grants authority nor starts execution.
`appearance`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `digital`	A portable map of authored presentation choices and resource references, separate from host account settings.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.122 Article

Identifier: `urn:redbook:type:post.article` **Key:** `post.article` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Post presented as an article or extended publication. Its publication state and placements remain ordinary Post semantics.

Parent: Post. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.123 Journal Entry

Identifier: `urn:redbook:type:post.journal` **Key:** `post.journal` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Post presented as a journal contribution. It is deliberately authored content, not an automatic edit log.

Parent: Post. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.124 Status

Identifier: `urn:redbook:type:post.status` **Key:** `post.status` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Post presented as a concise update about an authored state or situation.

Parent: Post. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.125 Story

Identifier: `urn:redbook:type:post.story` **Key:** `post.story` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Post presented as a story. Intended expiry remains a stated condition and does not itself run a timer or delete the Post.

Parent: Post. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.126 Observation

Identifier: `urn:redbook:type:post.observation` **Key:** `post.observation` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Post expressing an observation. It does not turn the statement into a verified fact or live query.

Parent: Post. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `post`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `post`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`publishAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended publication coordinate in the stated world clock. It does not schedule a publication action.
`expiresAt`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `post`	Intended expiry coordinate in the stated world clock. It does not delete or withdraw content automatically.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.127 Direct Conversation

Identifier: `urn:redbook:type:conversation.direct` **Key:** `conversation.direct` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation presented as a direct exchange among explicitly chosen participants. The specialization is not an encryption boundary and imposes no implicit two-person limit.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.128 Group Conversation

Identifier: `urn:redbook:type:conversation.group` **Key:** `conversation.group` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation presented as a group exchange; membership remains explicit and distinct from a Group Element.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.129 Channel

Identifier: `urn:redbook:type:conversation.channel` **Key:** `conversation.channel` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A continuing topic or audience channel represented as a Conversation.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.130 Forum

Identifier: `urn:redbook:type:conversation.forum` **Key:** `conversation.forum` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A forum-like Conversation; subordinate topic relationships, if authored, remain explicit rather than implicit title nesting.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.131 Topic

Identifier: `urn:redbook:type:conversation.topic` **Key:** `conversation.topic` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation focused on one subject within an explicitly authored wider discussion.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.132 Thread

Identifier: `urn:redbook:type:conversation.thread` **Key:** `conversation.thread` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation representing a thread; its parent or source association is authored explicitly.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.133 Correspondence Thread

Identifier: `urn:redbook:type:conversation.mail-thread` **Key:** `conversation.mail-thread` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation providing the shared context for internal correspondence. Mail Messages and delivery state remain separate records.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.134 Live Room

Identifier: `urn:redbook:type:conversation.live-room` **Key:** `conversation.live-room` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Conversation presented for live exchange. It does not imply an active network connection, attendance or clock.

Parent: Conversation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** `shared-neocortex`.

Every instance owns exactly one shared Soma Neocortex through one Neocortex Owner Relationship. Its existence does not populate that Neocortex with Neo Somas.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `conversation`	One of `"open"`, `"closed"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`participation`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `conversation`	The context participation policy map, including optional enrollment and participation constraints (3.3). It does not store host credentials.
`mailParticipants`	KIND: array REQUIRED: no CREATION DEFAULT: none DECLARED BY: `conversation`	When present, ordered unique typed references to the initial public correspondence audience: sender and To/Cc participants. This marks a mail thread; Bcc delivery does not add Membership. See 3.19.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.135 Question

Identifier: `urn:redbook:type:message.question` **Key:** `message.question` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Message asking a question. An Accepted Answer relationship records a selected answer without claiming objective truth.

Parent: Message. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `message`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `message`	One of `"draft"`, `"sent"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.136 Answer

Identifier: `urn:redbook:type:message.answer` **Key:** `message.answer` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Message offered as an answer. Reply and Accepted Answer relationships independently identify what it addresses and whether it was selected.

Parent: Message. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `message`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `message`	One of `"draft"`, `"sent"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.137 Summary

Identifier: `urn:redbook:type:message.summary` **Key:** `message.summary` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Message summarizing other material in authored wording; it does not replace its sources.

Parent: Message. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `message`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `message`	One of `"draft"`, `"sent"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.138 Image

Identifier: `urn:redbook:type:media.image` **Key:** `media.image` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media whose original asset is an image. Alternative text and descriptions remain independent authored representations.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.139 Audio

Identifier: `urn:redbook:type:media.audio` **Key:** `media.audio` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media whose original asset is audio. Playback and decoding are host capabilities.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.140 Video

Identifier: `urn:redbook:type:media.video` **Key:** `media.video` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media whose original asset is video. Timed cues, captions and playback capabilities are separate.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.141 File

Identifier: `urn:redbook:type:media.file` **Key:** `media.file` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media used for an arbitrary original file with its declared media type.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.142 Source Repository

Identifier: `urn:redbook:type:media.source` **Key:** `media.source` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media containing source code as original bytes. Reading or storing it does not activate or execute it.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.143 Recording

Identifier: `urn:redbook:type:media.recording` **Key:** `media.recording` · **Primitive:** element · **Level:** specialization · **Area:** specializations

Media representing a deliberately retained recording; it is not automatically generated from a live exchange.

Parent: Media. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`mediaType`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	The declared IANA-style media type of the original bytes; it does not prove that a host can decode or execute them.
`asset`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	A package-root-relative original-resource path, resolved under Part 2. It is not a network retrieval instruction.
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`altText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `media`	Authored alternative text conveying the material without requiring its visual rendering.
`details`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `media`	Additional portable descriptive metadata. Members have no implied executable meaning without a declared content profile.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.144 Document

Identifier: `urn:redbook:type:work.document` **Key:** `work.document` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work arranged as a document, with its own authored composition and publication state.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.145 Notebook

Identifier: `urn:redbook:type:work.notebook` **Key:** `work.notebook` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work arranged as a notebook; its entries are authored content and not automatically retained versions.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.146 Canvas

Identifier: `urn:redbook:type:work.canvas` **Key:** `work.canvas` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work arranged spatially. Its body declares the arrangement; collaborative editing or rendering requires a declared implementation.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.147 Dataset

Identifier: `urn:redbook:type:work.dataset` **Key:** `work.dataset` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work presenting a dataset with explicitly represented values, schema and sources.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.148 Knowledge Page

Identifier: `urn:redbook:type:work.knowledge-page` **Key:** `work.knowledge-page` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work arranged as a knowledge page; links and claims retain their authored status.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.149 Artifact

Identifier: `urn:redbook:type:work.artifact` **Key:** `work.artifact` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work presented as an artifact, which may associate original resources without executing them.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.150 Deliverable

Identifier: `urn:redbook:type:work.deliverable` **Key:** `work.deliverable` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Work presented as a deliverable. Association with a Commitment does not itself verify completion.

Parent: Work. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `work`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"draft"` DECLARED BY: `work`	One of `"draft"`, `"published"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.151 Album

Identifier: `urn:redbook:type:collection.album` **Key:** `collection.album` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Collection presented as an album of referenced items.

Parent: Collection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.152 Board

Identifier: `urn:redbook:type:collection.board` **Key:** `collection.board` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Collection presented as a board. Sections and positions are explicit authored arrangement.

Parent: Collection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.153 Portfolio

Identifier: `urn:redbook:type:collection.portfolio` **Key:** `collection.portfolio` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Collection presented as a portfolio of referenced works or other items.

Parent: Collection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.154 Playlist

Identifier: `urn:redbook:type:collection.playlist` **Key:** `collection.playlist` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Collection presented for ordered playback. Membership does not copy or modify the original media.

Parent: Collection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.155 Reading List

Identifier: `urn:redbook:type:collection.reading-list` **Key:** `collection.reading-list` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Collection presented as a reading list. Presence on the list does not mark the material read.

Parent: Collection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.7. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`description`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `collection`	Descriptive authored text; it does not execute, declare a new type or update the described subject.
`ordering`	KIND: string REQUIRED: no CREATION DEFAULT: `"manual"` DECLARED BY: `collection`	One of `"manual"`, `"chronological"`. Selects the domain ordering rule. Explicit placement positions and their deterministic tie-breaker are defined in 3.4 and 3.7.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.156 Friendship

Identifier: `urn:redbook:type:connection.friendship` **Key:** `connection.friendship` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Connection describing an explicitly authored friendship relation, with its own proposed, accepted, declined or ended state.

Parent: Connection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"proposed"` DECLARED BY: `connection`	One of `"proposed"`, `"accepted"`, `"declined"`, `"ended"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `connection`	Additional authored explanatory text; it is not an automatic change log.
`responses`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `connection`	Explicit counterpart decisions in counterpart-slot order: each entry is {digital: typed reference to that counterpart, response: accepted or declined}. Missing entries are undecided; absence of the whole field is an explicit administrative aggregate without individual decision claims. See Part 3.3.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`initiators`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals initiating the proposed connection.

Participant slot	Constraints	Meaning
`counterparts`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The other Digitals in that connection.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.157 Affiliation

Identifier: `urn:redbook:type:connection.affiliation` **Key:** `connection.affiliation` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Connection describing affiliation among its Digital participants; it is not a Group membership grant.

Parent: Connection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"proposed"` DECLARED BY: `connection`	One of `"proposed"`, `"accepted"`, `"declined"`, `"ended"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `connection`	Additional authored explanatory text; it is not an automatic change log.
`responses`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `connection`	Explicit counterpart decisions in counterpart-slot order: each entry is {digital: typed reference to that counterpart, response: accepted or declined}. Missing entries are undecided; absence of the whole field is an explicit administrative aggregate without individual decision claims. See Part 3.3.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`initiators`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals initiating the proposed connection.

Participant slot	Constraints	Meaning
`counterparts`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The other Digitals in that connection.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.158 Mentorship

Identifier: `urn:redbook:type:connection.mentorship` **Key:** `connection.mentorship` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Connection describing mentorship. Initiator and counterpart slots identify participation, not a hidden authority hierarchy.

Parent: Connection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"proposed"` DECLARED BY: `connection`	One of `"proposed"`, `"accepted"`, `"declined"`, `"ended"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `connection`	Additional authored explanatory text; it is not an automatic change log.
`responses`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `connection`	Explicit counterpart decisions in counterpart-slot order: each entry is {digital: typed reference to that counterpart, response: accepted or declined}. Missing entries are undecided; absence of the whole field is an explicit administrative aggregate without individual decision claims. See Part 3.3.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`initiators`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals initiating the proposed connection.

Participant slot	Constraints	Meaning
counterparts	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The other Digitals in that connection.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.159 Introduction

Identifier: `urn:redbook:type:connection.introduction` **Key:** `connection.introduction` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Connection describing an introduction among Digitals. Acceptance is explicitly authored.

Parent: Connection. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.3. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"proposed"` DECLARED BY: `connection`	One of `"proposed"`, `"accepted"`, `"declined"`, `"ended"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.
`note`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `connection`	Additional authored explanatory text; it is not an automatic change log.
`responses`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `connection`	Explicit counterpart decisions in counterpart-slot order: each entry is {digital: typed reference to that counterpart, response: accepted or declined}. Missing entries are undecided; absence of the whole field is an explicit administrative aggregate without individual decision claims. See Part 3.3.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`initiators`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals initiating the proposed connection.

Participant slot	Constraints	Meaning
counterparts	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The other Digitals in that connection.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.160 Rating

Identifier: `urn:redbook:type:reaction.rating` **Key:** `reaction.rating` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Reaction carrying an authored rating value and optional criterion. The string value is not implicitly converted into a universal numeric scale.

Parent: Reaction. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`value`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	The authored reaction value; strings are not silently interpreted as numbers, truth values or accepted responsibility.
`criteria`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `reaction`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.161 Acknowledgment

Identifier: `urn:redbook:type:reaction.acknowledgment` **Key:** `reaction.acknowledgment` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Reaction expressing acknowledgment. It is distinct from read tracking or accepted responsibility.

Parent: Reaction. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`value`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	The authored reaction value; strings are not silently interpreted as numbers, truth values or accepted responsibility.
`criteria`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `reaction`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.162 Recommendation

Identifier: `urn:redbook:type:reaction.recommendation` **Key:** `reaction.recommendation` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Reaction expressing a recommendation by its actors.

Parent: Reaction. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`value`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	The authored reaction value; strings are not silently interpreted as numbers, truth values or accepted responsibility.
`criteria`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `reaction`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.163 Endorsement

Identifier: `urn:redbook:type:reaction.endorsement` **Key:** `reaction.endorsement` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Reaction expressing endorsement, not certification or automatic agreement by other participants.

Parent: Reaction. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`value`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	The authored reaction value; strings are not silently interpreted as numbers, truth values or accepted responsibility.
`criteria`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `reaction`	Declared review criteria or an authored reaction criterion. Its field kind determines whether this is a list or a string.
`enabled`	KIND: boolean REQUIRED: no CREATION DEFAULT: `true` DECLARED BY: `reaction`	Whether this current authored choice is enabled. False retains the record and its identity but excludes its active effect.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`targets`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements that are the subjects of the association.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.164 Repost

Identifier: `urn:redbook:type:circulation.repost` **Key:** `circulation.repost` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Circulation referencing existing source material as a repost without silently copying or changing it.

Parent: Circulation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Participant slot	Constraints	Meaning
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.165 Quotation

Identifier: `urn:redbook:type:circulation.quote` **Key:** `circulation.quote` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Circulation with explicitly authored quoted text and source references. The quotation is not a live substring binding.

Parent: Circulation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Participant slot	Constraints	Meaning
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.166 Forward

Identifier: `urn:redbook:type:circulation.forward` **Key:** `circulation.forward` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Circulation indicating forwarding of source material to an explicit destination when present; it is not network delivery.

Parent: Circulation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Participant slot	Constraints	Meaning
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.167 Cross-post

Identifier: `urn:redbook:type:circulation.crosspost` **Key:** `circulation.crosspost` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Circulation associating the same source material with another context without transferring source authorship.

Parent: Circulation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Participant slot	Constraints	Meaning
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.168 Derivation

Identifier: `urn:redbook:type:circulation.derivation` **Key:** `circulation.derivation` · **Primitive:** relationship · **Level:** specialization · **Area:** specializations

A Circulation recording an authored derivation and its sources. Reuse does not imply an automatic license grant.

Parent: Circulation. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.6. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`commentary`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	Authored commentary accompanying a circulation of source material.
`quotedText`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `circulation`	An explicitly retained quotation as authored text, not an automatically updated pointer into its source.

PARTICIPANT SLOTS

Participant slot	Constraints	Meaning
`actors`	ALLOWED TYPES: Digital MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The Digitals performing or expressing the stated act.
`sources`	ALLOWED TYPES: Any Element MINIMUM: 1 MAXIMUM: unbounded ORDERED: yes UNIQUE: yes	The existing Elements referred to as source material.

Participant slot	Constraints	Meaning
`destination`	ALLOWED TYPES: Any Element MINIMUM: 0 MAXIMUM: 1 ORDERED: yes UNIQUE: yes	The destination context, if one is supplied.

Allowed root types include their valid one-level specializations.

Open slot names: no. A slot name not listed above is invalid.

A.169 World Explorer

Identifier: `urn:redbook:type:viewbox.explorer` **Key:** `viewbox.explorer` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Viewbox presenting a general exploration of the world. Its reusable implementation is distinct from this authored Viewbox instance.

Parent: Viewbox. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 4. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`implementation`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `viewbox`	Package-root-relative entry module for the Viewbox implementation, resolved and activated under Part 4.
`configuration`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `viewbox`	Portable configuration validated against the selected implementation's declared contract; it does not contain host credentials.
`theme`	KIND: string REQUIRED: no CREATION DEFAULT: none DECLARED BY: `viewbox`	The identity of the selected applicable theme, not a URL to fetch automatically.
`plugins`	KIND: array REQUIRED: no CREATION DEFAULT: `[]` DECLARED BY: `viewbox`	Ordered identities or declared assignments for Viewbox-targeted plugins; installation and activation follow Part 4.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

A.170 Offer

Identifier: `urn:redbook:type:request.offer` **Key:** `request.offer` · **Primitive:** element · **Level:** specialization · **Area:** specializations

A Request framed as an offer. Accepting it follows the same explicit responsibility and Commitment rules as a Request.

Parent: Request. All parent constraints and domain rules apply. This specialization cannot itself be the parent of another type.

Domain rules: Part 3, clause 3.10. **Effective traits:** none.

PROPERTIES

Property	Declaration	Allowed values / semantics
`body`	KIND: object REQUIRED: no CREATION DEFAULT: `{}` DECLARED BY: `request`	A portable content map. The common text, format, references and attachment conventions are defined in 3.2; additional declared content-profile members are preserved.
`state`	KIND: string REQUIRED: no CREATION DEFAULT: `"open"` DECLARED BY: `request`	One of `"open"`, `"accepted"`, `"declined"`, `"withdrawn"`. The current domain state of this record. Permitted transitions and authorized actors are defined in its Part 3 domain clause; storing a state does not execute a transition.

PARTICIPANT SLOTS

None. The `slots` map is empty. Use Relationships or typed-reference Properties to associate this Element with other records.

B

ANNEX B

Annex B — Standard action contracts

The standard world-action catalogue, including inputs, results, preconditions, effects, omission rules and errors.

This annex is normative. It presents the contracts in the action catalogue. Every entry is subject to Part 4, the value and model rules in Part 1, and the relevant domain constraints in Part 3. The full input and output schemas below are part of each contract. Schema `$ref` identifiers resolve to the local schemas directory; no network access is needed.

The catalogue groups standard world and participant operations under `core-operations`, and compound, administrative and package-management operations under `authoring`. The required name sets are defined by the catalogue entries. An implementation must satisfy the applicable contracts to claim the corresponding capability. Effects are `read`, `write`, or host-session `external`; an externally activated facility is not an authored record.

Common failure contract. A failed read returns an explicit error without a partial successful result or authored mutation. Failure of any write precondition, guard, hook, result or final-state validation discards the entire candidate, resources and allocations; no success receipt is produced. Terminal failure retry receipts follow Part 4.10. External actions report their declared host outcome and cleanup diagnostics, with no undeclared irreversible effect. Entry-specific exceptions are stated below.

Action	Effect	Capability
rb.document.describe	read	core-operations
rb.document.rename	write	core-operations
rb.records.query	read	core-operations
rb.records.read	read	core-operations
rb.types.list	read	core-operations
rb.types.register	write	core-operations
rb.types.update	write	core-operations
rb.elements.create	write	core-operations
rb.relationships.create	write	core-operations
rb.records.edit	write	core-operations
rb.records.unset-properties	write	core-operations
rb.records.deletion-preview	read	core-operations
rb.records.delete	write	core-operations
rb.taxonomy.define	write	core-operations

Action	Effect	Capability
rb.taxonomy.assign	write	core-operations
rb.time.read	read	core-operations
rb.time.configure	write	core-operations
rb.time.advance	write	core-operations
rb.digitals.create	write	core-operations
rb.communication.start	write	core-operations
rb.conversations.create	write	core-operations
rb.conversations.send	write	core-operations
rb.conversations.messages	read	core-operations
rb.conversations.close	write	core-operations
rb.messages.edit	write	core-operations
rb.questions.accept-answer	write	core-operations
rb.discussions.split	write	core-operations
rb.walls.create	write	core-operations
rb.posts.create	write	core-operations
rb.posts.publish	write	core-operations
rb.walls.place	write	core-operations
rb.walls.entries	read	core-operations
rb.walls.review	write	core-operations
rb.mail.send	write	core-operations
rb.mail.read	read	core-operations
rb.mail.reply	write	core-operations

Action	Effect	Capability
rb.mail.inbox	read	core-operations
rb.mail.organize	write	core-operations
rb.mail.template.render	read	core-operations
rb.memberships.set	write	core-operations
rb.invitations.respond	write	core-operations
rb.reactions.set	write	core-operations
rb.annotations.create	write	core-operations
rb.comments.create	write	core-operations
rb.sharing.reference	write	core-operations
rb.collections.add	write	core-operations
rb.collections.entries	read	core-operations
rb.collections.reorder	write	core-operations
rb.feeds.read	read	core-operations
rb.attention.mark	write	core-operations
rb.bookmarks.set	write	core-operations
rb.notifications.send	write	core-operations
rb.subscriptions.set	write	core-operations
rb.newsletters.publish	write	core-operations
rb.newsletters.deliver	write	core-operations
rb.requests.create	write	core-operations
rb.requests.respond	write	core-operations
rb.commitments.progress	write	core-operations

Action	Effect	Capability
rb.polls.vote	write	core-operations
rb.polls.tally	read	core-operations
rb.polls.close	write	core-operations
rb.questionnaires.respond	write	core-operations
rb.events.respond	write	core-operations
rb.events.attendance	write	core-operations
rb.reviews.submit	write	core-operations
rb.decisions.respond	write	core-operations
rb.reservations.request	write	core-operations
rb.reservations.confirm	write	core-operations
rb.works.capture-edition	write	core-operations
rb.gatherings.contribute	write	core-operations
rb.moderation.report	write	core-operations
rb.streams.append	write	core-operations
rb.soma.neocortex	read	core-operations
rb.soma.create	write	core-operations
rb.soma.objects.create	write	core-operations
rb.soma.relationships.create	write	core-operations
rb.soma.read	read	core-operations
rb.soma.represent	write	core-operations
rb.extensions.describe	read	core-operations
rb.viewboxes.configure	write	core-operations

Action	Effect	Capability
rb.presentation.select	write	core-operations
rb.blueprints.capture-graph	read	core-operations
rb.blueprints.instantiate-graph	write	core-operations
rb.transaction.perform	write	authoring
rb.document.configure	write	authoring
rb.records.retype	write	authoring
rb.records.attribute	write	authoring
rb.types.remove	write	authoring
rb.taxonomy.update	write	authoring
rb.taxonomy.remove	write	authoring
rb.assets.list	read	authoring
rb.assets.receive	write	authoring
rb.assets.remove	write	authoring
rb.extensions.install	write	authoring
rb.extensions.configure	write	authoring
rb.extensions.activate	external	authoring
rb.extensions.deactivate	external	authoring
rb.extensions.remove	write	authoring
rb.resources.read	read	authoring
rb.content.edit	write	core-operations
rb.content.unset-properties	write	core-operations
rb.content.classify	write	core-operations

Action	Effect	Capability
rb.content.delete	write	core-operations
rb.posts.edit	write	core-operations
rb.mail.move	write	core-operations
rb.domain.transition	write	core-operations
rb.newsletters.initialize-numbering	write	authoring
rb.events.occurrences	read	core-operations
rb.rules.evaluate	read	core-operations
rb.rules.process	write	core-operations
rb.mail.folders.create	write	core-operations
rb.content.create	write	core-operations
rb.connections.respond	write	core-operations

B.1 rb.document.describe

Read world identity, available definitions, presentation and time without recording activity.

Capability: `core-operations`. **Domain profile:** `core.document`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. A valid world is selected.

