

# Package: agentr (via r-universe)

July 7, 2026

**Type** Package

**Title** Specification and Review Scaffolding for AI Agent Workflows

**Version** 0.2.8.4

**Description** Specification, review, and scaffolding helpers for AI agent systems. The package standardizes workflow, memory, knowledge, interface, proposal, and review artifacts so humans and coding assistants can infer, inspect, revise, and hand off task designs. It intentionally excludes communication layers, provider-specific model client code, and full runtime execution engines so that design artifacts and implementation transport remain cleanly separated.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** R6, jsonlite, rlang, yaml

**Suggests** DiagrammeR, DiagrammeRsvg, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Repository** <https://oliverlds.r-universe.dev>

**Date/Publication** 2026-06-30 13:50:41 UTC

**RemoteUrl** <https://github.com/oliverlds/agentr>

**RemoteRef** HEAD

**RemoteSha** 58c95f1e3a30f4f1c618b7c714c1f8f848b64ac4

## Contents

add_child_task_node . . . . .	5
AEConfig . . . . .	6
AffectiveConfig . . . . .	7

AffectiveState . . . . .	9
AgentCore . . . . .	11
agentr_workspace_paths . . . . .	13
AgentScaffoldState . . . . .	13
AgentSpec . . . . .	15
append_decision_trace . . . . .	18
append_reflection_trace . . . . .	19
apply_design_feedback . . . . .	19
apply_initial_spec_message . . . . .	20
apply_knowledge_message . . . . .	20
apply_memory_message . . . . .	21
apply_node_detail_message . . . . .	21
apply_revision_message . . . . .	22
apply_scaffolder_message . . . . .	23
approve_workspace_proposal . . . . .	23
article_workflow_specs_from_json . . . . .	24
backup_agent . . . . .	25
build_agent_design_prompt . . . . .	25
build_article_workflow_extraction_prompt . . . . .	26
build_design_review_data . . . . .	27
build_implementation_prompt . . . . .	28
build_initial_spec_prompt . . . . .	29
build_knowledge_conflict_check_prompt . . . . .	29
build_knowledge_design_prompt . . . . .	30
build_knowledge_elicitation_prompt . . . . .	30
build_knowledge_normalization_prompt . . . . .	31
build_memory_revision_prompt . . . . .	31
build_memory_schema_prompt . . . . .	32
build_node_detail_prompt . . . . .	33
build_revision_prompt . . . . .	33
build_scaffolder_prompt . . . . .	34
build_workflow_extraction_prompt . . . . .	35
build_workspace_implementation_prompt . . . . .	36
child_task_node . . . . .	37
CognitiveConfig . . . . .	38
CognitiveState . . . . .	40
collect_scaffolder_questions . . . . .	43
combine_emotions . . . . .	43
compute_blended_emotions . . . . .	44
create_decision_trace . . . . .	44
create_reflection_trace . . . . .	45
decay_emotion_state . . . . .	46
default_emotion_state . . . . .	46
define_random_emotion_state . . . . .	47
describe_emotional_state . . . . .	47
design_feedback_item . . . . .	48
design_review_html . . . . .	49
DesignReviewSpec . . . . .	50

discover_task_specs . . . . .	52
export_design_review_html . . . . .	53
export_workspace_design_review . . . . .	53
IACConfig . . . . .	54
import_extracted_workflow . . . . .	56
inferencer_available . . . . .	57
inferencer_integration . . . . .	57
init_agentr_proposal_states . . . . .	58
init_agentr_workspace . . . . .	58
IntelligentAgent . . . . .	59
knowledge_action_methods . . . . .	61
knowledge_graph_data . . . . .	61
knowledge_graph_from_spec . . . . .	62
KnowledgeProposal . . . . .	62
KnowledgeProposalState . . . . .	64
KnowledgeSpec . . . . .	66
LACConfig . . . . .	68
list_workspace_proposals . . . . .	70
load_agent . . . . .	71
load_agent_spec . . . . .	71
load_design_feedback . . . . .	72
load_design_review_spec . . . . .	72
load_json_file . . . . .	73
load_knowledge_proposal . . . . .	73
load_knowledge_spec . . . . .	74
load_knowledge_spec_json . . . . .	74
load_knowledge_spec_yaml . . . . .	75
load_memory_spec . . . . .	75
load_memory_spec_json . . . . .	76
load_memory_spec_yaml . . . . .	76
load_subsystem_spec . . . . .	77
load_task_specs . . . . .	77
load_workflow_proposal . . . . .	78
load_workflow_spec . . . . .	78
load_workflow_spec_json . . . . .	79
load_workflow_spec_yaml . . . . .	79
load_yaml_file . . . . .	80
mark_node_agent_owned . . . . .	80
mark_node_human_owned . . . . .	81
memory_action_methods . . . . .	81
memory_field . . . . .	82
memory_persistence_policies . . . . .	83
memory_schema_graph_data . . . . .	83
memory_types . . . . .	84
MemoryProposal . . . . .	84
MemoryProposalState . . . . .	87
MemorySpec . . . . .	89
new_design_review_spec . . . . .	92

new_task_family_workflow . . . . .	92
new_workflow_spec . . . . .	93
normalize_subsystem_key . . . . .	94
parse_design_feedback_json . . . . .	94
parse_knowledge_message . . . . .	95
parse_memory_message . . . . .	95
parse_scaffolder_message . . . . .	96
PGConfig . . . . .	96
plot_knowledge_graph . . . . .	98
plot_workflow_graph . . . . .	98
preview_design_feedback . . . . .	99
preview_knowledge_message . . . . .	100
preview_memory_message . . . . .	100
preview_scaffolder_message . . . . .	101
print.agentr_workflow_proposal . . . . .	102
print.agentr_workflow_spec . . . . .	102
read_decision_traces . . . . .	103
read_reflection_traces . . . . .	103
reject_workspace_proposal . . . . .	104
render_knowledge_graphviz . . . . .	104
render_markdown_terminal . . . . .	105
render_memory_schema_graphviz . . . . .	106
render_schema_shape_graphviz . . . . .	107
render_task_preview . . . . .	108
render_task_previews . . . . .	109
render_workflow_graphviz . . . . .	109
RWMConfig . . . . .	110
save_agent . . . . .	112
save_agent_spec . . . . .	113
save_design_feedback . . . . .	113
save_design_review_spec . . . . .	114
save_knowledge_proposal . . . . .	114
save_knowledge_spec . . . . .	115
save_knowledge_spec_json . . . . .	115
save_knowledge_spec_yaml . . . . .	116
save_memory_spec . . . . .	116
save_memory_spec_json . . . . .	117
save_memory_spec_yaml . . . . .	117
save_subsystem_spec . . . . .	118
save_workflow_proposal . . . . .	118
save_workflow_spec . . . . .	119
save_workflow_spec_json . . . . .	119
save_workflow_spec_yaml . . . . .	120
Scaffolder . . . . .	120
scaffolder_action_methods . . . . .	132
schema_shape_graph_data . . . . .	133
set_workflow_node_automation_status . . . . .	133
set_workflow_node_owner . . . . .	134

SubsystemSpec . . . . .	135
task_family_metadata . . . . .	137
task_spec_paths . . . . .	138
terminal_ask_node_complete . . . . .	138
terminal_ask_node_rule . . . . .	139
terminal_ask_workflow_changes . . . . .	139
terminal_discuss_task . . . . .	140
terminal_scaffold_input . . . . .	140
validate_design_feedback . . . . .	141
validate_design_review_spec . . . . .	141
validate_knowledge_item . . . . .	142
validate_knowledge_proposal . . . . .	142
validate_knowledge_spec . . . . .	143
validate_memory_field . . . . .	143
validate_memory_proposal . . . . .	144
validate_memory_spec . . . . .	144
validate_scaffolder_message . . . . .	145
validate_task_specs . . . . .	145
validate_workflow_proposal . . . . .	146
validate_workflow_spec . . . . .	146
workflow_edge . . . . .	147
workflow_graph_data . . . . .	148
workflow_node . . . . .	148
workflow_proposal_graph_data . . . . .	150
workflow_spec_from_json . . . . .	151
workflow_spec_from_yaml . . . . .	151
WorkflowProposal . . . . .	152
WorkflowProposalState . . . . .	155

**Index****158**


---

add\_child\_task\_node    *Add a child task to a task-family workflow*

---

**Description**

Add a child task to a task-family workflow

**Usage**

```
add_child_task_node(workflow, node, tags = character())
```

**Arguments**

workflow	Existing task-family workflow.
node	One-row child-task node data frame.
tags	Optional tags for the child task.

**Value**

Updated task-family workflow.

---

AConfig

*AConfig*

---

**Description**

AConfig

AConfig

**Details**

Configuration for the action-execution subsystem.

**Methods**

`$initialize(enabled = TRUE, execution_mode = "guided", tool_budget = "standard", metadata = list())`  
 Create an action-execution config.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

**Public fields**

`enabled` Whether the subsystem is enabled.

`execution_mode` Execution-mode label.

`tool_budget` Optional tool budget label.

`metadata` Free-form metadata list.

**Methods****Public methods:**

- [AConfig\\$new\(\)](#)
- [AConfig\\$validate\(\)](#)
- [AConfig\\$as\\_list\(\)](#)
- [AConfig\\$print\(\)](#)
- [AConfig\\$clone\(\)](#)

**Method** `new()`: Create an action-execution config.

*Usage:*

```

AEConfig$new(
  enabled = TRUE,
  execution_mode = "guided",
  tool_budget = "standard",
  metadata = list()
)

```

*Arguments:*

enabled Whether the subsystem is enabled.  
 execution\_mode Execution-mode label.  
 tool\_budget Optional tool budget label.  
 metadata Free-form metadata list.

**Method** validate(): Validate the config.

*Usage:*

```
AEConfig$validate()
```

**Method** as\_list(): Return a serializable representation.

*Usage:*

```
AEConfig$as_list()
```

**Method** print(): Print a compact config summary.

*Usage:*

```
AEConfig$print(...)
```

*Arguments:*

... Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
AEConfig$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

AffectiveConfig

*AffectiveConfig*


---

**Description**

AffectiveConfig

AffectiveConfig

**Details**

Lightweight configuration for the affective layer inside RWM.

**Methods**

`$initialize(enabled = TRUE, style = "lightweight", persistence = "session", summary = NULL, metadata = list())` Create an affective-layer config.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

**Public fields**

`enabled` Whether the affective layer is enabled.

`style` Affective modeling style.

`persistence` Persistence mode for affective state.

`summary` Optional one-line summary.

`metadata` Free-form metadata list.

**Methods****Public methods:**

- [AffectiveConfig\\$new\(\)](#)
- [AffectiveConfig\\$validate\(\)](#)
- [AffectiveConfig\\$as\\_list\(\)](#)
- [AffectiveConfig\\$print\(\)](#)
- [AffectiveConfig\\$clone\(\)](#)

**Method** `new()`: Create an affective-layer config.

*Usage:*

```
AffectiveConfig$new(
  enabled = TRUE,
  style = "lightweight",
  persistence = "session",
  summary = NULL,
  metadata = list()
)
```

*Arguments:*

`enabled` Whether the affective layer is enabled.

`style` Affective modeling style.

`persistence` Persistence mode for affective state.

`summary` Optional one-line summary.

`metadata` Free-form metadata list.

**Method** `validate()`: Validate the config.

*Usage:*

```
AffectiveConfig$validate()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*  
 AffectiveConfig\$as\_list()

**Method print():** Print a compact config summary.

*Usage:*  
 AffectiveConfig\$print(...)

*Arguments:*  
 ... Unused print arguments.

**Method clone():** The objects of this class are cloneable with this method.

*Usage:*  
 AffectiveConfig\$clone(deep = FALSE)

*Arguments:*  
 deep Whether to make a deep clone.

---

 AffectiveState

*AffectiveState*


---

## Description

AffectiveState  
 AffectiveState

## Details

Minimal structured affective layer with inertia-aware updates.

## Methods

`$initialize(state = default_emotion_state())` Create an affective state container.  
`$decay(current_time = Sys.time())` Apply time-based decay to the stored affective state.  
`$update_primary(updates)` Blend named primary-emotion updates into the current state using inertia.  
`$describe(threshold = 0.2, include_bleneded = TRUE, method = "geometric")` Return a natural-language description of the current affective state.  
`$as_list()` Return the raw underlying affective-state list.

## Public fields

`state` A named list returned by `default_emotion_state()`.

**Methods****Public methods:**

- [AffectiveState\\$new\(\)](#)
- [AffectiveState\\$decay\(\)](#)
- [AffectiveState\\$update\\_primary\(\)](#)
- [AffectiveState\\$describe\(\)](#)
- [AffectiveState\\$as\\_list\(\)](#)
- [AffectiveState\\$clone\(\)](#)

**Method** `new()`: Create an `AffectiveState` with an initial emotion state.

*Usage:*

```
AffectiveState$new(state = default_emotion_state())
```

*Arguments:*

`state` Affective state used by `$initialize()`.

**Method** `decay()`: Apply time-based decay to the current affective state.

*Usage:*

```
AffectiveState$decay(current_time = Sys.time())
```

*Arguments:*

`current_time` Reference time used by `$decay()`.

**Method** `update_primary()`: Blend named primary-emotion updates into the current affective state.

*Usage:*

```
AffectiveState$update_primary(updates)
```

*Arguments:*

`updates` Named numeric updates used by `$update_primary()`.

**Method** `describe()`: Return a natural-language description of the current affective state.

*Usage:*

```
AffectiveState$describe(
  threshold = 0.2,
  include_blended = TRUE,
  method = "geometric"
)
```

*Arguments:*

`threshold` Threshold used by `$describe()`.

`include_blended` Logical flag used by `$describe()`.

`method` Combination method used by `$describe()`.

**Method** `as_list()`: Return the underlying affective state as a plain list.

*Usage:*

```
AffectiveState$as_list()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AffectiveState$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

AgentCore

*AgentCore*

---

## Description

AgentCore

AgentCore

## Details

Minimal agent container for the agentr cognitive core. An AgentCore combines cognitive and affective state layers and can optionally own a [Scaffolder](#) instance for human-in-the-loop workflow elicitation.

## Methods

`$initialize(id = "agentr-core", name = "agentr", cognition = CognitiveState$new(), affect = AffectiveState$new())` Create a minimal agent container with cognition and affect.

`$attach_scaffolder(scaffolder = NULL)` Attach an existing scaffolder or create a new [Scaffolder](#) owned by the agent.

`$snapshot()` Return a serializable snapshot of the agent's core state.

## Public fields

`id` Agent identifier.

`name` Human-readable agent name.

`cognition` A [CognitiveState](#) instance.

`affect` An [AffectiveState](#) instance.

`scaffolder` Optional [Scaffolder](#) instance.

`metadata` Free-form metadata list.

## Methods

### Public methods:

- [AgentCore\\$new\(\)](#)
- [AgentCore\\$attach\\_scaffolder\(\)](#)
- [AgentCore\\$snapshot\(\)](#)
- [AgentCore\\$clone\(\)](#)

**Method** `new()`: Create an `AgentCore` with cognition, affect, and free-form metadata.

*Usage:*

```
AgentCore$new(  
  id = "agentr-core",  
  name = "agentr",  
  cognition = CognitiveState$new(),  
  affect = AffectiveState$new(),  
  metadata = list()  
)
```

*Arguments:*

`id` Agent identifier used by `$initialize()`.  
`name` Human-readable agent name used by `$initialize()`.  
`cognition` A [CognitiveState](#) instance used by `$initialize()`.  
`affect` An [AffectiveState](#) instance used by `$initialize()`.  
`metadata` Free-form metadata list used by `$initialize()`.

**Method** `attach_scaffolder()`: Attach an existing scaffolder or create a new one owned by this agent.

*Usage:*

```
AgentCore$attach_scaffolder(scaffolder = NULL)
```

*Arguments:*

`scaffolder` Optional [Scaffolder](#) instance used by `$attach_scaffolder()`.

**Method** `snapshot()`: Return a serializable snapshot of the agent core state.

*Usage:*

```
AgentCore$snapshot()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
AgentCore$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

 agentr\_workspace\_paths

*Return standard agentr workspace paths*


---

**Description**

Return standard agentr workspace paths

**Usage**

```
agentr_workspace_paths(workspace)
```

**Arguments**

workspace      Workspace root directory.

**Value**

Named list of workspace paths.

---

 AgentScaffoldState

*AgentScaffoldState*


---

**Description**

AgentScaffoldState

AgentScaffoldState

**Details**

Top-level state container for approved agent designs and nested workflow state.

**Methods**

`$initialize(approved_agent_spec = NULL, proposal_state = list(status = "draft", proposals = list()))`  
 Create an agent scaffold state container.

`$validate()` Validate the state object.

`$set_approved_agent_spec(spec)` Store an approved AgentSpec.

`$approved_workflow()` Return the approved workflow, preferring the approved agent spec when present.

`$as_list()` Return a serializable representation.

**Public fields**

approved\_agent\_spec Current approved AgentSpec or NULL.  
 proposal\_state Free-form proposal lifecycle state list.  
 workflow\_state A WorkflowProposalState object.  
 metadata Free-form metadata list.

**Methods****Public methods:**

- [AgentScaffoldState\\$new\(\)](#)
- [AgentScaffoldState\\$validate\(\)](#)
- [AgentScaffoldState\\$set\\_approved\\_agent\\_spec\(\)](#)
- [AgentScaffoldState\\$approved\\_workflow\(\)](#)
- [AgentScaffoldState\\$as\\_list\(\)](#)
- [AgentScaffoldState#print\(\)](#)
- [AgentScaffoldState\\$clone\(\)](#)

**Method** `new()`: Create an agent scaffold state container.

*Usage:*

```
AgentScaffoldState$new(
  approved_agent_spec = NULL,
  proposal_state = list(status = "draft", proposals = list()),
  workflow_state = WorkflowProposalState$new(),
  metadata = list()
)
```

*Arguments:*

approved\_agent\_spec Current approved AgentSpec or NULL.  
 proposal\_state Free-form proposal lifecycle state list.  
 workflow\_state A WorkflowProposalState object.  
 metadata Free-form metadata list.

**Method** `validate()`: Validate the state object.

*Usage:*

```
AgentScaffoldState$validate()
```

**Method** `set_approved_agent_spec()`: Store an approved AgentSpec.

*Usage:*

```
AgentScaffoldState$set_approved_agent_spec(spec)
```

*Arguments:*

spec Agent spec used by `$set_approved_agent_spec()`.

**Method** `approved_workflow()`: Return the approved workflow.

*Usage:*

AgentScaffoldState\$approved\_workflow()

**Method** as\_list(): Return a serializable representation.

*Usage:*

AgentScaffoldState\$as\_list()

**Method** print(): Print a compact state summary.

*Usage:*

AgentScaffoldState#print(...)

*Arguments:*

... Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

AgentScaffoldState\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

AgentSpec

*AgentSpec*

---

## Description

AgentSpec

AgentSpec

## Details

Public agent-design artifact combining workflow, memory, knowledge, state, interface, and optional subsystem diagnostic labels.

## Methods

\$initialize(task, agent\_name = "agentr-agent", summary = NULL, subsystems = SubsystemSpec\$new(), wo  
Create an agent-design artifact.

\$validate() Validate the agent design.

\$selected\_subsystems() Return the selected subsystem names.

\$workflow\_spec() Return the embedded workflow specification.

\$design\_summary() Return a one-row summary table.

\$as\_list() Return a serializable representation.

\$save(file\_path) Save the object with save\_agent().

**Public fields**

task Source task description.  
 agent\_name Human-readable agent name.  
 summary One-line agent summary.  
 subsystems A SubsystemSpec object.  
 workflow Embedded workflow specification or NULL.  
 knowledge\_spec Embedded KnowledgeSpec or NULL.  
 memory\_spec Embedded MemorySpec or NULL.  
 state\_requirements Free-form list of state requirements.  
 state\_spec Optional structured state-spec list.  
 interfaces Free-form list of interfaces.  
 interface\_spec Optional structured interface-spec list.  
 autonomy\_spec Optional structured autonomy-spec list.  
 autonomy\_stage Optional autonomy-stage label.  
 implementation\_targets Free-form list of implementation targets.  
 metadata Free-form metadata list.

**Methods****Public methods:**

- [AgentSpec\\$new\(\)](#)
- [AgentSpec\\$validate\(\)](#)
- [AgentSpec\\$selected\\_subsystems\(\)](#)
- [AgentSpec\\$workflow\\_spec\(\)](#)
- [AgentSpec\\$design\\_summary\(\)](#)
- [AgentSpec\\$as\\_list\(\)](#)
- [AgentSpec\\$save\(\)](#)
- [AgentSpec\\$print\(\)](#)
- [AgentSpec\\$clone\(\)](#)

**Method** `new()`: Create an agent-design artifact.

*Usage:*

```

AgentSpec$new(
  task,
  agent_name = "agentr-agent",
  summary = NULL,
  subsystems = SubsystemSpec$new(),
  workflow = NULL,
  knowledge_spec = NULL,
  memory_spec = NULL,
  state_requirements = list(),
  state_spec = NULL,

```

```

    interfaces = list(),
    interface_spec = NULL,
    autonomy_spec = NULL,
    autonomy_stage = NULL,
    implementation_targets = list(),
    metadata = list()
)

```

*Arguments:*

**task** Source task description.  
**agent\_name** Human-readable agent name.  
**summary** One-line agent summary.  
**subsystems** A SubsystemSpec object or list payload.  
**workflow** Embedded workflow specification or NULL.  
**knowledge\_spec** Embedded KnowledgeSpec or NULL.  
**memory\_spec** Embedded MemorySpec or NULL.  
**state\_requirements** Free-form list of state requirements.  
**state\_spec** Optional structured state-spec list.  
**interfaces** Free-form list of interfaces.  
**interface\_spec** Optional structured interface-spec list.  
**autonomy\_spec** Optional structured autonomy-spec list.  
**autonomy\_stage** Optional autonomy-stage label.  
**implementation\_targets** Free-form list of implementation targets.  
**metadata** Free-form metadata list.

**Method** `validate()`: Validate the agent design.

*Usage:*

```
AgentSpec$validate()
```

**Method** `selected_subsystems()`: Return the selected subsystem names.

*Usage:*

```
AgentSpec$selected_subsystems()
```

**Method** `workflow_spec()`: Return the embedded workflow specification.

*Usage:*

```
AgentSpec$workflow_spec()
```

**Method** `design_summary()`: Return a one-row summary table.

*Usage:*

```
AgentSpec$design_summary()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*

```
AgentSpec$as_list()
```

**Method** `save()`: Save the object with `save_agent()`.

*Usage:*`AgentSpec$save(file_path)`*Arguments:*`file_path` Output path used by `$save()`.**Method** `print()`: Print a compact agent-design summary.*Usage:*`AgentSpec$print(...)`*Arguments:*`...` Unused print arguments.**Method** `clone()`: The objects of this class are cloneable with this method.*Usage:*`AgentSpec$clone(deep = FALSE)`*Arguments:*`deep` Whether to make a deep clone.