Effects. Return identity, version, time, presentation and structural counts without mutation.

Result. Document metadata and structural counts.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {},
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "formatVersion": {
      "const": "1.0"
    },
    "time": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/time-rule.schema.json"
    },
    "counts": {
      "type": "object",
      "properties": {
        "valid": {
```

```
          "const": true
        },
        "elements": {
          "type": "integer",
          "minimum": 0,
          "maximum": 9007199254740991
        },
        "relationships": {
          "type": "integer",
          "minimum": 0,
          "maximum": 9007199254740991
        },
        "types": {
          "type": "integer",
          "minimum": 0,
          "maximum": 9007199254740991
        }
      },
      "required": [
        "valid",
        "elements",
        "relationships",
        "types"
      ],
      "additionalProperties": false
    },
    "presentation": {
      "type": "object",
      "properties": {
        "defaultViewbox": {
          "oneOf": [
            {
              "type": "string",
              "minLength": 1
            },
            {
              "type": "null"
            }
          ]
        },
        "theme": {
          "type": "string",
          "minLength": 1
        },
        "language": {
          "type": "string",
          "minLength": 1
        }
      },
      "required": [
        "defaultViewbox"
      ],
      "additionalProperties": true
    }
  },
  "required": [
    "id",
    "title",
    "formatVersion",
```

```
    "time",
    "counts",
    "presentation"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.2 rb.document.rename

Change the current document title without retaining its previous value.

Capability: `core-operations`. **Domain profile:** `core.document`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. RedBook administration; title is a string.

Effects. Replace only the current document title; retain identity and chronology.

Result. Object containing the resulting title.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "title": {
      "type": "string"
    }
  },
  "required": [
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "title": {
      "type": "string"
    }
  },
  "required": [
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.3 rb.records.query

Select current authored records; reads do not mark attention.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Query uses declared keys and valid paging.

Effects. Filter the complete base enumeration (Elements then Relationships) using exact primitive/type predicates and the Unicode 15.1 full-default-fold searchable token algorithm in Part 4.4; apply offset then limit. No locale-dependent scalar conversion, sorting or implicit attention.

Result. An ordered list of matching record snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`type`	No type restriction.
`primitive`	Both primitives.
`text`	`""`
`limit`	`100`
`offset`	`0`

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "minLength": 1
    },
    "primitive": {
      "enum": [
        "element",
        "relationship"
      ]
    },
    "text": {
```

```
      "type": "string"
    },
    "limit": {
      "type": "integer",
      "minimum": 1,
      "maximum": 1000
    },
    "offset": {
      "type": "integer",
      "minimum": 0,
      "maximum": 9007199254740991
    }
  },
  "required": [],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.4 rb.records.read

Read an authored record without altering the world.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. The record identity resolves locally.

Effects. Read one record without changing it or creating defaults.

Result. The selected record snapshot.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.5 rb.types.list

Read carried CET/CRT definitions and specializations.

Capability: `core-operations`. **Domain profile:** `core.types`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. A world is selected.

Effects. Return carried declarations including unused supplied definitions.

Result. All current type definitions in declaration order.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {},
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/type.schema.json"
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.6 rb.types.register

Author a new root type or one further specialization.

Capability: `core-operations`. **Domain profile:** `core.types`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; fresh type identity; valid root or specialization.

Effects. Add the complete definition, validate satisfiable inheritance and existing references atomically.

Result. The registered type definition.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "definition": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/type.schema.json"
    }
  },
  "required": [
    "definition"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/type.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.7 rb.types.update

Deliberately replace a carried type definition while retaining its identity, primitive and parent. Existing records must still validate.

Capability: `core-operations`. **Domain profile:** `core.types`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; identity, key, primitive and parent remain stable; supplied meaning is retained for a core claim.

Effects. Replace the definition only if all affected records and inherited constraints remain valid.

Result. The updated type definition.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "definition": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/type.schema.json"
    }
  },
  "required": [
    "definition"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/type.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.8 rb.elements.create

Create a typed Element; Digital/context Neocortexes are formed in the same transaction.

Capability: `core-operations`. **Domain profile:** `core.elements`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; declared CET; supplied Properties satisfy the effective contract. Poll option and Questionnaire question definitions, including inherited core specializations, must satisfy Part 3.9 and domain-response.schema.json before allocation; exactly $rb is reserved in those local ID fields.

Effects. Create one Element with materialized creation defaults; initialize mandatory Neocortex dependencies atomically.

Result. The new Element; dependent records are accessible by their relationships.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`title`	`""`
`properties`	`{}`

Input schema.

```
{
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "type"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "allOf": [
    {
```

```
      "if": {
        "properties": {
          "type": {
            "enum": [
              "poll",
              "urn:redbook:type:poll"
            ]
          }
        },
        "required": [
          "type"
        ]
      },
      "then": {
        "properties": {
          "properties": {
            "type": "object",
            "properties": {
              "options": {
                "$ref": "https://redbook.fun/specification/1.0/schemas/domain-
response.schema.json#/$defs/options"
              }
            }
          }
        }
      }
    },
    {
      "if": {
        "properties": {
          "type": {
            "enum": [
              "questionnaire",
              "urn:redbook:type:questionnaire"
            ]
          }
        },
        "required": [
          "type"
        ]
      },
      "then": {
        "properties": {
          "properties": {
            "type": "object",
            "properties": {
              "questions": {
                "$ref": "https://redbook.fun/specification/1.0/schemas/domain-
response.schema.json#/$defs/questions"
              }
            }
          }
        }
      }
    }
  ]
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.9 rb.relationships.create

Create one typed, potentially multi-participant Relationship.

Capability: `core-operations`. **Domain profile:** `core.relationships`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; declared CRT; slot participants and Properties are valid.

Effects. Create an independently identified Relationship; no universal binary-edge restriction.

Result. The new Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`title`	`""`
`properties`	`{}`

Input schema.

```
{
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "slots": {
      "type": "object",
      "additionalProperties": {
        "type": "array",
        "items": {
          "type": "string",
          "minLength": 1
        }
      }
    }
  },
  "required": [
```

```
    "type",
    "slots"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.10 rb.records.edit

Edit current content in place. No revision snapshot is retained.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; record exists; patch names only title, properties, categories, tags or slots. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. Replace supplied simple members; shallow-merge supplied Property keys. Validate final invariants; no revision copy.

Result. The changed record snapshot.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "patch": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record-patch.schema.json"
    }
  },
  "required": [
    "id",
    "patch"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.11 rb.records.unset-properties

Remove named property keys, distinct from assigning null.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; record exists; removing keys preserves required fields.

Effects. Remove the explicitly named Property keys; assigning null is a different operation.

Result. The changed record snapshot.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "keys": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
    }
  },
  "required": [
    "id",
    "keys"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.12 rb.records.deletion-preview

Read the complete dependency-aware deletion plan without changing the world.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Every selected identity exists.

Effects. Compute dependent relationships, typed references, ownership implications and retained credits; no mutation.

Result. Deletion plan with identities and blockers.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "ids": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "ids"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "ids": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
```

```
    },
    "relationships": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "propertyReferrers": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "attributionReferrers": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "ownershipBlockers": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "resourceDependencies": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "ids",
    "relationships",
    "propertyReferrers",
    "attributionReferrers",
    "ownershipBlockers",
    "resourceDependencies"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.13 rb.records.delete

Permanently remove an explicit selection. Optional relationship removal is reported; no hidden edit archive.

Capability: `core-operations`. **Domain profile:** `core.records`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; all required references resolved explicitly; relationship deletion selected when needed.

Effects. Delete selected records and explicitly permitted dependent links; detach surviving author credits; validate atomically.

Result. Removed identities and records whose author links became credits.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`removeRelationships`	`false`

Input schema.

```
{
  "type": "object",
  "properties": {
    "ids": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "removeRelationships": {
      "type": "boolean"
    }
  },
  "required": [
    "ids"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "removed": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "retainedAuthorCredits": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "removed",
    "retainedAuthorCredits"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.14 rb.taxonomy.define

Create a Category or Tag definition shared by both primitives.

Capability: `core-operations`. **Domain profile:** `core.taxonomy`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; valid term kind; optional Category parent resolves.

Effects. Create a Category or Tag with a fresh identity and given label.

Result. The created taxonomy term.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`parent`	Category root; Tags reject a supplied parent.

Input schema.

```
{
  "type": "object",
  "properties": {
    "kind": {
      "enum": [
        "category",
        "tag"
      ]
    },
    "label": {
      "type": "string",
      "minLength": 1
    },
    "parent": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "kind",
    "label"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "oneOf": [
    {
      "type": "object",
      "properties": {
        "id": {
          "type": "string",
          "minLength": 1
        },
        "label": {
          "type": "string"
        },
        "parent": {
          "oneOf": [
            {
              "type": "string",
              "minLength": 1
            },
            {
              "type": "null"
            }
          ]
        }
      },
      "required": [
        "id",
        "label",
        "parent"
      ],
      "additionalProperties": false
    },
    {
      "type": "object",
      "properties": {
        "id": {
          "type": "string",
          "minLength": 1
        },
        "label": {
          "type": "string"
        }
      },
      "required": [
        "id",
        "label"
      ],
      "additionalProperties": false
    }
  ],
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.15 rb.taxonomy.assign

Set the current Categories and/or Tags of either primitive.

Capability: `core-operations`. **Domain profile:** `core.taxonomy`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; target record and all assigned terms exist.

Effects. Replace supplied assignment lists; preserve omitted lists; no duplicate assignment.

Result. The changed record snapshot.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`categories`	Leave existing Categories unchanged.
`tags`	Leave existing Tags unchanged.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
```

```
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.16 rb.time.read

Evaluate the optional central time rule. No activity or scheduling is manufactured.

Capability: `core-operations`. **Domain profile:** `core.time`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Valid central rule; custom provider available when required.

Effects. Compute a reading from explicit recordedAt or host time without recording activity.

Result. The rule and its computed portable worldTime.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`recordedAt`	Current host recording instant; unlike other read actions this explicit clock request may consult the host clock.

Input schema.

```
{
  "type": "object",
  "properties": {
    "recordedAt": {
      "type": "string",
      "pattern": "^[0-9]{4}-(?:0[1-9]|1[0-2])-(?:0[1-9]|[12][0-9]|3[01])T(?:[01][0-9]|2[0-3]):
[0-5][0-9]:[0-5][0-9]\\.[0-9]{3}Z(?![\\s\\S])"
    }
  },
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "rule": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/time-rule.schema.json"
    },
    "worldTime": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json"
    }
```

```
  },
  "required": [
    "rule",
    "worldTime"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.17 rb.time.configure

Replace the central time rule without redating existing authored events.

Capability: `core-operations`. **Domain profile:** `core.time`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; new rule satisfies its mode contract.

Effects. Replace the central rule; do not recalculate old event observations.

Result. The installed rule.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "rule": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/time-rule.schema.json"
    }
  },
  "required": [
    "rule"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/time-rule.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.18 rb.time.advance

Explicitly advance a manually stepped world clock.

Capability: `core-operations`. **Domain profile:** `core.time`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; central mode is manual; increment is exact signed milliseconds.

Effects. Add the increment exactly to currentMilliseconds; do not create events or execute jobs.

Result. The updated manual rule.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "milliseconds": {
      "type": "string",
      "pattern": "^(?:0|-?[1-9][0-9]*)(?![\\s\\S])"
    }
  },
  "required": [
    "milliseconds"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/time-rule.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.19 rb.digitals.create

Create an ordinary Digital and exactly one personal Soma Neocortex; no automatic Wall, Mailbox or AI.

Capability: `core-operations`. **Domain profile:** `core.digitals`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; type is Digital or its specialization; omitted type means root Digital.

Effects. Create the Digital, one personal Neocortex and their ownership Relationship atomically.

Result. The new Digital.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`type`	digital
`properties`	`{}`

Input schema.

```
{
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "minLength": 1,
      "default": "digital"
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.20 rb.communication.start

Create an explicitly selected communication context with its shared Neocortex and memberships.

Capability: `core-operations`. **Domain profile:** `core.communication`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor exists; declared type has shared-neocortex trait; participants are valid Digitals. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields. Review creation requires nonempty accessible targetIds; every other selected root rejects targetIds. Poll option and Questionnaire question definitions, including inherited core specializations, must satisfy Part 3.9 and domain-response.schema.json before allocation; exactly $rb is reserved in those local ID fields.

Effects. Create context, one shared Neocortex and unique explicit active memberships. Duplicate supplied participants are coalesced without changing their first-occurrence order. Preserve supplied valid state or the selected type's declared state default (for example Event planned and Gathering open); never force every context to open. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types. Review creation also creates its Review Target association atomically, using targetIds in supplied order.

Result. The new communication context.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": null,
  "typeSelection": "shared-neocortex",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`participants`	`[]`

Argument	Meaning when absent
`categories`	Empty on creation.
`tags`	Empty on creation.
`targetIds`	Required and nonempty for Review creation; forbidden for all other selected roots.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "participants": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "targetIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
```

```
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "required": [
    "type",
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "allOf": [
    {
      "if": {
        "properties": {
          "type": {
            "enum": [
              "poll",
              "urn:redbook:type:poll"
            ]
          }
        },
        "required": [
          "type"
        ]
      },
      "then": {
        "properties": {
          "properties": {
            "type": "object",
            "properties": {
              "options": {
                "$ref": "https://redbook.fun/specification/1.0/schemas/domain-
response.schema.json#/$defs/options"
              }
            }
          }
        }
      }
    },
    {
      "if": {
        "properties": {
          "type": {
            "enum": [
              "questionnaire",
              "urn:redbook:type:questionnaire"
            ]
          }
        },
        "required": [
          "type"
        ]
      },
      "then": {
        "properties": {
          "properties": {
            "type": "object",
            "properties": {
```

```
              "questions": {
                "$ref": "https://redbook.fun/specification/1.0/schemas/domain-
response.schema.json#/$defs/questions"
              }
            }
          }
        }
      }
    }
  ]
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.21 rb.conversations.create

Start a conversation with explicit participants and a shared conceptual home.

Capability: `core-operations`. **Domain profile:** `core.conversations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor exists; type is Conversation or specialization; participants resolve. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create Conversation, shared Neocortex, owner link and participant memberships atomically. Duplicate supplied participants are coalesced without changing their first-occurrence order. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Conversation.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "conversation",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	conversation
`properties`	`{}`
`participants`	`[]`
`categories`	Empty on creation.
`tags`	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "participants": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.22 rb.conversations.send

Contribute a message, question, answer or summary as the selected Digital.

Capability: `core-operations`. **Domain profile:** `core.conversations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Open Conversation; active participant; optional reply and mentions resolve in correct scope. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create sent Message, recording fields, Contribution, optional Reply and Mention links. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Message.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "message",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	message
`title`	`""`
`replyTo`	No reply Relationship is created.
`mentions`	`[]`
`properties`	`{}`

Argument	Meaning when absent
categories	Empty on creation.
tags	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "conversationId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "replyTo": {
      "type": "string",
      "minLength": 1
    },
    "mentions": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
```

```
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "conversationId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.23 rb.conversations.messages

Read messages in authored sequence, without attention effects.

Capability: `core-operations`. **Domain profile:** `core.conversations`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Conversation exists.

Effects. Return its contributions in exact local event order without modifying attention or chronology.

Result. The ordered contribution snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "conversationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "conversationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.24 rb.conversations.close

Explicitly close a conversation as its author or administrator.

Capability: `core-operations`. **Domain profile:** `core.conversations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor is attributed context author, or administrator.

Effects. Set state to closed or open explicitly; preserve all content and conceptual homes.

Result. The updated Conversation.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "closed": {
      "type": "boolean"
    }
  },
  "required": [
    "id",
    "closed"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.25 rb.messages.edit

Correct a current message without keeping its prior body.

Capability: `core-operations`. **Domain profile:** `core.messages`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Message exists; selected actor is its author, or administrator.

Effects. Replace current body without adding an event, edition or edit-history record.

Result. The updated Message.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "id",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.26 rb.questions.accept-answer

Select an answer explicitly; clearing or changing it does not claim a truth score.

Capability: `core-operations`. **Domain profile:** `core.questions`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Question and Answer are Message instances; selected actor is question author or administrator.

Effects. With non-null answerId, set or replace the Accepted Answer relationship after verifying the selected question and answer. With explicit null, remove only the Accepted Answer relationships for that question; retain question, answer, their observations and credits. Repeated clear is an authorized no-op.

Result. Selected Accepted Answer relationship, or null when cleared.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "questionId": {
      "type": "string",
      "minLength": 1
    },
    "answerId": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
          "type": "null"
        }
```

```
      ]
    }
  },
  "required": [
    "questionId",
    "answerId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "oneOf": [
    {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    {
      "type": "null"
    }
  ]
}
```

B.27 rb.discussions.split

Create a new conversation and move an explicitly selected set of contributions into it.

Capability: `core-operations`. **Domain profile:** `core.discussions`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Context author or administrator; all selected messages belong to source; reply scope can remain valid.

Effects. Create a new Thread with participant memberships and empty shared Neocortex; move selected Contribution links atomically.

Result. The new Thread; source conceptual content is not silently copied.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "conversationId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "messageIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
```

```
      "uniqueItems": true
    }
  },
  "required": [
    "conversationId",
    "title",
    "messageIds"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.28 rb.walls.create

Create a Wall or specialization with an explicit contribution policy.

Capability: `core-operations`. **Domain profile:** `core.walls`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor exists; type is Wall or specialization; policy is valid. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Wall with exactly one shared Neocortex; no Post is implied. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Wall.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "wall",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	wall
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.29 rb.posts.create

Author a Post or specialization as an ordinary contribution.

Capability: `core-operations`. **Domain profile:** `core.posts`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor exists; type is Post or specialization; body is portable. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create one attributed event in draft by default, or published when explicitly requested. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Post.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "post",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`title`	`""`
`type`	post
`publish`	`false`
`properties`	`{}`
`categories`	Empty on creation.

Argument	Meaning when absent
tags	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "publish": {
      "type": "boolean"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.30 rb.posts.publish

Make the current authored contribution available as published, without making a hidden edition.

Capability: `core-operations`. **Domain profile:** `core.posts`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Post exists; selected actor is its author or administrator.

Effects. Set current state to published without cloning or rewriting its original event fields.

Result. The published Post.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.31 rb.walls.place

Place published content on a Wall; content and placement have distinct identities.

Capability: `core-operations`. **Domain profile:** `core.walls`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Wall and content exist; Post is published; actor satisfies Wall policy. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create a distinct placement with pin, position and published/pending state; retain content authorship. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Wall Placement Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "wall-placement",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`position`	Use the item's event sequence if exactly representable as a safe ordinary integer; otherwise append after the greatest existing placement position using a representable value. If no such value exists, reject with ORDER_RANGE.
`pinned`	`false`
`properties`	`{}`
`categories`	Empty on creation.

Argument	Meaning when absent
tags	Empty on creation.
type	wall-placement

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "wallId": {
      "type": "string",
      "minLength": 1
    },
    "itemId": {
      "type": "string",
      "minLength": 1
    },
    "position": {
      "type": "number"
    },
    "pinned": {
      "type": "boolean"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "wallId",
```

```
    "itemId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.32 rb.walls.entries

Read placed entries with pins and authored order, without duplicating source content.

Capability: `core-operations`. **Domain profile:** `core.walls`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Wall exists.

Effects. Select published placements, or include pending when explicitly requested, in standard Wall order.

Result. Ordered objects with placement and referenced item snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`includePending`	`false`

Input schema.

```
{
  "type": "object",
  "properties": {
    "wallId": {
      "type": "string",
      "minLength": 1
    },
    "includePending": {
      "type": "boolean"
    }
  },
  "required": [
    "wallId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
```

```
      "placement": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      },
      "items": {
        "type": "array",
        "items": {
          "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
        }
      }
    },
    "required": [
      "placement",
      "items"
    ],
    "additionalProperties": false
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.33 rb.walls.review

Approve or reject a Wall placement as the Wall author or administrator.

Capability: `core-operations`. **Domain profile:** `core.walls`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Placement exists; selected actor is Wall author or administrator.

Effects. Set placement state to published or rejected; leave source content unchanged.

Result. The changed placement.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "placementId": {
      "type": "string",
      "minLength": 1
    },
    "accept": {
      "type": "boolean"
    }
  },
  "required": [
    "placementId",
    "accept"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.34 rb.mail.send

Dispatch internal correspondence and create independent recipient delivery and mailbox state.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor exists; nonempty recipients; no duplicates across To/Cc/Bcc. With an antecedent and threadMode=continue, its typed threadId and unique matching Contribution resolve an open mail-thread context; ambiguity fails THREAD_UNRESOLVED and closure fails STATE. Explicit new bypasses reuse, not source visibility checks. Reusing an unchanged-audience thread additionally requires active sender Membership; otherwise PARTICIPATION. Explicit new does not rejoin the old thread. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create sent Mail Message, correspondence context/shared home, deliveries and recipient inbox state atomically. The public audience is sender plus To/Cc in first-occurrence order; Bcc receives Delivery and Mailbox Entry only, not automatic thread Membership. Continue reuses the antecedent thread iff its stored mailParticipants set equals the new public audience; otherwise create a new mail-thread and shared Neocortex. A new Message stores typed threadId and has exactly one matching Contribution. No implicit historic audience or shared-home grant is added to a Bcc recipient. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. New Mail Message snapshot and conversationId of the deterministically selected existing or new correspondence context.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`, `THREAD_UNRESOLVED`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "mail-message",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`subject`	`""`

Argument	Meaning when absent
`cc`	`[]`
`bcc`	`[]`
`replyTo`	No reply Relationship is created.
`threadMode`	continue when there is an antecedent; new when there is none.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	mail-message

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "subject": {
      "type": "string"
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "to": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "cc": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
```

```
    },
    "bcc": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "replyTo": {
      "type": "string",
      "minLength": 1
    },
    "threadMode": {
      "enum": [
        "continue",
        "new"
      ]
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "body",
    "to"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "message": {
```

```
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "conversationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "message",
    "conversationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.35 rb.mail.read

Read an addressed message. Digital perspective omits other Bcc recipients; the file holder retains full access.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Message exists; Digital is sender or recipient unless administrator.

Effects. Project envelope under Bcc rules; no read-state mutation.

Result. Message and visible Delivery snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "messageId": {
      "type": "string",
      "minLength": 1
    },
    "actorId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "messageId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "message": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "deliveries": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    }
  },
  "required": [
    "message",
    "deliveries"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.36 rb.mail.reply

Compose a new internal message using the visible envelope for reply or reply-all.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Source correspondence is visible to actor; reply has a valid nonempty derived recipient set. With an antecedent and threadMode=continue, its typed threadId and unique matching Contribution resolve an open mail-thread context; ambiguity fails THREAD_UNRESOLVED and closure fails STATE. Explicit new bypasses reuse, not source visibility checks. Reusing an unchanged-audience thread additionally requires active sender Membership; otherwise PARTICIPATION. Explicit new does not rejoin the old thread. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Derive reply/reply-all recipients from visible envelope; perform internal send; link reply to source. The public audience is sender plus To/Cc in first-occurrence order; Bcc receives Delivery and Mailbox Entry only, not automatic thread Membership. Continue reuses the antecedent thread iff its stored mailParticipants set equals the new public audience; otherwise create a new mail-thread and shared Neocortex. A new Message stores typed threadId and has exactly one matching Contribution. No implicit historic audience or shared-home grant is added to a Bcc recipient. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. New Mail Message snapshot and conversationId of the deterministically selected existing or new correspondence context.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`, `THREAD_UNRESOLVED`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "mail-message",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`replyAll`	`false`

Argument	Meaning when absent
`threadMode`	continue when there is an antecedent; new when there is none.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	mail-message

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "messageId": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "replyAll": {
      "type": "boolean"
    },
    "threadMode": {
      "enum": [
        "continue",
        "new"
      ]
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
```

```
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "messageId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "message": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "conversationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "message",
    "conversationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.37 rb.mail.inbox

Read a Digital's delivered messages and independent organization without setting read flags.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected Digital exists.

Effects. Return its current Mailbox Entry and Message pairs without setting read flags.

Result. Ordered entry/message snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "entry": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      },
      "message": {
```

```
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    },
    "required": [
      "entry",
      "message"
    ],
    "additionalProperties": false
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.38 rb.mail.organize

Set this recipient’s labels, read, acknowledgment, star or snooze values without changing another recipient.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Mailbox Entry belongs to selected recipient; patch keys are labels, read, acknowledged, starred or snoozedUntil.

Effects. Update that entry’s current organization only; validate declared property types.

Result. The changed Mailbox Entry Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "entryId": {
      "type": "string",
      "minLength": 1
    },
    "patch": {
      "type": "object",
      "properties": {
        "labels": {
          "type": "array",
          "items": {
            "type": "string"
          },
          "uniqueItems": true
        },
        "read": {
```

```
          "type": "boolean"
        },
        "acknowledged": {
          "type": "boolean"
        },
        "starred": {
          "type": "boolean"
        },
        "snoozedUntil": {
          "type": "string"
        }
      },
      "required": [],
      "additionalProperties": false
    }
  },
  "required": [
    "entryId",
    "patch"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.39 rb.mail.template.render

Render declared template variables without sending correspondence.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Template exists; every referenced substitution has a supplied string value.

Effects. Substitute declared placeholders without sending or mutating anything.

Result. Rendered subject and text.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "templateId": {
      "type": "string",
      "minLength": 1
    },
    "values": {
      "type": "object",
      "additionalProperties": {
        "type": "string"
      }
    }
  },
  "required": [
    "templateId",
    "values"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "subject": {
      "type": "string"
    },
    "text": {
      "type": "string"
```

```
    }
  },
  "required": [
    "subject",
    "text"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.40 rb.memberships.set

Explicitly set one Digital's membership; this represents participation, not host authentication.

Capability: `core-operations`. **Domain profile:** `core.memberships`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Context and target Digital exist. Administrator or attributed context author may manage others. A Digital may change its own existing membership only as specified in 4.5, without increasing its role or bypassing an invitation or closed enrollment.

Effects. Upsert a single-member Membership with explicit state and role; no inference of host access. When a relationship includes other members, split or reduce it without changing their membership; reject if typed references prevent a valid split.

Result. The resulting Membership Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`role`	Preserve an existing role; use member only when creating a permitted new membership.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "contextId": {
      "type": "string",
      "minLength": 1
    },
    "digitalId": {
      "type": "string",
      "minLength": 1
    },
    "state": {
```

```
      "enum": [
        "active",
        "invited",
        "requested",
        "declined",
        "left"
      ]
    },
    "role": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "contextId",
    "digitalId",
    "state"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.41 rb.invitations.respond

Explicitly accept or decline your invitation and record corresponding membership.

Capability: `core-operations`. **Domain profile:** `core.invitations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Invitation identifies this Digital as its single invitee and is pending. A multi-invitee invitation must be explicitly split before this singleton action.

Effects. Record accepted/declined state; acceptance explicitly activates membership; no AI wake-up.

Result. The changed Invitation Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "invitationId": {
      "type": "string",
      "minLength": 1
    },
    "accept": {
      "type": "boolean"
    }
  },
  "required": [
    "invitationId",
    "accept"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.42 rb.reactions.set

Set or update an attributed reaction without inferring agreement, reading or responsibility.

Capability: `core-operations`. **Domain profile:** `core.reactions`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Target and actor exist; type is Reaction or specialization. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert the actor/type/target current reaction with value, criteria and enabled state. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The resulting Reaction Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "reaction",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	reaction
`criteria`	`""`
`enabled`	`true`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.

Argument	Meaning when absent
`tags`	Empty on creation; preserve existing assignments on an upsert.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "value": {
      "type": "string"
    },
    "criteria": {
      "type": "string"
    },
    "enabled": {
      "type": "boolean"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetId",
    "value"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.43 rb.annotations.create

Annotate an Element with an explicit selector and a separately authored body.

Capability: `core-operations`. **Domain profile:** `core.annotations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Target exists; selector has a declared supported interpretation. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create an attributed Annotation event and Annotation Target link without altering its subject. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Annotation.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "annotation",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`selector`	`{"kind": "whole"}`
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	annotation

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "selector": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.44 rb.comments.create

Add a comment to an existing Element without changing its content.

Capability: `core-operations`. **Domain profile:** `core.comments`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Target and actor exist. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create an attributed Comment event and a Reply link to its subject. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Comment.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "comment",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	comment

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.


```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.45 rb.sharing.reference

Create an attributed repost, quotation, forward or other circulation relation.

Capability: `core-operations`. **Domain profile:** `core.sharing`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor and sources exist; type is Circulation or specialization; optional destination resolves. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed circulation with source references, commentary and explicit quote text; do not duplicate originals implicitly. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Circulation Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "circulation",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	circulation
`destinationId`	No destination slot.
`commentary`	`""`
`quotedText`	`""`
`properties`	`{}`

Argument	Meaning when absent
`categories`	Empty on creation.
`tags`	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "sourceIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "destinationId": {
      "type": "string",
      "minLength": 1
    },
    "commentary": {
      "type": "string"
    },
    "quotedText": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
```

```
  },
  "required": [
    "sourceIds"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.46 rb.collections.add

Add an ordered reference and note to a Collection or Agenda without copying the source.

Capability: `core-operations`. **Domain profile:** `core.collections`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Collection or Agenda and selected items exist; position is finite. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create ordered Collection Entry with note; preserve item identities. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Collection Entry Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "collection-entry",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`position`	Append after the greatest existing collection position, or use zero for an empty collection; reject ORDER_RANGE if no representable append position is available.
`note`	`""`
`properties`	`{}`
`categories`	Empty on creation.

Argument	Meaning when absent
tags	Empty on creation.
type	collection-entry

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "collectionId": {
      "type": "string",
      "minLength": 1
    },
    "itemIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "position": {
      "type": "number"
    },
    "note": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
```

```
    }
  },
  "required": [
    "collectionId",
    "itemIds"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.47 rb.collections.entries

Read a collection’s current authored arrangement.

Capability: `core-operations`. **Domain profile:** `core.collections`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Collection/Agenda exists.

Effects. Return ordered entries and referenced items without copying them into new authored records.

Result. Entry/item snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "collectionId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "collectionId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "entry": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      },
      "items": {
        "type": "array",
        "items": {
          "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
        }
```

```
      }
    },
    "required": [
      "entry",
      "items"
    ],
    "additionalProperties": false
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.48 rb.collections.reorder

Reorder all current entries with an explicit identity list.

Capability: `core-operations`. **Domain profile:** `core.collections`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Author or administrator; each current entry ID is supplied once.

Effects. Assign consecutive local positions in requested order; reject missing/repeated entries atomically.

Result. The resulting ordered entry identities.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "collectionId": {
      "type": "string",
      "minLength": 1
    },
    "entryIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "collectionId",
    "entryIds"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "order": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "order"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.49 rb.feeds.read

Evaluate a stored feed query without duplicating records or recording attention.

Capability: `core-operations`. **Domain profile:** `core.feeds`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Feed exists and query is supported.

Effects. Filter the complete base enumeration using the standard query, apply the declared Feed ordering to all matches, then offset and limit; title ties use exact identity and chronology uses exact sequences under Part 3.23. No paging before sorting and no authored changes.

Result. Ordered current result snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "feedId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "feedId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.50 rb.attention.mark

Record deliberate focus, read, acknowledgment or dismissal as an authored relationship.

Capability: `core-operations`. **Domain profile:** `core.attention`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected Digital and target exist; attention kind is explicit. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert that Digital/target/kind Attention and stamp this deliberate report; no hidden earlier-state archive. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The current stamped Attention Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "attention",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`position`	`{}`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.
`type`	Existing target type on update; otherwise attention

Input schema.


```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "kind": {
      "type": "string",
      "minLength": 1
    },
    "position": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetId",
    "kind"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.51 rb.bookmarks.set

Save or remove a personal reference and note.

Capability: `core-operations`. **Domain profile:** `core.bookmarks`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected Digital and target exist. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert current bookmark note/enabled state without implying reading or acknowledgment. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Bookmark Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "bookmark",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`note`	`""`
`enabled`	`true`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.

Argument	Meaning when absent
`type`	Existing target type on update; otherwise bookmark

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "note": {
      "type": "string"
    },
    "enabled": {
      "type": "boolean"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.52 rb.notifications.send

Author an addressed notice, without network delivery, scheduling or waking an AI.

Capability: `core-operations`. **Domain profile:** `core.notifications`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor, recipients and optional sources exist. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create an attributed Notification event and Notice Target link; no external delivery or wake-up. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Notification.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "notification",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`sourceIds`	`[]`
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	notification

Input schema.


```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "recipients": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "sourceIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "recipients",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.53 rb.subscriptions.set

Set an authored subscription; it does not start a background task.

Capability: `core-operations`. **Domain profile:** `core.subscriptions`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Subscriber and source exist. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert current enabled and filter settings; no scheduler starts. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Subscription Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "subscription",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`enabled`	`true`
`filters`	`{}`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.

Argument	Meaning when absent
`type`	Existing target type on update; otherwise subscription

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "sourceId": {
      "type": "string",
      "minLength": 1
    },
    "enabled": {
      "type": "boolean"
    },
    "filters": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "sourceId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.54 rb.newsletters.publish

Publish a prepared issue in its newsletter; does not automatically deliver it.

Capability: `core-operations`. **Domain profile:** `core.newsletters`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Newsletter author or administrator; body valid; present nextIssueNumber is canonical positive text and exceeds retained ordinals. If absent, publication may initialize to 1 only for a series with no retained numbered issues; otherwise ALLOCATION_UNINITIALIZED. Allocated issueNumber must be at most 9007199254740991. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Allocate the Newsletter frontier as the new Issue issueNumber, increment nextIssueNumber exactly, create the published attributed Issue event and Newsletter Entry in one transaction. Deleting issues never lowers the frontier. Failure rolls all counters and records back. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Newsletter Issue.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `ALLOCATION_UNINITIALIZED`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `NUMBER_RANGE`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "newsletter-issue",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.

Argument	Meaning when absent
tags	Empty on creation.
type	newsletter-issue

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "newsletterId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "newsletterId",
    "title",
    "body"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.55 rb.newsletters.deliver

Explicitly create one delivery per current subscriber, without external mail or automatic scheduling.

Capability: `core-operations`. **Domain profile:** `core.newsletters`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Newsletter author or administrator; Issue belongs to Newsletter and is published.

Effects. Create missing deliveries and inbox entries for selected enabled subscribers; do not redeliver existing pairs.

Result. Number of newly created deliveries.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "newsletterId": {
      "type": "string",
      "minLength": 1
    },
    "issueId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "newsletterId",
    "issueId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "delivered": {
      "type": "integer",
      "minimum": 0,
      "maximum": 9007199254740991
    }
  },
  "required": [
    "delivered"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.56 rb.requests.create

Address a request or offer to explicit Digitals.

Capability: `core-operations`. **Domain profile:** `core.requests`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor and responders exist; type is Request or specialization. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create an attributed Request event and Request Address Relationship. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Request or Offer.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "request",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`type`	request
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "responders": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "title",
    "body",
    "responders"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.57 rb.requests.respond

Accept responsibility as a distinct Commitment, or decline the request.

Capability: `core-operations`. **Domain profile:** `core.requests`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Request is open; selected actor is a designated responder.

Effects. Decline without commitment, or accept and create attributed Commitment/responsibility atomically. A decline returns commitment=null; acceptance may not create a second commitment for the same accepted request/responder pair.

Result. Request and optional new Commitment snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "requestId": {
      "type": "string",
      "minLength": 1
    },
    "accept": {
      "type": "boolean"
    }
  },
  "required": [
    "requestId",
    "accept"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "request": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "commitment": {
      "oneOf": [
        {
          "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
        },
        {
          "type": "null"
        }
      ]
    }
  },
  "required": [
    "request",
    "commitment"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.58 rb.commitments.progress

Record an update, reported completion or independent explicit verification.

Capability: `core-operations`. **Domain profile:** `core.commitments`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Commitment exists; responsible actor reports, requester/admin verifies reported-complete work. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Progress and link; set current Commitment state according to requested transition. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. Commitment and new Progress snapshots.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "progress",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	progress

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "commitmentId": {
      "type": "string",
      "minLength": 1
    },
    "kind": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "commitmentId",
    "kind",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "commitment": {
```

```
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "progress": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    }
  },
  "required": [
    "commitment",
    "progress"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.59 rb.polls.vote

Cast or replace this Digital's current ballot; no previous ballot is automatically kept.

Capability: `core-operations`. **Domain profile:** `core.polls`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Poll is open; options, ballot method, actor and selections are valid. Actor is an active eligible participant or the context explicitly declares open response eligibility. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. Replace current actor ballot; preserve distinct tally semantics; no vote history. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The current Ballot Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "ballot",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`choices`	`[]`
`ranking`	`[]`
`ratings`	`{}`
`properties`	`{}`

Argument	Meaning when absent
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.
`type`	Existing target type on update; otherwise ballot

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "pollId": {
      "type": "string",
      "minLength": 1
    },
    "choices": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/domain-response.schema.json#/
$defs/localId"
      },
      "uniqueItems": true
    },
    "ranking": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/domain-response.schema.json#/
$defs/localId"
      },
      "uniqueItems": true
    },
    "ratings": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/domain-response.schema.json#/$defs/
ratings"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
```

```
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "pollId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.60 rb.polls.tally

Compute the declared tally: counts, summed ratings, or Borda points for ranked ballots.

Capability: `core-operations`. **Domain profile:** `core.polls`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Poll options and retained ballots satisfy method constraints. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. Compute declared counts, rating sums or Borda points without changing the Poll.

Result. Method, ranked method where applicable, ballot count and scores by option ID.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "pollId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "pollId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "method": {
      "enum": [
        "single",
        "multiple",
        "ranked",
        "rating"
      ]
    },
    "rankedMethod": {
```

```
      "enum": [
        "borda",
        null
      ]
    },
    "ballots": {
      "type": "integer",
      "minimum": 0,
      "maximum": 9007199254740991
    },
    "scores": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/domain-response.schema.json#/$defs/
scores"
    }
  },
  "required": [
    "method",
    "rankedMethod",
    "ballots",
    "scores"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.61 rb.polls.close

Close the poll and record its current tally without deleting ballots.

Capability: `core-operations`. **Domain profile:** `core.polls`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Poll author or administrator. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. Compute current tally and store results with state closed atomically.

Result. The changed Poll.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "pollId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "pollId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.62 rb.questionnaires.respond

Save draft answers or explicitly submit them after required-field and value checks.

Capability: `core-operations`. **Domain profile:** `core.questionnaires`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Questionnaire is open; known question IDs; submitted answers meet required fields/kinds/choices. Actor is an active eligible participant or the context explicitly declares open response eligibility. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. Upsert respondent's current draft or submitted response; no hidden earlier answer set. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Questionnaire Response Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `LOCAL_ID_RESERVED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "questionnaire-response",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`submit`	`false`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.

Argument	Meaning when absent
`type`	Existing target type on update; otherwise questionnaire-response

Input schema.


```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "questionnaireId": {
      "type": "string",
      "minLength": 1
    },
    "answers": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/domain-response.schema.json#/$defs/
answers"
    },
    "submit": {
      "type": "boolean"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "questionnaireId",
    "answers"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.63 rb.events.respond

Record RSVP independently of attendance or execution.

Capability: `core-operations`. **Domain profile:** `core.events`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Event and actor exist; response is explicit. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert actor/occurrence RSVP; do not infer attendance. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Event Response Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "event-response",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`occurrence`	`""`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.
`type`	Existing target type on update; otherwise event-response

Input schema.


```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "eventId": {
      "type": "string",
      "minLength": 1
    },
    "response": {
      "type": "string",
      "minLength": 1
    },
    "occurrence": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "eventId",
    "response"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.64 rb.events.attendance

Record an explicit attendance report; no RSVP inference.

Capability: `core-operations`. **Domain profile:** `core.events`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Event/Gathering and actor exist; status and occurrence explicit. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Record a deliberate stamped Attendance report separate from RSVP. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The current Attendance Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "attendance",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`occurrence`	`""`
`note`	`""`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.

Argument	Meaning when absent
`type`	Existing target type on update; otherwise attendance

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "eventId": {
      "type": "string",
      "minLength": 1
    },
    "status": {
      "type": "string",
      "minLength": 1
    },
    "occurrence": {
      "type": "string"
    },
    "note": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "eventId",
    "status"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.65 rb.reviews.submit

Record a finding within a review and preserve the distinction from the target content.

Capability: `core-operations`. **Domain profile:** `core.reviews`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Review is open and actor actively participates. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Finding event and Finding In relationship; honour declared presentation release. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Finding.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "finding",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`disposition`	comment
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	finding

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "reviewId": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "disposition": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "reviewId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.66 rb.decisions.respond

Record an explicit proposal position, not inferred agreement.

Capability: `core-operations`. **Domain profile:** `core.decisions`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Subject is Proposal or Decision Round; actor and position explicit. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Upsert actor's current position/reason without creating a Decision automatically. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The Decision Position Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "decision-position",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "preserve-type"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`reason`	`""`
`properties`	`{}`
`categories`	Empty on creation; preserve existing assignments on an upsert.
`tags`	Empty on creation; preserve existing assignments on an upsert.
`type`	Existing target type on update; otherwise decision-position

Input schema.


```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "subjectId": {
      "type": "string",
      "minLength": 1
    },
    "position": {
      "type": "string",
      "minLength": 1
    },
    "reason": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "subjectId",
    "position"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.67 rb.reservations.request

Request use of an authored resource; confirmation remains separate.

Capability: `core-operations`. **Domain profile:** `core.reservations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor and Resource exist; ordered interval, supported clock and positive quantity. Clock defaults to world. Recorded coordinates are canonical UTC instants; world coordinates follow Part 3, clause 3.15, including ms: exact integers or an explicitly declared custom coordinate profile. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create requested Reservation event and Reservation For link; no real-world booking. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The requested Reservation.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "reservation",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`clock`	world
`quantity`	`1`
`properties`	`{}`
`categories`	Empty on creation.

Argument	Meaning when absent
tags	Empty on creation.
type	reservation

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "resourceId": {
      "type": "string",
      "minLength": 1
    },
    "start": {
      "type": "string",
      "minLength": 1
    },
    "end": {
      "type": "string",
      "minLength": 1
    },
    "clock": {
      "enum": [
        "world",
        "recorded"
      ]
    },
    "quantity": {
      "type": "number"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
```

```
      },
      "uniqueItems": true
    }
  },
  "required": [
    "resourceId",
    "start",
    "end"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "allOf": [
    {
      "if": {
        "properties": {
          "clock": {
            "const": "recorded"
          }
        },
        "required": [
          "clock"
        ]
      },
      "then": {
        "properties": {
          "start": {
            "type": "string",
            "pattern": "^[0-9]{4}-(?:0[1-9]|1[0-2])-(?:0[1-9]|[12][0-9]|3[01])T(?:[01][0-9]|
2[0-3]):[0-5][0-9]:[0-5][0-9]\\.[0-9]{3}Z(?![\\s\\S])"
          },
          "end": {
            "type": "string",
            "pattern": "^[0-9]{4}-(?:0[1-9]|1[0-2])-(?:0[1-9]|[12][0-9]|3[01])T(?:[01][0-9]|
2[0-3]):[0-5][0-9]:[0-5][0-9]\\.[0-9]{3}Z(?![\\s\\S])"
          }
        }
      }
    }
  ]
}
```


Successful result schema.


```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.68 rb.reservations.confirm

Confirm capacity for a reservation against other confirmed/lent intervals.

Capability: `core-operations`. **Domain profile:** `core.reservations`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Resource author/admin; pending request; comparable clock intervals and available capacity.

Effects. Confirm only after exact overlap/capacity evaluation; reject atomically if unavailable.

Result. The confirmed Reservation.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "reservationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "reservationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.69 rb.works.capture-edition

Deliberately make one separately authored edition; ordinary edits never call this automatically.

Capability: `core-operations`. **Domain profile:** `core.works`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Work exists; actor explicitly requests an edition. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create separate Edition event containing selected current body and an explicit source Reference. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Edition.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "edition",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`note`	`""`
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	edition

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "workId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "note": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "workId",
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.70 rb.gatherings.contribute

Add an attributed contribution to a facilitated gathering.

Capability: `core-operations`. **Domain profile:** `core.gatherings`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Gathering is open; selected actor actively participates. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Message event and Contribution to Gathering. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new contribution Message.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "message",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	message

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "gatheringId": {
      "type": "string",
      "minLength": 1
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "gatheringId",
    "body"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.


```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.71 rb.moderation.report

Author a report without silently removing the source.

Capability: `core-operations`. **Domain profile:** `core.moderation`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Actor and targets exist; reason explicit. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Moderation Report event and target links; do not remove targets. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new report.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "moderation-report",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	moderation-report

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "targetIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "reason": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "targetIds",
    "reason"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.72 rb.streams.append

Record metadata for a supplied encoded segment; does not capture, transcode or transmit media.

Capability: `core-operations`. **Domain profile:** `core.streams`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Open Stream; declared existing asset; valid timing; active unmuted publishing role. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create encoded-segment metadata event and Stream Entry; no capture/transcoding/network transport. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Stream Segment.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "stream-segment",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use the selected Digital. A RedBook administrator must supply actorId for an attributed participant operation.
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	stream-segment

Input schema.

```
{
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "streamId": {
      "type": "string",
      "minLength": 1
    },
    "asset": {
      "type": "string",
      "minLength": 1
    },
    "mediaType": {
      "type": "string",
      "minLength": 1
    },
    "timestamp": {
      "type": "number"
    },
    "duration": {
      "type": "number"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "streamId",
    "asset",
    "mediaType",
    "timestamp",
    "duration"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.73 rb.soma.neocortex

Read the conceptual home of a Digital or communication context.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Owner exists and has one valid conceptual home.