---

`append_decision_trace` *Append a decision trace*

---

**Description**

Append a decision trace

**Usage**`append_decision_trace(trace, path)`**Arguments**

<code>trace</code>	Trace list.
<code>path</code>	JSONL or RDS path.

**Value**

Invisibly returns TRUE.

---

append\_reflection\_trace *Append a reflection trace*

---

**Description**

Append a reflection trace

**Usage**

```
append_reflection_trace(trace, path)
```

**Arguments**

trace	Trace list.
path	JSONL or RDS path.

**Value**

Invisibly returns TRUE.

---

apply\_design\_feedback *Apply structured design feedback*

---

**Description**

Applies feedback through existing scaffolder review/discussion mechanisms when a scaffolder is supplied. Non-workflow feedback is preserved as structured design discussion metadata; it is not auto-executed.

**Usage**

```
apply_design_feedback(x, feedback, review_spec = NULL)
```

**Arguments**

x	A <a href="#">Scaffolder</a> object.
feedback	Feedback item or list of items.
review_spec	Optional review spec used for target-id warnings.

**Value**

The mutated Scaffolder object.

---

apply\_initial\_spec\_message

*Apply an initial LLM response into a workspace proposal state*

---

### Description

Apply an initial LLM response into a workspace proposal state

### Usage

```
apply_initial_spec_message(
    workspace,
    target = c("workflow", "agent", "memory", "knowledge"),
    message,
    comment = NULL
)
```

### Arguments

workspace	Workspace root directory.
target	Target state: workflow, agent, memory, or knowledge.
message	JSON string, parsed list, or path to a JSON response file.
comment	Optional initial task context for workflow or agent targets.

### Value

Mutated state object.

---

apply\_knowledge\_message

*Apply a knowledge message*

---

### Description

Apply a knowledge message

### Usage

```
apply_knowledge_message(state, message)
```

### Arguments

state	A <a href="#">KnowledgeProposalState</a> object.
message	Parsed or raw knowledge message.

**Value**

Mutated state object.

---

apply\_memory\_message    *Apply a memory message*

---

**Description**

Apply a memory message

**Usage**

```
apply_memory_message(state, message)
```

**Arguments**

state	A <a href="#">MemoryProposalState</a> object.
message	Parsed or raw memory message.

**Value**

Mutated state object.

---

apply\_node\_detail\_message  
*Apply a node-detail LLM response into a workspace workflow proposal*

---

**Description**

This is a convenience wrapper for `apply_revision_message(target = "workflow", node_id = ...)`. It previews the proposed node schema or nested workflow edits and stores them as a workflow proposal; approved workflow state is not mutated until the proposal is explicitly approved.

**Usage**

```
apply_node_detail_message(workspace, node_id, message, agent_spec_path = NULL)
```

**Arguments**

workspace	Workspace root directory.
node_id	Workflow node id to revise.
message	JSON string, parsed list, or path to a JSON response file.
agent_spec_path	Optional path to approved <a href="#">AgentSpec</a> .rds.

**Value**

Preview result.

---

apply\_revision\_message

*Apply a revision LLM response into a workspace proposal state*

---

**Description**

Workflow revisions are previewed and stored as proposals; approved specs are not mutated by this function.

**Usage**

```
apply_revision_message(  
  workspace,  
  target = c("workflow", "agent", "memory", "knowledge"),  
  message,  
  agent_spec_path = NULL,  
  node_id = NULL  
)
```

**Arguments**

workspace	Workspace root directory.
target	Target state: workflow, agent, memory, or knowledge.
message	JSON string, parsed list, or path to a JSON response file.
agent_spec_path	Optional path to approved <a href="#">AgentSpec</a> .rds.
node_id	Optional workflow node id. When supplied for workflow targets, only node-detail actions for this node are accepted.

**Value**

Mutated state object or preview result.

---

`apply_scaffolder_message`*Apply a machine-readable scaffolder message*

---

**Description**

Parses and dispatches a machine-readable scaffolder message into concrete calls on a [Scaffolder](#) instance.

**Usage**

```
apply_scaffolder_message(  
  scaffolder,  
  message,  
  allowed_methods = scaffolder_action_methods(),  
  stop_on_error = TRUE  
)
```

**Arguments**

<code>scaffolder</code>	A <a href="#">Scaffolder</a> instance.
<code>message</code>	Parsed message list, JSON string, or path to a downloaded <code>.json</code> file.
<code>allowed_methods</code>	Character vector of allowed method names.
<code>stop_on_error</code>	Whether to stop on the first action error. When <code>FALSE</code> , errors are collected in the returned result object.

**Value**

A standardized list with `applied_actions`, `workflow_after`, `human_prompts`, and `errors`.

---

`approve_workspace_proposal`*Approve a workspace proposal*

---

**Description**

Approve a workspace proposal

**Usage**

```
approve_workspace_proposal(  
  workspace,  
  type = c("workflow", "agent", "memory", "knowledge"),  
  proposal_id,  
  note = NULL,  
  agent_spec_path = NULL  
)
```

**Arguments**

workspace	Workspace root directory.
type	Proposal type: workflow, agent, memory, or knowledge.
proposal_id	Proposal identifier.
note	Optional approval note.
agent_spec_path	Optional path to approved <a href="#">AgentSpec</a> .rds.

**Value**

Approved proposal or spec object.

---

article\_workflow\_specs\_from\_json

*Build workflow specifications from article extraction JSON*

---

**Description**

Converts the article-level JSON object produced from [build\\_article\\_workflow\\_extraction\\_prompt\(\)](#) into one validated workflow specification per element of workflows.

**Usage**

```
article_workflow_specs_from_json(x)
```

**Arguments**

x	Parsed list, raw JSON string, or path to a .json file.
---	--

**Value**

A named list of validated workflow specifications.

---

backup_agent	<i>Backup an agentr object with a timestamped filename</i>
--------------	--

---

**Description**

Saves a timestamped backup of an agentr core object to a specified directory.

**Usage**

```
backup_agent(agent, dir)
```

**Arguments**

agent	An object created by agentr.
dir	Backup directory. Must be supplied explicitly.

**Value**

Invisibly returns the backup file path.

---

build_agent_design_prompt	<i>Build an LLM prompt for agent design decisions</i>
---------------------------	---

---

**Description**

Creates a prompt that targets subsystem-first agent design while keeping the workflow as a nested component inside the proposed agent specification.

**Usage**

```
build_agent_design_prompt(scaffolder, format = "json")
```

**Arguments**

scaffolder	A <a href="#">Scaffolder</a> instance.
format	Prompt payload format. Use "json" or "markdown".

**Value**

Character string prompt.

---

```
build_article_workflow_extraction_prompt
```

*Build a workflow extraction prompt from an article*

---

### Description

Creates a prompt for a reasoning model to infer one or more agentr-compatible workflow specifications from an article describing agentic AI application cases, demonstrations, or methods.

### Usage

```
build_article_workflow_extraction_prompt(
    article_context,
    article_title = NULL,
    task = NULL,
    case_names = NULL,
    extraction_mode = "both",
    format = "json",
    target_agent = "reasoning_model",
    extraction_goal =
        "Infer agentr workflow specs from article-described application cases.",
    constraints = character(),
    extra_context = list()
)
```

### Arguments

article_context	Character string or character vector containing the article text, excerpt, URL, abstract, notes, or section summaries.
article_title	Optional article title.
task	Optional task summary for the extraction.
case_names	Optional case names to prioritize when extracting workflows.
extraction_mode	Extraction mode: "case_workflows", "global_workflow", or "both".
format	Prompt payload format. Use "json" for SDK-facing structured payloads and "markdown" for prompts pasted into a reasoning-model chat interface.
target_agent	Target reasoning agent label.
extraction_goal	Optional extraction goal note.
constraints	Optional character vector of extraction constraints.
extra_context	Optional named list of additional context.

### Value

Character string prompt.

---

 build\_design\_review\_data

*Build design-review data*


---

### Description

Packages an agent design and optional proposal states into a stable, JSON-ready review bundle. This prepares the data contract for a future JS/HTML review interface; it does not render a UI.

### Usage

```
build_design_review_data(
  x = NULL,
  workflow = NULL,
  memory_spec = NULL,
  knowledge_spec = NULL,
  graph_spec = NULL,
  workflow_state = NULL,
  knowledge_state = NULL,
  memory_state = NULL,
  proposal_states = list(),
  review_id = .design_review_id(),
  metadata = list()
)
```

### Arguments

x	Optional <a href="#">AgentSpec</a> , <a href="#">IntelligentAgent</a> , <a href="#">Scaffolder</a> , <a href="#">agentr_workflow_spec</a> , <a href="#">WorkflowProposal</a> , or <a href="#">KnowledgeSpec</a> .
workflow	Optional workflow spec overriding the workflow inferred from x.
memory_spec	Optional <a href="#">MemorySpec</a> overriding memory inferred from x.
knowledge_spec	Optional <a href="#">KnowledgeSpec</a> overriding knowledge inferred from x.
graph_spec	Optional plain graph representation overriding graph knowledge inferred from knowledge_spec.
workflow_state	Optional <a href="#">WorkflowProposalState</a> .
knowledge_state	Optional <a href="#">KnowledgeProposalState</a> .
memory_state	Optional <a href="#">MemoryProposalState</a> .
proposal_states	Additional named proposal-state snapshots.
review_id	Optional review bundle id.
metadata	Additional metadata list.

### Value

A [DesignReviewSpec](#) object.

---

```
build_implementation_prompt
```

*Build an implementation prompt for a coding agent*

---

### Description

Creates a second-stage prompt that turns workflow scaffolding output into an implementation-oriented handoff for a coding assistant.

### Usage

```
build_implementation_prompt(
  x,
  language,
  format = "json",
  target_agent = "coding_assistant",
  runtime = NULL,
  style = NULL,
  constraints = character(),
  extra_context = list(),
  include_knowledge = TRUE,
  knowledge_scope = c("referenced", "approved", "all")
)
```

### Arguments

x	A <a href="#">Scaffolder</a> instance, workflow specification, or implementation-spec-like list. AgentSpec and IntelligentAgent inputs are also supported.
language	Target implementation language, for example "R" or "Python".
format	Prompt payload format. Use "json" for SDK-facing structured payloads and "markdown" for prompts that a human may paste into a coding chat interface.
target_agent	Target coding assistant label.
runtime	Optional runtime or framework note.
style	Optional implementation style note.
constraints	Optional character vector of implementation constraints.
extra_context	Optional named list of additional context.
include_knowledge	Whether approved knowledge should be included in the implementation handoff when available.
knowledge_scope	Knowledge-selection scope when include_knowledge is TRUE: referenced items only, all approved items, or all items.

### Value

Character string prompt.

---

build\_initial\_spec\_prompt  
*Build an initial design prompt into a workspace*

---

**Description**

Build an initial design prompt into a workspace

**Usage**

```
build_initial_spec_prompt(  
    workspace,  
    target = c("workflow", "agent", "memory", "knowledge"),  
    comment,  
    out = NULL,  
    format = c("markdown", "json")  
)
```

**Arguments**

workspace	Workspace root directory.
target	Prompt target: workflow, agent, memory, or knowledge.
comment	Natural-language task or design context.
out	Optional output file path.
format	Prompt payload format.

**Value**

Output prompt path.

---

build\_knowledge\_conflict\_check\_prompt  
*Build a knowledge conflict-check prompt*

---

**Description**

Build a knowledge conflict-check prompt

**Usage**

```
build_knowledge_conflict_check_prompt(  
    knowledge_spec = NULL,  
    candidate,  
    format = c("markdown", "json")  
)
```

**Arguments**

knowledge\_spec Existing [KnowledgeSpec](#) or NULL.  
 candidate Proposed knowledge item.  
 format Output format, "markdown" or "json".

**Value**

Prompt string.

---

build\_knowledge\_design\_prompt  
*Build a knowledge design prompt*

---

**Description**

Build a knowledge design prompt

**Usage**

```
build_knowledge_design_prompt(knowledge_state, format = c("markdown", "json"))
```

**Arguments**

knowledge\_state  
 A [KnowledgeProposalState](#) object.  
 format Output format, "markdown" or "json".

**Value**

Prompt string.

---

build\_knowledge\_elicitation\_prompt  
*Build a knowledge elicitation prompt*

---

**Description**

Build a knowledge elicitation prompt

**Usage**

```
build_knowledge_elicitation_prompt(  

  context = NULL,  

  format = c("markdown", "json")  

)
```

**Arguments**

context	Optional context text.
format	Output format, "markdown" or "json".

**Value**

Prompt string.

---

*build\_knowledge\_normalization\_prompt*  
*Build a knowledge normalization prompt*

---

**Description**

Build a knowledge normalization prompt

**Usage**

```
build_knowledge_normalization_prompt(  
    raw_statement,  
    format = c("markdown", "json")  
)
```

**Arguments**

raw_statement	Raw human knowledge statement.
format	Output format, "markdown" or "json".

**Value**

Prompt string.

---

*build\_memory\_revision\_prompt*  
*Build a memory revision prompt*

---

**Description**

Build a memory revision prompt

**Usage**

```
build_memory_revision_prompt(  
    memory_state,  
    feedback = NULL,  
    format = c("markdown", "json")  
)
```

**Arguments**

memory_state	A <a href="#">MemoryProposalState</a> object.
feedback	Optional human feedback text or structured list.
format	Output format, "markdown" or "json".

**Value**

Prompt string.

---

build\_memory\_schema\_prompt  
*Build a memory schema prompt*

---

**Description**

Build a memory schema prompt

**Usage**

```
build_memory_schema_prompt(  
    context = NULL,  
    current_memory = NULL,  
    format = c("markdown", "json")  
)
```

**Arguments**

context	Optional context text.
current_memory	Optional <a href="#">MemorySpec</a> or NULL.
format	Output format, "markdown" or "json".

**Value**

Prompt string.

---

`build_node_detail_prompt`*Build a node-detail revision prompt*

---

**Description**

Creates a constrained prompt for revising only one workflow node's interface schema or nested workflow detail. The expected response uses `set_node_schema()` and/or `set_node_nested_workflow()` actions.

**Usage**

```
build_node_detail_prompt(  
  workflow,  
  node_id,  
  include_nested_workflow = TRUE,  
  feedback = NULL,  
  format = c("json", "markdown")  
)
```

**Arguments**

<code>workflow</code>	Workflow spec containing the target node.
<code>node_id</code>	Workflow node id to revise.
<code>include_nested_workflow</code>	Whether to include existing nested workflow payload in the prompt.
<code>feedback</code>	Optional human revision feedback.
<code>format</code>	Prompt payload format. Use "json" or "markdown".

**Value**

Character string prompt.

---

`build_revision_prompt` *Build a revision prompt into a workspace*

---

**Description**

Build a revision prompt into a workspace

**Usage**

```

build_revision_prompt(
  workspace,
  target = c("workflow", "agent", "memory", "knowledge"),
  comment,
  out = NULL,
  agent_spec_path = NULL,
  node_id = NULL,
  format = c("markdown", "json")
)

```

**Arguments**

workspace	Workspace root directory.
target	Revision target: workflow, agent, memory, or knowledge.
comment	Human revision feedback.
out	Optional output file path.
agent_spec_path	Optional path to approved <a href="#">AgentSpec</a> .rds.
node_id	Optional workflow node id. When supplied for workflow targets, the prompt is constrained to node schema and nested-workflow revision.
format	Prompt payload format.

**Value**

Output prompt path.

---

build\_scaffolder\_prompt

*Build an LLM prompt for scaffolding decisions*

---

**Description**

Creates a prompt that describes the scaffolder's available methods, the task context, the current workflow state, and the required machine-readable JSON response format.

**Usage**

```

build_scaffolder_prompt(scaffolder, task = NULL, format = "json")

```

**Arguments**

scaffolder	A <a href="#">Scaffolder</a> instance.
task	Optional task text. Defaults to the scaffolder's current task.
format	Prompt payload format. Use "json" for SDK-facing structured payloads and "markdown" for prompts that a human may paste into a chat interface.

**Value**

Character string prompt.

---

```
build_workflow_extraction_prompt
```

*Build a workflow extraction prompt from existing code*

---

**Description**

Creates a prompt for a reasoning model to infer an agentr-compatible workflow specification from ad hoc code, scripts, or module summaries that already exist.

**Usage**

```
build_workflow_extraction_prompt(
    code_context,
    task = NULL,
    language = NULL,
    format = "json",
    target_agent = "reasoning_model",
    extraction_goal = "Infer a workflow specification consistent with agentr.",
    constraints = character(),
    extra_context = list()
)
```

**Arguments**

code_context	Character string or character vector describing the existing code, snippets, file summaries, or execution flow to inspect.
task	Optional task summary associated with the code.
language	Optional source-code language, for example "R" or "Python".
format	Prompt payload format. Use "json" for SDK-facing structured payloads and "markdown" for prompts pasted into a reasoning-model chat interface.
target_agent	Target reasoning agent label.
extraction_goal	Optional extraction goal note.
constraints	Optional character vector of extraction constraints.
extra_context	Optional named list of additional context.

**Value**

Character string prompt.

---

```
build_workspace_implementation_prompt
```

*Build an implementation handoff prompt from workspace artifacts*

---

### Description

This creates a prompt for a coding assistant or implementation team. It does not execute the approved design.

### Usage

```
build_workspace_implementation_prompt(
  workspace,
  agent_spec_path = NULL,
  out = NULL,
  language = "R",
  target_agent = "coding_assistant",
  runtime = NULL,
  style = NULL,
  constraints = character(),
  include_knowledge = TRUE,
  knowledge_scope = c("referenced", "approved", "all"),
  format = c("markdown", "json")
)
```

### Arguments

workspace	Workspace root directory.
agent_spec_path	Optional path to approved <code>AgentSpec</code> .rds.
out	Optional output prompt path.
language	Target implementation language.
target_agent	Target implementation agent.
runtime	Optional runtime note.
style	Optional implementation style note.
constraints	Character vector of implementation constraints.
include_knowledge	Include approved knowledge in the prompt.
knowledge_scope	Knowledge inclusion scope.
format	Prompt format.

### Value

Output prompt path.

---

child_task_node	<i>Create a child-task workflow node</i>
-----------------	--

---

## Description

Creates a workflow node that represents one child task inside a parent task-family workflow. The child task can point to a saved workflow through `subworkflow_ref` and/or embed a reviewable `nested_workflow`.