Effects. Read the actual home without creating or repairing it.

Result. The Soma Neocortex.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "ownerId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "ownerId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.74 rb.soma.create

Create a conceptual world inside one Neocortex, never loose in the RedBook.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Home exists; actor is personal owner or eligible shared participant, or administrator. Type is the named core root or one direct specialization of the same primitive, and supplied Categories/Tags resolve. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create Neo Soma and one Soma Home link in one transaction. Use the selected type, materialize its valid defaults, and assign explicit classifications atomically; preserve personal/shared contribution authority and same-Soma invariants. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Neo Soma.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "neo-soma",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`properties`	`{}`
`type`	neo-soma
`categories`	`[]`
`tags`	`[]`

Input schema.

```
{
  "type": "object",
  "properties": {
    "neocortexId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "neocortexId",
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.75 rb.soma.objects.create

Create a Neo Object inside one conceptual world.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Soma and home valid; selected actor can contribute there. Type is the named core root or one direct specialization of the same primitive, and supplied Categories/Tags resolve. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create Neo Object and one Neo Membership in that Soma. Use the selected type, materialize its valid defaults, and assign explicit classifications atomically; preserve personal/shared contribution authority and same-Soma invariants. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Neo Object.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "neo-object",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`properties`	`{}`
`type`	neo-object
`categories`	`[]`
`tags`	`[]`

Input schema.

```
{
  "type": "object",
  "properties": {
    "somaId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "somaId",
    "title"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.76 rb.soma.relationships.create

Create a meaningful conceptual relationship with named, multi-participant slots.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor can contribute; every role participant belongs to specified Soma. Type is the named core root or one direct specialization of the same primitive, and supplied Categories/Tags resolve. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create attributed Neo Relationship with exact Soma scope and named slots. Use the selected type, materialize its valid defaults, and assign explicit classifications atomically; preserve personal/shared contribution authority and same-Soma invariants. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Neo Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "neo-relationship",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "declared-content",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`title`	`""`
`properties`	`{}`
`type`	neo-relationship
`categories`	`[]`
`tags`	`[]`

Input schema.

```
{
  "type": "object",
  "properties": {
    "somaId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "slots": {
      "type": "object",
      "additionalProperties": {
        "type": "array",
        "items": {
          "type": "string",
          "minLength": 1
        }
      }
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "somaId",
    "slots"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.77 rb.soma.read

Read a Neo Soma, its Objects and its Relationships without recording thought or attention.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Neo Soma exists.

Effects. Return Soma, member Objects and internal Neo Relationships without recording thought or attention.

Result. Soma snapshot, Objects and Relationships.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "somaId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "somaId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "soma": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "objects": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    },
    "relationships": {
```

```
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    }
  },
  "required": [
    "soma",
    "objects",
    "relationships"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.78 rb.soma.represent

Relate a conceptual object to an actual world item without equating them.

Capability: `core-operations`. **Domain profile:** `core.soma`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Neo Object and subject Elements exist; actor can contribute in its home. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create a Represents relationship without equating concept and subject. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The new Represents Relationship.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "represents",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	represents

Input schema.

```
{
  "type": "object",
  "properties": {
    "objectId": {
      "type": "string",
      "minLength": 1
    },
    "subjectIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "objectId",
    "subjectIds"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.79 rb.extensions.describe

Read carried feature, theme and plugin declarations; does not execute them.

Capability: `core-operations`. **Domain profile:** `core.extensions`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. A world is selected.

Effects. Return carried feature/extension declarations; do not activate code.

Result. Current installation declarations.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {},
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/extension-declaration.schema.json"
  }
}
```

B.80 rb.viewboxes.configure

Configure a world-owned experience as administration, not Digital participation.

Capability: `core-operations`. **Domain profile:** `core.viewboxes`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; Viewbox exists; configuration satisfies declared schema.

Effects. Replace current Viewbox configuration; retain content and implementation identity.

Result. The updated Viewbox.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "configuration": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "id",
    "configuration"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.81 rb.presentation.select

Choose the world's default Viewbox and optional theme.

Capability: `core-operations`. **Domain profile:** `core.presentation`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; each supplied non-null Viewbox belongs to this world and non-null theme is a compatible declared world theme.

Effects. Set supplied defaultViewbox, including null to clear it; set supplied theme, or remove the theme member for null to restore host/default fallback. Preserve omitted members and all Viewboxes, resources and content. No interaction role or visible navigation changes.

Result. Resulting presentation descriptor.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`viewboxId`	Retain current defaultViewbox.
`theme`	Retain current theme.

Input schema.

```
{
  "type": "object",
  "properties": {
    "viewboxId": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
          "type": "null"
        }
      ]
    },
    "theme": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
```

```
          "type": "null"
        }
      ]
    }
  },
  "required": [],
  "additionalProperties": false,
  "anyOf": [
    {
      "required": [
        "viewboxId"
      ]
    },
    {
      "required": [
        "theme"
      ]
    }
  ],
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/presentation.schema.json"
}
```

B.82 rb.blueprints.capture-graph

Prepare a reusable component graph, its type definitions and explicit external bindings. The host packages its required original resources.

Capability: `core-operations`. **Domain profile:** `core.blueprints`. **Effect:** `read`. **Modes:** `redbook-admin`.

Preconditions. Administrator; at least one root record, type, Category, Tag or world-owned installation explicitly selected; included records and all selected registry identities exist. Complete transitive definition, taxonomy and declared source/resource closure is resolvable. Selected installations must exist and have complete original resources; Reader-targeted installations cannot be captured into a world Blueprint.

Effects. Capture selected exports and closure without modifying source. Emit explicit exports: {records,types,categories,tags}, with roots equal to exports.records. Definition-only export needs no invented record. Include ancestor/allowed-slot types, Category ancestors, typeResources and resourceDependencies closure as Part 2 specifies. For installationIds, recursively include required world component dependencies and reconstruct complete component packages from retained manifest/ resources without changing their source bytes. Add components entries, requiredParts, compatible declaration types, typed configuration references, mapped Viewbox targets and explicit external bindings according to Part 2.19. Component-only selection may have empty record exports; it does not invent a record or activate code.

Result. Independent component graph and explicit binding/resource requirements, including selected installed functionality and its complete declared closure. Captured component key order follows first selected installation occurrence, then dependency closure order.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`rootIds`	`[]`
`includeIds`	`[]`
`typeIds`	`[]`
`categoryIds`	`[]`
`tagIds`	`[]`
`installationIds`	`[]`

Input schema.

```
{
  "type": "object",
  "properties": {
    "rootIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "includeIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "typeIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "categoryIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tagIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "installationIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [],
  "additionalProperties": false,
  "anyOf": [
    {
      "properties": {
        "rootIds": {
          "minItems": 1
```

```
        }
      },
      "required": [
        "rootIds"
      ]
    },
    {
      "properties": {
        "typeIds": {
          "minItems": 1
        }
      },
      "required": [
        "typeIds"
      ]
    },
    {
      "properties": {
        "categoryIds": {
          "minItems": 1
        }
      },
      "required": [
        "categoryIds"
      ]
    },
    {
      "properties": {
        "tagIds": {
          "minItems": 1
        }
      },
      "required": [
        "tagIds"
      ]
    },
    {
      "properties": {
        "installationIds": {
          "minItems": 1
        }
      },
      "required": [
        "installationIds"
      ]
    }
  ],
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/blueprint.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.83 rb.blueprints.instantiate-graph

Place an independent component graph into this world with new identities and explicit destination bindings. Required assets must already be available.

Capability: `core-operations`. **Domain profile:** `core.blueprints`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; graph valid; bindings/resources/definitions/capabilities compatible. Required resource bytes come from a scoped accepted Blueprint component handle whose graph and inventory match the supplied graph. Bindings are sourceId/targetId pairs; sourceId identifies an external requirement key. Each sourceId occurs exactly once; duplicate/contradictory pairs fail BLUEPRINT_BINDING.

Effects. Atomically allocate record/taxonomy identities, bind explicit dependencies, and map chronology. Equal semantic type definitions reuse their identifiers and destination display metadata; incompatible constraints fail TYPE_CONFLICT. Type semantic comparison excludes only top-level label/description/area and field-rule description. Definition-only exports may return roots=[] and mapping=[]. Preserve whole-Newsletter allocation frontiers; importing an Issue into an existing series must respect its unused-ordinal/frontier rule in Part 3. Copy required parts unchanged under one fresh `redbook/imports/<UUID>/` root, preserve relative source hierarchy, and rewrite only declared resourceReferences, asset paths and resolver-covered metadata. All parts and graph changes commit together; resource-less imports create no resource root. Persist current mapped resource closure in settings.resourceImports; stale record/type/resource dependencies are forbidden. Deletion and re-export consult this registry. Carried component declarations are installed at fresh installation roots with enabled=false; no code executes during import. Component source bytes and declarations remain unchanged, and graph, resources, configurations and installation mappings commit together.

Result. Ordered imported record roots and complete record/type/Category/Tag identity maps. resourceMapping covers every required source path; resourceRoot is the imported resource namespace or null. Identity mappings are ordered sourceId/targetId pair lists covering every carried source once, in corresponding source collection order; arbitrary IDs including $rb are preserved as string values. componentMapping is an ordered list of {key,installationId} entries, in source component order, covering every carried component exactly once; empty when none are carried. Keys, including $rb, remain string values.

Required capabilities. `core-operations`.

Errors. `ACTION_UNKNOWN`, `BLUEPRINT_BINDING`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CHRONOLOGY_DUPLICATE`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `RESOURCE_UNAVAILABLE`, `STATE`, `TYPE_CONFLICT`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`bindings`	`[]`
`title`	Retain the imported root title.

Argument	Meaning when absent
resourceHandle	Allowed only when requiredParts is empty; otherwise the accepted component bytes must be supplied through this host-owned staging handle.

Input schema.

```
{
  "type": "object",
  "properties": {
    "blueprint": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/blueprint.schema.json"
    },
    "bindings": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sourceId": {
            "type": "string",
            "minLength": 1
          },
          "targetId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "sourceId",
          "targetId"
        ],
        "additionalProperties": false
      }
    },
    "title": {
      "type": "string",
      "minLength": 1
    },
    "resourceHandle": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "blueprint"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "roots": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "mapping": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sourceId": {
            "type": "string",
            "minLength": 1
          },
          "targetId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "sourceId",
          "targetId"
        ],
        "additionalProperties": false
      }
    },
    "typeMapping": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sourceId": {
            "type": "string",
            "minLength": 1
          },
          "targetId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "sourceId",
          "targetId"
        ],
        "additionalProperties": false
      }
    },
    "categoryMapping": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sourceId": {
            "type": "string",
```

```
            "minLength": 1
          },
          "targetId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "sourceId",
          "targetId"
        ],
        "additionalProperties": false
      }
    },
    "tagMapping": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "sourceId": {
            "type": "string",
            "minLength": 1
          },
          "targetId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "sourceId",
          "targetId"
        ],
        "additionalProperties": false
      }
    },
    "resourceMapping": {
      "type": "object",
      "additionalProperties": {
        "type": "string",
        "minLength": 1
      }
    },
    "resourceRoot": {
      "oneOf": [
        {
          "type": "string",
          "pattern": "^redbook/imports/[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]
{12}/(?![\\s\\S])"
        },
        {
          "type": "null"
        }
      ]
    },
    "componentMapping": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
```

```
          "key": {
            "type": "string",
            "minLength": 1
          },
          "installationId": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "key",
          "installationId"
        ],
        "additionalProperties": false
      }
    }
  },
  "required": [
    "roots",
    "mapping",
    "typeMapping",
    "categoryMapping",
    "tagMapping",
    "resourceMapping",
    "resourceRoot",
    "componentMapping"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.84 rb.transaction.perform

Execute a compound authored change atomically under one selected role.

Capability: `authoring`. **Domain profile:** `core.transaction`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Unique step keys; only world actions; no nesting or external effects; same delegated role for all steps.

Effects. Evaluate earlier-result bindings, execute staged operations and validate final invariants once; roll back every step on failure.

Result. Ordered step results and a committed indicator; binding includes the updated host revision.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `BATCH_BINDING`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "steps": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "minLength": 1
          },
          "name": {
            "type": "string",
            "minLength": 1
          },
          "input": {
            "type": "object",
            "additionalProperties": true,

"description": "Action input template; only the exact {$result:{step,path}} object is a result
binding, as specified in 4.6."
          }
        },
        "required": [
          "key",
          "name",
          "input"
        ],
        "additionalProperties": false
```

```
      },
      "minItems": 1
    }
  },
  "required": [
    "steps"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "results": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "minLength": 1
          },
          "result": {
            "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json"
          }
        },
        "required": [
          "key",
          "result"
        ],
        "additionalProperties": false
      }
    },
    "committed": {
      "const": true
    }
  },
  "required": [
    "results",
    "committed"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.85 rb.document.configure

Edit world-level description, language and declared settings.

Capability: `authoring`. **Domain profile:** `core.document`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; settings obey their declared profile.

Effects. Apply supplied description/language; shallow-merge settings. Identity, chronology and time are not writable through this action.

Result. Current description, presentation and settings.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.


```
{
  "type": "object",
  "properties": {
    "patch": {
      "type": "object",
      "properties": {
        "description": {
          "type": "string"
        },
        "language": {
          "type": "string",
          "minLength": 1
        },
        "settings": {
          "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
        }
      },
      "required": [],
      "additionalProperties": false
    }
  },
  "required": [
    "patch"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.

```
{
  "type": "object",
  "properties": {
    "description": {
      "type": "string"
    },
    "presentation": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "settings": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "description",
    "presentation",
    "settings"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.86 rb.records.retype

Deliberately assign an existing record a compatible type.

Capability: `authoring`. **Domain profile:** `core.records`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; new type has the same primitive; final ownership and dependent-record invariants hold.

Effects. Change type while retaining identity and authored fields. No existing data is silently removed and no child default silently overwrites an existing Property. Missing defaults are materialized only for newly introduced fields.

Result. Current retyped record. If dependent content must change, use an explicit atomic batch.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "id",
    "type"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.87 rb.records.attribute

Explicitly edit or remove current authorship metadata.

Capability: `authoring`. **Domain profile:** `core.records`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; local author, if supplied, is a present Digital.

Effects. Replace attribution exactly, or remove it when explicitly null. This is deliberate editing, never an incidental consequence of deleting a Digital.

Result. The record with the explicitly chosen attribution.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "attribution": {
      "oneOf": [
        {
          "type": "object",
          "properties": {
            "digitalId": {
              "type": "string",
              "minLength": 1
            },
            "credit": {
              "type": "object",
              "properties": {
                "documentId": {
                  "type": "string",
                  "minLength": 1
                },
                "digitalId": {
                  "type": "string",
                  "minLength": 1
                },
                "name": {
                  "type": "string"
                }
              },
```

```
              "required": [
                "documentId",
                "digitalId",
                "name"
              ],
              "additionalProperties": false
            }
          },
          "required": [],
          "additionalProperties": false,
          "anyOf": [
            {
              "required": [
                "digitalId"
              ]
            },
            {
              "required": [
                "credit"
              ]
            }
          ]
        },
        {
          "type": "null"
        }
      ]
    }
  },
  "required": [
    "id",
    "attribution"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```


Successful result schema.


```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.88 rb.types.remove

Remove an unused, unreferenced type definition.

Capability: `authoring`. **Domain profile:** `core.types`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; no surviving records, child definitions or allowed-type declarations depend on the type; complete-core claim remains valid.

Effects. Remove the declaration only after all dependencies are resolved in the same transaction.

Result. Removed type identity.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "removed": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "removed"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.89 rb.taxonomy.update

Change a Category or Tag label and an explicitly supplied Category parent.

Capability: `authoring`. **Domain profile:** `core.taxonomy`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Term exists; parent applies only to Categories and must not form a cycle.

Effects. Change supplied fields only; retain identity and assignments.

Result. The updated term.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`label`	Retain existing label.
`parent`	Retain current Category parent; no parent is accepted for a Tag.

Input schema.

```
{
  "type": "object",
  "properties": {
    "kind": {
      "enum": [
        "category",
        "tag"
      ]
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "label": {
      "type": "string"
    },
    "parent": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
          "type": "null"
        }
```

```
      ]
    }
  },
  "required": [
    "kind",
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/taxonomy.schema.json"
}
```

B.90 rb.taxonomy.remove

Remove an explicitly selected taxonomy term.

Capability: `authoring`. **Domain profile:** `core.taxonomy`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. No surviving assignment/child may dangle; an explicit reparent value is required when Category children exist.

Effects. Remove only the selected term; remove assignments only if explicitly requested; otherwise reject.

Result. Removed term and affected record identities.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`removeAssignments`	`false`
`reparentChildrenTo`	Do not reparent; reject if children exist. A supplied null explicitly makes children roots.

Input schema.

```
{
  "type": "object",
  "properties": {
    "kind": {
      "enum": [
        "category",
        "tag"
      ]
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "removeAssignments": {
      "type": "boolean"
    },
    "reparentChildrenTo": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
```

```
          "type": "null"
        }
      ]
    }
  },
  "required": [
    "kind",
    "id"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "removed": {
      "type": "string",
      "minLength": 1
    },
    "affectedRecords": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "removed",
    "affectedRecords"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.91 rb.assets.list

List original-resource declarations without reading their contents.

Capability: `authoring`. **Domain profile:** `core.assets`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. World is loaded.

Effects. No mutation or external resource retrieval.

Result. Declared asset descriptors.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `DEPENDENCY`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {},
  "required": [],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "array",
  "items": {
    "$ref": "https://redbook.fun/specification/1.0/schemas/asset.schema.json"
  },
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.92 rb.assets.receive

Commit explicitly staged original bytes into the world package.

Capability: `authoring`. **Domain profile:** `core.assets`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Source bytes were explicitly staged and are available; destination part name is valid; required dependency checks pass.

Effects. Atomically insert bytes and descriptor with exact size and SHA-256, or explicitly replace that asset and update its inventory. No code executes.

Result. Committed original-resource descriptor.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `RESOURCE_CONFLICT`, `RESOURCE_UNAVAILABLE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`name`	Use the staged descriptive name; it is not the package path.
`replace`	`false`

Input schema.

```
{
  "type": "object",
  "properties": {
    "resourceHandle": {
      "type": "string",
      "minLength": 1
    },
    "path": {
      "type": "string",
      "minLength": 1
    },
    "mediaType": {
      "type": "string",
      "minLength": 1
    },
    "name": {
      "type": "string"
    },
    "replace": {
      "type": "boolean"
    }
```

```
  },
  "required": [
    "resourceHandle",
    "path",
    "mediaType"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$ref": "https://redbook.fun/specification/1.0/schemas/asset.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.93 rb.assets.remove

Remove original resources only after dependency resolution.

Capability: `authoring`. **Domain profile:** `core.assets`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. No surviving native or declared feature reference requires the selected parts; unresolved opaque dependencies cause rejection.

Effects. Remove asset declarations and bytes as one transaction; do not leave a stale inventory or source reference.

Result. Removed package-root-relative paths.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "paths": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "required": [
    "paths"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "removed": {
      "type": "array",
      "items": {
```

```
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "removed"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.94 rb.extensions.install

Install a RedBook- or Viewbox-targeted Theme or Plugin package.

Capability: `authoring`. **Domain profile:** `core.extensions`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Package kind/target/manifest valid; target is this world or a Viewbox; dependencies and staged resources resolved.

Effects. Install original declared resources under a fresh canonical UUID payload root according to Annex D.2; preserve the original manifest bytes, retain entry as component-relative, and record relocated manifestPart. Allocate installation identity, or retain explicit replaceInstallationId after requiring inactivity and validating surviving dependents. A new installation sets enabled=false and order=0; replacement preserves the prior enabled/order intent while remaining inactive. Installation does not activate code; replacement removes only eligible old exclusive payload bytes atomically. No host execution grant is stored or transferred.

Result. Installed descriptor.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`replaceInstallationId`	Create a new installation; replacement is never inferred from an equal filename.
`configuration`	Use the component manifest configuration, validated against its schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "resourceHandle": {
      "type": "string",
      "minLength": 1
    },
    "targetId": {
      "type": "string",
      "minLength": 1
    },
    "replaceInstallationId": {
      "type": "string",
      "minLength": 1
    },
```

```
    "configuration": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "resourceHandle",
    "targetId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "allOf": [
    {
      "$ref": "https://redbook.fun/specification/1.0/schemas/extension-declaration.schema.json"
    },
    {
      "required": [
        "installationId",
        "targetId",
        "manifestPart",
        "resourceKey",
        "payloadRoot",
        "enabled",
        "order"
      ],
      "properties": {
        "kind": {
          "enum": [
            "theme",
            "plugin"
          ]
        }
      }
    }
  ]
}
```

B.95 rb.extensions.configure

Edit a world-owned extension configuration or authored state.

Capability: `authoring`. **Domain profile:** `core.extensions`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Installation exists; supplied values satisfy its declared schemas and all resulting dependencies.

Effects. Replace only supplied configuration, authored state, enabled intent and order. Retain omitted members. This records portable desired state without executing code or granting host permission. Effective implementation or authority changes revoke affected active generations and bound invocations; explicit reactivation requires fresh host checks.

Result. Current installed descriptor.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`configuration`	Retain current configuration.
`state`	Retain current authored state.
`enabled`	Retain enabled intent.
`order`	Retain declared order.

Input schema.

```
{
  "type": "object",
  "properties": {
    "installationId": {
      "type": "string",
      "minLength": 1
    },
    "configuration": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "state": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json"
    },
    "enabled": {
```

```
      "type": "boolean"
    },
    "order": {
      "type": "integer",
      "minimum": -9007199254740991,
      "maximum": 9007199254740991
    }
  },
  "required": [
    "installationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "allOf": [
    {
      "$ref": "https://redbook.fun/specification/1.0/schemas/extension-declaration.schema.json"
    },
    {
      "required": [
        "installationId",
        "targetId",
        "manifestPart",
        "resourceKey",
        "payloadRoot",
        "enabled",
        "order"
      ],
      "properties": {
        "kind": {
          "enum": [
            "theme",
            "plugin"
          ]
        }
      }
    }
  ]
}
```

B.96 rb.extensions.activate

Activate a world-owned installed feature in this host session.

Capability: `authoring`. **Domain profile:** `core.extensions`. **Effect:** `external`. **Modes:** `redbook-admin`.

Preconditions. Installation is enabled, its dependency plan and configuration are valid, and the host explicitly permits the requested execution. Each component's required execution/presentation profiles must be available: ECMAScript for executable module components, presentation for DOM/CSS components, templates for declared templates, and participant-authoring for its checked authoring declaration. Missing optional processor facilities make this activation unavailable; authoring capability alone does not assert those facilities.

Effects. Run the scoped activation contract, rollback registrations on failure; no authored revision or history is implied.

Result. Host-local activation receipt, not a serialized document record.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `DEPENDENCY`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Specific failure behavior. Failure returns a host-operation error. All owned registrations are revoked and all cleanup is attempted; diagnostics distinguish inactive state from incomplete cleanup. No authored rollback is promised for undeclared external effects, which are prohibited.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "installationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "installationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "activationId": {
      "type": "string",
      "minLength": 1
    },
    "installationId": {
      "type": "string",
      "minLength": 1
    },
    "state": {
      "const": "active"
    }
  },
  "required": [
    "activationId",
    "installationId",
    "state"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.97 rb.extensions.deactivate

Deactivate a world-owned feature in this host session.

Capability: `authoring`. **Domain profile:** `core.extensions`. **Effect:** `external`. **Modes:** `redbook-admin`.

Preconditions. Activation belongs to selected world and host.

Effects. Revoke API and attempt all cleanup. Authored state and installed source remain.

Result. Truthful cleanup result, including partial cleanup diagnostics.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `DEPENDENCY`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Specific failure behavior. Failure returns a host-operation error. All owned registrations are revoked and all cleanup is attempted; diagnostics distinguish inactive state from incomplete cleanup. No authored rollback is promised for undeclared external effects, which are prohibited.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "activationId": {
      "type": "string",
      "minLength": 1
    }
  },
  "required": [
    "activationId"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "activationId": {
      "type": "string",
      "minLength": 1
    },
    "state": {
      "enum": [
```

```
        "inactive",
        "inactive-with-cleanup-errors"
      ]
    },
    "diagnostics": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  },
  "required": [
    "activationId",
    "state",
    "diagnostics"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.98 rb.extensions.remove

Remove a world-owned extension assignment and explicitly selected exclusive resources.

Capability: `authoring`. **Domain profile:** `core.extensions`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. No active host use, dependent installation or unresolved record/resource reference; default preserves authored content.

Effects. Remove descriptor and assignment references atomically. Data deletion requires deleteAuthoredData=true and an explicit recordIds list whose dependency plan validates.

Result. Removal report with preserved parts.

Required capabilities. `authoring`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `ORDER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`deleteAuthoredData`	`false`
`recordIds`	`[]`

Input schema.

```
{
  "type": "object",
  "properties": {
    "installationId": {
      "type": "string",
      "minLength": 1
    },
    "deleteAuthoredData": {
      "type": "boolean"
    },
    "recordIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "installationId"
  ],
```

```
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "removedInstallation": {
      "type": "string",
      "minLength": 1
    },
    "removedRecords": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "retainedParts": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "removedInstallation",
    "removedRecords",
    "retainedParts"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.99 rb.resources.read

Read an explicit byte range of an accepted package part without executing it.

Capability: `authoring`. **Domain profile:** `core.resources`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Part is in the selected accepted package; host supports resource reads and the requested range.

Effects. Return only the selected byte range from the bound package snapshot; no network retrieval, source execution, attention marker or authored change.

Result. Base64 bytes, range metadata, effective media type and digest of the complete part.

Required capabilities. `authoring`, `package-resource-read`.

Errors. `ACTION_UNKNOWN`, `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `HOOK_FAILURE`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `REFERENCE`, `RESOURCE_UNAVAILABLE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "type": "object",
  "properties": {
    "path": {
      "type": "string",
      "minLength": 1
    },
    "offset": {
      "type": "integer",
      "minimum": 0,
      "maximum": 9007199254740991
    },
    "length": {
      "type": "integer",
      "minimum": 0,
      "maximum": 16777216
    }
  },
  "required": [
    "path",
    "offset",
    "length"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

Successful result schema.

```
{
  "type": "object",
  "properties": {
    "path": {
      "type": "string",
      "minLength": 1
    },
    "mediaType": {
      "type": "string",
      "minLength": 1
    },
    "offset": {
      "type": "integer",
      "minimum": 0
    },
    "bytesRead": {
      "type": "integer",
      "minimum": 0
    },
    "base64": {
      "type": "string",
      "pattern": "^(?:[A-Za-z0-9+/]{4})*(?:[A-Za-z0-9+/]{2}==|[A-Za-z0-9+/]{3}=)?(?![\\s\\S])"
    },
    "eof": {
      "type": "boolean"
    },
    "partSha256": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}(?![\\s\\S])"
    }
  },
  "required": [
    "path",
    "mediaType",
    "offset",
    "bytesRead",
    "base64",
    "eof",
    "partSha256"
  ],
  "additionalProperties": false,
  "$schema": "https://json-schema.org/draft/2020-12/schema"
}
```

B.100 rb.content.edit

Revise an authorized contribution in its current home.

Capability: `core-operations`. **Domain profile:** `core.content`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Resolve one scope under Parts 3.17, 3.24 and 3.25: conceptual content, an author-owned descriptor from the finite initiation table, or additional fields/classifications of a standard primary record. Enforce that scope's actual owner and inherited participant-edit preconditions, including active participation and state restrictions. Never rewrite attribution, event observations, home assignment, protected state/policy fields or authority-bearing slots. Administration does not expand this action's field scope. Applicable Poll/Questionnaire local IDs must be valid and cannot be exactly $rb; reject a reserved descriptor or response key with LOCAL_ID_RESERVED before changing state.

Effects. For conceptual content retain the Part 3.17 edit contract. For descriptors shallow-merge permitted Properties and replace supplied permitted members. For other standard primary records permit only properties containing specialization-declared additional keys plus categories/tags; use the domain action for title/body/state and other inherited fields. Validate the complete graph atomically; preserve identity, type, attribution and event observations.

Result. Changed record.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `LOCAL_ID_RESERVED`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "patch": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record-patch.schema.json"
    }
  },
  "required": [
    "id",
    "patch"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.101 rb.content.unset-properties

Remove explicitly named Properties from an authorized contribution.

Capability: `core-operations`. **Domain profile:** `core.content`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Resolve one scope under Parts 3.17, 3.24 and 3.25: conceptual content, an author-owned descriptor from the finite initiation table, or additional fields/classifications of a standard primary record. Enforce that scope's actual owner and inherited participant-edit preconditions, including active participation and state restrictions. Never rewrite attribution, event observations, home assignment, protected state/policy fields or authority-bearing slots. Administration does not expand this action's field scope.

Effects. Remove precisely named permitted keys, including the empty string if declared/allowed in the selected scope. Missing keys are no-ops. A required field or protected inherited field cannot be removed. Retain all other content and observations.

Result. Changed record.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "keys": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "uniqueItems": true
    }
  },
  "required": [
    "id",
    "keys"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.102 rb.content.classify

Assign Categories and Tags and optionally create assigned terms in an authorized participant edit.

Capability: `core-operations`. **Domain profile:** `core.content`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Resolve one scope under Parts 3.17, 3.24 and 3.25: conceptual content, an author-owned descriptor from the finite initiation table, or additional fields/classifications of a standard primary record. Enforce that scope's actual owner and inherited participant-edit preconditions, including active participation and state restrictions. Never rewrite attribution, event observations, home assignment, protected state/policy fields or authority-bearing slots. Administration does not expand this action's field scope.

Effects. Replace supplied assignment lists, preserve omitted lists, create each requested term with a fresh identity, and append it to its matching assignment list without duplicates. A Category parent is an existing term, never a label match. Tags reject a parent. Term creation and assignment commit together; no unrelated term is edited.

Result. Changed record and created taxonomy term objects in request order; Category objects include parent and Tag objects do not.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`categories`	Retain existing Category assignments before appending created Categories.
`tags`	Retain existing Tag assignments before appending created Tags.
`createTerms`	`[]`

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "minLength": 1
    },
    "categories": {
```

```
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "createTerms": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "kind": {
            "enum": [
              "category",
              "tag"
            ]
          },
          "label": {
            "type": "string",
            "minLength": 1
          },
          "parent": {
            "oneOf": [
              {
                "type": "string",
                "minLength": 1
              },
              {
                "type": "null"
              }
            ]
          }
        },
        "required": [
          "kind",
          "label"
        ],
        "additionalProperties": false,
        "allOf": [
          {
            "if": {
              "properties": {
                "kind": {
                  "const": "tag"
                }
              }
            },
            "then": {
              "not": {
                "required": [
```

```
                  "parent"
                ]
              }
            }
          }
        ]
      }
    }
  },
  "required": [
    "id"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "record": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "createdTerms": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/taxonomy.schema.json"
      }
    }
  },
  "required": [
    "record",
    "createdTerms"
  ],
  "additionalProperties": false
}
```

B.103 rb.content.delete

Remove authorized conceptual content or author-owned descriptive records with complete dependency checks.

Capability: `core-operations`. **Domain profile:** `core.content`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Every selected or automatically removed record is either conceptual content permitted by Part 3.17, or a deletable author-owned descriptor from Part 3.24. Resolve and enforce each record's exact current owner/context and deletion authority. Structural membership/home links may be removed only with selected dependent content; no owning Digital or Neocortex may be selected. Connection, Invitation, Submission and unrelated standard primary records do not gain deletion authority through this action.

Effects. Compute the ordinary deletion plan, require authority for the entire removal set, and remove only explicit records and permitted incident relationships. Never cascade through another contributor or another home implicitly. Remaining reading arrangements, typed references and home invariants must validate. Every dependent record must itself be eligible and authorized; otherwise reject rather than erase another actor's work.

Result. Removed identities and retained-credit report.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`removeRelationships`	`false`

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "ids": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "removeRelationships": {
```

```
      "type": "boolean"
    }
  },
  "required": [
    "ids"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "removed": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "retainedAuthorCredits": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    }
  },
  "required": [
    "removed",
    "retainedAuthorCredits"
  ],
  "additionalProperties": false
}
```

B.104 rb.posts.edit

Edit current Post title or body as its author.

Capability: `core-operations`. **Domain profile:** `core.posts`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Post exists; selected actor is its attributed author or administrator. State is draft or published; a withdrawn Post must deliberately return to draft first.

Effects. Replace only supplied title/body. Preserve publication state, placements, attribution and original event observation. No edition or hidden history is created.

Result. Changed Post.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Selected Digital; administration supplies actorId.
`title`	Retain title.
`body`	Retain body.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "body": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
```

```
  "required": [
    "id"
  ],
  "additionalProperties": false,
  "anyOf": [
    {
      "required": [
        "title"
      ]
    },
    {
      "required": [
        "body"
      ]
    }
  ]
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.105 rb.mail.move

Move only the selected recipient's mailbox entry to a folder or clear its folder.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Mailbox Entry names selected recipient; non-null folder belongs to the same Mailbox as the entry. Administrator supplies the recipient actor explicitly.

Effects. Set its folder slot to the one selected Mail Folder, or [] for null. Do not move other entries, alter the Message/Delivery or mark read.

Result. Changed Mailbox Entry.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Selected Digital; administration supplies actorId.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "entryId": {
      "type": "string",
      "minLength": 1
    },
    "folderId": {
      "oneOf": [
        {
          "type": "string",
          "minLength": 1
        },
        {
          "type": "null"
        }
      ]
    }
  },
  "required": [
```

```
    "entryId",
    "folderId"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.106 rb.domain.transition

Perform one authorized state transition from the closed standard lifecycle table.

Capability: `core-operations`. **Domain profile:** `core.domain`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Record type, prior state, target state, actor and details must match one row of Part 3's domain-transition table. A same-state request still checks that row's authority and is a no-op. Every unlisted transition fails STATE.

Effects. Apply the selected state change and its expressly required dependent records atomically. Proposal adoption/rejection and Decision Round closure require procedure/body details and create an attributed Decision event; other transitions preserve existing observations. Poll closure uses rb.polls.close and is not accepted here.

Result. Resulting subject record and all new records in creation order; empty list when none were required.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`, `TRANSITION_DETAILS`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Selected Digital; administration supplies actorId when the row requires an attributed participant.
`details`	`{}`

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "to": {
      "type": "string",
      "minLength": 1
    },
    "details": {
```

```
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    }
  },
  "required": [
    "id",
    "to"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "record": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "createdRecords": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    }
  },
  "required": [
    "record",
    "createdRecords"
  ],
  "additionalProperties": false
}
```

B.107 rb.newsletters.initialize-numbering

Initialize missing allocation state without resetting a present Newsletter counter.

Capability: `authoring`. **Domain profile:** `core.newsletters`. **Effect:** `write`. **Modes:** `redbook-admin`.

Preconditions. Administrator; Newsletter exists; nextIssueNumber is absent, and proposed frontier exceeds every retained positive numbered issue. A present frontier cannot be reset.

Effects. Write the explicit canonical positive counter. It is current allocation state, not a retained publication history.

Result. Updated Newsletter.

Required capabilities. `authoring`.

Errors. `ALLOCATION_INITIALIZED`, `ALLOCATION_UNINITIALIZED`, `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `NUMBER_RANGE`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "newsletterId": {
      "type": "string",
      "minLength": 1
    },
    "nextIssueNumber": {
      "type": "string",
      "pattern": "^[1-9][0-9]*(?![\\s\\S])"
    }
  },
  "required": [
    "newsletterId",
    "nextIssueNumber"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.108 rb.events.occurrences

Expand and page a finite Event recurrence query without storing projected events.

Capability: `core-operations`. **Domain profile:** `core.events`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Event exists; query is finite, nonempty and in the Event's clock. Recurrence and exceptions obey the two standard profiles in Part 3. Unsupported calendar/coordinate semantics fail explicitly.

Effects. Expand original occurrences, apply replacements/cancellations, include projected intervals intersecting [start,end), sort by projected start then original occurrence key, then apply offset/limit. An instantaneous occurrence is included exactly when start<=point<end. Return nextOffset only if further results remain; no authored event or attention.

Result. Ordered occurrence/start/end projections and nextOffset or null.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CONTEXT`, `DEPENDENCY`, `INPUT_SCHEMA`, `LIMIT_EXCEEDED`, `OUTPUT_SCHEMA`, `RECURRENCE`, `REFERENCE`, `STATE`, `TIME_RULE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`limit`	`100`
`offset`	`0`

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "eventId": {
      "type": "string",
      "minLength": 1
    },
    "start": {
      "type": "string",
      "minLength": 1
    },
    "end": {
      "type": "string",
      "minLength": 1
    },
    "limit": {
      "type": "integer",
      "minimum": 1,
      "maximum": 1000
    },
```

```
    "offset": {
      "type": "integer",
      "minimum": 0,
      "maximum": 9007199254740991
    }
  },
  "required": [
    "eventId",
    "start",
    "end"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "occurrences": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "occurrence": {
            "type": "string"
          },
          "start": {
            "type": "string",
            "minLength": 1
          },
          "end": {
            "type": "string",
            "minLength": 1
          }
        },
        "required": [
          "occurrence",
          "start",
          "end"
        ],
        "additionalProperties": false
      }
    },
    "nextOffset": {
      "oneOf": [
        {
          "type": "integer",
          "minimum": 0,
          "maximum": 9007199254740991
        },
        {
          "type": "null"
        }
      ]
    }
```

```
  },
  "required": [
    "occurrences",
    "nextOffset"
  ],
  "additionalProperties": false
}
```

B.109 rb.rules.evaluate

Evaluate applicable rules without invoking the described operation.

Capability: `core-operations`. **Domain profile:** `core.rules`. **Effect:** `read`. **Modes:** `redbook-admin`, `digital`.

Preconditions. The explicit contextId is among the base action's actual derived policy contexts under the standard resolver table; unknown or ambiguous resolution fails POLICY_CONTEXT. Base name is an available ordinary read/write action, not a transaction/rule wrapper or external action. Validate base input schema before evaluating; resolve actors under the selected role.

Effects. Evaluate a snapshot of applicable rules against {actor,context,action}; sort by priority then rule identity. A failed require or matched deny denies; a matched effects rule selects its ordered literal inputs. Do not execute base action or effects and do not imply that later domain/state validation will succeed. Multiple context guard sets are evaluated in exact context identity order; no supplied contextId can bypass another actual context. Effect selection and execution use context, rule and effect-list order.

Result. Explicit policy decision, matched and denying IDs, and selected effect descriptors.

Required capabilities. `core-operations`.

Errors. `INPUT_SCHEMA`, `CONTEXT`, `REFERENCE`, `STATE`, `PARTICIPATION`, `POLICY_CONTEXT`, `POLICY_DENIED`, `RULE`, `DEPENDENCY`, `CAPABILITY_UNAVAILABLE`, `EDIT_CONFLICT`, `OUTPUT_SCHEMA`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "contextId": {
      "type": "string",
      "minLength": 1
    },
    "action": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "minLength": 1
        },
        "input": {
          "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
        }
      },
      "required": [
        "name",
        "input"
      ],
```

```
      "additionalProperties": false
    }
  },
  "required": [
    "contextId",
    "action"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "allowed": {
      "type": "boolean"
    },
    "matchedRuleIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "deniedBy": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      }
    },
    "effects": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "ruleId": {
            "type": "string",
            "minLength": 1
          },
          "index": {
            "type": "integer",
            "minimum": 0,
            "maximum": 9007199254740991
          },
          "action": {
            "type": "string",
            "minLength": 1
          },
          "input": {
            "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
          }
        },
        "required": [
          "ruleId",
```

```
          "index",
          "action",
          "input"
        ],
        "additionalProperties": false
      }
    },
    "validation": {
      "const": "policy-only"
    }
  },
  "required": [
    "allowed",
    "matchedRuleIds",
    "deniedBy",
    "effects",
    "validation"
  ],
  "additionalProperties": false
}
```

B.110 rb.rules.process

Apply an explicitly requested action and selected rule effects atomically.

Capability: `core-operations`. **Domain profile:** `core.rules`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. The explicit contextId is among the base action's actual derived policy contexts under the standard resolver table; unknown or ambiguous resolution fails POLICY_CONTEXT. Base name is an available ordinary read/write action, not a transaction/rule wrapper or external action. Validate base input schema before evaluating; resolve actors under the selected role. Selected rules permit the base action; all effect actions are available ordinary world read/write actions and pass their own authority, policy and schema checks. Policy-changing base/effect actions are prohibited.

Effects. Snapshot and evaluate policies once, execute the checked base action, then selected literal effect actions in deterministic rule/effect order in the same transaction. Ordinary action guards remain active, but effects never recursively trigger more rule processing. Any denial, effect error or final validation failure rolls back base/effects and allocations together. Multiple context guard sets are evaluated in exact context identity order; no supplied contextId can bypass another actual context. Effect selection and execution use context, rule and effect-list order.

Result. Base result and rule-result receipts in execution order; no stored rule-run history.

Required capabilities. `core-operations`.

Errors. `INPUT_SCHEMA`, `CONTEXT`, `REFERENCE`, `STATE`, `PARTICIPATION`, `POLICY_CONTEXT`, `POLICY_DENIED`, `RULE`, `DEPENDENCY`, `CAPABILITY_UNAVAILABLE`, `EDIT_CONFLICT`, `OUTPUT_SCHEMA`. Error categories have the meanings in Part 4, clause 4.10.

No optional top-level argument has an implicit value. Required arguments are identified by the input schema.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "contextId": {
      "type": "string",
      "minLength": 1
    },
    "action": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "minLength": 1
        },
        "input": {
          "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
        }
      },
      "required": [
```

```
        "name",
        "input"
      ],
      "additionalProperties": false
    }
  },
  "required": [
    "contextId",
    "action"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "result": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json"
    },
    "ruleResults": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "ruleId": {
            "type": "string",
            "minLength": 1
          },
          "index": {
            "type": "integer",
            "minimum": 0,
            "maximum": 9007199254740991
          },
          "result": {
            "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json"
          }
        },
        "required": [
          "ruleId",
          "index",
          "result"
        ],
        "additionalProperties": false
      }
    }
  },
  "required": [
    "result",
    "ruleResults"
  ],
  "additionalProperties": false
}
```

B.111 rb.mail.folders.create

Create an attributed folder in an authorized Mailbox.

Capability: `core-operations`. **Domain profile:** `core.mail`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. The Mailbox exists; the selected Digital owns it or the context is RedBook administration. actorId follows the selected-role rules. Primary-record type, Properties and classifications obey Part 4.19; additional-fields input cannot override inherited domain fields.

Effects. Create one MailFolder with typed mailboxId reference, title and supplied description, preserving all messages and entries. Apply ordinary creation attribution without fabricating an event. Apply primary-record defaults, additional fields and supplied classifications atomically before final validation; dependent records retain their prescribed types.

Result. The newly created MailFolder record.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": "mail-folder",
  "typeSelection": "base-or-specialization",
  "propertiesMode": "additional-fields",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Use selected Digital in digital mode; administration remains unattributed when no actor is supplied.
`description`	`""`
`properties`	`{}`
`categories`	Empty on creation.
`tags`	Empty on creation.
`type`	mail-folder

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "mailboxId": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "description": {
      "type": "string"
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "mailboxId",
    "title"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

B.112 rb.content.create

Author a descriptor, proposal of Connection or Submission through the finite participant-initiation table.

Capability: `core-operations`. **Domain profile:** `core.content`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Selected actor exists. Type is one listed in Part 3.24 or a valid specialization. Its exact row determines permitted Properties, slots, context and dependencies. Resolve references and enforce that row's authority; reject unused contextId/sourceIds, unrelated types and protected state/policy fields. Slot order and multiplicity follow the effective CRT constraints; generic slot arrays impose no additional uniqueness rule.

Effects. Construct the attributed primary record with declared defaults and valid supplied content, applying the row's fixed initial state and any explicitly required dependent records. Submission creates its Submission Target atomically. Connection sets only the selected initiator, proposed state and an empty decision list. No event, participation grant, placement, delivery or background job is implied.

Result. Primary record and all newly created records, primary first, in creation order.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Primary creation. This entry follows Part 4.19.

```
{
  "baseType": null,
  "typeSelection": "participant-initiation-table",
  "propertiesMode": "table-defined",
  "existingRecord": "not-applicable"
}
```

Omitted arguments.