## Usage

```
child_task_node(
  id,
  label,
  subworkflow_ref = NA_character_,
  nested_workflow = NULL,
  input_schema = list(),
  output_schema = list(),
  human_required = TRUE,
  owner = "human",
  automation_status = "human_in_loop",
  target_automation_status = NA_character_,
  implementation_hint = NA_character_,
  rule_spec = NA_character_,
  knowledge_refs = character(),
  trace_required = NA
)
```

## Arguments

<code>id</code>	Child-task node id.
<code>label</code>	Child-task label.
<code>subworkflow_ref</code>	Optional reference to a saved child workflow.
<code>nested_workflow</code>	Optional embedded child workflow.
<code>input_schema</code>	Structured input schema for the child task.
<code>output_schema</code>	Structured output schema for the child task.
<code>human_required</code>	Whether the child task requires human review.
<code>owner</code>	Current child-task owner.
<code>automation_status</code>	Current child-task automation status.
<code>target_automation_status</code>	Target automation status.

implementation\_hint      Optional implementation hint.  
 rule\_spec                  Optional child-task rule.  
 knowledge\_refs      Character vector of related knowledge ids.  
 trace\_required      Whether trace collection is required.

**Value**

One-row workflow node data frame.

---

CognitiveConfig	<i>CognitiveConfig</i>
-----------------	------------------------

---

**Description**

CognitiveConfig

CognitiveConfig

**Details**

Lightweight configuration for the cognitive layer inside RWM.

**Methods**

`$initialize(enabled = TRUE, persistence = "session", memory_types = character(), summary = NULL, me`  
 Create a cognitive-layer config.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

**Public fields**

`enabled` Whether the cognitive layer is enabled.

`persistence` Persistence mode for cognitive state.

`memory_types` Character vector of memory categories to keep.

`summary` Optional one-line summary.

`metadata` Free-form metadata list.

## Methods

### Public methods:

- [CognitiveConfig\\$new\(\)](#)
- [CognitiveConfig\\$validate\(\)](#)
- [CognitiveConfig\\$as\\_list\(\)](#)
- [CognitiveConfig\\$print\(\)](#)
- [CognitiveConfig\\$clone\(\)](#)

**Method** `new()`: Create a cognitive-layer config.

*Usage:*

```
CognitiveConfig$new(  
  enabled = TRUE,  
  persistence = "session",  
  memory_types = character(),  
  summary = NULL,  
  metadata = list()  
)
```

*Arguments:*

`enabled` Whether the cognitive layer is enabled.

`persistence` Persistence mode for cognitive state.

`memory_types` Character vector of memory categories to keep.

`summary` Optional one-line summary.

`metadata` Free-form metadata list.

**Method** `validate()`: Validate the config.

*Usage:*

```
CognitiveConfig$validate()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*

```
CognitiveConfig$as_list()
```

**Method** `print()`: Print a compact config summary.

*Usage:*

```
CognitiveConfig$print(...)
```

*Arguments:*

`...` Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
CognitiveConfig$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

CognitiveState

*CognitiveState*


---

### Description

CognitiveState

CognitiveState

### Details

Minimal structured cognitive layer for agent state representation. This class intentionally provides only a lightweight API for 0.1.3. Its `bayes_update()` method is a placeholder interface rather than a full inference engine.

### Methods

`$initialize(beliefs = list(), knowledge = list(), goals = list(), task_context = list(), confidence = list())` Create a lightweight cognitive state container.

`$set_belief(name, value, confidence = NULL)` Store or update a named belief and optional confidence value.

`$add_knowledge(entry, label = NULL)` Append a knowledge record with timestamped provenance.

`$set_goal(id, description, status = "proposed")` Store or update a goal record.

`$set_context(...)` Merge named task-context fields into the current cognitive state.

`$bayes_update(target, evidence, prior = NULL, note = NULL)` Record a placeholder Bayesian-style update artifact.

`$as_list()` Return the cognitive state as a plain list.

`$record_update(type, key, value, confidence = NULL)` Append a structured update record to the update log.

### Public fields

`beliefs` Named list of beliefs.

`knowledge` List of observations, notes, or external facts.

`goals` List of goal records.

`task_context` Free-form task context list.

`confidence` Named numeric vector of confidence scores.

`update_log` List of update events.

**Methods****Public methods:**

- `CognitiveState$new()`
- `CognitiveState$set_belief()`
- `CognitiveState$add_knowledge()`
- `CognitiveState$set_goal()`
- `CognitiveState$set_context()`
- `CognitiveState$bayes_update()`
- `CognitiveState$as_list()`
- `CognitiveState$record_update()`
- `CognitiveState$clone()`

**Method** `new()`: Create a `CognitiveState` with beliefs, knowledge, goals, and context.

*Usage:*

```
CognitiveState$new(
  beliefs = list(),
  knowledge = list(),
  goals = list(),
  task_context = list(),
  confidence = numeric(),
  update_log = list()
)
```

*Arguments:*

`beliefs` Named list used by `$initialize()`.

`knowledge` List used by `$initialize()` and `$add_knowledge()`.

`goals` Goal list used by `$initialize()`.

`task_context` Task context list used by `$initialize()`.

`confidence` Confidence vector used by `$initialize()` and `$set_belief()`.

`update_log` Update log used by `$initialize()`.

**Method** `set_belief()`: Store or update a named belief and optional confidence value.

*Usage:*

```
CognitiveState$set_belief(name, value, confidence = NULL)
```

*Arguments:*

`name` Belief name used by `$set_belief()`.

`value` Belief or update value used by `$set_belief()` and `$record_update()`.

`confidence` Confidence vector used by `$initialize()` and `$set_belief()`.

**Method** `add_knowledge()`: Append a timestamped knowledge record to the cognitive state.

*Usage:*

```
CognitiveState$add_knowledge(entry, label = NULL)
```

*Arguments:*

`entry` Knowledge entry used by `$add_knowledge()`.

label Optional knowledge label used by \$add\_knowledge().

**Method** set\_goal(): Store or update a structured goal record.

*Usage:*

CognitiveState\$set\_goal(id, description, status = "proposed")

*Arguments:*

id Goal identifier used by \$set\_goal().

description Goal description used by \$set\_goal().

status Goal status used by \$set\_goal().

**Method** set\_context(): Merge named task-context fields into the current state.

*Usage:*

CognitiveState\$set\_context(...)

*Arguments:*

... Named task-context updates used by \$set\_context().

**Method** bayes\_update(): Record a placeholder Bayesian-style update artifact.

*Usage:*

CognitiveState\$bayes\_update(target, evidence, prior = NULL, note = NULL)

*Arguments:*

target Update target used by \$bayes\_update().

evidence Evidence payload used by \$bayes\_update().

prior Optional prior payload used by \$bayes\_update().

note Optional note used by \$bayes\_update().

**Method** as\_list(): Return the cognitive state as a plain list.

*Usage:*

CognitiveState\$as\_list()

**Method** record\_update(): Append a structured update event to the update log.

*Usage:*

CognitiveState\$record\_update(type, key, value, confidence = NULL)

*Arguments:*

type Update type used by \$record\_update().

key Update key used by \$record\_update().

value Belief or update value used by \$set\_belief() and \$record\_update().

confidence Confidence vector used by \$initialize() and \$set\_belief().

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

CognitiveState\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

collect\_scaffolder\_questions *Collect human-facing questions from scaffolding output*

---

**Description**

Extracts pending human questions from a standardized dispatch result or, if no dispatch result is supplied, from the scaffolder interaction log.

**Usage**

```
collect_scaffolder_questions(scaffolder, dispatch_result = NULL)
```

**Arguments**

scaffolder      A [Scaffolder](#) instance.  
dispatch\_result      Optional result object returned by [apply\\_scaffolder\\_message\(\)](#).

**Value**

Data frame of human-facing prompts.

---

combine\_emotions      *Combine two emotion values*

---

**Description**

Combine two emotion values

**Usage**

```
combine_emotions(a, b, method = "geometric", w1 = 0.5, w2 = 0.5)
```

**Arguments**

a                      First value.  
b                      Second value.  
method                Combination method.  
w1                     Weight for a when method = "weighted".  
w2                     Weight for b when method = "weighted".

**Value**

Numeric scalar.

---

compute\_blended\_emotions

*Compute blended emotions from primary emotions*

---

### Description

Compute blended emotions from primary emotions

### Usage

```
compute_blended_emotions(primary, method = "geometric")
```

### Arguments

primary	Named numeric vector of primary emotions.
method	Combination method passed to <a href="#">combine_emotions()</a> .

### Value

Named list of blended emotions.

---

create\_decision\_trace *Create a decision trace*

---

### Description

Create a decision trace

### Usage

```
create_decision_trace(  
  trace_id,  
  agent_id,  
  workflow_node_id,  
  context = list(),  
  human_decision,  
  rationale,  
  outcome = NULL,  
  reflection = NULL,  
  candidate_knowledge_refs = character(),  
  reusable_rule_candidate = TRUE  
)
```

**Arguments**

trace_id	Trace identifier.
agent_id	Agent identifier.
workflow_node_id	Workflow node identifier.
context	Optional context list.
human_decision	Human decision text.
rationale	Decision rationale.
outcome	Optional outcome text.
reflection	Optional reflection text.
candidate_knowledge_refs	Optional candidate knowledge ids.
reusable_rule_candidate	Whether this trace suggests a reusable rule.

**Value**

Trace list.

---

create\_reflection\_trace  
*Create a reflection trace*

---

**Description**

Create a reflection trace

**Usage**

```
create_reflection_trace(
  trace_id,
  agent_id,
  workflow_node_id,
  reflection,
  outcome = NULL
)
```

**Arguments**

trace_id	Trace identifier.
agent_id	Agent identifier.
workflow_node_id	Workflow node identifier.
reflection	Reflection text.
outcome	Optional outcome text.

**Value**

Trace list.

---

decay\_emotion\_state    *Apply time-based decay to an affective state*

---

**Description**

Apply time-based decay to an affective state

**Usage**

```
decay_emotion_state(emotion_state, current_time = Sys.time())
```

**Arguments**

emotion\_state    State list created by [default\\_emotion\\_state\(\)](#) or [define\\_random\\_emotion\\_state\(\)](#).  
current\_time    Reference time.

**Value**

Updated affective state list.

---

default\_emotion\_state    *Create a default affective state*

---

**Description**

Initializes a minimal affective state with Plutchik-style primary dimensions, an inertia factor, and a timestamp for time-based decay.

**Usage**

```
default_emotion_state(decay_rate = 0.98, inertia = 0.85)
```

**Arguments**

decay\_rate    Hourly decay rate between 0 and 1.  
inertia    Inertia factor between 0 and 1 for incremental updates.

**Value**

A named list.

---

define\_random\_emotion\_state  
*Create a randomized affective state*

---

**Description**

Create a randomized affective state

**Usage**

```
define_random_emotion_state(  
  total_intensity = 1,  
  sparsity = 0,  
  decay_rate = 0.98,  
  inertia = 0.85  
)
```

**Arguments**

total_intensity	Total sum of primary emotion values.
sparsity	Proportion of primary emotions to zero out.
decay_rate	Hourly decay rate between 0 and 1.
inertia	Inertia factor between 0 and 1 for incremental updates.

**Value**

A named list.

---

describe\_emotional\_state  
*Describe an affective state in natural language*

---

**Description**

Describe an affective state in natural language

**Usage**

```
describe_emotional_state(  
  emotion_state,  
  threshold = 0.2,  
  include_blended = TRUE,  
  method = "geometric"  
)
```

**Arguments**

emotion_state	State list created by <code>default_emotion_state()</code> or <code>define_random_emotion_state()</code> .
threshold	Minimum intensity required for a dominant affect label.
include_blended	Whether to include blended affect.
method	Combination method passed to <code>combine_emotions()</code> .

**Value**

Character string.

---

design\_feedback\_item *Create a structured design-feedback item*

---

**Description**

Feedback items are the machine-readable output expected from a future JS/HTML review layer. They are intentionally structured rather than free text so they can be routed into workflow, memory, or knowledge revision prompts.

**Usage**

```
design_feedback_item(
  target,
  field,
  issue,
  suggestion,
  severity = "medium",
  issue_type = "unclear",
  id = NULL,
  target_id = NULL,
  item_id = NA_character_,
  location = list(),
  status = "open",
  source = "human",
  created_at = Sys.time(),
  metadata = list()
)
```

**Arguments**

target	Review target, such as "workflow_node", "memory_schema", "knowledge_item", or "graph_edge".
field	Field path or semantic field name being reviewed.
issue	Concise issue description.

suggestion	Concise suggested change.
severity	Severity label: low, medium, or high.
issue_type	Issue type.
id	Optional feedback id.
target_id	Optional target identifier, such as a node id or memory-field id.
item_id	Optional target item id, such as a node id or memory-field id.
location	Optional structured location metadata.
status	Feedback status.
source	Feedback source.
created_at	Creation timestamp.
metadata	Additional metadata list.

**Value**

A validated design-feedback item list.

---

design\_review\_html      *Build standalone design-review HTML*

---

**Description**

Creates a standalone, offline HTML/JavaScript review page from a design review bundle or supported design object. The page is review-only: it renders design artifacts and exports structured feedback JSON, but it does not run workflow nodes, call LLM providers, or mutate saved R objects.

**Usage**

```
design_review_html(
  x,
  include_workflow = TRUE,
  include_knowledge = TRUE,
  include_memory_schema = TRUE,
  include_feedback_panel = TRUE,
  self_contained = TRUE,
  title = NULL,
  graph_layout = c("grid", "layered", "swimlane", "process"),
  edge_style = c("curved", "straight", "orthogonal"),
  node_color_theme = c("default", "subsystems"),
  ...
)
```

**Arguments**

x	A <a href="#">DesignReviewSpec</a> or any input accepted by <a href="#">build_design_review_data()</a> .
include_workflow	Whether to render workflow graph information.
include_knowledge	Whether to render narrative and graph-shaped knowledge.
include_memory_schema	Whether to render memory/state/interface schema.
include_feedback_panel	Whether to include the structured feedback form and JSON export controls.
self_contained	Reserved for future asset handling. The current implementation is always self-contained and uses no remote resources.
title	Optional page title.
graph_layout	Workflow graph layout. "grid" preserves the original row/column placement; "layered" places nodes by DAG depth; "swimlane" groups nodes into responsibility lanes; "process" renders loop-heavy workflows as a vertical process spine with side branches.
edge_style	Workflow edge routing style: "straight", "curved", or "orthogonal".
node_color_theme	Initial node-color theme: "default" uses human-gate, deterministic-automation, and external stochastic LLM categories. Parent nodes with nested workflows inherit the most restrictive descendant category in the default theme. "subsystems" uses subsystem tags such as rwm, pg, ae, la, and iac when available.
...	Additional arguments passed to <a href="#">build_design_review_data()</a> when x is not already a <a href="#">DesignReviewSpec</a> .

**Value**

HTML string.

---

DesignReviewSpec	<i>DesignReviewSpec</i>
------------------	-------------------------

---

**Description**

DesignReviewSpec

DesignReviewSpec

**Details**

Data contract for a future JS/HTML human review layer. It packages the current design artifacts into stable sections that can be rendered, commented on, and converted back into structured feedback.

**Methods**

`$initialize(...)` Create a design-review data bundle.  
`$validate()` Validate the bundle sections.  
`$to_list()` Return a JSON-ready list.  
`$print(...)` Print a compact summary.

**Public fields**

`review_id` Review bundle identifier.  
`agent_name` Agent name.  
`task` Source task.  
`generated_at` Bundle creation timestamp.  
`workflow_graph` Workflow graph section.  
`memory_schema` Memory schema section.  
`narrative_knowledge` Narrative knowledge section.  
`graph_knowledge` Graph-shaped knowledge section.  
`proposal_states` Proposal-state snapshots.  
`feedback_schema` Structured feedback schema.  
`metadata` Free-form metadata.

**Methods****Public methods:**

- [DesignReviewSpec\\$new\(\)](#)
- [DesignReviewSpec\\$validate\(\)](#)
- [DesignReviewSpec\\$to\\_list\(\)](#)
- [DesignReviewSpec\\$print\(\)](#)
- [DesignReviewSpec\\$clone\(\)](#)

**Method** `new()`: Create a design-review data bundle.

*Usage:*

```
DesignReviewSpec$new(
  review_id = .design_review_id(),
  agent_name = NA_character_,
  task = NA_character_,
  generated_at = Sys.time(),
  workflow_graph = .workflow_review_section(NULL),
  memory_schema = .memory_review_section(NULL),
  narrative_knowledge = .narrative_knowledge_review_section(NULL),
  graph_knowledge = .graph_knowledge_review_section(NULL),
  proposal_states = list(),
  feedback_schema = .design_review_feedback_schema(),
  metadata = list()
)
```

*Arguments:*

review\_id Review bundle identifier.  
 agent\_name Agent name.  
 task Source task.  
 generated\_at Bundle creation timestamp.  
 workflow\_graph Workflow graph section.  
 memory\_schema Memory schema section.  
 narrative\_knowledge Narrative knowledge section.  
 graph\_knowledge Graph-shaped knowledge section.  
 proposal\_states Proposal-state snapshots.  
 feedback\_schema Structured feedback schema.  
 metadata Free-form metadata.

**Method** validate(): Validate the design-review bundle.

*Usage:*

DesignReviewSpec\$validate()

**Method** to\_list(): Return a JSON-ready list.

*Usage:*

DesignReviewSpec\$to\_list()

**Method** print(): Print a compact summary.

*Usage:*

DesignReviewSpec#print(...)

*Arguments:*

... Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

DesignReviewSpec\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

discover\_task\_specs    *Discover task-local spec files*

---

**Description**

Discover task-local spec files

**Usage**

```
discover_task_specs(task_dir, docs_dir = "docs")
```

**Arguments**

- `task_dir` Task root directory.
- `docs_dir` Documentation/spec directory relative to `task_dir`, or an absolute path.

**Value**

Data frame with spec type, path, and existence flag.

---

`export_design_review_html`  
*Export standalone design-review HTML*

---

**Description**

Export standalone design-review HTML

**Usage**

`export_design_review_html(x, path, ...)`

**Arguments**

- `x` A [DesignReviewSpec](#) or any input accepted by [build\\_design\\_review\\_data\(\)](#).
- `path` Output HTML path.
- `...` Arguments passed to [design\\_review\\_html\(\)](#).

**Value**

Invisibly returns the normalized output path.

---

`export_workspace_design_review`  
*Export workspace design-review HTML*

---

**Description**

Export workspace design-review HTML

**Usage**

```
export_workspace_design_review(
  workspace,
  agent_spec_path = NULL,
  out = NULL,
  title = "agentr design review",
  graph_layout = c("grid", "layered", "swimlane", "process"),
  edge_style = c("curved", "straight", "orthogonal")
)
```

**Arguments**

workspace	Workspace root directory.
agent_spec_path	Optional path to approved <a href="#">AgentSpec</a> .rds.
out	Optional output HTML path.
title	Review title.
graph_layout	Workflow graph layout passed to <a href="#">design_review_html()</a> .
edge_style	Workflow edge style passed to <a href="#">design_review_html()</a> .

**Value**

Output HTML path.

---

IACConfig

*IACConfig*


---

**Description**

IACConfig

IACConfig

**Details**

Configuration for Inter-Agent Communication.

**Methods**

`$initialize(enabled = TRUE, channels = character(), structured_io = TRUE, metadata = list())`

Create an Inter-Agent Communication config.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

**Public fields**

enabled Whether the subsystem is enabled.  
channels Character vector of communication channels.  
structured\_io Whether strongly structured I/O is required.  
metadata Free-form metadata list.

**Methods****Public methods:**

- `IACConfig$new()`
- `IACConfig$validate()`
- `IACConfig$as_list()`
- `IACConfig$print()`
- `IACConfig$clone()`

**Method** `new()`: Create an Inter-Agent Communication config.

*Usage:*

```
IACConfig$new(  
  enabled = TRUE,  
  channels = character(),  
  structured_io = TRUE,  
  metadata = list()  
)
```

*Arguments:*

enabled Whether the subsystem is enabled.  
channels Character vector of communication channels.  
structured\_io Whether strongly structured I/O is required.  
metadata Free-form metadata list.

**Method** `validate()`: Validate the config.

*Usage:*

```
IACConfig$validate()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*

```
IACConfig$as_list()
```

**Method** `print()`: Print a compact config summary.

*Usage:*

```
IACConfig$print(...)
```

*Arguments:*

... Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
IACConfig$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

```
import_extracted_workflow
```

*Import extracted workflow JSON into agentr*

---

**Description**

Imports workflow JSON from a reasoning model into a workflow specification and optionally stores it as a workflow proposal on a [Scaffolder](#).

**Usage**

```
import_extracted_workflow(
  x,
  scaffolder = NULL,
  source = "model",
  notes = NULL,
  store_proposal = !is.null(scaffolder),
  approve = FALSE
)
```

**Arguments**

x	Parsed list, raw JSON string, or path to a .json file.
scaffolder	Optional <a href="#">Scaffolder</a> instance.
source	Proposal source used when storing on a scaffolder.
notes	Optional proposal notes.
store_proposal	Whether to store a workflow proposal when a scaffolder is supplied.
approve	Whether to approve the stored proposal immediately.

**Value**

A workflow specification or a list containing workflow, proposal\_id, and proposal.

---

inferencer\_available *Check whether inferencer is available*

---

**Description**

Check whether inferencer is available

**Usage**

```
inferencer_available()
```

**Value**

Logical scalar.

---

inferencer\_integration  
*Build optional integration metadata for inferencer*

---

**Description**

Returns a lightweight descriptor rather than a duplicated provider client.

**Usage**

```
inferencer_integration(profile = NULL, prompt_template = NULL)
```

**Arguments**

profile           Optional integration profile name.  
prompt\_template   Optional prompt template identifier.

**Value**

Named list.

---

init\_agentr\_proposal\_states

*Initialize proposal-state artifacts for an agentr workspace*

---

### Description

Initialize proposal-state artifacts for an agentr workspace

### Usage

```
init_agentr_proposal_states(workspace, agent_spec_path = NULL)
```

### Arguments

workspace      Workspace root directory.  
agent\_spec\_path      Optional path to an approved [AgentSpec](#) .rds.

### Value

Named list containing initialized state objects.

---

init\_agentr\_workspace *Initialize a generic agentr lifecycle workspace*

---

### Description

Creates workspace-scoped directories for specs, proposal states, prompts, reviews, traces, responses, and handoff prompts. It does not seed domain-specific content.

### Usage

```
init_agentr_workspace(workspace, comment = NULL, create_readme = TRUE)
```

### Arguments

workspace      Workspace root directory.  
comment      Optional workspace note.  
create\_readme      Whether to create a minimal workspace README.

### Value

Named list of created workspace paths.

---

IntelligentAgent	<i>IntelligentAgent</i>
------------------	-------------------------

---

**Description**

IntelligentAgent

IntelligentAgent

**Details**

Runtime-oriented container for an approved agent design.

**Methods**

`$initialize(id = "intelligent-agent", name = NULL, spec, workflow = NULL, subsystems = NULL, runtime_state = NULL)`

Create a runtime-oriented agent container from an AgentSpec.

`$validate()` Validate the runtime container.

`$selected_subsystems()` Return the selected subsystem names.

`$snapshot()` Return a serializable runtime snapshot.

**Public fields**

`id` Runtime identifier.

`name` Human-readable agent name.

`spec` An AgentSpec object.

`workflow` Current workflow specification.

`subsystems` Selected SubsystemSpec object.

`runtime_state` Free-form runtime state list.

`metadata` Free-form metadata list.

**Methods****Public methods:**

- `IntelligentAgent$new()`
- `IntelligentAgent$validate()`
- `IntelligentAgent$selected_subsystems()`
- `IntelligentAgent$snapshot()`
- `IntelligentAgent$print()`
- `IntelligentAgent$clone()`

**Method** `new()`: Create a runtime-oriented agent container from an AgentSpec.

*Usage:*

```
IntelligentAgent$new(  
  id = "intelligent-agent",  
  name = NULL,  
  spec,  
  workflow = NULL,  
  subsystems = NULL,  
  runtime_state = list(),  
  metadata = list()  
)
```

*Arguments:*

`id` Runtime identifier.  
`name` Human-readable agent name.  
`spec` An AgentSpec object.  
`workflow` Current workflow specification.  
`subsystems` Selected SubsystemSpec object.  
`runtime_state` Free-form runtime state list.  
`metadata` Free-form metadata list.

**Method** `validate()`: Validate the runtime container.

*Usage:*

```
IntelligentAgent$validate()
```

**Method** `selected_subsystems()`: Return the selected subsystem names.

*Usage:*

```
IntelligentAgent$selected_subsystems()
```

**Method** `snapshot()`: Return a serializable runtime snapshot.

*Usage:*

```
IntelligentAgent$snapshot()
```

**Method** `print()`: Print a compact runtime summary.

*Usage:*

```
IntelligentAgent$print(...)
```

*Arguments:*

... Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
IntelligentAgent$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`knowledge_action_methods`*List LLM-callable knowledge methods*

---

**Description**

List LLM-callable knowledge methods

**Usage**

```
knowledge_action_methods()
```

**Value**

Character vector of allowed knowledge methods.

---

`knowledge_graph_data` *Build graph data from knowledge, memory, or a graph representation*

---

**Description**

`agentr` treats graph structure as a representation shape rather than as a separate first-class spec. This helper returns graph-ready node and edge data from a [KnowledgeSpec](#), [MemorySpec](#), or plain `list(nodes, edges, metadata)` graph representation.

**Usage**

```
knowledge_graph_data(x)
```

**Arguments**

`x` A [KnowledgeSpec](#), [MemorySpec](#), or plain graph list with nodes and edges.

**Value**

A list with nodes, edges, and metadata.

---

knowledge\_graph\_from\_spec

*Build a graph representation from a knowledge or memory spec*

---

### Description

This is a compatibility-oriented alias for `knowledge_graph_data()`. It no longer returns a separate KnowledgeGraphSpec; graph is now a representation shape embedded in knowledge or memory specs.

### Usage

`knowledge_graph_from_spec(x)`

### Arguments

`x` A `KnowledgeSpec`, `MemorySpec`, or graph representation list.

### Value

A list with graph-ready nodes, edges, and metadata.

---

KnowledgeProposal

*KnowledgeProposal*

---

### Description

KnowledgeProposal

KnowledgeProposal

### Details

Proposal object for one candidate knowledge item.

### Public fields

`id` Proposal identifier.

`item` Proposed knowledge item.

`status` Proposal status.

`notes` Optional notes.

`conflict_report` Optional conflict report.

`history` Lifecycle history.

`metadata` Free-form metadata.

**Methods****Public methods:**

- `KnowledgeProposal$new()`
- `KnowledgeProposal$validate()`
- `KnowledgeProposal$discuss()`
- `KnowledgeProposal$transition()`
- `KnowledgeProposal$approve()`
- `KnowledgeProposal$reject()`
- `KnowledgeProposal$to_list()`
- `KnowledgeProposal$print()`
- `KnowledgeProposal$clone()`

**Method** `new()`: Create a knowledge proposal.

*Usage:*

```
KnowledgeProposal$new(
  item,
  id = if (is.list(item) && !is.null(item$id)) paste0("knowledge_proposal_",
    as.character(item$id)[1]) else "knowledge_proposal_1",
  status = "pending",
  notes = NULL,
  conflict_report = list(),
  history = list(),
  metadata = list(),
  created_at = Sys.time(),
  updated_at = created_at,
  approved_at = as.POSIXct(NA),
  rejected_at = as.POSIXct(NA),
  superseded_by = NA_character_,
  supersedes = NA_character_
)
```

**Method** `validate()`: Validate the proposal.

*Usage:*

```
KnowledgeProposal$validate()
```

**Method** `discuss()`: Append a discussion note.

*Usage:*

```
KnowledgeProposal$discuss(
  note,
  source = "human",
  confidence = NA_character_,
  timestamp = Sys.time()
)
```

**Method** `transition()`: Apply a status transition.

*Usage:*

```
KnowledgeProposal$transition(status, note = NULL, timestamp = Sys.time())
```

**Method** `approve()`: Approve the proposal.

*Usage:*

```
KnowledgeProposal$approve(note = NULL)
```

**Method** `reject()`: Reject the proposal.

*Usage:*

```
KnowledgeProposal$reject(note = NULL)
```

**Method** `to_list()`: Return a serializable representation.

*Usage:*

```
KnowledgeProposal$to_list()
```

**Method** `print()`: Print a compact summary.

*Usage:*

```
KnowledgeProposal$print(...)
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
KnowledgeProposal$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

KnowledgeProposalState

*KnowledgeProposalState*

---

## Description

KnowledgeProposalState

KnowledgeProposalState

## Details

State container for approved knowledge plus active and historical proposals.

## Public fields

`approved_knowledge_spec` Approved [KnowledgeSpec](#).

`proposals` Named list of [KnowledgeProposal](#) objects.

`history` Proposal-state history.

**Methods****Public methods:**

- `KnowledgeProposalState$new()`
- `KnowledgeProposalState$validate()`
- `KnowledgeProposalState$add_proposal()`
- `KnowledgeProposalState$get_proposal()`
- `KnowledgeProposalState$list_proposals()`
- `KnowledgeProposalState$discuss_proposal()`
- `KnowledgeProposalState$approve_proposal()`
- `KnowledgeProposalState$reject_proposal()`
- `KnowledgeProposalState$approved_spec()`
- `KnowledgeProposalState$as_list()`
- `KnowledgeProposalState$clone()`

**Method** `new()`: Create a knowledge proposal state container.

*Usage:*

```
KnowledgeProposalState$new(  
  approved_knowledge_spec = KnowledgeSpec$new(),  
  proposals = list(),  
  history = list()  
)
```

**Method** `validate()`: Validate the state object.

*Usage:*

```
KnowledgeProposalState$validate()
```

**Method** `add_proposal()`: Add a proposal object.

*Usage:*

```
KnowledgeProposalState$add_proposal(proposal)
```

**Method** `get_proposal()`: Return one stored proposal.

*Usage:*

```
KnowledgeProposalState$get_proposal(proposal_id)
```

**Method** `list_proposals()`: List proposals with optional status filtering.

*Usage:*

```
KnowledgeProposalState$list_proposals(status = NULL)
```

**Method** `discuss_proposal()`: Append a discussion note to one proposal.

*Usage:*

```
KnowledgeProposalState$discuss_proposal(proposal_id, note, source = "human")
```

**Method** `approve_proposal()`: Approve one proposal and add its item to approved knowledge.

*Usage:*

KnowledgeProposalState\$approve\_proposal(proposal\_id, note = NULL)

**Method** reject\_proposal(): Reject one proposal.

*Usage:*

KnowledgeProposalState\$reject\_proposal(proposal\_id, note = NULL)

**Method** approved\_spec(): Return the approved knowledge specification.

*Usage:*

KnowledgeProposalState\$approved\_spec()

**Method** as\_list(): Return a serializable representation.

*Usage:*

KnowledgeProposalState\$as\_list()

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

KnowledgeProposalState\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

KnowledgeSpec

*KnowledgeSpec*

---

## Description

KnowledgeSpec

KnowledgeSpec

## Details

Curated domain and epistemic knowledge used to guide agent behavior. `items` stores narrative knowledge items, `graph` stores an optional graph-shaped representation, and `vector_refs` reserves references to external vector stores.

## Methods

`$initialize(items = list(), graph = NULL, vector_refs = list(), metadata = list())`  
Create a knowledge specification.

`$add_item(item)` Add a narrative knowledge item.

`$get_item(id)` Return a narrative knowledge item by id.

`$list_items(type = NULL, domain = NULL)` List narrative knowledge items, optionally filtered.

`$validate()` Validate the knowledge specification.

`$to_list()` Return a serializable list.

`$print(...)` Print a compact summary.

**Public fields**

items Named list of narrative knowledge items.  
graph Optional graph representation list with nodes and edges.  
vector\_refs List of external vector-knowledge references.  
metadata Free-form metadata list.

**Methods****Public methods:**

- KnowledgeSpec\$new()
- KnowledgeSpec\$add\_item()
- KnowledgeSpec\$get\_item()
- KnowledgeSpec\$list\_items()
- KnowledgeSpec\$validate()
- KnowledgeSpec\$to\_list()
- KnowledgeSpec\$print()
- KnowledgeSpec\$clone()

**Method new():** Create a knowledge specification.

*Usage:*

```
KnowledgeSpec$new(  
  items = list(),  
  graph = NULL,  
  vector_refs = list(),  
  metadata = list()  
)
```

*Arguments:*

items List of narrative knowledge items.  
graph Optional graph representation list with nodes and edges.  
vector\_refs List of external vector-knowledge references.  
metadata Free-form metadata list.

**Method add\_item():** Add a knowledge item.

*Usage:*

```
KnowledgeSpec$add_item(item)
```

*Arguments:*

item Knowledge item used by \$add\_item().

**Method get\_item():** Return a knowledge item by id.

*Usage:*

```
KnowledgeSpec$get_item(id)
```

*Arguments:*

id Knowledge item id used by \$get\_item().

**Method** `list_items()`: List knowledge items with optional filters.

*Usage:*

`KnowledgeSpec$list_items(type = NULL, domain = NULL)`

*Arguments:*

`type` Optional knowledge type filter used by `$list_items()`.

`domain` Optional domain filter used by `$list_items()`.

**Method** `validate()`: Validate the knowledge specification.

*Usage:*

`KnowledgeSpec$validate()`

**Method** `to_list()`: Return a serializable representation.

*Usage:*

`KnowledgeSpec$to_list()`

**Method** `print()`: Print a compact summary.

*Usage:*

`KnowledgeSpec$print(...)`

*Arguments:*

`...` Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

`KnowledgeSpec$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

---

LAConfig

*LAConfig*

---

## Description

LAConfig

LAConfig

## Details

Configuration for Learning & Adaptation.

## Methods

`$initialize(enabled = TRUE, learning_mode = "feedback_driven", feedback_sources = character(), pers`  
 Create a learning and adaptation config.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

**Public fields**

enabled Whether the subsystem is enabled.  
learning\_mode Learning-mode label.  
feedback\_sources Character vector of feedback sources.  
persistence Persistence mode for learned artifacts.  
metadata Free-form metadata list.

**Methods****Public methods:**

- [LAConfig\\$new\(\)](#)
- [LAConfig\\$validate\(\)](#)
- [LAConfig\\$as\\_list\(\)](#)
- [LAConfig\\$print\(\)](#)
- [LAConfig\\$clone\(\)](#)

**Method** `new()`: Create a learning and adaptation config.

*Usage:*

```
LAConfig$new(  
  enabled = TRUE,  
  learning_mode = "feedback_driven",  
  feedback_sources = character(),  
  persistence = "session",  
  metadata = list()  
)
```

*Arguments:*

enabled Whether the subsystem is enabled.  
learning\_mode Learning-mode label.  
feedback\_sources Character vector of feedback sources.  
persistence Persistence mode for learned artifacts.  
metadata Free-form metadata list.

**Method** `validate()`: Validate the config.

*Usage:*

```
LAConfig$validate()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*

```
LAConfig$as_list()
```

**Method** `print()`: Print a compact config summary.

*Usage:*

```
LAConfig$print(...)
```

*Arguments:*

. . . Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
LAConfig$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

list\_workspace\_proposals

*List workspace proposals*

---

**Description**

List workspace proposals

**Usage**

```
list_workspace_proposals(  
  workspace,  
  type = c("workflow", "agent", "memory", "knowledge"),  
  status = NULL  
)
```

**Arguments**

workspace	Workspace root directory.
type	Proposal type: workflow, agent, memory, or knowledge.
status	Optional status filter.

**Value**

Data frame summary.

---

load_agent	<i>Load an agentr object from a file</i>
------------	--

---

**Description**

Loads an agentr core object from a saved .rds file.

**Usage**

```
load_agent(file_path)
```

**Arguments**

file\_path      File path from which to load the object.

**Value**

An object created by agentr.

---

load_agent_spec	<i>Load an AgentSpec from a file</i>
-----------------	--------------------------------------

---

**Description**

Loads a saved [AgentSpec](#) object from an .rds file.

**Usage**

```
load_agent_spec(file_path)
```

**Arguments**

file\_path      File path from which to load the object.

**Value**

An [AgentSpec](#) object.

---

load\_design\_feedback    *Load structured design feedback*

---

**Description**

Load structured design feedback

**Usage**

load\_design\_feedback(path)

**Arguments**

path                    Input .rds path.

**Value**

A design-feedback item or list of items.

---

load\_design\_review\_spec  
*Load a design-review specification*

---

**Description**

Load a design-review specification

**Usage**

load\_design\_review\_spec(path)

**Arguments**

path                    Input .rds path.

**Value**

A [DesignReviewSpec](#) object.

---

load_json_file	<i>Load a JSON file</i>
----------------	-------------------------

---

**Description**

Load a JSON file

**Usage**

```
load_json_file(path, simplifyVector = TRUE)
```

**Arguments**

path                    Path to a JSON file.  
simplifyVector    Passed to `jsonlite::fromJSON()`.

**Value**

Parsed JSON content.

---

load_knowledge_proposal	<i>Load a knowledge proposal</i>
-------------------------	----------------------------------

---

**Description**

Load a knowledge proposal

**Usage**

```
load_knowledge_proposal(path)
```

**Arguments**

path                    File path.

**Value**

A `KnowledgeProposal` object.

load\_knowledge\_spec    *Load a knowledge specification*

---

**Description**

Load a knowledge specification

**Usage**

```
load_knowledge_spec(path, format = c("rds", "json", "yaml"))
```

**Arguments**

path	File path.
format	File format, either rds, json, or yaml.

**Value**

A [KnowledgeSpec](#) object.

---

load\_knowledge\_spec\_json  
*Load a knowledge specification from JSON*

---

**Description**

Load a knowledge specification from JSON

**Usage**

```
load_knowledge_spec_json(path)
```

**Arguments**

path	File path.
------	------------

**Value**

A [KnowledgeSpec](#) object.

---

`load_knowledge_spec_yaml`*Load a knowledge specification from YAML*

---

**Description**

Load a knowledge specification from YAML

**Usage**

```
load_knowledge_spec_yaml(path)
```

**Arguments**

path            File path.

**Value**

A [KnowledgeSpec](#) object.

---

`load_memory_spec`*Load a MemorySpec from a file*

---

**Description**

Loads a saved [MemorySpec](#) object from an `.rds`, `.json`, or `.yaml` file.

**Usage**

```
load_memory_spec(file_path, format = c("rds", "json", "yaml"))
```

**Arguments**

file\_path        File path from which to load the object.

format           File format, either `rds`, `json`, or `yaml`.

**Value**

A [MemorySpec](#) object.

---

load\_memory\_spec\_json *Load a MemorySpec from JSON*

---

**Description**

Load a MemorySpec from JSON

**Usage**

```
load_memory_spec_json(file_path)
```

**Arguments**

file\_path      File path from which to load the JSON.

**Value**

A [MemorySpec](#) object.

---

load\_memory\_spec\_yaml *Load a MemorySpec from YAML*

---

**Description**

Load a MemorySpec from YAML

**Usage**

```
load_memory_spec_yaml(file_path)
```

**Arguments**

file\_path      File path from which to load the YAML.

**Value**

A [MemorySpec](#) object.

---

load_subsystem_spec	<i>Load a SubsystemSpec from a file</i>
---------------------	---

---

**Description**

Loads a saved [SubsystemSpec](#) object from an `.rds` file.

**Usage**

```
load_subsystem_spec(file_path)
```

**Arguments**

file_path	File path from which to load the object.
-----------	--

**Value**

A [SubsystemSpec](#) object.

---

load_task_specs	<i>Load task-local specs</i>
-----------------	------------------------------

---

**Description**

Loads conventional task-local YAML specs when present. Missing specs are returned as NULL unless `missing = "error"` is requested.

**Usage**

```
load_task_specs(task_dir, docs_dir = "docs", missing = c("null", "error"))
```

**Arguments**

task_dir	Task root directory.
docs_dir	Documentation/spec directory relative to task_dir, or an absolute path.
missing	How to handle missing spec files.

**Value**

Named list containing loaded specs, path metadata, and discovery manifest.

---

load\_workflow\_proposal

*Load a workflow proposal*

---

### **Description**

Loads a previously saved workflow proposal from an .rds file.

### **Usage**

```
load_workflow_proposal(file_path)
```

### **Arguments**

file\_path      File path from which to load the proposal.

### **Value**

Workflow proposal object.

---

load\_workflow\_spec

*Load a workflow specification*

---

### **Description**

Load a workflow specification

### **Usage**

```
load_workflow_spec(file_path, format = c("rds", "json", "yaml"))
```

### **Arguments**

file\_path      File path from which to load the workflow.

format          File format, either rds, json, or yaml.

### **Value**

Workflow specification.

---

load\_workflow\_spec\_json

*Load a workflow specification from JSON*

---

**Description**

Load a workflow specification from JSON

**Usage**

load\_workflow\_spec\_json(file\_path)

**Arguments**

file\_path      File path from which to load the workflow JSON.

**Value**

Workflow specification.

---

load\_workflow\_spec\_yaml

*Load a workflow specification from YAML*

---

**Description**

Load a workflow specification from YAML

**Usage**

load\_workflow\_spec\_yaml(file\_path)

**Arguments**

file\_path      File path from which to load the workflow YAML.

**Value**

Workflow specification.

---

load_yaml_file	<i>Load a YAML file</i>
----------------	-------------------------

---

**Description**

Load a YAML file

**Usage**

load\_yaml\_file(path)

**Arguments**

path	Path to a YAML file.
------	----------------------

**Value**

Parsed YAML content.