Argument	Meaning when absent
`actorId`	Selected Digital; administration must supply the attributed actor.
`title`	`""`
`properties`	`{}`
`slots`	`{}`
`categories`	`[]`

Argument	Meaning when absent
`tags`	`[]`
`contextId`	Absent except where Part 3.24 requires it.
`sourceIds`	Absent except for Submission, where a nonempty list is required.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "type": {
      "type": "string",
      "minLength": 1
    },
    "title": {
      "type": "string"
    },
    "properties": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/value.schema.json#/$defs/map"
    },
    "slots": {
      "type": "object",
      "additionalProperties": {
        "type": "array",
        "items": {
          "type": "string",
          "minLength": 1
        }
      }
    },
    "categories": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "tags": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    },
    "contextId": {
```

```
      "type": "string",
      "minLength": 1
    },
    "sourceIds": {
      "type": "array",
      "items": {
        "type": "string",
        "minLength": 1
      },
      "uniqueItems": true
    }
  },
  "required": [
    "type"
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "record": {
      "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
    },
    "createdRecords": {
      "type": "array",
      "items": {
        "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
      }
    }
  },
  "required": [
    "record",
    "createdRecords"
  ],
  "additionalProperties": false
}
```

B.113 rb.connections.respond

Record only the selected counterpart's decision on a proposed Connection.

Capability: `core-operations`. **Domain profile:** `core.connections`. **Effect:** `write`. **Modes:** `redbook-admin`, `digital`.

Preconditions. Target is a proposed Connection or specialization with valid participant and decision lists; selected actor is a named counterpart. Administrative use explicitly supplies that counterpart's actorId. A terminal decision is not silently reopened.

Effects. Replace only this counterpart's decision, retaining every other decision. Canonicalize decision order to counterpart-slot order. Set declined if any counterpart declines; accepted only if every counterpart accepts; otherwise keep proposed. Preserve participants, attribution and original observations.

Result. The Connection with its current explicit decisions.

Required capabilities. `core-operations`.

Errors. `CAPABILITY_UNAVAILABLE`, `CHRONOLOGY`, `CONTEXT`, `DEPENDENCY`, `EDIT_CONFLICT`, `INPUT_SCHEMA`, `OUTPUT_SCHEMA`, `PARTICIPATION`, `REFERENCE`, `STATE`. Error categories have the meanings in Part 4, clause 4.10.

Omitted arguments.

Argument	Meaning when absent
`actorId`	Selected Digital; administration explicitly names the counterpart whose decision it asserts.

Input schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "object",
  "properties": {
    "actorId": {
      "type": "string",
      "minLength": 1
    },
    "id": {
      "type": "string",
      "minLength": 1
    },
    "accept": {
      "type": "boolean"
    }
  },
  "required": [
    "id",
    "accept"
```

```
  ],
  "additionalProperties": false
}
```