---

mark_node_agent_owned	<i>Mark a workflow node as agent-owned</i>
-----------------------	--

---

**Description**

Mark a workflow node as agent-owned

**Usage**

mark\_node\_agent\_owned(workflow, node\_id)

**Arguments**

workflow	Workflow specification.
node_id	Node identifier.

**Value**

Updated workflow specification.

---

mark\_node\_human\_owned *Mark a workflow node as human-owned*

---

**Description**

Mark a workflow node as human-owned

**Usage**

```
mark_node_human_owned(  
    workflow,  
    node_id,  
    reason,  
    target_automation_status = NULL,  
    trace_required = TRUE  
)
```

**Arguments**

workflow	Workflow specification.
node_id	Node identifier.
reason	Human-owned reason.
target_automation_status	Optional target automation status.
trace_required	Whether traces are required.

**Value**

Updated workflow specification.

---

memory\_action\_methods *List LLM-callable memory methods*

---

**Description**

List LLM-callable memory methods

**Usage**

```
memory_action_methods()
```

**Value**

Character vector of allowed memory methods.

---

memory_field	<i>Create a memory field record</i>
--------------	-------------------------------------

---

**Description**

Create a memory field record

**Usage**

```
memory_field(
  id,
  label,
  memory_type = c("context", "semantic", "episodic", "procedural"),
  description = NA_character_,
  schema = list(),
  persistence = c("session", "cold_start_rds", "jsonl_trace", "external_store", "none"),
  update_policy = list(),
  source = NA_character_,
  review = list(status = "draft"),
  provenance = list(),
  metadata = list()
)
```

**Arguments**

id	Memory field identifier.
label	Human-readable field label.
memory_type	Memory type: context, semantic, episodic, or procedural.
description	Optional field description.
schema	Structured schema constraints for the field.
persistence	Persistence policy.
update_policy	Free-form update-policy description or list.
source	Optional source label.
review	Review metadata list.
provenance	Provenance metadata list.
metadata	Additional metadata list.

**Value**

A validated memory field list.

---

memory\_persistence\_policies  
*Memory persistence policies*

---

**Description**

Memory persistence policies

**Usage**

```
memory_persistence_policies()
```

**Value**

Character vector of supported memory persistence policies.

---

memory\_schema\_graph\_data  
*Convert a MemorySpec into graph-ready data*

---

**Description**

Converts memory fields and their optional schema shapes into graph-ready node and edge data. This is a design-review projection of memory structure, not a runtime memory store.

**Usage**

```
memory_schema_graph_data(x, include_field_schemas = TRUE)
```

**Arguments**

x                   A [MemorySpec](#) object.  
include\_field\_schemas   Whether to include nested schema-shape nodes for each memory field's schema.

**Value**

A list with vertices and edges data frames.

---

memory_types	<i>Memory types</i>
--------------	---------------------

---

**Description**

Memory types

**Usage**

memory\_types()

**Value**

Character vector of supported memory type labels.

---

MemoryProposal	<i>MemoryProposal</i>
----------------	-----------------------

---

**Description**

MemoryProposal

MemoryProposal

**Details**

Proposal object for a candidate memory schema.

**Public fields**

id Proposal identifier.

memory\_spec Proposed [MemorySpec](#).

status Proposal status.

notes Optional notes.

history Lifecycle history.

metadata Free-form metadata.

created\_at Creation timestamp.

updated\_at Last update timestamp.

approved\_at Approval timestamp, or NA.

rejected\_at Rejection timestamp, or NA.

superseded\_by Proposal identifier that superseded this proposal, or NA.

supersedes Proposal identifier superseded by this proposal, or NA.

**Methods****Public methods:**

- `MemoryProposal$new()`
- `MemoryProposal$validate()`
- `MemoryProposal$discuss()`
- `MemoryProposal$transition()`
- `MemoryProposal$approve()`
- `MemoryProposal$reject()`
- `MemoryProposal$to_list()`
- `MemoryProposal$print()`
- `MemoryProposal$clone()`

**Method** `new()`: Create a memory proposal.

*Usage:*

```
MemoryProposal$new(
  memory_spec,
  id = paste0("memory_proposal_", format(Sys.time(), "%Y%m%d%H%M%S")),
  status = "pending",
  notes = NULL,
  history = list(),
  metadata = list(),
  created_at = Sys.time(),
  updated_at = created_at,
  approved_at = as.POSIXct(NA),
  rejected_at = as.POSIXct(NA),
  superseded_by = NA_character_,
  supersedes = NA_character_
)
```

*Arguments:*

`memory_spec` Proposed [MemorySpec](#) or serializable memory-spec list.

`id` Proposal identifier.

`status` Proposal status.

`notes` Optional proposal notes.

`history` Lifecycle history entries.

`metadata` Free-form metadata.

`created_at` Creation timestamp.

`updated_at` Last update timestamp.

`approved_at` Approval timestamp, or NA.

`rejected_at` Rejection timestamp, or NA.

`superseded_by` Proposal identifier that superseded this proposal, or NA.

`supersedes` Proposal identifier superseded by this proposal, or NA.

**Method** `validate()`: Validate the proposal.

*Usage:*

```
MemoryProposal$validate()
```

**Method** `discuss()`: Append a discussion note.

*Usage:*

```
MemoryProposal$discuss(  
  note,  
  source = "human",  
  confidence = NA_character_,  
  timestamp = Sys.time()  
)
```

*Arguments:*

`note` Discussion or transition note.  
`source` Discussion source.  
`confidence` Optional confidence label.  
`timestamp` Event timestamp.

**Method** `transition()`: Apply a lifecycle transition.

*Usage:*

```
MemoryProposal$transition(status, note = NULL, timestamp = Sys.time())
```

*Arguments:*

`status` Proposal status.  
`note` Discussion or transition note.  
`timestamp` Event timestamp.

**Method** `approve()`: Approve the proposal.

*Usage:*

```
MemoryProposal$approve(note = NULL)
```

*Arguments:*

`note` Discussion or transition note.

**Method** `reject()`: Reject the proposal.

*Usage:*

```
MemoryProposal$reject(note = NULL)
```

*Arguments:*

`note` Discussion or transition note.

**Method** `to_list()`: Return a serializable representation.

*Usage:*

```
MemoryProposal$to_list()
```

**Method** `print()`: Print a compact summary.

*Usage:*

```
MemoryProposal$print(...)
```

*Arguments:*

... Unused.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

MemoryProposal\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

MemoryProposalState    *MemoryProposalState*

---

## Description

MemoryProposalState

MemoryProposalState

## Details

State container for approved memory schema plus candidate proposals.

## Public fields

approved\_memory\_spec Approved [MemorySpec](#).

proposals Named list of [MemoryProposal](#) objects.

history Proposal-state history.

## Methods

### Public methods:

- [MemoryProposalState\\$new\(\)](#)
- [MemoryProposalState\\$validate\(\)](#)
- [MemoryProposalState\\$add\\_proposal\(\)](#)
- [MemoryProposalState\\$get\\_proposal\(\)](#)
- [MemoryProposalState\\$list\\_proposals\(\)](#)
- [MemoryProposalState\\$discuss\\_proposal\(\)](#)
- [MemoryProposalState\\$approve\\_proposal\(\)](#)
- [MemoryProposalState\\$reject\\_proposal\(\)](#)
- [MemoryProposalState\\$approved\\_spec\(\)](#)
- [MemoryProposalState\\$as\\_list\(\)](#)
- [MemoryProposalState\\$clone\(\)](#)

**Method** new(): Create a memory proposal state container.

*Usage:*

```
MemoryProposalState$new(  
  approved_memory_spec = MemorySpec$new(),  
  proposals = list(),  
  history = list()  
)
```

*Arguments:*

approved\_memory\_spec Approved [MemorySpec](#) or serializable memory-spec list.  
proposals Initial proposals.  
history Proposal-state history.

**Method** `validate()`: Validate the state object.

*Usage:*

```
MemoryProposalState$validate()
```

**Method** `add_proposal()`: Add a proposal.

*Usage:*

```
MemoryProposalState$add_proposal(proposal)
```

*Arguments:*

proposal [MemoryProposal](#) object or serializable proposal record.

**Method** `get_proposal()`: Return one proposal.

*Usage:*

```
MemoryProposalState$get_proposal(proposal_id)
```

*Arguments:*

proposal\_id Proposal identifier.

**Method** `list_proposals()`: List proposals with optional status filtering.

*Usage:*

```
MemoryProposalState$list_proposals(status = NULL)
```

*Arguments:*

status Optional status filter.

**Method** `discuss_proposal()`: Discuss one proposal.

*Usage:*

```
MemoryProposalState$discuss_proposal(proposal_id, note, source = "human")
```

*Arguments:*

proposal\_id Proposal identifier.  
note Discussion or transition note.  
source Discussion source.

**Method** `approve_proposal()`: Approve one proposal and replace the approved memory spec.

*Usage:*

```
MemoryProposalState$approve_proposal(proposal_id, note = NULL)
```

*Arguments:*

proposal\_id Proposal identifier.

note Discussion or transition note.

**Method** reject\_proposal(): Reject one proposal.

*Usage:*

```
MemoryProposalState$reject_proposal(proposal_id, note = NULL)
```

*Arguments:*

proposal\_id Proposal identifier.

note Discussion or transition note.

**Method** approved\_spec(): Return the approved memory specification.

*Usage:*

```
MemoryProposalState$approved_spec()
```

**Method** as\_list(): Return a serializable representation.

*Usage:*

```
MemoryProposalState$as_list()
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
MemoryProposalState$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

MemorySpec

*MemorySpec*

---

## Description

MemorySpec

MemorySpec

## Details

First-class memory schema for an agent design. MemorySpec records which memory fields exist, what type of memory they represent, how they persist across cold-start runs, and how they are expected to update. It can also carry an optional graph-shaped representation of memory relationships.

**Methods**

`$initialize(fields = list(), graph = NULL, metadata = list())` Create a memory specification.

`$add_field(field)` Add one memory field.

`$get_field(id)` Return one memory field by id.

`$list_fields(memory_type = NULL, persistence = NULL)` Return memory fields, optionally filtered.

`$validate()` Validate the memory specification.

`$to_list()` Return a serializable list.

`$print(...)` Print a compact summary.

**Public fields**

`fields` Named list of memory field records.

`graph` Optional graph representation list with nodes and edges.

`metadata` Free-form metadata list.

**Methods****Public methods:**

- [MemorySpec\\$new\(\)](#)
- [MemorySpec\\$add\\_field\(\)](#)
- [MemorySpec\\$get\\_field\(\)](#)
- [MemorySpec\\$list\\_fields\(\)](#)
- [MemorySpec\\$validate\(\)](#)
- [MemorySpec\\$to\\_list\(\)](#)
- [MemorySpec\\$print\(\)](#)
- [MemorySpec\\$clone\(\)](#)

**Method** `new()`: Create a memory specification.

*Usage:*

```
MemorySpec$new(fields = list(), graph = NULL, metadata = list())
```

*Arguments:*

`fields` List of memory field records.

`graph` Optional graph representation list with nodes and edges.

`metadata` Free-form metadata list.

**Method** `add_field()`: Add one memory field.

*Usage:*

```
MemorySpec$add_field(field)
```

*Arguments:*

`field` Memory field record used by `$add_field()`.

**Method** `get_field()`: Return one memory field by id.

*Usage:*

```
MemorySpec$get_field(id)
```

*Arguments:*

`id` Memory field id used by `$get_field()`.

**Method** `list_fields()`: Return memory fields, optionally filtered by type or persistence policy.

*Usage:*

```
MemorySpec$list_fields(memory_type = NULL, persistence = NULL)
```

*Arguments:*

`memory_type` Optional memory type filter used by `$list_fields()`.

`persistence` Optional persistence-policy filter used by `$list_fields()`.

**Method** `validate()`: Validate the memory specification.

*Usage:*

```
MemorySpec$validate()
```

**Method** `to_list()`: Return a serializable list.

*Usage:*

```
MemorySpec$to_list()
```

**Method** `print()`: Print a compact memory schema summary.

*Usage:*

```
MemorySpec$print(...)
```

*Arguments:*

`...` Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
MemorySpec$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`new_design_review_spec`*Create a design-review specification*

---

**Description**

Convenience constructor matching the public plan API.

**Usage**

```
new_design_review_spec(...)
```

**Arguments**

... Arguments passed to [DesignReviewSpec\\$new\(\)](#).

**Value**

A [DesignReviewSpec](#) object.

---

`new_task_family_workflow`*Create a task-family workflow*

---

**Description**

Creates a root workflow whose nodes are child tasks. By default the root workflow has no edges because sibling child tasks are interpreted as independent unless explicit dependencies are supplied.

**Usage**

```
new_task_family_workflow(  
  id,  
  label,  
  objective,  
  nodes = .empty_workflow_nodes(),  
  edges = .empty_workflow_edges(),  
  shared_inputs = character(),  
  shared_review_concerns = character(),  
  task_tags = list(),  
  metadata = list()  
)
```

**Arguments**

id	Task-family identifier.
label	Task-family label.
objective	Family-level objective.
nodes	Data frame of child-task nodes.
edges	Optional root-level dependency edges among child tasks.
shared_inputs	Character vector of shared input names.
shared_review_concerns	Character vector of shared review concerns.
task_tags	Optional named list mapping child-task ids to tags.
metadata	Additional root workflow metadata.

**Value**

agentr\_workflow\_spec.

---

new\_workflow\_spec      *Create a workflow specification*

---

**Description**

Workflow specifications are outputs of reasoning and scaffolding rather than fixed package logic. The object captures DAG-like workflow structure and the minimal metadata needed for downstream implementation translation.

**Usage**

```
new_workflow_spec(
  nodes = workflow_node("task", "Task"),
  edges = data.frame(from = character(), to = character(), relation = character(),
    stringsAsFactors = FALSE),
  task = NULL,
  metadata = list()
)
```

**Arguments**

nodes	Data frame of workflow nodes.
edges	Data frame of workflow edges.
task	Optional source task text.
metadata	Additional metadata list.

**Value**

An object of class agentr\_workflow\_spec.

normalize\_subsystem\_key

*Normalize subsystem keys*

---

### **Description**

Accepts canonical subsystem keys and legacy mixed-case variants, then returns the canonical keys used throughout agentr.

### **Usage**

```
normalize_subsystem_key(x)
```

### **Arguments**

x                      Character vector of subsystem keys.

### **Value**

Character vector containing canonical subsystem keys.

---

parse\_design\_feedback\_json

*Parse design feedback JSON*

---

### **Description**

Parse design feedback JSON

### **Usage**

```
parse_design_feedback_json(x)
```

### **Arguments**

x                      JSON string, parsed list, or .json file path.

### **Value**

A list of validated design-feedback items.

---

parse\_knowledge\_message

*Parse a knowledge message*

---

**Description**

Parse a knowledge message

**Usage**

parse\_knowledge\_message(x)

**Arguments**

x                   JSON string, parsed list, or .json file path.

**Value**

Parsed knowledge message list.

---

parse\_memory\_message   *Parse a memory message*

---

**Description**

Parse a memory message

**Usage**

parse\_memory\_message(x)

**Arguments**

x                   JSON string, parsed list, or .json file path.

**Value**

Parsed memory message list.

---

```
parse_scaffolder_message
```

*Parse an LLM scaffolder message*

---

### Description

Parses a machine-readable scaffolder message from JSON text or a .json file path into an R list.

### Usage

```
parse_scaffolder_message(text)
```

### Arguments

text                    Character string containing JSON or a path to a .json file.

### Value

Parsed list.

---

```
PGConfig
```

*PGConfig*

---

### Description

PGConfig

PGConfig

### Details

Configuration for the Perception & Grounding subsystem.

### Methods

`$initialize(enabled = TRUE, planning_mode = "task_decomposition", decomposition_style = "dag", meta)`  
 Create a Perception & Grounding config. Legacy field names are preserved for compatibility.

`$validate()` Validate the config.

`$as_list()` Return a serializable representation.

### Public fields

`enabled` Whether the subsystem is enabled.

`planning_mode` Legacy field name retained for compatibility; interpreted as a grounding/perception mode label.

`decomposition_style` Legacy field name retained for compatibility; interpreted as a representation-structuring style.

`metadata` Free-form metadata list.

## Methods

### Public methods:

- [PGConfig\\$new\(\)](#)
- [PGConfig\\$validate\(\)](#)
- [PGConfig\\$as\\_list\(\)](#)
- [PGConfig\\$print\(\)](#)
- [PGConfig\\$clone\(\)](#)

**Method** `new()`: Create a Perception & Grounding config.

*Usage:*

```
PGConfig$new(  
  enabled = TRUE,  
  planning_mode = "task_decomposition",  
  decomposition_style = "dag",  
  metadata = list()  
)
```

*Arguments:*

`enabled` Whether the subsystem is enabled.  
`planning_mode` Planning mode label.  
`decomposition_style` Workflow decomposition style.  
`metadata` Free-form metadata list.

**Method** `validate()`: Validate the config.

*Usage:*

```
PGConfig$validate()
```

**Method** `as_list()`: Return a serializable representation.

*Usage:*

```
PGConfig$as_list()
```

**Method** `print()`: Print a compact config summary.

*Usage:*

```
PGConfig$print(...)
```

*Arguments:*

... Unused print arguments.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PGConfig$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

plot\_knowledge\_graph *Plot a graph-shaped knowledge or memory representation*

---

### Description

Plot a graph-shaped knowledge or memory representation

### Usage

```
plot_knowledge_graph(  
  x,  
  rankdir = "TB",  
  label_width = 28,  
  show_edge_labels = TRUE,  
  show_tooltips = FALSE  
)
```

### Arguments

x                    A [KnowledgeSpec](#), [MemorySpec](#), or graph representation list.

rankdir             Graphviz rank direction, for example "TB" or "LR".

label\_width         Approximate wrapping width for node labels.

show\_edge\_labels    Whether to show edge relation labels.

show\_tooltips      Whether to include Graphviz tooltip attributes.

### Value

A DiagrammeR graph object.

---

plot\_workflow\_graph *Plot a workflow graph with DiagrammeR*

---

### Description

Creates a DiagrammeR graph from a workflow. This is preferred over base igraph plotting for readable workflow DAG visualization.

**Usage**

```
plot_workflow_graph(
  x,
  rankdir = "TB",
  label_width = 28,
  show_edge_labels = FALSE,
  show_tooltips = FALSE,
  same_rank = NULL
)
```

**Arguments**

x	A workflow specification or a <a href="#">Scaffolder</a> instance.
rankdir	Graphviz rank direction, for example "TB" or "LR".
label_width	Approximate wrapping width for node labels.
show_edge_labels	Whether to show edge relation labels.
show_tooltips	Whether to include Graphviz tooltip attributes. Defaults to FALSE because long prose tooltips can trigger Viz.js parse failures in some DiagrammeR renderers.
same_rank	Optional list of node-id character vectors to keep at the same Graphviz rank.

**Value**

A DiagrammeR graph object.

---

```
preview_design_feedback
```

*Preview design feedback application*

---

**Description**

Preview design feedback application

**Usage**

```
preview_design_feedback(x, feedback, review_spec = NULL)
```

**Arguments**

x	A <a href="#">Scaffolder</a> or design object.
feedback	Feedback item or list of items.
review_spec	Optional review spec used for target-id warnings.

**Value**

A non-mutating preview list.

preview\_knowledge\_message

*Preview a knowledge message*

---

### Description

Preview a knowledge message

### Usage

```
preview_knowledge_message(state, message)
```

### Arguments

state	A <a href="#">KnowledgeProposalState</a> object.
message	Parsed or raw knowledge message.

### Value

Preview list.

---

preview\_memory\_message

*Preview a memory message*

---

### Description

Preview a memory message

### Usage

```
preview_memory_message(state, message)
```

### Arguments

state	A <a href="#">MemoryProposalState</a> object.
message	Parsed or raw memory message.

### Value

Preview list.

---

`preview_scaffolder_message`

*Preview a machine-readable scaffolder message without mutating live workflow*

---

## Description

Applies a message to a deep clone of the scaffolder, returns the preview result, and optionally stores the resulting workflow as a proposal on the original scaffolder for later approval or discussion.

## Usage

```
preview_scaffolder_message(  
  scaffolder,  
  message,  
  allowed_methods = scaffolder_action_methods(),  
  stop_on_error = TRUE,  
  store_proposal = TRUE,  
  source = "model",  
  proposal_notes = NULL  
)
```

## Arguments

<code>scaffolder</code>	A <a href="#">Scaffolder</a> instance.
<code>message</code>	Parsed message list, JSON string, or path to a downloaded .json file.
<code>allowed_methods</code>	Character vector of allowed method names.
<code>stop_on_error</code>	Whether to stop on the first action error. When FALSE, errors are collected in the returned result object.
<code>store_proposal</code>	Whether to store the previewed workflow as a proposal on the original scaffolder.
<code>source</code>	Proposal source label used when storing a proposal.
<code>proposal_notes</code>	Optional proposal notes. Defaults to top-level message\$notes when available.

## Value

A standardized list with `proposal_id`, `proposal`, `preview_dispatch`, `workflow_after`, `human_prompts`, and `errors`.

```
print.agentr_workflow_proposal
    Format a workflow proposal
```

---

**Description**

Format a workflow proposal

**Usage**

```
## S3 method for class 'agentr_workflow_proposal'
print(x, ...)
```

**Arguments**

x	Workflow proposal object.
...	Unused.

---

```
print.agentr_workflow_spec
    Format a workflow specification
```

---

**Description**

Format a workflow specification

**Usage**

```
## S3 method for class 'agentr_workflow_spec'
print(x, ...)
```

**Arguments**

x	Workflow specification.
...	Unused.

---

read\_decision\_traces    *Read decision traces*

---

**Description**

Read decision traces

**Usage**

read\_decision\_traces(path)

**Arguments**

path                    JSONL or RDS path.

**Value**

List of traces.

---

read\_reflection\_traces  
                          *Read reflection traces*

---

**Description**

Read reflection traces

**Usage**

read\_reflection\_traces(path)

**Arguments**

path                    JSONL or RDS path.

**Value**

List of traces.

---

```
reject_workspace_proposal
```

*Reject a workspace proposal*

---

### Description

Reject a workspace proposal

### Usage

```
reject_workspace_proposal(  
  workspace,  
  type = c("workflow", "agent", "memory", "knowledge"),  
  proposal_id,  
  note = NULL  
)
```

### Arguments

workspace	Workspace root directory.
type	Proposal type: workflow, agent, memory, or knowledge.
proposal_id	Proposal identifier.
note	Optional rejection note.

### Value

Rejected proposal.

---

```
render_knowledge_graphviz
```

*Render a graph representation as Graphviz DOT, DiagrammeR, or SVG*

---

### Description

This helper renders graph-shaped knowledge or memory. It accepts a [KnowledgeSpec](#), [MemorySpec](#), or plain `list(nodes, edges, metadata)` graph representation. It does not require or create a separate graph spec.

**Usage**

```
render_knowledge_graphviz(
  x,
  rankdir = "TB",
  as = c("dot", "diagrammer", "svg"),
  label_width = 28,
  show_edge_labels = TRUE,
  show_tooltips = FALSE
)
```

**Arguments**

`x` A [KnowledgeSpec](#), [MemorySpec](#), or graph representation list.

`rankdir` Graphviz rank direction, for example "TB" or "LR".

`as` Output format: raw "dot" text, a "diagrammer" object, or exported "svg" text.

`label_width` Approximate wrapping width for node labels.

`show_edge_labels` Whether to show edge relation labels.

`show_tooltips` Whether to include Graphviz tooltip attributes.

**Value**

A Graphviz DOT string, DiagrammeR graph object, or SVG string.

---

```
render_markdown_terminal
```

*Render markdown-like text for terminal output*

---

**Description**

Render markdown-like text for terminal output

**Usage**

```
render_markdown_terminal(txt)
```

**Arguments**

`txt` Character string.

**Value**

Rendered character string with ANSI styling.

---

`render_memory_schema_graphviz`*Render a MemorySpec as Graphviz DOT, DiagrammeR, or SVG*

---

## Description

Renders memory fields and, optionally, the schema shape of each field. The output is for inspection and review of memory design, not runtime memory execution.

## Usage

```
render_memory_schema_graphviz(  
  x,  
  include_field_schemas = TRUE,  
  rankdir = "TB",  
  as = c("dot", "diagrammer", "svg"),  
  label_width = 28,  
  show_edge_labels = TRUE,  
  show_tooltips = FALSE  
)
```

## Arguments

<code>x</code>	A <a href="#">MemorySpec</a> object.
<code>include_field_schemas</code>	Whether to include nested schema-shape nodes for each memory field's schema.
<code>rankdir</code>	Graphviz rank direction, for example "TB" or "LR".
<code>as</code>	Output format: raw "dot" text, a "diagrammer" object, or exported "svg" text.
<code>label_width</code>	Approximate wrapping width for node labels.
<code>show_edge_labels</code>	Whether to show edge relation labels.
<code>show_tooltips</code>	Whether to include Graphviz tooltip attributes.

## Value

A Graphviz DOT string, DiagrammeR graph object, or SVG string.

---

`render_schema_shape_graphviz`*Render a schema shape as Graphviz DOT, DiagrammeR, or SVG*

---

### Description

Renders a structural preview of a nested schema object, such as a workflow node's `input_schema` or `output_schema`.

### Usage

```
render_schema_shape_graphviz(  
  x,  
  root_id = "schema",  
  root_label = "schema",  
  rankdir = "TB",  
  as = c("dot", "diagrammer", "svg"),  
  label_width = 28,  
  show_edge_labels = TRUE,  
  show_tooltips = FALSE  
)
```

### Arguments

<code>x</code>	Schema object to inspect.
<code>root_id</code>	Root node identifier.
<code>root_label</code>	Human-readable root node label.
<code>rankdir</code>	Graphviz rank direction, for example "TB" or "LR".
<code>as</code>	Output format: raw "dot" text, a "diagrammer" object, or exported "svg" text.
<code>label_width</code>	Approximate wrapping width for node labels.
<code>show_edge_labels</code>	Whether to show edge relation labels.
<code>show_tooltips</code>	Whether to include Graphviz tooltip attributes.

### Value

A Graphviz DOT string, DiagrammeR graph object, or SVG string.

---

render\_task\_preview     *Render one task-local design-review preview*

---

### Description

Loads conventional task-local YAML specs and renders docs/review.html. When present, memory and narrative knowledge specs are included alongside the workflow graph. Graph-shaped knowledge is read from knowledge\_spec.yaml when present.

### Usage

```
render_task_preview(
  task_dir,
  docs_dir = "docs",
  out = NULL,
  title = NULL,
  require_workflow = TRUE,
  graph_layout = c("grid", "layered", "swimlane", "process"),
  edge_style = c("curved", "straight", "orthogonal"),
  node_color_theme = c("default", "subsystems"),
  ...
)
```

### Arguments

task_dir	Task root directory.
docs_dir	Documentation/spec directory relative to task_dir, or an absolute path.
out	Optional output HTML path. Defaults to docs/review.html.
title	Optional review title. Defaults to the workflow task title, then the task directory name.
require_workflow	Whether workflow_spec.yaml must exist.
graph_layout	Workflow graph layout passed to <a href="#">export_design_review_html()</a> .
edge_style	Workflow edge style passed to <a href="#">export_design_review_html()</a> .
node_color_theme	Initial node-color theme passed to <a href="#">export_design_review_html()</a> .
...	Additional arguments passed to <a href="#">export_design_review_html()</a> .

### Value

Invisibly returns the normalized output HTML path.

---

render\_task\_previews *Render task-local design-review previews under a workspace*

---

### Description

Scans a workspace for workflow\_spec.yaml files under task-local docs/ directories and renders one review HTML file per discovered task. This helper only loads specs and writes review artifacts; it does not execute task code.

### Usage

```
render_task_previews(
  root,
  tasks_dir = "tasks",
  docs_dir = "docs",
  recursive = TRUE,
  require_workflow = TRUE,
  ...
)
```

### Arguments

root	Workspace root directory.
tasks_dir	Tasks directory relative to root, or an absolute path.
docs_dir	Documentation/spec directory name relative to each task.
recursive	Whether to scan nested task folders. Defaults to TRUE so node-folder subwork-flow specs are rendered too.
require_workflow	Whether discovered task previews require a workflow spec. Discovered paths always have one; this is forwarded to <a href="#">render_task_preview()</a> .
...	Additional arguments passed to <a href="#">render_task_preview()</a> .

### Value

Data frame with rendered task directories and review paths.

---

render\_workflow\_graphviz  
*Render a workflow as Graphviz DOT, DiagrammeR, or SVG*

---

### Description

Converts a workflow specification into a Graphviz-friendly representation. The DiagrammeR path is preferred for visual inspection of workflow DAGs.

**Usage**

```
render_workflow_graphviz(
  x,
  rankdir = "TB",
  as = c("dot", "diagrammer", "svg"),
  label_width = 28,
  show_edge_labels = FALSE,
  show_tooltips = FALSE,
  same_rank = NULL
)
```

**Arguments**

x	A workflow specification or a <a href="#">Scaffolder</a> instance.
rankdir	Graphviz rank direction, for example "TB" or "LR".
as	Output format: raw "dot" text, a "diagrammer" object, or exported "svg" text.
label_width	Approximate wrapping width for node labels.
show_edge_labels	Whether to show edge relation labels.
show_tooltips	Whether to include Graphviz tooltip attributes. Defaults to FALSE because long prose tooltips can trigger Viz.js parse failures in some DiagrammeR renderers.
same_rank	Optional list of node-id character vectors to keep at the same Graphviz rank.

**Value**

A Graphviz DOT string, DiagrammeR graph object, or SVG string.

---

RWMConfig

*RWMConfig*


---

**Description**

RWMConfig  
RWMConfig

**Details**

Configuration for the Reasoning & World Model subsystem.

**Methods**

`$initialize(cognitive = CognitiveConfig$new(), affective = NULL, persistence = "session", summary = ...)` Create a Reasoning & World Model config.

`$validate()` Validate the config.

`$selected_layers()` Return the active inner layers.

`$as_list()` Return a serializable representation.

**Public fields**

cognitive A CognitiveConfig object or NULL.  
 affective An AffectiveConfig object or NULL.  
 persistence Persistence mode for the overall subsystem.  
 summary Optional one-line summary.  
 metadata Free-form metadata list.

**Methods****Public methods:**

- [RWMConfig\\$new\(\)](#)
- [RWMConfig\\$validate\(\)](#)
- [RWMConfig\\$selected\\_layers\(\)](#)
- [RWMConfig\\$as\\_list\(\)](#)
- [RWMConfig\\$print\(\)](#)
- [RWMConfig\\$clone\(\)](#)

**Method new():** Create an RWM config.

*Usage:*

```
RWMConfig$new(
  cognitive = CognitiveConfig$new(),
  affective = NULL,
  persistence = "session",
  summary = NULL,
  metadata = list()
)
```

*Arguments:*

cognitive A CognitiveConfig object or list payload.  
 affective An AffectiveConfig object or list payload.  
 persistence Persistence mode for the overall subsystem.  
 summary Optional one-line summary.  
 metadata Free-form metadata list.

**Method validate():** Validate the config.

*Usage:*

```
RWMConfig$validate()
```

**Method selected\_layers():** Return the active inner layers.

*Usage:*

```
RWMConfig$selected_layers()
```

**Method as\_list():** Return a serializable representation.

*Usage:*

```
RWMConfig$as_list()
```

**Method** print(): Print a compact config summary.

*Usage:*

RWMConfig#print(...)

*Arguments:*

... Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

RWMConfig\$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

---

save_agent	<i>Save an agentr object to a file</i>
------------	--

---

## Description

Saves an [AgentCore](#), [CognitiveState](#), [AffectiveState](#), or [Scaffolder](#) object to a specified .rds file. AgentSpec, SubsystemSpec, MemorySpec, DesignReviewSpec, AgentScaffoldState, and IntelligentAgent are also supported.

## Usage

```
save_agent(agent, file_path)
```

## Arguments

agent	An object created by agentr.
file_path	File path where the object should be saved.

## Value

Invisibly returns TRUE.

---

save_agent_spec	<i>Save an AgentSpec to a file</i>
-----------------	------------------------------------

---

**Description**

Saves an [AgentSpec](#) object to a specified .rds file.

**Usage**

```
save_agent_spec(spec, file_path)
```

**Arguments**

spec	An <a href="#">AgentSpec</a> object.
file_path	File path where the object should be saved.

**Value**

Invisibly returns TRUE.

---

save_design_feedback	<i>Save structured design feedback</i>
----------------------	--

---

**Description**

Save structured design feedback

**Usage**

```
save_design_feedback(x, path)
```

**Arguments**

x	A design-feedback item or list of items.
path	Output .rds path.

**Value**

Invisibly returns TRUE.

---

save\_design\_review\_spec  
*Save a design-review specification*

---

**Description**

Save a design-review specification

**Usage**

```
save_design_review_spec(x, path)
```

**Arguments**

x	A <a href="#">DesignReviewSpec</a> object.
path	Output .rds path.

**Value**

Invisibly returns TRUE.

---

save\_knowledge\_proposal  
*Save a knowledge proposal*

---

**Description**

Save a knowledge proposal

**Usage**

```
save_knowledge_proposal(x, path)
```

**Arguments**

x	A <a href="#">KnowledgeProposal</a> object.
path	File path.

**Value**

Invisibly returns TRUE.

---

save\_knowledge\_spec    *Save a knowledge specification*

---

**Description**

Save a knowledge specification

**Usage**

```
save_knowledge_spec(x, path, format = c("rds", "json", "yaml"))
```

**Arguments**

x	A <a href="#">KnowledgeSpec</a> object.
path	File path.
format	File format, either rds, json, or yaml.

**Value**

Invisibly returns TRUE.

---

save\_knowledge\_spec\_json  
*Save a knowledge specification as JSON*

---

**Description**

Save a knowledge specification as JSON

**Usage**

```
save_knowledge_spec_json(x, path)
```

**Arguments**

x	A <a href="#">KnowledgeSpec</a> object.
path	File path.

**Value**

Invisibly returns TRUE.

---

`save_knowledge_spec_yaml`*Save a knowledge specification as YAML*

---

**Description**

Save a knowledge specification as YAML

**Usage**

```
save_knowledge_spec_yaml(x, path)
```

**Arguments**

<code>x</code>	A <a href="#">KnowledgeSpec</a> object.
<code>path</code>	File path.

**Value**

Invisibly returns TRUE.

---

`save_memory_spec`*Save a MemorySpec to a file*

---

**Description**

Saves a [MemorySpec](#) object to a specified `.rds`, `.json`, or `.yaml` file.

**Usage**

```
save_memory_spec(spec, file_path, format = c("rds", "json", "yaml"))
```

**Arguments**

<code>spec</code>	A <a href="#">MemorySpec</a> object.
<code>file_path</code>	File path where the object should be saved.
<code>format</code>	File format, either <code>rds</code> , <code>json</code> , or <code>yaml</code> .

**Value**

Invisibly returns TRUE.

---

save\_memory\_spec\_json *Save a MemorySpec as JSON*

---

**Description**

Save a MemorySpec as JSON

**Usage**

```
save_memory_spec_json(spec, file_path)
```

**Arguments**

spec	A <a href="#">MemorySpec</a> object.
file_path	File path where the JSON should be saved.

**Value**

Invisibly returns TRUE.

---

save\_memory\_spec\_yaml *Save a MemorySpec as YAML*

---

**Description**

Save a MemorySpec as YAML

**Usage**

```
save_memory_spec_yaml(spec, file_path)
```

**Arguments**

spec	A <a href="#">MemorySpec</a> object.
file_path	File path where the YAML should be saved.

**Value**

Invisibly returns TRUE.

---

save\_subsystem\_spec     *Save a SubsystemSpec to a file*

---

**Description**

Saves a [SubsystemSpec](#) object to a specified .rds file.

**Usage**

```
save_subsystem_spec(spec, file_path)
```

**Arguments**

spec                    A [SubsystemSpec](#) object.  
file\_path              File path where the object should be saved.

**Value**

Invisibly returns TRUE.

---

save\_workflow\_proposal  
                          *Save a workflow proposal*

---

**Description**

Saves an `agentr_workflow_proposal` object so it can be reviewed, approved, or visualized in a later session.

**Usage**

```
save_workflow_proposal(proposal, file_path)
```

**Arguments**

proposal                Workflow proposal object.  
file\_path              File path where the proposal should be saved.

**Value**

Invisibly returns TRUE.

---

save_workflow_spec	<i>Save a workflow specification</i>
--------------------	--------------------------------------

---

**Description**

Save a workflow specification

**Usage**

```
save_workflow_spec(workflow, file_path, format = c("rds", "json", "yaml"))
```

**Arguments**

workflow	Workflow specification.
file_path	File path where the workflow should be saved.
format	File format, either rds, json, or yaml.

**Value**

Invisibly returns TRUE.

---

save_workflow_spec_json	<i>Save a workflow specification as JSON</i>
-------------------------	--

---

**Description**

Save a workflow specification as JSON

**Usage**

```
save_workflow_spec_json(workflow, file_path)
```

**Arguments**

workflow	Workflow specification.
file_path	File path where the JSON should be saved.

**Value**

Invisibly returns TRUE.

---

```
save_workflow_spec_yaml
```

*Save a workflow specification as YAML*

---

### Description

Save a workflow specification as YAML

### Usage

```
save_workflow_spec_yaml(workflow, file_path)
```

### Arguments

workflow	Workflow specification.
file_path	File path where the YAML should be saved.

### Value

Invisibly returns TRUE.

---

Scaffolder

*Scaffolder*

---

### Description

Scaffolder

Scaffolder

### Details

Human-in-the-loop scaffolding interface for iterative workflow elicitation. A Scaffolder keeps a persistent task-evaluation artifact, supports free-form discussion rounds before structured graph edits, separates workflow-level and node-level review, treats node/edge edits as first-class operations, and manages previewable workflow proposals with explicit lifecycle state.

### Methods

```
$initialize(agent = NULL, completion_threshold = 0.75) Create a scaffolder with empty workflow state and review metadata.
```

```
$evaluate_task(task, summary = NULL, workflow_complete = NA, blockers = NULL, next_focus = NULL) Create or refresh the persistent task-evaluation artifact.
```