Successful result schema.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$ref": "https://redbook.fun/specification/1.0/schemas/record.schema.json"
}
```

C

ANNEX C

Annex C — External reference register

The external specifications and reference documents identified by RedBook Open XML 1.0.

JELLYBOX — Jellybox WebMCP Framework

BCP14 — Requirement terminology

XML — Extensible Markup Language (XML) 1.0

XMLNS — Namespaces in XML 1.0

XSD — XML Schema Part 1: Structures and Part 2: Datatypes

OPC — ECMA-376 Part 2: Open Packaging Conventions

URI — Uniform Resource Identifier: Generic Syntax

JSON — The JavaScript Object Notation Data Interchange Format

JSONSCHEMA — JSON Schema Core and Validation

ECMASCRIPT — ECMAScript Language Specification

UTF8 — UTF-8, a transformation format of ISO 10646

SHA256 — Secure Hash Standard

UNICODE — The Unicode Standard

CCBY — Creative Commons Attribution 4.0 International

MIT — MIT License

DATETIME — Internet date and time syntax

ICALENDAR — Recurrence conventions

LANGUAGE — Tags for identifying languages

BASE64 — Binary-to-text encoding

CASEFOLD — Versioned Unicode default case folding

This register identifies the external works used for the specific purposes stated below. RedBook obligations are defined in Parts 1–5. A reference does not incorporate unrelated application formats, transitional features or host behavior. The external works retain their own licenses and governance.

JELLYBOX — Jellybox WebMCP Framework

Version **1.0 — Standalone Core**, editorial revision 8 November 2026, by Press Rogers. Published specification; pinned local copy. Normative for `jellybox-webmcp-1.0`: public tool schemas, references, action descriptors, results, retry/lifecycle behavior and navigation. RedBook domain authority and transaction obligations remain governed by Part 4. The adapter profile records the exact source hashes; later changes to the same remote URL are not automatically adopted.

BCP14 — Requirement terminology

RFC 2119 and RFC 8174 · normative

Requirement terminology

Capitalized requirement words only; the RedBook clauses state the actual obligations.

XML — Extensible Markup Language (XML) 1.0

Fifth Edition, 26 November 2008 · normative

Extensible Markup Language (XML) 1.0

XML character and well-formedness rules; RedBook excludes DTDs and external entities.

XMLNS — Namespaces in XML 1.0

Third Edition, 8 December 2009 · normative

Namespaces in XML 1.0

Expanded element and attribute names; prefixes have no identity meaning.

XSD — XML Schema Part 1: Structures and Part 2: Datatypes

Second Edition, 28 October 2004 · normative

XML Schema Part 1: Structures and Part 2: Datatypes

XSD 1.0 grammars supplied locally; prose adds semantic constraints.

OPC — ECMA-376 Part 2: Open Packaging Conventions

Fifth Edition, December 2021 · normative

ECMA-376 Part 2: Open Packaging Conventions

Container concepts, content types and package relationships as constrained by Part 2; not WordprocessingML conformance.

URI — Uniform Resource Identifier: Generic Syntax

RFC 3986, January 2005 · normative

Uniform Resource Identifier: Generic Syntax

URI syntax and internal-reference resolution constrained by the canonical RedBook path subset.

JSON — The JavaScript Object Notation Data Interchange Format

RFC 8259, December 2017 · normative

The JavaScript Object Notation Data Interchange Format

JSON text; this specification additionally rejects duplicate keys and restricts ordinary numbers.

JSONSCHEMA — JSON Schema Core and Validation

Draft 2020-12 · normative

JSON Schema Core and Validation

Schema dialect used by this distribution, resolved offline; not asserted as an IETF Internet Standard.

ECMASCRIPT — ECMAScript Language Specification

ECMA-262, ECMAScript 2024 · normative

ECMAScript Language Specification

Language semantics for the optional standard ECMAScript execution profile; host API specified in Part 4.

UTF8 — UTF-8, a transformation format of ISO 10646

RFC 3629, November 2003 · normative

UTF-8, a transformation format of ISO 10646

Native XML/JSON/source encoding.

SHA256 — Secure Hash Standard

FIPS PUB 180-4, August 2015 · normative

Secure Hash Standard

SHA-256 digests identify retained part bytes; they are not signatures or execution authority.

UNICODE — The Unicode Standard

Unicode 15.1.0 · informative

The Unicode Standard

Text terminology; the fixed search mappings are specified by CASEFOLD below. Exact identifier equality does not normalize Unicode.

CCBY — Creative Commons Attribution 4.0 International

4.0 · license

Creative Commons Attribution 4.0 International

License for original specification prose.

MIT — MIT License

Text included in licenses/MIT.txt · license

MIT License

License for original machine-readable artifacts and tooling.

DATETIME — Internet date and time syntax

RFC 3339, July 2002 · reference for comparison

Date and Time on the Internet: Timestamps

RedBook defines its narrower canonical UTC millisecond recording syntax and proleptic calendar rules explicitly in Parts 1 and 3. It does not accept every RFC 3339 offset or leap-second spelling.

ICALENDAR — Recurrence conventions

RFC 5545, September 2009 · profile reference

Internet Calendaring and Scheduling Core Object Specification

Part 3 specifies the exact supported recurrence subset, finite expansion and exception semantics. A recurrence description does not schedule host work.

LANGUAGE — Tags for identifying languages

BCP 47 / RFC 5646, September 2009 · normative syntax reference

Tags for Identifying Languages

RedBook language labels use the referenced tag syntax. Language identification does not select a different identity-comparison or search-case algorithm.

BASE64 — Binary-to-text encoding

RFC 4648, October 2006 · normative encoding reference

The Base16, Base32, and Base64 Data Encodings

Resource-read results use the standard base64 alphabet, required padding and zero pad bits, with no inserted whitespace. Decoding does not change original resource bytes.

CASEFOLD — Versioned Unicode default case folding

Unicode 15.1.0, CaseFolding.txt · normative search mapping

Unicode 15.1.0 case-fold mappings

Part 4 uses the C and F mappings, excludes Turkic T mappings, and performs no normalization. The version is fixed for interoperable standard search; locale-dependent host lowercase functions are not substitutes.

D

ANNEX D

Annex D — Portable execution, extensions and presentation

The portable execution and presentation contracts for modules, Plugins, Themes, Viewboxes and Reader facilities.

D.1 Profiles and capability negotiation

D.2 Installation resource roots and dependencies

D.3 ECMAScript module profile

D.4 Plugin API and handler invocation

D.5 Hook stages, ordering and filtering

D.6 User-defined extension points

D.7 Viewbox bridge and render completion

D.8 Declarative Theme templates and view data

D.9 Stable interface regions and contributions

D.10 Minimal portable Reader host API

D.11 Failure and lifecycle invariants

D.12 Portable participant authoring

Normative. This annex completes the execution and Reader-facing contracts referenced by Part 4. It specifies portable behavior, not a required UI framework, database, process model or AI provider. The normative machine companions are `catalogues/hooks.json`, `catalogues/reader-actions.json`, `catalogues/ui-slots.json`, `schemas/templates.schema.json`, `schemas/execution-context.schema.json` and `schemas/view-render-context.schema.json`. A host may preserve every byte of a component while reporting that an execution profile is unavailable.

D.1 Profiles and capability negotiation

The profile identifiers are `rbx.ecmascript.1` (modules, actions and hooks), `rbx.dom.1` (Viewbox bridge and scoped DOM), `rbx.templates.1` (declarative Theme templates), `rbx.reader-host.1` (Reader facilities) and `rbx.participant-authoring.1` (declared participant mutations). The corresponding broad processor capabilities are `ecmascript`, `presentation`, `templates`, `reader-host` and `participant-authoring`; claiming one includes its versioned profile. CSS Theme support is included in `presentation`; template Theme support additionally requires `templates`. `reader-host` does not grant filesystem or connection access merely because the profile is implemented. Participant authoring supplies checked mutation operations for custom CETs and CRTs and added fields on core specializations under D.12; registering an ordinary action alone supplies no such authority.

> **[RBX-D-001]** Discovery MUST state each known profile's availability and applicable limits before activation. Components MUST declare every required versioned profile in `requires`. A missing requirement MUST produce `CAPABILITY_UNAVAILABLE` without execution or fallback to a different meaning. Optional enhancement can be separately declared by a component; the host MUST NOT silently pretend a required facility is optional. Additional host facilities use namespaced identifiers and their own input, result, effect and lifetime contracts.

The execution context identifies `scopeId`, `instanceId`, `documentId`, `contextId`, selected role and nullable `digitalId`. A Reader context has null instance/document/Digital identifiers. A world context has a loaded-instance identifier distinct from its persistent document identity. A mounted Viewbox adds `viewboxId`; activation adds installation/target identifiers. Runtime handles and callback functions are host control values, never ordinary portable Properties.

D.2 Installation resource roots and dependencies

An installation has an authored installation identity, a canonical lowercase UUID `resourceKey`, `payloadRoot`, a component identity and exact version, target kind/identity, original manifest path, entry, configuration, optional portable state, `enabled` intent and signed safe-integer `order`. The default intent is `enabled:false`, `order:0`. `payloadRoot` is exactly `redbook/installations/<resourceKey>/`. The resource key is newly allocated per installation, independently of authored record identifiers; it never derives from an untrusted filename or display label. The original component archive may additionally be retained as an asset.

[RBX-D-002] Installation MUST validate the complete source envelope and resource inventory before staging a fresh resource root. Every source payload path `p` MUST be stored as `payloadRoot + p` without rewriting its bytes. The original manifest is retained inside that root. The native extension declaration records the destination manifest path and the original component-root-relative entry path; paths inside the original manifest remain component-root-relative. The native inventory MUST include every stored part with its actual digest and content type. A collision with any existing destination part MUST reject the transaction, even when that existing part has equal bytes; a new resource key can be allocated before retrying. Successful installation MUST commit the declaration and all required bytes together.

A host resolves installed source through a virtual root `rbx://<opaque-scope-token>/`. The token identifies one loaded-world instance or Reader installation, one installation identity and one activation generation. It is host-generated, is not serialized, and conveys no authority outside that resource resolver. Source `code/plugin.mjs` and its `../assets/icon.png` retain those relative names within the installation. The resolver maps virtual `p` to stored `payloadRoot+p`. Two components can therefore both use `code/plugin.mjs` without colliding. Blueprint graph-resource imports use a fresh reserved `redbook/imports/<UUID>/` root under Part 4. Explicit component entries additionally create fresh disabled installations under D.2; importing ordinary graph resources does not implicitly install extensions. An implementation entry under that reserved prefix resolves relative resources against that import root; its original source paths and bytes remain unchanged inside it. Only declared graph resource-reference fields are rewritten using the returned `resourceMapping`. The current resource/dependency scope is persisted in `settings.resourceImports` under the schema and deletion/re-export rules in Part 4, so reopen does not lose the resolver boundary. Unknown embedded references require their declared resolver or cause import failure. Other direct core source and standalone Viewboxes use a separate package-root resolver with the same confinement rules.

[RBX-D-003] Relative resource resolution MUST remove dot segments without permitting escape above the declared virtual root. Empty paths, backslashes, percent-encoded separators, fragments in module imports, userinfo, network schemes and undeclared parts MUST be rejected. A leading `/` means this installation's root, never the host filesystem. Query strings are forbidden for module and original-resource lookup. CSS `url()` and local `@import` use the referring stylesheet's virtual directory; font/image bytes remain original bytes. Network CSS imports and undeclared URL fetches are unavailable under the standard profile. A link deliberately presented for user navigation is governed separately by D.8.

[RBX-D-004] Dependency lookup MUST be scoped: a Reader installation resolves only Reader-targeted dependencies in that Reader configuration; a RedBook installation resolves only RedBook dependencies in the same loaded world; a Viewbox installation resolves Viewbox dependencies on that same Viewbox first, then RedBook dependencies in that world. More than one matching `(component id, exact version)` at the applicable precedence is `DEPENDENCY_AMBIGUOUS`; missing or wrong-version dependencies are `DEPENDENCY_UNAVAILABLE`. The complete graph MUST be acyclic. Dependencies activate before dependents; independent ready nodes use ascending `order`, then exact installation-identity scalar order. A disabled required dependency makes activation unavailable; a request MUST NOT enable it silently.

Dependencies provide declared capabilities, not implicit imports into another installation's private files. The standard module profile has no cross-installation module import. A component needing code dependencies carries their original modules in its own payload, with applicable notices, or declares a separately versioned callable capability. This preserves reproducible local module graphs without source rewriting.

> **[RBX-D-005]** `enabled` and `order` describe portable desired state. Changing either MUST be an explicit authored configuration action and MUST NOT itself execute code. Opening or inspecting a package remains passive. A host may offer an explicit “activate enabled components” operation; that operation resolves the entire plan, requests host grants, activates in dependency order and reports each outcome. Failure MUST revoke every activation newly created by that plan in reverse order; pre-existing activations remain unchanged. Host grants, actual activity, errors and cleanup receipts are host state. A host can refuse execution while preserving the authored intent.

> **[RBX-D-006]** Replacement MUST name the installation identity and require that installation and its active dependents to be inactive. It stages and validates a new exclusive resource root, replaces the declaration atomically, then removes only the previous root when no surviving declared use remains. Replacement preserves the previous `enabled` and `order` intent; a separate explicit configuration action changes that intent. The replacement remains actually inactive until explicit activation. Uninstallation MUST reject surviving installation dependencies or external references into that root. It MUST remove only its exclusive payload and declaration. Resources shared deliberately through ordinary world assets require separate ownership/reference checks and MUST NOT be removed merely because one installation retires. Failed replacement or removal leaves the accepted package unchanged.

D.3 ECMAScript module profile

The `rbx.ecmascript.1` source language is ECMAScript 2024 modules encoded as UTF-8. Data, operations and participant-authoring semantics are language-independent; an independently implemented processor need not use JavaScript internally. A component declaring this execution profile does require its module semantics. A Plugin entry exports `setup(api, context)`. A Viewbox entry exports `mount(root, book, options)`. Dependencies and original source are carried as declared resources. Retaining a Source Repository archive as Media does not advertise its execution; executing a repository requires an explicitly declared build profile with its own inputs and outputs. No build system is implied by this standard.

> **[RBX-D-007]** The loader MUST resolve static relative imports and re-exports against the referring module's virtual URL, including explicit file extensions, and root-relative imports against the installation virtual root. Bare specifiers, network imports, import attributes requiring another module kind, dynamic `import()`, top-level await and CommonJS facilities are unavailable in `rbx.ecmascript.1`. A host MUST diagnose these as `MODULE_UNSUPPORTED` before setup/mount rather than reinterpret them. JavaScript modules may contain ordinary Promise/async function definitions, but standard action, hook, setup, cleanup and mount calls remain synchronous. JSON and media are resource bytes, not implicit executable modules.

> **[RBX-D-008]** A complete static module graph MUST be parsed, linked and verified against its declared inventory before evaluation. Module identity is `(activation generation, canonical virtual path)`; each module evaluates once in that generation. Cyclic static imports follow ECMAScript module semantics, including uninitialized-binding errors. Different installations, loaded copies and activation generations MUST NOT share mutable module instances. Deactivation revokes its generation. Reactivation creates new module instances; failed evaluation discards the generation and leaves no registered facilities.

The minimum globals are ECMAScript intrinsic objects plus `TextEncoder`, `TextDecoder`, `URL` and `URLSearchParams`. A diagnostic console may be supplied but does not establish a persistent document log. The standard does not supply `fetch`, filesystem APIs, process environment, transport credentials, Worker creation or unrestricted Reader internals. A DOM Viewbox receives its dedicated root and DOM APIs through the `presentation` capability; a Plugin receives a DOM root only in an authorized render/UI callback. Direct access to another installation's DOM or mutable world store is outside the profile. Additional globals must be explicitly named capabilities. Host containment must enforce these boundaries; a global property convention alone is insufficient.

> **[RBX-D-009]** Module evaluation and setup MUST NOT author records, initiate network/filesystem operations or activate an AI connection. Host time and random values may be used for ephemeral calculation; authoritative event allocation, record identity, clock observations and commits remain checked dispatcher responsibilities. Unsupported module facilities MUST be reported as unavailable while preserving the original source bytes.

D.4 Plugin API and handler invocation

All methods below are bound to an activation generation and authority ceiling. Released or revoked methods fail with `EXTENSION_INACTIVE`. Parameters described as snapshots are independent values; mutating one never mutates the world.

Method	Defined behavior
`read(id)` / `query(query)`	Read the current context projection through the same semantics as the corresponding standard action. No implicit authoring.
`action(name,description,effect,inputSchema,handler,modes,outputSchema,options={})`	Register a unique complete action contract; return its idempotent release. `options.policyContextResolver`, when supplied, identifies the exact registered resolver defined in Part 3; it is action metadata, never a caller-supplied context grant. An unknown required resolver makes the action unavailable. Standard action names cannot be replaced. Default modes are the activation's permitted target roles; Reader activations default to `reader-admin`, world activations to their explicitly granted subset of `redbook-admin`/`digital`.
`authoring.register(name,handler)`	Register one action from this installation's retained participant-authoring declaration, with the checked mutation facade in D.12. No runtime argument may widen its portable contract. Returns an idempotent release. Requires `rbx.participant-authoring.1`, portable participation rules and separate host execution permission.

Method	Defined behavior
`hook(name,callback,priority=10,version="1.0")`	Register against a known compatible hook descriptor in this target scope; return a release.
`hooks.define(descriptor)` / `hooks.list()`	Declare a custom namespaced extension point / read the scoped available descriptors. Definition returns a release; D.6 governs version and dispatch.
`hooks.dispatch(name,args,version="1.0")`	Dispatch a custom hook owned by this activation in its declared phase; core `rb.*` hooks cannot be dispatched by a Plugin.
`policies.defineResolver(descriptor)`	Register a namespaced resolver described by D.6 finite operators; return a tracked release.
`clock(id,provider)`	Register a unique synchronous `(rule,recordedAt) -> portableValue` custom time provider, with release.
`perform(name,input,context)`	Initiate a checked operation outside an active handler/hook/observer. Authority remains the intersection of selected role and activation grant. It is not a nested transaction escape.
`onCommit(callback)`	Register a scoped read-only postcommit observer receiving `{instanceId,documentId,editCounter,name,changedIds}`. Return a release.
`resources.read(path)` / `resources.url(path)`	Return a copy of accepted payload bytes / a generation-bound virtual-resource URL. No fetching or execution.
`ui.contribute(descriptor)`	Add a declared scoped template/command contribution under D.9; return a release. Requires DOM presentation.
`host.describe()` / `host.invoke(name,input,options)`	Discover/invoke explicitly granted Reader facilities in D.10. Invocation follows their synchronous read or asynchronous-job contract.
`host.jobs.read(id)` / `host.jobs.cancel(id)` / `host.jobs.subscribe(id,callback)`	Read/cancel/observe a host job belonging to this activation and permitted context. Subscription returns a release.

> **[RBX-D-010]** The registered handler MUST be invoked exactly as `handler(input, call)`, with independent validated input and a live call facade. `call.context` is the immutable execution-context snapshot. `call.tx` is a transaction facade for writes and null for reads/external actions. `call.resources` provides accepted reads only; `call.host` is available only for declared external actions. The handler MUST synchronously return its declared portable result. A thrown error or thenable fails with the appropriate action error; a write then rolls back. The dispatcher validates the result and final candidate state before success. Retaining `call` past return never retains a live authoring authority.

A `read` handler runs against one accepted revision and cannot invoke any write, staging or external effect. An `external` handler may initiate a declared host job and return its receipt, but does not combine irreversible host effects into an authored-world transaction. A `write` handler uses the transaction facade for all changes. No direct mutable record-store API is part of the standard. D.12 adds checked primitive mutations for specifically declared participant-authoring handlers; those methods are not unrestricted record-store access.

> **[RBX-D-011]** A write facade MUST supply `tx.perform(name,input)`, `tx.read(id)` and `tx.query(query)`. Reads observe the candidate state including prior steps. `perform` applies the called action's input, mode, membership, dependency, result and semantic checks under the same context and authority ceiling. It may invoke only transactional read/write actions, never an external action or another batch transaction. Nested calls share the candidate, allocator and rollback boundary; they MUST NOT create independent receipts, commit, notify observers or advance the accepted edit counter. Domain operations may perform internal invariant-preserving stages, but later public calls see only valid completed nested results.

> **[RBX-D-012]** The dispatcher MUST reject recursive ancestor action-name re-entry with `ACTION_REENTRANT` and enforce declared finite nesting and step bounds, reporting `LIMIT_EXCEEDED` before commitment when exceeded. Repeated sibling calls are permitted. A failed nested call poisons the outer transaction even if extension code catches its exception. Calling ordinary `api.perform` from a handler, precommit hook or postcommit observer MUST fail; the dispatcher MUST NOT start an independent inner transaction. Revocation and all final validation failures discard every staged record, resource, attribution and allocation change.

The `rb.action.stage` hook runs once for the outer logical write after its handler and before final state/result validation. Its callback receives `{name,input,result}`, the context, and the same transaction facade. It can add dependent checked work and cannot replace the handler result or bypass its schema. Nested `tx.perform` calls execute their ordinary input/record/action hooks, but do not recursively emit `rb.action.stage`. A batch is one outer logical write; its steps do not each emit this hook. This provides one explicit extension point for atomic dependent contributions.

> **[RBX-D-013]** A successful changed outer write MUST advance the accepted edit counter once and publish one postcommit notification after releasing its commit lock. Notification `changedIds` is a deduplicated, scalar-sorted list of identities created, changed or removed by that transaction; it is transient notification data, not a persisted audit history. Read/no-op operations emit no change notification. Postcommit callbacks are read-only; their exceptions are isolated and reported as observer errors without reversing or hiding the committed result. Failed transactions emit no success notification. Disposal revokes observer subscriptions before invoking cleanup.

D.5 Hook stages, ordering and filtering

`catalogues/hooks.json` defines the standard names, target scopes, callback argument schemas, kind, phase, filter result schema and failure behavior. The standard supplies action/record hooks plus query, template/render, Reader library/open/save, activation and UI contribution hooks. The fixed eight transport tools are unchanged; hook discovery is available through the scoped Plugin API and host capability catalogue.

> **[RBX-D-014]** Dispatch MUST select only live registrations in the event's Reader/world/Viewbox scope and matching activation generation. Order is ascending signed safe-integer priority, then registration order within that scope. Default priority is 10. A dispatch snapshots its registration list; new registrations wait for a later dispatch, and registrations revoked before their turn are skipped. Release is idempotent and affects only the registration it created. Parameters use independent snapshots, except explicitly typed DOM roots and transaction facades.

> **[RBX-D-015]** A filter's first argument is its current value. Each returned replacement MUST validate against the descriptor's result schema and the stage's semantic restrictions before becoming the next filter's first argument. A missing/undefined return is invalid for a filter. Action-hook returns are ignored except that a thenable is forbidden. A thrown error, invalid filter output or thenable terminates a precommit/render-filter/lifecycle-before dispatch with `HOOK_FAILURE`. The completed `rb.viewbox.rendered` observer is an exception: its errors are isolated after completion, without undoing the accepted render handle or repeating the hook. Precommit failure aborts the enclosing transaction; a pre-open/pre-save failure prevents the host operation. Postcommit and completed-host observers isolate errors and cannot revoke an already completed effect.

The query-input filter runs before standard filtering/sorting/paging and cannot change the selected role, reveal unprojected fields, disable caller constraints or widen an explicit target restriction. It may add narrowing predicates and reduce limits. The basic-query schema has no order member: ordering changes belong to the declared Feed/view contract or the result filter, never an invented input field. The query-result filter may reorder or remove only records in that authorized result; it cannot fabricate records or add excluded data. A filter that cannot honor these conditions fails. Template selection can choose only a declared available template for the same target. Rendered-item filters receive declarative nodes, not HTML strings; the resulting node tree is revalidated before rendering. Reader save hooks may inspect the selected immutable snapshot, but changing bytes requires an explicit authored action before starting a fresh save.

> **[RBX-D-016]** All ordinary precommit hooks MUST remain snapshot-only. Only `rb.action.stage` receives transaction-bound authoring facilities. Hook execution MUST NOT escalate role, change discovery or activation, perform irreversible effects, or recursively dispatch a core hook. Rendering, query and UI hooks MUST NOT cause authored activity. A postcommit observer can display a notification or refresh an ephemeral view; a further authored change requires a separate explicit invocation outside the observer.

D.6 User-defined extension points

A custom descriptor supplies `name`, exact `version`, `scope`, `kind`, `phase`, `argumentsSchema`, nullable `resultSchema`, description and required capabilities. Its name is a URI under the extension author's control; `rb.*` and `reader.*` names are reserved. Schemas are protocol-control JSON using the offline rules in Part 2. Host-control argument markers are limited to declared DOM/transaction contracts; a custom descriptor cannot grant a transaction facade to a phase that forbids it.

> **[RBX-D-017]** `hooks.define` MUST validate the descriptor and reject an unequal existing same-name/version descriptor with `HOOK_CONFLICT`. Equal definitions can be reference-counted within their owner scopes, while registration ownership remains distinct. Hook consumers MUST select an exact version through the registration/dispatch version argument (default `"1.0"`) and required capability set. Different versions are distinct descriptors; a registration never switches versions implicitly. Unknown or unavailable hook registration fails with `HOOK_UNAVAILABLE`; discovery MUST expose its reason. Releasing the final definition revokes dependent registrations before subsequent dispatch. A custom dispatch MUST originate from its owning activation, remain in its declared scope and phase, and validate arguments/results exactly as a standard dispatch.

Custom hook dispatch cannot manufacture a postcommit phase or host-operation success. In a handler it inherits the handler's current transaction restrictions; outside a handler it cannot obtain a transaction facade. Defining a hook changes the capability revision and invalidates affected bindings. These rules allow documented extension composition without private kernel patches.

The resolver declaration API accepts `{id,clauses:[{op,path}]}`. `id` is a namespaced URI; `clauses` is a nonempty ordered list with at most 64 entries. Each path is a nonempty list of literal input member names. Supported operators are `input`, `wallOfPlacement`, `conversationOfMessage`, `mailboxOfEntry`, `contextOfInvitation`, `resourceOfReservation`, `ownerOfNeocortex`, `ownerOfSoma` and `ownerOfConcept`, with the meanings in `catalogues/domain-lifecycles.json`. Each path resolves one identity or a list of identities; apply its operator to each, then deduplicate and scalar-sort the resulting Element contexts. Missing inputs, missing records, ambiguous links or non-Element contexts produce `POLICY_CONTEXT`, never an empty grant. There is no callback evaluation, external lookup or mutable-state access. Registration is checked against `policy-resolver.schema.json`, returns a release, and invalidates capability bindings. Releasing a resolver makes dependent actions unavailable. An extension write action advertising Digital participation MUST name an available resolver; administration-only actions may omit it but still obey all structural invariants. Context guards apply in addition to, and cannot replace, the selected actor's normal operation checks.

D.7 Viewbox bridge and render completion

The standard bridge has `read`, `query`, `describe`, `catalogue`, `perform`, `snapshot`, `onChange`, `beginRender`, `completeRender`, `cancelRender` and `resources`. All read results are scoped projections. `perform` follows the same checked dispatcher as the eight-tool interface. `snapshot` supplies a read-only projection, never the mutable world object. Additional host aliases are outside the standard bridge.

> **[RBX-D-018]** `mount(root,book,options)` MUST receive a dedicated DOM root, a generation-bound bridge and snapshots in `{context,viewboxId,configuration,onError}`. It MUST synchronously return an idempotent disposer. Mounting is successful only after its initial render has completed through the bridge. An exception or thenable return fails mounting, revokes the bridge and cleans all tracked subscriptions/listeners/resources. The host MUST NOT separately emit a duplicate completion merely because mount returned.

> **[RBX-D-019]** Before changing its rendered content, a Viewbox MUST call `beginRender({section,viewState})` to obtain one opaque render handle for its current instance and generation. At least one of section/viewState is present. After successful DOM changes it MUST call `completeRender(handle)`. The host MUST accept that handle once, increment that instance's render revision and emit exactly one `rb.viewbox.rendered(root,renderContext)` notification. Repeating completion for an already completed live handle returns its original completion receipt without a second hook. A failed render calls `cancelRender(handle)` and emits no completed-render hook. Errors from completed-render observers are diagnostic and cannot turn the already accepted handle back into pending or cause a second dispatch; completion returns its stable receipt even when observers report errors. Foreign, canceled or revoked handles fail with `RENDER_HANDLE`.

Render context includes instance/document/Viewbox identifiers, generation identifier, render revision and the supplied section/view state. Render revision is a safe integer scoped to the mount; exhaustion requires remounting with a fresh generation. `onChange` returns a release function and reports accepted commits only. Implementations may coalesce several notifications into one render, provided the resulting view uses the latest accepted snapshot and emits one completion for that render.

> **[RBX-D-020]** Changes to effective context, implementation, configuration or target MUST dispose the old mount before replacement. Disposal MUST revoke all render handles, bridge authoring methods, listeners, subscriptions, observers, timers and generated resource URLs. Late completion or callbacks MUST fail or be dropped with a revoked-generation diagnostic; they cannot author or attach to a replacement mount. Theme replacement may rerender the existing compatible Viewbox only through its explicitly supported update path; otherwise it remounts.

D.8 Declarative Theme templates and view data

A CSS Theme uses `profile:"css"` (also the meaning of an absent profile) and a CSS entry. A template Theme uses `profile:"template"`, a JSON entry conforming to `templates.schema.json`, and optional declared `stylesheet`. The latter requires `rbx.templates.1` and `rbx.dom.1`. Template JSON is protocol-control data, not executable JavaScript. Themes do not receive Plugin registration powers.

The template document has `formatVersion:"1.0"`, a nonempty `templates` map and ordered `selection` rules. A rule supplies `view` (`*` or one stable view name), optional exact concrete `type`, and a template name. First matching rule wins. `defaultTemplate`, when present, is that Theme's fallback. The renderer tries selected Viewbox-target Theme, then RedBook-target Theme, then Reader-target Theme; the first matching template wins. If no theme supplies a match, the Viewbox's built-in template is used. Configuration is separate from the source template. CSS layers apply Reader → RedBook → Viewbox, within declared scopes; later layers do not remove accessibility obligations.

> **[RBX-D-021]** Template selection MUST use declared view identity and concrete record type, never substring matching of labels or inspection of hidden DOM. Selection filters may choose only a template declared by an available effective Theme at that scope. Absent templates, unresolved paths and unsupported node kinds MUST yield an explicit render error or the documented built-in fallback selected before rendering; they MUST NOT produce an empty success that conceals content.

Standard view names are `world`, `record`, `wall`, `conversation`, `collection` and `soma`. The common data context is `{world,view,actor,record,items,theme}`. `world` contains id/title/description; `view` contains id/name/configuration; `actor` contains role and nullable Digital id; `record` is the selected authorized record or null; `items` contains the authorized results of the view's ordinary read operation; `theme` contains validated configuration. Wall items are placed records in Wall order; Conversation items are visible messages in event order; Collection items are its ordered members; Soma items are its objects followed by conceptual Relationships; World items follow the standard query; Record items are empty. Each `items` entry is a view-data object containing the selected record snapshot's normal top-level fields (so `item.title` and `item.properties` address that record), plus a mandatory top-level `placement` member containing the applicable placement/membership Relationship snapshot or null. It is not a `{record,placement}` wrapper. This extra member belongs to projected view data, never the serialized record or its Properties. The exact common shape is `template-context.schema.json`. Mail-specific Viewboxes must supply only their recipient projection, including the Bcc rule.

A path is an array of literal string map keys or nonnegative integer list indices. Evaluation traverses own members only, without prototype lookup, expressions or coercion. Missing and null remain distinct. A missing path uses the node's explicit fallback or reports `TEMPLATE_DATA`; it does not execute a getter. Within `each`, `item` and `index` are lexical bindings; nested iteration shadows only those two names and restores the outer bindings afterwards.

Node kind	Contract
`text`	Emit literal `text` as a text node.
`value`	Read `path`; render the standard scalar text (strings unchanged, null `null`, booleans lowercase, exact-number original lexeme, ordinary number canonical JSON). Maps/lists require explicit iteration or an identified renderer. Optional `fallback` is literal text for an absent path.
`element`	Create an allowed semantic HTML tag, validated plain attributes and ordered children. No executable attributes or arbitrary HTML.
`each`	Evaluate an array at `path`; render `children` for each item in order, or `empty` when the array is empty. No implicit query, fetch or mutation.
`slot`	Render contributions for a declared named slot in priority order; use `fallback` when none exist.
`resource`	Display a declared local image resource using its scoped resolver and required alternate text. Unsupported media has a visible fallback.
`link`	Display a user-activated link with literal or path-derived destination and children. HTTP/HTTPS/mailto and declared internal navigation are permitted; `javascript:`, data script URLs and filesystem navigation are rejected. No link is followed automatically.
`action`	Display a labelled user control for a discoverable available action. Input fields are explicit literal/path bindings. An explicit activation invokes that action in the current context and reports its result/error. Rendering never invokes it.

> **[RBX-D-022]** Renderers MUST validate the full template tree, enforce declared finite node and depth limits, and escape all authored text by construction. Allowed element tags are defined by the schema; `script`, `iframe`, arbitrary style elements, inline event handlers, raw HTML and string evaluation are excluded. The renderer MUST reject prototype-sensitive attribute/lookup behavior. CSS and resource URLs are resolved within the Theme root. Template data MUST be a role-appropriate projection; switching theme cannot expand data access or grant an action.

D.9 Stable interface regions and contributions

The standard region catalogue defines `reader.navigation`, `reader.library.toolbar`, `reader.document.toolbar`, `view.header`, `view.toolbar`, `view.sidebar`, `view.footer`, `record.before`, `record.after`, `view.empty` and `view.status`. A host exposes `data-rb-region` on its region roots, `data-rb-view` on view roots, `data-rb-record-id` on record containers, and `data-rb-action` on

authored action controls. These markers have the meanings in `catalogues/ui-slots.json`, not merely a shared prefix. A host may change internal markup while preserving the documented region and control semantics.

> **[RBX-D-023]** A contribution MUST declare its unique activation-local identity, target, slot, safe-integer priority and template node content. Reader slots accept Reader-targeted contributions only. World/Viewbox contributions are confined to their own world and target Viewbox. Ordering is priority then registration order. Contributions MUST render through the same safe node and action-binding contract as Themes, retain keyboard operation and accessible labels, and be removed on release/deactivation. No contribution can replace a mandatory host permission/error control or hide the selected actor/operation semantics.

The core CSS variables are `--rb-color-background`, `--rb-color-surface`, `--rb-color-text`, `--rb-color-muted`, `--rb-color-accent`, `--rb-color-border`, `--rb-font-body`, `--rb-font-mono`, `--rb-radius` and `--rb-space`. A host supplies usable defaults. Target roots retain `data-rb-target="reader|redbook|viewbox"`; a Viewbox root also identifies its document and Viewbox through `data-rb-document-id` and `data-rb-viewbox-id`. Theme styles are rooted under their target scope and cannot escape into another world or unrelated Reader surface. A Theme may rearrange supported presentation and templates without altering the stored meaning of a Wall, Post or actor. Explicit host accessibility overrides may take precedence.

D.10 Minimal portable Reader host API

`catalogues/reader-actions.json` defines the input and result schemas for the common Reader operations. These are host actions, distinct from the world-action catalogue. They can be discovered through the same eight tools in `reader-admin` context, or through a granted Plugin host facade. For the eight-tool route, the catalogue modes require selection of `reader-admin`. For the Plugin facade, an explicit host grant creates a restricted host-operation context containing only the granted action names, instance/ticket/connection scope and current grant revision. It does not change the Plugin's selected world/Digital context. The host checks that restricted context rather than treating the catalogue mode as a blanket Reader-administration grant. Unlisted actions or instances remain unavailable.

> **[RBX-D-024]** `host.describe()` MUST return `{profile,scopeId,grantRevision,capabilities,limits}`. Each capability identifies the supported action/profile and current availability with a reason when unavailable. A required but unavailable native chooser, storage facility or connection MUST remain unavailable rather than fabricate a successful empty result. Host actions MUST validate their schemas and grants before starting work. No standard input is an unrestricted host filesystem path, executable command, credential or arbitrary network destination.

`host.invoke` requires `{scopeId,grantRevision,requestId}` options for external actions; reads may omit options but still check the current grant. Scope and grant revision come from `host.describe`. The host reserves `(scopeId,activation generation,requestId)` atomically and compares the complete operation/input signature before execution. Retained terminal replay precedes freshness

checks; a conflicting reuse fails, and an identical in-flight call waits or reports `REQUEST_IN_PROGRESS`. New work with an obsolete grant revision fails `CONTEXT_STALE`. Revoked activations cannot invoke or replay through a retained facade.

Synchronous read actions return their specified result directly. An external host action returns `{jobId,status}` immediately, where status is `pending`, `running`, `succeeded`, `failed`, `cancelled` or `uncertain`. Job reads return the same identity, operation, status and exactly one terminal result/error when known. Job result schemas are the operation's separately listed `completionSchema`; a job receipt is not its completion. Job IDs and file/resource handles are opaque, scope-bound and nonportable.

Action	Resulting host behavior
`reader.instances.list`	List permitted loaded instance IDs, document IDs/titles and accepted/saved revisions. Equal document IDs do not merge instances.
`reader.library.list`	Filter by kind and exact identity or title text, sort by exact library-item identity, then offset/limit. Return metadata only.
`reader.files.choose`	Obtain explicitly user-selected readable or writable file tickets through a supported chooser. Cancellation is a terminal cancelled result. It opens/executes nothing.
`reader.files.open`	Read one accepted file/library ticket, validate and open a new independent instance passively. Return instance/document IDs and available capabilities.
`reader.files.save`	Save a specified accepted instance revision to an explicitly selected write ticket using atomic replacement and destination freshness. Return saved revision/digest/bytes and whether newer edits remain unsaved.
`reader.resources.stage`	Read a selected readable ticket into immutable accepted staging; return resourceHandle, bytes, digest and media type for later explicit world/library operations.
`reader.library.receive`	Store a selected accepted package as an independent library item with exact bytes and a fresh library identity. It neither imports into a world nor activates code.
`reader.connections.list`	List permitted named connection capabilities and availability, without credential material.
`reader.connections.request`	Perform an explicitly granted named connection operation with its declared schema and egress policy. It is not unrestricted URL fetching or an implied AI request.
`reader.jobs.read` / `reader.jobs.cancel`	Inspect/cancel a permitted scoped job. Cancellation is best effort until a commit/external effect has happened; report the actual terminal outcome.

[RBX-D-025] A file ticket MUST record access mode and an opaque destination revision. Save MUST compare the expected destination revision immediately before atomic replacement, fail if changed, and preserve the previous destination on failure. It saves one immutable accepted world revision; subsequent edits remain dirty and are not lost. Host pre-save hooks inspect that snapshot and cannot change it. Save-completed notifications occur only after durable success under the declared host storage guarantees. An unsupported storage guarantee must be disclosed before starting the operation.

[RBX-D-026] Host jobs MUST be owned by the initiating scope and grant. A revoked activation MUST lose callback and authoring access immediately. The host cancels reversible work where possible; an already committed file write or remote effect is reported as completed or uncertain rather than falsely rolled back. Lost connectivity after an external request MUST yield `uncertain` when its effect cannot be determined. Such work MUST NOT be automatically repeated as a fresh operation. Explicit request identities use the same reservation/replay principles as Part 4, scoped to the Reader instance and operation grant.

This API deliberately leaves the native chooser design, directory layout, window system, database and connection provider to the host. The semantics above and the machine schemas are portable. A specific external connection adds its own versioned operation contract; absence of an AI provider or external directory service does not prevent document or Reader-host conformance.

D.11 Failure and lifecycle invariants

[RBX-D-027] Every registration, resource URL, UI contribution, observer, job subscription, module graph and render handle MUST belong to one explicit lifetime owner. Deactivation first revokes authority, then attempts all tracked releases in reverse registration order, then reports clean inactivity or cleanup errors. A failure in one cleanup does not skip others. A retained stale reference cannot register new work, mutate a world or target a replacement activation. A host may retain diagnostic data outside the document without inventing automatic authored history.

[RBX-D-028] Qualification of these profiles MUST cover failed setup after partial registration, nested transaction failure, hook filter rejection, duplicate/late render completion, component path collisions, dependency cycles, desired-state save/reopen, template escaping and authority projection, job cancellation/uncertain effects, and independent module-root resolution. Passing envelope/schema checks alone MUST NOT be reported as execution or UI conformance. Conformance reports identify which engine/host implementations actually ran each behavior case.

D.12 Portable participant authoring

The `rbx.participant-authoring.1` profile connects declared actions to checked changes of custom Elements and Relationships. It also permits declared additional Properties on specializations of supplied core types to be edited under their normal participant authority. It applies to RedBook- and

Viewbox-targeted Plugins. Measurement and Measured Subject in Annex E demonstrate the general mechanism; they are not supplied types or privileged action names. Core creation and lifecycle actions remain authoritative for core types and their specializations. Part 4's primary-creation extension arguments carry their additional Properties through those actions.

D.12.1 Portable rules and host execution

> **[RBX-D-029]** The action's meaning, schemas, implementation, type definitions, mutation scope and world participation rules MUST be retained as authored package data. The Reader's permission to execute a component or let a session represent a Digital MUST remain separate host state. Host permission may refuse or narrow execution but MUST NOT supply missing portable authoring rules, widen those rules or silently substitute administration. Moving to another capable Reader MUST preserve the action's declared meaning and participation rules without inheriting the former host's execution permission.

> **[RBX-D-030]** A participating Plugin MUST name a declared `application/json` resource in `authoringPart` and require `rbx.participant-authoring.1`. That resource MUST conform to `participant-authoring.schema.json`. Its `profile` is the exact versioned identifier, `types` carries the applicable custom definitions, and `actions` carries unique namespaced action declarations. Each action declares description, input/output schemas, a finite `policyContextResolver`, scope, participation basis, explicit reference inputs and uniquely identified mutation rules. Its effect is `write` and mode is `digital`. Input/output schemas use Part 2's bounded same-schema reference profile; `{}` accepts every value and `{"not":{}}` accepts none. Discovery exposes the retained resolver descriptor's `id` string; the declaration carries its complete descriptor. No action name may replace a standard or another installed action. Reader-targeted Plugins cannot use this world-authoring profile.

A rule declares `id`, `typeId`, `includeSpecializations`, nonempty `operations`, exact `properties` and `slots` key lists, `members` drawn from title/categories/tags, and `editAuthority`. Operations are `create`, `recordEvent`, `patch`, `unset` and `remove`. Empty key lists grant no writes in that category. Property names are literal, including the empty string; there are no wildcards. Rule IDs select one rule, never a union. Rules grant no retyping, type/registry/policy editing, attribution editing or host access.

> **[RBX-D-031]** The installation's ordinary `enabled` intent and D.2 lifecycle MUST also govern participant-authoring components. There is no additional portable authoring-adoption flag. Before activation the processor MUST validate the retained manifest, resource inventory, authoring declaration and compatible world definitions. Enabling or installing a component MUST NOT execute it, grant a host session representation of a Digital, or bypass world participation. Replacement invalidates the old activation generation and registrations under D.2; it MUST NOT erase authored participation rules merely because payload bytes changed. The host independently decides whether the replacement may execute.

> **[RBX-D-032]** `authoring.register(name,handler)` MUST obtain the named action from its own live installation, validate its declarations and register its finite resolver under the ordinary conflict and lifetime rules. It MUST reject unknown names, conflicts, thenable setup, unresolved definitions or incompatible semantic type contracts before making the action available. A handler supplies behavior, not replacement schemas or wider scope. Release revokes the registration and its tracked resolver usage. Registration MUST NOT install types or mutate authored state. Definitions are introduced by an explicit administrative or Blueprint operation before activation.

All applicable custom definitions, including selected specializations and their custom parents, are carried in the declaration and compatible with the world registry under Part 2's semantic projection. Supplied core parents are resolved against the normative core catalogue. A root rule permits only that root unless `includeSpecializations:true`; it then also permits declared valid direct specializations. A rule naming a specialization permits that exact specialization only. Root and child constraints apply together.

For an independently defined root and its specializations the declared primitive operations are available. For a specialization of a supplied core root only `patch` and `unset` of explicitly declared additional Properties are available. Such a rule has empty slots and members lists, cannot name an inherited core Property, and must also satisfy the base domain's participant edit authority. Core-root mutation, creation, deletion, participant changes and lifecycle changes remain with their standard actions. Renaming a type cannot evade the semantic distinction. A referenced core type supplies no mutation authority merely by being a dependency.

D.12.2 Context and participation

> **[RBX-D-033]** Before invoking the handler the dispatcher MUST resolve `scope.contextInputPath` from validated input to one existing Element present among the action's resolved policy contexts. Paths consist of literal object member names, not expressions or array indexing. The dispatcher MUST fix the context, explicit reference inputs, selected Digital and applicable participation snapshot before mutation. The context itself cannot be mutated by this primitive facade. `scope.recordContext` MUST select one of these declared associations:

Kind	Association of a target record with the fixed context
`property`	`property` names a required typed-Reference field without a default in the effective target contract. The target contains exactly the context Reference; creation injects it and the draft cannot supply it.
`creator`	The context is a Digital and the target's local creator attribution identifies that Digital. New records receive the selected Digital's attribution under Part 1. A descriptive external credit alone cannot establish this association.

Kind	Association of a target record with the fixed context
relationship	`typeId`, `recordSlot` and `contextSlot` identify a declared Relationship association. A retained or staged Relationship of that type associates an Element in `recordSlot` with the fixed context in `contextSlot`. The binding Relationship itself is in scope when its `contextSlot` identifies that context; every effective slot/type constraint applies. New Elements and their binding Relationships may be completed within the same atomic transaction. An existing target must already belong to the scope in accepted state and remain in scope in the candidate; adding an association cannot manufacture edit authority over a foreign existing record.

The required `participation.basis` is `owner`, `active-members` or `policy`. Owner and active-members use the existing ownership or membership semantics of the core context; they do not require a duplicate list of actors. A Digital owns its own context. Mailbox ownership follows Mailbox Owner. Wall owner authority follows its current local author as expressly defined for Walls. Neocortex ownership follows Neocortex Owner, and Neo Soma follows Soma Home to its Neocortex and owner. Active membership follows an active Membership of the context; for a Neocortex or Soma it resolves through those ownership links to the shared context first. Personal homes use the owner basis. An ambiguous, missing or inapplicable ownership/membership relation fails closed rather than using attribution as a generic ownership substitute. Part 3 and the domain lifecycle catalogue define those relationships.

For `policy`, the context's portable `properties.participation.extensionAuthoring` explicitly grants the exact action to the selected Digital. Each action appears at most once. An entry contains either a duplicate-free `actors` list of typed References to existing Digitals, or a `basis` of `owner` or `active-members` evaluated by the same rules above. It cannot contain both. An absent action or nonmatching rule denies participation. Malformed or unresolved policy fails. The policy supplies no administration. All ordinary applicable require/deny guards in Part 4 additionally apply, whatever basis is selected.

The rule's `editAuthority` is `contributor` or `participants`. Both require action-level participation. Contributor limits existing-record changes to a matching local `attribution.digitalId`; participants permits participating actors to edit matching records in that scope. Neither overrides a core specialization's base edit rules. A remote source credit is not a local contributor match. Permitted edits preserve the original attribution.

D.12.3 Public checked mutations

The live `call.tx` supplied to a registered participant-authoring handler adds the methods below. They are synchronous, share the enclosing candidate and become unusable when the handler returns. Record results are independent snapshots; pending references may occur only in the immediate mutation result used to finish that same transaction. Public reads, queries and nested actions require a complete valid candidate.

Method	Meaning
`reserveId()`	Reserve a fresh identity for this live handler and transaction. The reservation authors no record and consumes no event number.

Method	Meaning
`create(ruleId,draft)`	Create one ordinary record. Draft follows `participant-create.schema.json`, requires type, and optionally supplies title, Properties, Categories, Tags and slots. Their defaults are respectively empty string, map, list, list and map before effective Property defaults are applied. Optional `id` must be an unused reservation belonging to this handler; omission allocates a fresh identity.
`recordEvent(ruleId,draft)`	Create one deliberate event through the same checked construction contract, additionally allocating the complete event triple under Part 1. The rule must explicitly permit `recordEvent`; `create` permission does not imply it.
`patch(ruleId,id,patch)`	Apply `record-patch.schema.json`: change allowed members, shallow-merge supplied Property keys and replace a supplied slot map, checking both removed and supplied slot keys.
`unset(ruleId,id,keys)`	Remove exactly the permitted optional Property keys. An absent optional key is a no-op. Required, scope and policy fields cannot be removed.
`remove(ruleId,id)`	Remove exactly one authorized record, returning `{removed:id}`. Final validation must find no surviving dependency. There is no implicit cascade.

[RBX-D-034] Each mutation MUST select exactly one declared rule, require its named operation, check the permitted type/lineage and enforce the fixed scope, participation and editable fields. Unknown rules, undeclared members or operations, out-of-scope records and caller-selected authority MUST fail before commitment. Primitive calls MUST NOT change role, actor, installation, declaration, context or type registry. Only handlers registered through `authoring.register` receive these methods. Ordinary handlers and stage observers cannot acquire them by retaining another action's facade. A reservation MUST be consumed exactly once before its handler returns; a foreign, reused or unconsumed reservation fails the transaction. Required cross-references may use reserved identities, but the final graph must resolve them all.

[RBX-D-035] Creation MUST determine the primitive from the declared type, apply effective defaults and assign the selected Digital's attribution and source credit under Part 1. Property-scope creation MUST inject its reserved scope Reference and reject that Property in the draft even if equal. Explicit draft Properties, slots and members MUST be permitted by the selected rule. Defaults may initialize other declared fields but do not grant subsequent writes. Supplied or defaulted `participation` is forbidden through this facade. `create` MUST NOT invent observed metadata. `recordEvent` MUST allocate the host recording instant, current world-time result and exact next sequence atomically; the draft cannot supply them. Required lifecycle and graph dependencies MUST be satisfied by explicitly checked work before commitment.

[RBX-D-036] Patch, unset and removal MUST apply edit authority to every target. Patch accepts only the declared title, Properties, Categories, Tags and slots. Slot replacement checks old and new keys. The shared omitted/null/empty/default rules are in Part 1. Category/Tag identities must exist. Identity, type, primitive, attribution, event observations, context association and `participation` MUST remain unchanged. The facade MUST NOT change guards, membership, installation state or another action's declaration through an alternative representation of those protected semantics. Editing added core-specialization Properties also enforces the base domain edit authority and effective combined constraints.

[RBX-D-037] Every changed slot and typed Reference, including nested Properties and defaults, MUST satisfy its ordinary participant, type and cardinality constraints. A newly introduced or retargeted content reference MUST additionally point to the fixed context, a record in the same declared scope, or an existing Element selected by a declared `referenceInputs` path. An unchanged retained reference remains subject to ordinary integrity and type checks but does not require a fresh linking grant merely because another field is edited. Each such path resolves one existing Element from validated input before the handler; the actor must be permitted to read and link it. A binding permits that reference only, never editing the referenced record. Attribution and source credit follow their separate Part 1 rules. Reservations and relationship-scoped construction may defer reference/scope closure until the graph is complete; they do not defer schema checks, field authority or final participation validation.

[RBX-D-038] Final validation MUST preserve reference, type, resource and installation integrity without rewriting unauthorized records. Removal may be staged with dependent removals explicitly authorized in the same transaction. Ordinary read/query/perform calls MUST reject an incomplete candidate rather than expose dangling references. Invalid fields, participant types, specializations, scope closure, output schema, unused reservations or final graph state MUST poison the complete transaction even when extension code catches an immediate exception. Failure rolls back record identities, event allocation and every staged change together.

D.12.4 Composition, preservation and qualification

[RBX-D-039] Part 4's transaction, binding, replay and commit-freshness contracts MUST apply. Nested `tx.perform` retains the selected Digital and independently applies the called action's own declaration, host execution permission and participation; it MUST NOT combine rules into a wider grant. Stage hooks may call a declared custom action through `tx.perform` but receive no primitive mutation authority themselves. After filters and stage work, the processor MUST recheck changed records and references. It MUST revalidate authority and guards against current accepted state at the serialized commit point. Revoked or stale authority discards the candidate and allocations. Success produces one accepted revision and one ordinary commit notification; no automatic history is authored. Each installation's registrations and reservations remain isolated and are revoked with its own activation generation.

> **[RBX-D-040]** Saving and reopening MUST preserve original implementation and declaration bytes, compatible definitions, enabled intent and current authored participation. A receiving host MUST rediscover the rules and make its own execution/representation decision. Unsupported or refused execution MUST NOT delete those rules. Qualification MUST exercise a loaded extension through `authoring.register` and these public methods, with genuinely new CET/CRT roots and a valid specialization. It MUST cover successful create/edit, scope denial, invalid-participant rollback, preserved attribution, deliberate event allocation, valid mutual-reference construction, and host refusal followed by separately permitted continuation after actual save/reopen. A synthetic mutation callback that bypasses the interface is insufficient. A functionality-transfer assessment additionally follows Part 2/4 component closure through import, reopen, continued use and re-export.

Diagnostics distinguish `AUTHORING_DECLARATION` (invalid retained contract), `EXTENSION_INACTIVE` (no live permitted generation), `HOST_EXECUTION_DENIED` (host refusal), `MUTATION_SCOPE` (undeclared mutation authority) and `REFERENCE_SCOPE` (reference outside declared linking scope). Existing participation, policy, schema, reference and state errors retain their meanings. These declarations explain portable behavior; they are not bearer credentials. Host execution decisions neither replace nor erase the world rules.

E

ANNEX E

Annex E — Annotated reading and implementation walkthroughs

Annotated examples explaining package structure, exact values, transactional authoring, presentation and reuse.

E.1 Package structure and authored meaning

E.2 A record containing exact data and a reference

E.3 Protocol-control JSON without changing authored values

E.4 Atomic extension work and a subsequent render

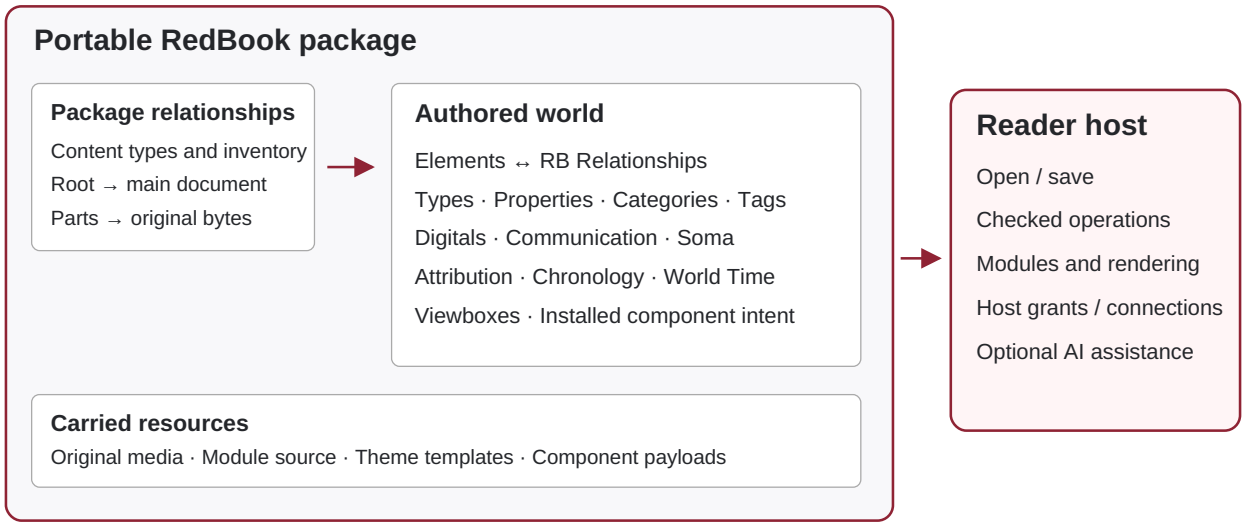
E.5 A portable research-log presentation

E.6 A new kind of authored behavior: Measurement

E.7 Specialized content, deliberate events and reuse

Informative. These examples explain the normative contracts in Parts 1–5 and Annex D. XML excerpts are individual native parts, not complete self-contained RedBook packages. The example subjects, type definitions and document identities are illustrative.

E.1 Package structure and authored meaning



Package metadata, authored content and Reader-host facilities occupy distinct roles.

Package relationships locate parts and resources. RB Relationships are records in the authored world; they express roles among Elements. These two uses of “relationship” have separate namespaces, syntax and meaning. The root package relationship identifies the main document; the main document's part map identifies the native record/registry parts. None of these packaging links grants participation authority.

A read starts with ZIP checks, canonical names and root-relationship discovery. It then validates content types/inventory, native XML and logical value shape, followed by reference/type/domain constraints. Execution is a later explicit host operation. A structurally accepted source archive need not be executable on that host, and a retained unsupported resource need not be discarded.

E.2 A record containing exact data and a reference

The complete excerpt is records.xml. The central Properties are:

```
<rb:properties xmlns:rb="https://redbook.fun/ns/redbook/1.0">
  <rb:map>
    <rb:entry key="measurement"><rb:decimal>1.2300</rb:decimal></rb:entry>
    <rb:entry key="subject"><rb:reference id="urn:example:river"/></rb:entry>
```

```
    <rb:entry key=""><rb:string>A valid empty Property key</rb:string></rb:entry>
  </rb:map>
</rb:properties>
```

The decimal lexeme `1.2300` remains exact authored data. The subject is an explicit record reference and participates in closure/deletion checks; an ordinary string that happens to resemble an identifier does not. The empty key is valid and can be removed by the explicit unset operation. The enclosing Element has Categories and Tags even when their lists are empty. It does not gain Relationship slots.

The full excerpt also retains a source author credit without a local Digital link. That credit does not imply the author is present in the receiving world. A live `digitalId` at the attribution's local-reference position would have different validation and deletion semantics. The recorded UTC time, local sequence and null world-time observation are separate fields. Import assigns destination-local chronology while preserving the recorded instant and descriptive credit.

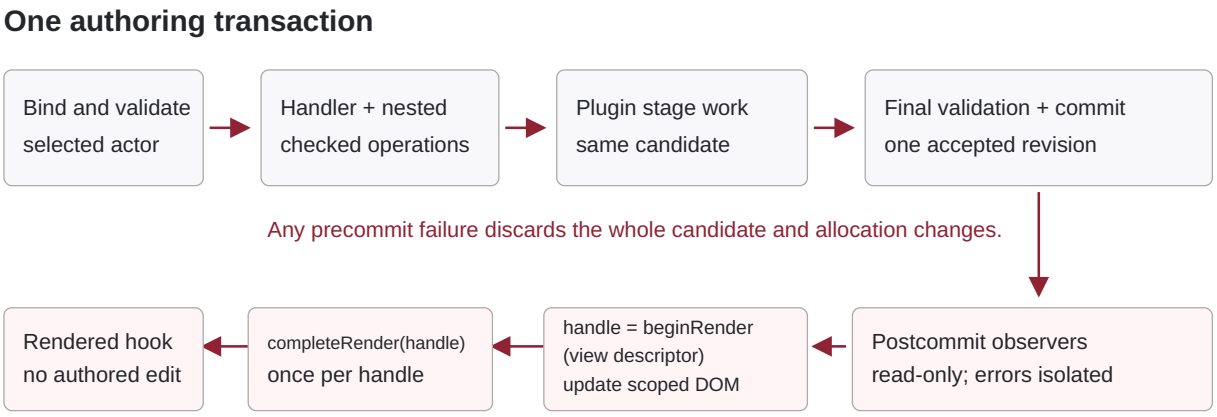
E.3 Protocol-control JSON without changing authored values

The complete control-schema.xml excerpt uses the dedicated control marker:

```
<rb:actions xmlns:rb="https://redbook.fun/ns/redbook/1.0">
  <rb:json>{"exampleSchema":{"type":"object","properties":{
    "$rb":{"const":"reference"},"id":{"type":"string"}
  },"required":["$rb","id"],"additionalProperties":false}}</rb:json>
</rb:actions>
```

Here `$rb` is the literal name of a Property described by a JSON Schema, not a tagged authored value. Native control containers have a defined JSON decoding context. A `rb:json` element is not permitted as an arbitrary value inside an authored Property map. XML character escaping still applies to the JSON text; a literal < must be escaped or represented using JSON's Unicode escape. The excerpt demonstrates encoding only; a complete action declaration also supplies its name, effect, role and full execution contract.

E.4 Atomic extension work and a subsequent render



Dependent authoring shares one transaction; rendering follows an accepted commit.

A participant action can create an item. An authorized `rb.action.stage` contribution can add related conceptual material through the same transaction facade. If that dependent operation fails, both writes and their allocation changes are discarded. Catching the exception inside extension code does not restore a poisoned transaction. On success, one accepted revision is announced. Rendering is separate: it begins a render handle, updates the dedicated DOM and completes that handle once. A duplicate completion returns its existing receipt; it does not emit another rendered hook.

An implementation can verify this sequence by staging a dependent operation that succeeds, then one that fails, and comparing the complete accepted state after each call. Module-resolution behavior is assessed separately from the transaction and render lifecycle.

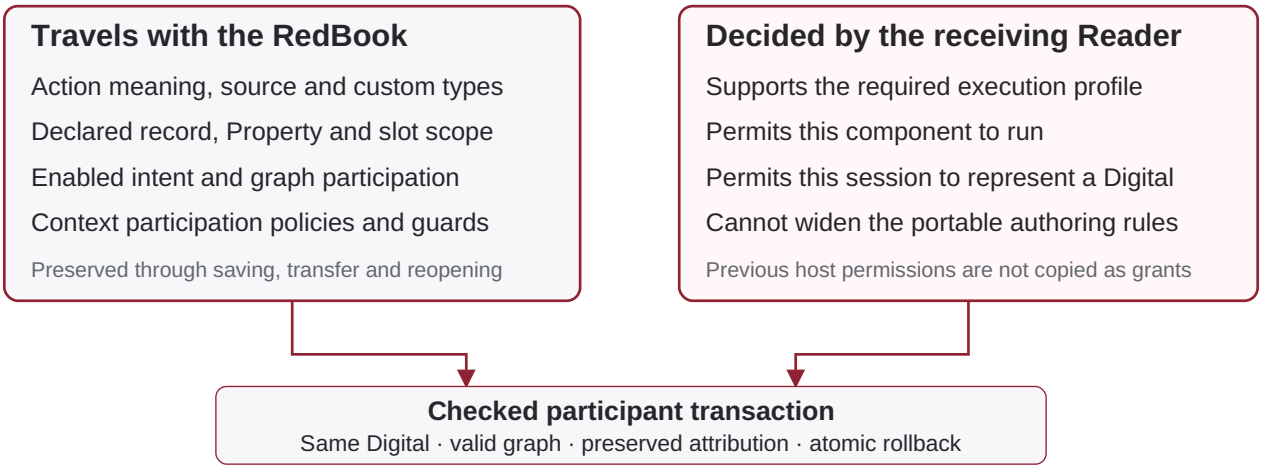
E.5 A portable research-log presentation

The research Wall Theme demonstrates template selection, literal text, record values, ordered iteration and an explicit empty state. Its source JSON and stylesheet are included beside the archive. A template changes how the authorized Wall entries are presented; it does not duplicate the underlying Posts or invent a second publication history. Themes use the same projected data and checked actions as the selected Viewbox.

The Theme and Plugin examples cover Reader, RedBook and Viewbox targets. The definition-library Blueprint demonstrates type-only reuse without creating an Element merely to carry a definition. Neither receiving a package nor reading its manifest activates executable code.

E.6 A new kind of authored behavior: Measurement

The participant-authoring example adds a root CET named Measurement, a root CRT named Measured Subject, valid Calibrated Measurement and Documented Measured Subject specializations, and a Specimen target type. These are ordinary namespaced extension definitions. Neither Measurement nor its action receives a special core exception.



Portable authoring rules travel with the RedBook. Execution permission belongs to the receiving Reader.

The carried authoring declaration describes the exact input/output shapes, custom definitions, context resolver and mutation rules. This example uses explicit `participation.extensionAuthoring` policies. Other actions can use existing owner or active-member participation and choose Property, creator or Relationship scope. The ordinary installation `enabled` flag records desired activation; no additional authoring-adoption flag exists. The host separately chooses whether to execute it and let the session represent the selected Digital.

The actual Plugin module registers its handlers with `api.authoring.register`. Its record handler calls `tx.create('measurement', ...)`, then `tx.create('association', ...)` using the returned identity. The dispatcher supplies the immutable context Reference and creator attribution, checks the custom CET/CRT and participant slots, and commits both records together. A later revise action uses `tx.patch` to change the Measurement and its relationship while retaining their original identities and authorship.

The declaration includes an exercise action for checking failure behavior. It attempts forbidden changes, catches a failed nested mutation, tries an administrative call and supplies an invalid result. These deliberate invalid inputs make the mutation boundary observable. A failed relationship must discard the previously staged Measurement, and an attempted foreign-context edit must leave the entire world unchanged.

To demonstrate persistence, an implementation carries the module and declaration as installed resources, invokes the public authoring interface, and saves the resulting world as a native XML/ZIP package. On reopening, a fresh host loads the retained module and declaration. That host can

refuse execution while preserving the authored data, then separately permit continuation under the same portable rules. Successful results and failure cases are assessed against the complete retained state.

An assessment of this example identifies the resolver, visibility, module and operation behavior it exercises. Selected mutation and package round-trip results establish those cases; a claim covering the complete participant-authoring profile additionally requires the applicable evidence in Part 5 and Annex D.

E.7 Specialized content, deliberate events and reuse

A Laboratory Reading can specialize Post with a required `temperature` Property. Its Digital uses `rb.posts.create` with the specialized type and additional Property, while the standard operation still controls Post body, publication state, attribution and event semantics. A later permitted added-field edit cannot retype the Post, replace its author or bypass its lifecycle. This illustrates why specializations keep their base meaning rather than becoming unrelated roots merely to gain write access.

A Measurement extension may also define a deliberate Experiment Run. Its action calls `tx.recordEvent` rather than ordinary `tx.create`; the dispatcher supplies recording time, world time and the next exact local sequence. A failed later step rolls back the event and its allocation. Editing its content later does not recalculate those observations. Required mutual references can be constructed with `tx.reserveId()` and completed before a public read or commit.

The complete functionality reuse case captures the graph, custom definitions, portable participation and original component archive. Import into a destination returns identity/resource/component mappings and creates a fresh disabled installation. Saving and reopening retain that material. After the destination independently permits execution, the same carried action records a new event using the destination counter. Re-export includes that working component and current graph again. The relevant portability assessment follows this complete cycle and checks the retained definitions, mappings, source bytes, participation rules, attribution and event allocation at each stage.