```
$discuss_task(feedback, source = "human", node_id = NULL, confidence = NA_real_) Record a free-form human, model, or system discussion round.
```

`$decompose_task(task = self$task, candidates = NULL, suggestions = NULL, nodes = NULL, edges = NULL)`  
 Create or replace the workflow from linear candidates or non-linear graph suggestions.

`$ask_human_complete(node_id)` Create a prompt asking whether a workflow node is complete.

`$ask_human_changes()` Create a prompt asking what workflow or edge changes should happen next.

`$ask_human_rule(node_id)` Create a prompt requesting a node-specific rule.

`$review_workflow(status = "pending", notes = NULL, confidence = NA_real_)` Store workflow-level completeness or revision review state.

`$review_node(node_id, status = "pending", notes = NULL, confidence = NA_real_, complete = NULL)`  
 Store node-level correctness or completion review state.

`$edit_workflow(add = NULL, insert = NULL, remove = NULL, add_edges = NULL, remove_edges = NULL, rule_specs = NULL)`  
 Apply first-class node and edge edits to the current workflow.

`$set_node_schema(node_id, input_schema = NULL, output_schema = NULL)` Set input/output schema metadata for one workflow node.

`$set_node_nested_workflow(node_id, subworkflow_ref = NULL, nested_workflow = NULL)`  
 Attach a nested workflow reference or embedded nested workflow to one node.

`$apply_human_feedback(completeness = NULL, add = NULL, remove = NULL, rule_specs = list(), confidence = NULL)`  
 Compatibility wrapper for structured human workflow edits.

`$recommend_subsystems(task = self$task)` Recommend optional subsystem/capability labels for the current task and workflow.

`$subsystem_recommendations()` Return the current subsystem recommendation records.

`$subsystem_recommendation_rationale(subsystem = NULL)` Return stored recommendation rationale for one subsystem or all subsystems.

`$select_subsystems(subsystems)` Store the selected subsystem configuration.

`$selected_subsystems()` Return the currently selected subsystem names.

`$label_workflow_subsystems(labels)` Assign subsystem owners to workflow nodes.

`$edit_workflow_subsystems(set = NULL, add = NULL, remove = NULL, clear = NULL)` Edit workflow-node subsystem ownership incrementally.

`$propose_agent_spec(agent_name = "agentr-agent", summary = NULL, subsystems = NULL, workflow = NULL)`  
 Store a draft agent-spec proposal.

`$list_agent_spec_proposals(status = NULL)` Return stored agent-spec proposal summaries.

`$get_agent_spec_proposal(proposal_id)` Return a stored agent-spec proposal by id.

`$discuss_agent_spec_proposal(proposal_id, feedback, source = "human", confidence = NA_real_)`  
 Attach discussion feedback to a draft agent-spec proposal.

`$approve_agent_spec_proposal(proposal_id, approve_linked_workflow = TRUE)` Approve a stored agent-spec proposal and optionally approve its linked workflow proposal.

`$approve_agent_spec(agent_name = "agentr-agent", summary = NULL, state_requirements = list(), interaction_requirements = list(), subsystems = NULL)`  
 Approve an AgentSpec built from the current task, workflow, and optional subsystem labels.

`$agent_spec()` Return the approved agent spec or a draft spec built from current state.

`$propose_workflow(workflow, source = "model", notes = NULL)` Store a pending workflow proposal for preview and review.

`$list_workflow_proposals(status = NULL)` Return a summary table of stored workflow proposals.

`$get_workflow_proposal(proposal_id)` Return a stored `WorkflowProposal` object by identifier.

`$approve_workflow_proposal(proposal_id)` Promote a stored workflow proposal to the live workflow and supersede older active proposals when applicable.

`$discuss_workflow_proposal(proposal_id, feedback, source = "human", confidence = NA_real_)` Attach free-form discussion feedback to a non-approved workflow proposal and transition it into discussion state when needed.

`$workflow_spec()` Validate and return the current workflow specification.

`$implementation_spec()` Return an implementation-facing summary of workflow nodes and rules.

`$low_confidence_nodes()` Return workflow nodes below the completion threshold.

`$get_node(node_id)` Return a single workflow node by identifier.

`$record_interaction(type, payload)` Append an interaction event to the scaffolder log.

### Public fields

`agent` Optional `AgentCore` owner.

`task` Current task text.

`workflow` Current workflow specification.

`workflow_state` Public workflow proposal state container.

`agent_state` Public agent scaffold state container.

`proposal_log` Stored workflow proposals across pending, discussion, approved, superseded, and rejected lifecycle states.

`interaction_log` List of scaffolding interactions.

`completion_threshold` Threshold used to flag low-confidence nodes.

### Methods

#### Public methods:

- `Scaffolder$new()`
- `Scaffolder$evaluate_task()`
- `Scaffolder$discuss_task()`
- `Scaffolder$decompose_task()`
- `Scaffolder$ask_human_complete()`
- `Scaffolder$ask_human_changes()`
- `Scaffolder$ask_human_rule()`
- `Scaffolder$review_workflow()`
- `Scaffolder$review_node()`
- `Scaffolder$edit_workflow()`
- `Scaffolder$set_node_schema()`
- `Scaffolder$set_node_nested_workflow()`

- Scaffolder\$apply\_human\_feedback()
- Scaffolder\$recommend\_subsystems()
- Scaffolder\$subsystem\_recommendations()
- Scaffolder\$subsystem\_recommendation\_rationale()
- Scaffolder\$select\_subsystems()
- Scaffolder\$selected\_subsystems()
- Scaffolder\$label\_workflow\_subsystems()
- Scaffolder\$edit\_workflow\_subsystems()
- Scaffolder\$propose\_agent\_spec()
- Scaffolder\$list\_agent\_spec\_proposals()
- Scaffolder\$get\_agent\_spec\_proposal()
- Scaffolder\$discuss\_agent\_spec\_proposal()
- Scaffolder\$approve\_agent\_spec\_proposal()
- Scaffolder\$approve\_agent\_spec()
- Scaffolder\$agent\_spec()
- Scaffolder\$propose\_workflow()
- Scaffolder\$list\_workflow\_proposals()
- Scaffolder\$get\_workflow\_proposal()
- Scaffolder\$approve\_workflow\_proposal()
- Scaffolder\$discuss\_workflow\_proposal()
- Scaffolder\$workflow\_spec()
- Scaffolder\$implementation\_spec()
- Scaffolder\$low\_confidence\_nodes()
- Scaffolder\$get\_node()
- Scaffolder\$record\_interaction()
- Scaffolder\$clone()

**Method** new(): Create a Scaffolder with empty workflow, review, and discussion state.

*Usage:*

```
Scaffolder$new(agent = NULL, completion_threshold = 0.75)
```

*Arguments:*

agent Optional [AgentCore](#) used by \$initialize().

completion\_threshold Confidence threshold used by \$initialize().

**Method** evaluate\_task(): Create or refresh the persistent task-evaluation artifact.

*Usage:*

```
Scaffolder$evaluate_task(
  task,
  summary = NULL,
  workflow_complete = NA,
  blockers = NULL,
  next_focus = NULL
)
```

*Arguments:*

task Task text used by \$evaluate\_task() and \$decompose\_task().  
summary Optional task summary used by \$evaluate\_task().  
workflow\_complete Optional task-level completeness flag used by \$evaluate\_task().  
blockers Optional blocker strings used by \$evaluate\_task().  
next\_focus Optional next-focus note used by \$evaluate\_task().

**Method** discuss\_task(): Record a free-form human, model, or system discussion round.

*Usage:*

```
Scaffolder$discuss_task(
  feedback,
  source = "human",
  node_id = NULL,
  confidence = NA_real_
)
```

*Arguments:*

feedback Free-form discussion feedback used by \$discuss\_task().  
source Discussion source used by \$discuss\_task().  
source Proposal source used by proposal methods.  
node\_id Workflow node identifier used by node-specific methods.  
confidence Confidence value used by review/edit helpers.

**Method** decompose\_task(): Replace the workflow with nodes and edges derived from task suggestions.

*Usage:*

```
Scaffolder$decompose_task(
  task = self$task,
  candidates = NULL,
  suggestions = NULL,
  nodes = NULL,
  edges = NULL,
  notes = NULL
)
```

*Arguments:*

task Task text used by \$evaluate\_task() and \$decompose\_task().  
candidates Optional candidate node labels used by \$decompose\_task().  
suggestions Optional free-form or structured graph suggestions used by \$decompose\_task().  
nodes Optional node list accepted directly by \$decompose\_task().  
edges Optional edge list accepted directly by \$decompose\_task().  
notes Optional decomposition notes accepted directly by \$decompose\_task().  
notes Optional review notes.  
notes Optional proposal notes.

**Method** ask\_human\_complete(): Build a prompt asking whether a node is complete.

*Usage:*

```
Scaffolder$ask_human_complete(node_id)
```

*Arguments:*

node\_id Workflow node identifier used by node-specific methods.

**Method** ask\_human\_changes(): Build a prompt asking what workflow or edge changes should happen next.

*Usage:*

```
Scaffolder$ask_human_changes()
```

**Method** ask\_human\_rule(): Build a prompt asking for a node-specific rule.

*Usage:*

```
Scaffolder$ask_human_rule(node_id)
```

*Arguments:*

node\_id Workflow node identifier used by node-specific methods.

**Method** review\_workflow(): Store workflow-level completeness or revision review state.

*Usage:*

```
Scaffolder$review_workflow(
  status = "pending",
  notes = NULL,
  confidence = NA_real_
)
```

*Arguments:*

status Review status used by \$review\_workflow() and \$review\_node().  
 notes Optional decomposition notes accepted directly by \$decompose\_task().  
 notes Optional review notes.  
 notes Optional proposal notes.  
 confidence Confidence value used by review/edit helpers.

**Method** review\_node(): Store node-level review status, notes, confidence, and completion state.

*Usage:*

```
Scaffolder$review_node(
  node_id,
  status = "pending",
  notes = NULL,
  confidence = NA_real_,
  complete = NULL
)
```

*Arguments:*

node\_id Workflow node identifier used by node-specific methods.  
 status Review status used by \$review\_workflow() and \$review\_node().  
 notes Optional decomposition notes accepted directly by \$decompose\_task().

notes Optional review notes.  
 notes Optional proposal notes.  
 confidence Confidence value used by review/edit helpers.  
 complete Optional node completion flag used by `$review_node()`.

**Method** `edit_workflow()`: Apply first-class node and edge edits to the current workflow.

*Usage:*

```
Scaffolder$edit_workflow(
  add = NULL,
  insert = NULL,
  remove = NULL,
  add_edges = NULL,
  remove_edges = NULL,
  rule_specs = list(),
  confidence = list()
)
```

*Arguments:*

`add` List of node records to add in `$edit_workflow()`.  
`insert` List of insertion specs used by `$edit_workflow()`.  
`remove` Character vector of node ids to remove in `$edit_workflow()`.  
`add_edges` List of edge records to add in `$edit_workflow()`.  
`remove_edges` List of edge specs to remove in `$edit_workflow()`.  
`rule_specs` Named list of rule specs used by `$edit_workflow()`.  
`confidence` Confidence value used by review/edit helpers.

**Method** `set_node_schema()`: Set structured input and output schema metadata for one workflow node.

*Usage:*

```
Scaffolder$set_node_schema(node_id, input_schema = NULL, output_schema = NULL)
```

*Arguments:*

`node_id` Workflow node identifier used by node-specific methods.  
`input_schema` Structured input schema used by `$set_node_schema()`.  
`output_schema` Structured output schema used by `$set_node_schema()`.

**Method** `set_node_nested_workflow()`: Attach a nested workflow reference or embedded nested workflow to one node.

*Usage:*

```
Scaffolder$set_node_nested_workflow(
  node_id,
  subworkflow_ref = NULL,
  nested_workflow = NULL
)
```

*Arguments:*

`node_id` Workflow node identifier used by node-specific methods.

subworkflow\_ref Nested workflow reference used by `$set_node_nested_workflow()`.  
 nested\_workflow Embedded nested workflow used by `$set_node_nested_workflow()`.

**Method** `apply_human_feedback()`: Apply legacy structured human feedback to the workflow.

*Usage:*

```
Scaffolder$apply_human_feedback(
  completeness = NULL,
  add = NULL,
  remove = NULL,
  rule_specs = list(),
  confidence = list()
)
```

*Arguments:*

completeness Named list of completion flags used by `$apply_human_feedback()`.  
 add List of node records to add in `$edit_workflow()`.  
 remove Character vector of node ids to remove in `$edit_workflow()`.  
 rule\_specs Named list of rule specs used by `$edit_workflow()`.  
 confidence Confidence value used by review/edit helpers.

**Method** `recommend_subsystems()`: Recommend optional subsystem/capability labels for the current task and workflow.

*Usage:*

```
Scaffolder$recommend_subsystems(task = self$task)
```

*Arguments:*

task Task text used by `$evaluate_task()` and `$decompose_task()`.

**Method** `subsystem_recommendations()`: Return the current subsystem recommendation records.

*Usage:*

```
Scaffolder$subsystem_recommendations()
```

**Method** `subsystem_recommendation_rationale()`: Return stored recommendation rationale.

*Usage:*

```
Scaffolder$subsystem_recommendation_rationale(subsystem = NULL)
```

*Arguments:*

subsystem Optional subsystem name used by `$subsystem_recommendation_rationale()`.

**Method** `select_subsystems()`: Store the selected subsystem configuration.

*Usage:*

```
Scaffolder$select_subsystems(subsystems)
```

*Arguments:*

subsystems Selected subsystems used by `$select_subsystems()`.

**Method** `selected_subsystems()`: Return the currently selected subsystem names.

*Usage:*

Scaffolder\$selected\_subsystems()

**Method** label\_workflow\_subsystems(): Assign subsystem owners to workflow nodes.

*Usage:*

```
Scaffolder$label_workflow_subsystems(labels)
```

*Arguments:*

labels Named node-to-subsystem assignments used by \$label\_workflow\_subsystems().

**Method** edit\_workflow\_subsystems(): Edit workflow-node subsystem ownership incrementally.

*Usage:*

```
Scaffolder$edit_workflow_subsystems(
  set = NULL,
  add = NULL,
  remove = NULL,
  clear = NULL
)
```

*Arguments:*

set Named node-to-subsystem assignments used by \$edit\_workflow\_subsystems().

add List of node records to add in \$edit\_workflow().

remove Character vector of node ids to remove in \$edit\_workflow().

clear Character vector of node ids whose ownership labels should be cleared by \$edit\_workflow\_subsystems().

**Method** propose\_agent\_spec(): Store a draft agent-spec proposal.

*Usage:*

```
Scaffolder$propose_agent_spec(
  agent_name = "agentr-agent",
  summary = NULL,
  subsystems = NULL,
  workflow = NULL,
  workflow_proposal_id = NULL,
  state_requirements = list(),
  interfaces = list(),
  implementation_targets = list(),
  metadata = list(),
  source = "model",
  notes = NULL
)
```

*Arguments:*

agent\_name Agent name used by \$approve\_agent\_spec().

summary Optional task summary used by \$evaluate\_task().

subsystems Selected subsystems used by \$select\_subsystems().

workflow Proposed workflow used by proposal methods.

workflow\_proposal\_id Workflow proposal id used to seed an agent-spec proposal.

state\_requirements State requirements used by \$approve\_agent\_spec().

*interfaces* Interfaces used by `$approve_agent_spec()`.  
*implementation\_targets* Implementation targets used by `$approve_agent_spec()`.  
*metadata* Agent-spec metadata used by `$approve_agent_spec()`.  
*source* Discussion source used by `$discuss_task()`.  
*source* Proposal source used by proposal methods.  
*notes* Optional decomposition notes accepted directly by `$decompose_task()`.  
*notes* Optional review notes.  
*notes* Optional proposal notes.

**Method** `list_agent_spec_proposals()`: Return stored agent-spec proposal summaries.

*Usage:*

```
Scaffolder$list_agent_spec_proposals(status = NULL)
```

*Arguments:*

*status* Review status used by `$review_workflow()` and `$review_node()`.

**Method** `get_agent_spec_proposal()`: Return a stored agent-spec proposal by id.

*Usage:*

```
Scaffolder$get_agent_spec_proposal(proposal_id)
```

*Arguments:*

*proposal\_id* Workflow proposal identifier.

**Method** `discuss_agent_spec_proposal()`: Attach discussion feedback to a draft agent-spec proposal.

*Usage:*

```
Scaffolder$discuss_agent_spec_proposal(
  proposal_id,
  feedback,
  source = "human",
  confidence = NA_real_
)
```

*Arguments:*

*proposal\_id* Workflow proposal identifier.

*feedback* Free-form discussion feedback used by `$discuss_task()`.

*source* Discussion source used by `$discuss_task()`.

*source* Proposal source used by proposal methods.

*confidence* Confidence value used by review/edit helpers.

**Method** `approve_agent_spec_proposal()`: Approve a stored agent-spec proposal and optionally approve its linked workflow proposal.

*Usage:*

```
Scaffolder$approve_agent_spec_proposal(
  proposal_id,
  approve_linked_workflow = TRUE
)
```

*Arguments:*

proposal\_id Workflow proposal identifier.

approve\_linked\_workflow Whether linked workflow proposals should be approved when approving an agent-spec proposal.

**Method** approve\_agent\_spec(): Approve an agent spec built from the current scaffolding state.

*Usage:*

```
Scaffolder$approve_agent_spec(
  agent_name = "agentr-agent",
  summary = NULL,
  state_requirements = list(),
  interfaces = list(),
  implementation_targets = list(),
  metadata = list()
)
```

*Arguments:*

agent\_name Agent name used by \$approve\_agent\_spec().

summary Optional task summary used by \$evaluate\_task().

state\_requirements State requirements used by \$approve\_agent\_spec().

interfaces Interfaces used by \$approve\_agent\_spec().

implementation\_targets Implementation targets used by \$approve\_agent\_spec().

metadata Agent-spec metadata used by \$approve\_agent\_spec().

**Method** agent\_spec(): Return the approved agent spec or a draft spec built from current state.

*Usage:*

```
Scaffolder$agent_spec()
```

**Method** propose\_workflow(): Store a pending workflow proposal for preview and review.

*Usage:*

```
Scaffolder$propose_workflow(workflow, source = "model", notes = NULL)
```

*Arguments:*

workflow Proposed workflow used by proposal methods.

source Discussion source used by \$discuss\_task().

source Proposal source used by proposal methods.

notes Optional decomposition notes accepted directly by \$decompose\_task().

notes Optional review notes.

notes Optional proposal notes.

**Method** list\_workflow\_proposals(): Return a summary table of stored workflow proposals.

*Usage:*

```
Scaffolder$list_workflow_proposals(status = NULL)
```

*Arguments:*

status Review status used by \$review\_workflow() and \$review\_node().

**Method** `get_workflow_proposal()`: Return a stored WorkflowProposal object by identifier.

*Usage:*

```
Scaffolder$get_workflow_proposal(proposal_id)
```

*Arguments:*

`proposal_id` Workflow proposal identifier.

**Method** `approve_workflow_proposal()`: Promote a stored workflow proposal to the live workflow and supersede older active proposals when applicable.

*Usage:*

```
Scaffolder$approve_workflow_proposal(proposal_id)
```

*Arguments:*

`proposal_id` Workflow proposal identifier.

**Method** `discuss_workflow_proposal()`: Attach free-form discussion feedback to a non-approved workflow proposal and transition it into discussion state when needed.

*Usage:*

```
Scaffolder$discuss_workflow_proposal(  
  proposal_id,  
  feedback,  
  source = "human",  
  confidence = NA_real_  
)
```

*Arguments:*

`proposal_id` Workflow proposal identifier.

`feedback` Free-form discussion feedback used by `$discuss_task()`.

`source` Discussion source used by `$discuss_task()`.

`source` Proposal source used by proposal methods.

`confidence` Confidence value used by review/edit helpers.

**Method** `workflow_spec()`: Validate and return the current workflow specification.

*Usage:*

```
Scaffolder$workflow_spec()
```

**Method** `implementation_spec()`: Return an implementation-facing summary of workflow nodes and rules.

*Usage:*

```
Scaffolder$implementation_spec()
```

**Method** `low_confidence_nodes()`: Return workflow nodes whose confidence falls below the completion threshold.

*Usage:*

```
Scaffolder$low_confidence_nodes()
```

**Method** `get_node()`: Return a single workflow node by identifier.

*Usage:*`Scaffolder$get_node(node_id)`*Arguments:*`node_id` Workflow node identifier used by node-specific methods.

**Method** `record_interaction()`: Append an interaction record to the scaffolder log.

*Usage:*`Scaffolder$record_interaction(type, payload)`*Arguments:*`type` Interaction type used by `$record_interaction()`.`payload` Interaction payload used by `$record_interaction()`.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*`Scaffolder$clone(deep = FALSE)`*Arguments:*`deep` Whether to make a deep clone.

---

`scaffolder_action_methods`*List LLM-callable scaffolder methods*

---

**Description**

Returns the method names that an external reasoning system is allowed to request through a machine-readable scaffolding message.

**Usage**`scaffolder_action_methods()`**Value**

Character vector of allowed method names.

---

`schema_shape_graph_data`*Convert a schema shape into graph-ready data*

---

**Description**

Converts a nested R list, JSON-schema-like object, vector, or data frame into node and edge data frames. The output is a structural preview of the schema shape, not a validator.

**Usage**

```
schema_shape_graph_data(x, root_id = "schema", root_label = "schema")
```

**Arguments**

<code>x</code>	Schema object to inspect. Common inputs are workflow-node <code>input_schema</code> or <code>output_schema</code> lists.
<code>root_id</code>	Root node identifier.
<code>root_label</code>	Human-readable root node label.

**Value**

A list with vertices and edges data frames.

---

`set_workflow_node_automation_status`*Set one workflow node automation status*

---

**Description**

Set one workflow node automation status

**Usage**

```
set_workflow_node_automation_status(  
  workflow,  
  node_id,  
  automation_status,  
  target_automation_status = NULL  
)
```

**Arguments**

<code>workflow</code>	Workflow specification.
<code>node_id</code>	Node identifier.
<code>automation_status</code>	Automation-status value.
<code>target_automation_status</code>	Optional target automation-status value.

**Value**

Updated workflow specification.

---

`set_workflow_node_owner`

*Set one workflow node owner*

---

**Description**

Set one workflow node owner

**Usage**

```
set_workflow_node_owner(workflow, node_id, owner)
```

**Arguments**

<code>workflow</code>	Workflow specification.
<code>node_id</code>	Node identifier.
<code>owner</code>	Owner value.

**Value**

Updated workflow specification.

---

SubsystemSpec	<i>SubsystemSpec</i>
---------------	----------------------

---

**Description**

SubsystemSpec

SubsystemSpec

**Details**

Sparse diagnostic inventory of the selected agent subsystems.

**Methods**

`$initialize(rwm = NULL, pg = NULL, ae = NULL, iac = NULL, la = NULL, metadata = list())`

Create an optional subsystem diagnostic inventory.

`$validate()` Validate the subsystem diagnostic labels.

`$selected_subsystems()` Return the selected subsystem names.

`$persistence_requirements()` Return persistence requirements for selected subsystems.

`$communication_requirements()` Return communication requirements for selected subsystems.

`$summary()` Return a one-row summary table.

`$as_list()` Return a serializable representation.

**Public fields**

`rwm` An RWMConfig object or NULL.

`pg` A PGConfig object or NULL.

`ae` An AEConfig object or NULL.

`iac` An IACConfig object or NULL.

`la` A LAConfig object or NULL.

`metadata` Free-form metadata list.

**Methods****Public methods:**

- [SubsystemSpec\\$new\(\)](#)
- [SubsystemSpec\\$validate\(\)](#)
- [SubsystemSpec\\$selected\\_subsystems\(\)](#)
- [SubsystemSpec\\$persistence\\_requirements\(\)](#)
- [SubsystemSpec\\$communication\\_requirements\(\)](#)
- [SubsystemSpec\\$summary\(\)](#)
- [SubsystemSpec\\$as\\_list\(\)](#)

- [SubsystemSpec#print\(\)](#)
- [SubsystemSpec\\$clone\(\)](#)

**Method new():** Create an optional subsystem diagnostic inventory.

*Usage:*

```
SubsystemSpec$new(
  rwm = NULL,
  pg = NULL,
  ae = NULL,
  iac = NULL,
  la = NULL,
  metadata = list()
)
```

*Arguments:*

rwm An RWMConfig object or list payload.  
 pg A PGConfig object or list payload.  
 ae An AEConfig object or list payload.  
 iac An IACConfig object or list payload.  
 la A LAConfig object or list payload.  
 metadata Free-form metadata list.

**Method validate():** Validate the subsystem diagnostic labels.

*Usage:*

```
SubsystemSpec$validate()
```

**Method selected\_subsystems():** Return the selected subsystem names.

*Usage:*

```
SubsystemSpec$selected_subsystems()
```

**Method persistence\_requirements():** Return persistence requirements for selected subsystems.

*Usage:*

```
SubsystemSpec$persistence_requirements()
```

**Method communication\_requirements():** Return communication requirements for selected subsystems.

*Usage:*

```
SubsystemSpec$communication_requirements()
```

**Method summary():** Return a one-row summary table.

*Usage:*

```
SubsystemSpec$summary()
```

**Method as\_list():** Return a serializable representation.

*Usage:*

```
SubsystemSpec$as_list()
```

**Method** print(): Print a compact subsystem summary.

*Usage:*

```
SubsystemSpec$print(...)
```

*Arguments:*

... Unused print arguments.

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
SubsystemSpec$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

task\_family\_metadata *Create task-family metadata*

---

## Description

Task-family metadata describes a parent design space whose root workflow contains child-task nodes. It is stored under workflow\$metadata\$task\_family so the workflow remains a normal agentr\_workflow\_spec.

## Usage

```
task_family_metadata(
  id,
  label,
  objective,
  child_tasks = character(),
  dependency_policy = "independent_children_no_root_edges",
  shared_inputs = character(),
  shared_review_concerns = character(),
  task_tags = list(),
  metadata = list()
)
```

## Arguments

id	Task-family identifier.
label	Human-readable task-family label.
objective	Family-level objective.
child_tasks	Character vector of child-task node ids.
dependency_policy	Description of how root-level child nodes should be interpreted.

shared\_inputs Character vector of shared input names.  
 shared\_review\_concerns  
                   Character vector of shared review concerns.  
 task\_tags Optional named list mapping child-task ids to tags.  
 metadata Additional metadata.

**Value**

List suitable for workflow\$metadata\$task\_family.

---

task_spec_paths	<i>Return conventional task-local spec paths</i>
-----------------	--

---

**Description**

Coding-assistant workflows commonly keep editable agentr specs under a task-local docs/ directory. This helper returns the conventional paths without creating or modifying files.

**Usage**

```
task_spec_paths(task_dir, docs_dir = "docs")
```

**Arguments**

task\_dir Task root directory.  
 docs\_dir Documentation/spec directory relative to task\_dir, or an absolute path.

**Value**

Named list of task-local paths.

---

terminal_ask_node_complete	<i>Ask for workflow-node completeness in the terminal</i>
----------------------------	---

---

**Description**

Ask for workflow-node completeness in the terminal

**Usage**

```
terminal_ask_node_complete(scaffolder, node_id)
```

**Arguments**

scaffolder	A <a href="#">Scaffolder</a> instance.
node_id	Workflow node identifier.

**Value**

A list containing the prompt and response.

---

terminal\_ask\_node\_rule

*Ask for a node-specific rule in the terminal*

---

**Description**

Ask for a node-specific rule in the terminal

**Usage**

```
terminal_ask_node_rule(scaffolder, node_id)
```

**Arguments**

scaffolder	A <a href="#">Scaffolder</a> instance.
node_id	Workflow node identifier.

**Value**

A list containing the prompt and response.

---

terminal\_ask\_workflow\_changes

*Ask for workflow changes in the terminal*

---

**Description**

Ask for workflow changes in the terminal

**Usage**

```
terminal_ask_workflow_changes(scaffolder)
```

**Arguments**

scaffolder	A <a href="#">Scaffolder</a> instance.
------------	--

**Value**

A list containing the prompt and response.

---

terminal\_discuss\_task *Capture free-form terminal feedback and record it in the scaffolder*

---

### Description

Capture free-form terminal feedback and record it in the scaffolder

### Usage

```
terminal_discuss_task(  
  scaffolder,  
  prompt = "Share feedback for the current task or workflow.",  
  source = "human",  
  node_id = NULL  
)
```

### Arguments

scaffolder	A <a href="#">Scaffolder</a> instance.
prompt	Character string shown to the human.
source	Discussion source label.
node_id	Optional workflow node identifier.

### Value

A list containing the prompt and recorded response.

---

terminal\_scaffold\_input  
*Prompt for terminal input during scaffolding*

---

### Description

Prompt for terminal input during scaffolding

### Usage

```
terminal_scaffold_input(prompt)
```

### Arguments

prompt	Character string shown to the human.
--------	--------------------------------------

### Value

Character string entered by the user.

---

`validate_design_feedback`*Validate structured design feedback*

---

**Description**

Validate structured design feedback

**Usage**

```
validate_design_feedback(x, review_spec = NULL)
```

**Arguments**

<code>x</code>	A feedback item, list of feedback items, or parsed feedback bundle containing a feedback field.
<code>review_spec</code>	Optional <a href="#">DesignReviewSpec</a> or review-spec list used to warn when feedback target ids no longer exist in the reviewed design.

**Value**

The validated feedback, invisibly.

---

`validate_design_review_spec`*Validate a design-review specification*

---

**Description**

Validate a design-review specification

**Usage**

```
validate_design_review_spec(x)
```

**Arguments**

<code>x</code>	A <a href="#">DesignReviewSpec</a> object or serializable design-review list.
----------------	---

**Value**

The validated object, invisibly.

validate\_knowledge\_item

*Validate a knowledge specification item*

---

**Description**

Validate a knowledge specification item

**Usage**

validate\_knowledge\_item(item)

**Arguments**

item                    Knowledge-item list.

**Value**

Validated item, invisibly.

---

validate\_knowledge\_proposal

*Validate a knowledge proposal*

---

**Description**

Validate a knowledge proposal

**Usage**

validate\_knowledge\_proposal(x)

**Arguments**

x                        Knowledge proposal record or object.

**Value**

Validated proposal, invisibly.

---

validate\_knowledge\_spec

*Validate a knowledge specification*

---

**Description**

Validate a knowledge specification

**Usage**

validate\_knowledge\_spec(x)

**Arguments**

x                    A [KnowledgeSpec](#) object.

**Value**

The validated object, invisibly.

---

validate\_memory\_field    *Validate a memory field*

---

**Description**

Validate a memory field

**Usage**

validate\_memory\_field(x)

**Arguments**

x                    Memory field list.

**Value**

The validated field, invisibly.

validate\_memory\_proposal

*Validate a memory proposal*

---

**Description**

Validate a memory proposal

**Usage**

validate\_memory\_proposal(x)

**Arguments**

x                   Memory proposal record or object.

**Value**

The validated proposal, invisibly.

---

validate\_memory\_spec   *Validate a MemorySpec*

---

**Description**

Validate a MemorySpec

**Usage**

validate\_memory\_spec(x)

**Arguments**

x                   A [MemorySpec](#) object.

**Value**

The validated object, invisibly.

---

```
validate_scaffolder_message
    Validate a machine-readable scaffolder message
```

---

**Description**

Validate a machine-readable scaffolder message

**Usage**

```
validate_scaffolder_message(x, allowed_methods = scaffolder_action_methods())
```

**Arguments**

`x`                      Parsed scaffolder message.  
`allowed_methods`        Character vector of allowed method names.

**Value**

The validated message, invisibly.

---

```
validate_task_specs    Validate task-local specs
```

---

**Description**

Validates conventional task-local YAML specs when present and reports missing or invalid files without mutating the task directory.

**Usage**

```
validate_task_specs(
  task_dir,
  docs_dir = "docs",
  require = character(),
  stop_on_error = FALSE
)
```

**Arguments**

`task_dir`                Task root directory.  
`docs_dir`                Documentation/spec directory relative to `task_dir`, or an absolute path.  
`require`                Character vector of spec types that must exist. Supported values are workflow, memory, and knowledge.  
`stop_on_error`        Whether to stop when required or present specs are invalid.

**Value**

Data frame with one row per spec type.

---

```
validate_workflow_proposal
      Validate a workflow proposal
```

---

**Description**

Checks that a workflow proposal has the expected structure, valid lifecycle status, and a valid embedded workflow specification.

**Usage**

```
validate_workflow_proposal(x)
```

**Arguments**

x                      Workflow proposal object.

**Value**

The validated proposal, invisibly.

---

```
validate_workflow_spec
      Validate a workflow specification
```

---

**Description**

Validate a workflow specification

**Usage**

```
validate_workflow_spec(x, knowledge_spec = NULL, warn_missing_knowledge = TRUE)
```

**Arguments**

x                      Workflow specification.  
 knowledge\_spec      Optional [KnowledgeSpec](#) used to warn about missing, non-approved, or inactive knowledge\_refs.  
 warn\_missing\_knowledge      Whether to emit warnings about unresolved knowledge references when knowledge\_spec is supplied.

**Value**

The validated object, invisibly.

---

workflow_edge	<i>Create a workflow edge record</i>
---------------	--------------------------------------

---

## Description

Create a workflow edge record

## Usage

```
workflow_edge(  
  from,  
  to,  
  relation = "depends_on",  
  confidence = NA_real_,  
  notes = NA_character_,  
  condition = NA_character_,  
  branch_group = NA_character_,  
  mutually_exclusive = NA  
)
```

## Arguments

from	Source node id.
to	Target node id.
relation	Edge relation label. Common relations include depends_on, branch, exclusive_branch, reads, writes, updates, prompts_with, validates_against, and produces.
confidence	Optional edge confidence score between 0 and 1.
notes	Optional edge notes.
condition	Optional branch or transition condition.
branch_group	Optional identifier for related branch alternatives.
mutually_exclusive	Whether the edge is mutually exclusive with other edges in the same branch group.

## Value

One-row data frame.

---

workflow\_graph\_data     *Convert a workflow specification into graph-ready data*

---

### Description

Returns node and edge data frames in a shape that is directly usable by graph packages and renderers such as DiagrammeR.

### Usage

```
workflow_graph_data(  
  x,  
  highlight_low_confidence = TRUE,  
  confidence_threshold = 0.6  
)
```

### Arguments

x                     A workflow specification or a [Scaffolder](#) instance.  
highlight\_low\_confidence             Whether to add a low-confidence flag based on confidence\_threshold.  
confidence\_threshold             Threshold for low-confidence highlighting.

### Value

A list with vertices and edges data frames.

---

workflow\_node             *Create a workflow node record*

---

### Description

Create a workflow node record

### Usage

```
workflow_node(  
  id,  
  label,  
  confidence = NA_real_,  
  human_required = TRUE,  
  rule_spec = NA_character_,  
  implementation_hint = NA_character_,  
  complete = FALSE,
```

```

review_status = "pending",
review_notes = NA_character_,
review_confidence = NA_real_,
node_kind = "action",
owner = NA_character_,
automation_status = NA_character_,
human_owned_reason = NA_character_,
target_automation_status = NA_character_,
trace_required = NA,
knowledge_refs = character(),
source_path = NA_character_,
retrieval_mode = NA_character_,
persistence = NA_character_,
linked_spec_ids = character(),
subworkflow_ref = NA_character_,
input_schema = list(),
output_schema = list(),
nested_workflow = NULL
)

```

### Arguments

id	Node identifier.
label	Human-readable node label.
confidence	Provisional confidence score between 0 and 1.
human_required	Whether human confirmation is required.
rule_spec	Optional node-specific rule specification.
implementation_hint	Optional implementation hint.
complete	Whether the node is considered complete.
review_status	Node-level review status.
review_notes	Optional node-level review notes.
review_confidence	Optional confidence attached to the latest review.
node_kind	Optional workflow node kind. Supported values are action, status, knowledge, memory, file, api, schema, and data.
owner	Optional current owner for the node.
automation_status	Optional current automation status for the node.
human_owned_reason	Optional explanation for why the node remains human-owned.
target_automation_status	Optional target automation status for the node.
trace_required	Optional logical flag indicating whether decision traces should be collected.

knowledge_refs	Optional character vector of referenced knowledge-item ids.
source_path	Optional path, URI, or symbolic source for data/resource nodes.
retrieval_mode	Optional retrieval mode for data/resource nodes.
persistence	Optional persistence description for data/resource nodes.
linked_spec_ids	Optional character vector of linked knowledge, memory, schema, or interface spec ids.
subworkflow_ref	Optional identifier or path for a nested workflow.
input_schema	Optional structured input schema for the node.
output_schema	Optional structured output schema for the node.
nested_workflow	Optional nested workflow spec or list with nodes and edges, used by review UIs for drilldown.

**Value**

One-row data frame.

---

workflow\_proposal\_graph\_data

*Convert a workflow proposal into graph-ready data*

---

**Description**

Exports graph-ready vertex and edge tables for a stored workflow proposal. This accepts either a workflow proposal object directly or a `Scaffolder` plus a proposal id.

**Usage**

```
workflow_proposal_graph_data(x, proposal_id = NULL)
```

**Arguments**

x	A workflow proposal object or a <code>Scaffolder</code> instance.
proposal_id	Optional proposal id when x is a <code>Scaffolder</code> .

**Value**

A list with vertices and edges.

---

`workflow_spec_from_json`*Build a workflow specification from extracted JSON*

---

**Description**

Converts reasoning-model output produced from `build_workflow_extraction_prompt()` into a validated `agentr_workflow_spec` object.

**Usage**`workflow_spec_from_json(x)`**Arguments**

`x` Parsed list, raw JSON string, or path to a `.json` file.

**Value**

A validated workflow specification.

---

`workflow_spec_from_yaml`*Build a workflow specification from YAML*

---

**Description**

Build a workflow specification from YAML

**Usage**`workflow_spec_from_yaml(file_path)`**Arguments**

`file_path` Path to a `.yaml` or `.yml` workflow specification file.

**Value**

A validated workflow specification.

---

WorkflowProposal

*WorkflowProposal*


---

## Description

WorkflowProposal

WorkflowProposal

## Details

Public workflow proposal object with explicit lifecycle state and persistence helpers.

## Methods

`$initialize(...)` Create a workflow proposal object.

`$validate()` Validate the proposal state and embedded workflow.

`$as_list()` Return the proposal as a serializable proposal record.

`$summary()` Return a one-row summary data frame.

`$transition(to_status, timestamp = Sys.time(), superseded_by = NULL, supersedes = NULL)`  
Apply a valid lifecycle transition.

`$discuss(feedback, source = "human", confidence = NA_real_, timestamp = Sys.time())`  
Append a discussion round and transition into discussion state when needed.

`$graph_data()` Export graph-ready data from the proposed workflow.

`$save(file_path)` Save the proposal to disk.

## Public fields

`id` Workflow proposal identifier.

`status` Workflow proposal lifecycle status.

`source` Proposal source label.

`notes` Optional proposal notes.

`workflow` Proposed workflow specification.

`discussion_rounds` Stored discussion rounds.

`created_at` Proposal creation time.

`updated_at` Latest proposal update time.

`approved_at` Approval time.

`superseded_by` Newer proposal id that superseded this proposal.

`supersedes` Older proposal id superseded by this proposal.

`rejected_at` Rejection time.

## Methods

### Public methods:

- `WorkflowProposal$new()`
- `WorkflowProposal$validate()`
- `WorkflowProposal$as_list()`
- `WorkflowProposal$summary()`
- `WorkflowProposal$transition()`
- `WorkflowProposal$discuss()`
- `WorkflowProposal$graph_data()`
- `WorkflowProposal$save()`
- `WorkflowProposal$clone()`

**Method** `new()`: Create a `WorkflowProposal`.

*Usage:*

```
WorkflowProposal$new(  
  id,  
  workflow,  
  status = "pending",  
  source = "model",  
  notes = NULL,  
  discussion_rounds = list(),  
  created_at = Sys.time(),  
  updated_at = created_at,  
  approved_at = as.POSIXct(NA),  
  superseded_by = NA_character_,  
  supersedes = NA_character_,  
  rejected_at = as.POSIXct(NA)  
)
```

*Arguments:*

`id` Workflow proposal identifier.  
`workflow` Proposed workflow specification.  
`status` Workflow proposal lifecycle status.  
`source` Proposal source label.  
`notes` Optional proposal notes.  
`discussion_rounds` Stored discussion rounds.  
`created_at` Proposal creation time.  
`updated_at` Latest proposal update time.  
`approved_at` Approval time.  
`superseded_by` Newer proposal id that superseded this proposal.  
`supersedes` Older proposal id superseded by this proposal.  
`rejected_at` Rejection time.

**Method** `validate()`: Validate the proposal.

*Usage:*

```
WorkflowProposal$validate()
```

**Method** `as_list()`: Return a serializable proposal record.

*Usage:*

```
WorkflowProposal$as_list()
```

**Method** `summary()`: Return a one-row proposal summary.

*Usage:*

```
WorkflowProposal$summary()
```

**Method** `transition()`: Apply a valid lifecycle transition.

*Usage:*

```
WorkflowProposal$transition(
  to_status,
  timestamp = Sys.time(),
  superseded_by = NULL,
  supersedes = NULL
)
```

*Arguments:*

`to_status` Target lifecycle status used by `$transition()`.  
`timestamp` Timestamp used by `$transition()` and `$discuss()`.  
`superseded_by` Newer proposal id that superseded this proposal.  
`supersedes` Older proposal id superseded by this proposal.

**Method** `discuss()`: Append a discussion round to the proposal.

*Usage:*

```
WorkflowProposal$discuss(
  feedback,
  source = "human",
  confidence = NA_real_,
  timestamp = Sys.time()
)
```

*Arguments:*

`feedback` Discussion feedback used by `$discuss()`.  
`source` Proposal source label.  
`confidence` Optional discussion confidence used by `$discuss()`.  
`timestamp` Timestamp used by `$transition()` and `$discuss()`.

**Method** `graph_data()`: Export graph-ready proposal workflow data.

*Usage:*

```
WorkflowProposal$graph_data()
```

**Method** `save()`: Save the proposal to disk.

*Usage:*

```
WorkflowProposal$save(file_path)
```

*Arguments:*

`file_path` File path used by `$save()`.

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
WorkflowProposal$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

WorkflowProposalState *WorkflowProposalState*

---

**Description**

WorkflowProposalState

WorkflowProposalState

**Details**

Public state container for the approved workflow plus stored workflow proposals.

**Methods**

`$initialize(approved_workflow = new_workflow_spec(...), proposals = list())` Create a workflow proposal state container.

`$set_approved_workflow(workflow)` Replace the approved workflow.

`$add_proposal(proposal)` Store a proposal object.

`$get_proposal(proposal_id)` Return a stored proposal by id.

`$list_proposals(status = NULL)` Return a summary table of proposals.

`$latest_proposal()` Return the latest stored proposal or NULL.

`$active_proposals()` Return active proposal objects.

`$proposal_history()` Return proposal history in insertion order.

`$approve_proposal(proposal_id, timestamp = Sys.time())` Approve a proposal, update the approved workflow, and supersede older active proposals.

`$as_list()` Return a serializable state snapshot.

**Public fields**

`approved_workflow` Current approved workflow specification.

`proposals` Named list of WorkflowProposal objects.

**Methods****Public methods:**

- WorkflowProposalState\$new()
- WorkflowProposalState\$set\_approved\_workflow()
- WorkflowProposalState\$add\_proposal()
- WorkflowProposalState\$get\_proposal()
- WorkflowProposalState\$list\_proposals()
- WorkflowProposalState\$latest\_proposal()
- WorkflowProposalState\$active\_proposals()
- WorkflowProposalState\$proposal\_history()
- WorkflowProposalState\$approve\_proposal()
- WorkflowProposalState\$as\_list()
- WorkflowProposalState\$clone()

**Method** new(): Create a WorkflowProposalState.

*Usage:*

```
WorkflowProposalState$new(
  approved_workflow = new_workflow_spec(nodes = .empty_workflow_nodes(), edges =
    .empty_workflow_edges(), task = NULL, metadata = list(evaluation = NULL,
    workflow_review = NULL, discussion_rounds = list())),
  proposals = list()
)
```

*Arguments:*

approved\_workflow Current approved workflow specification used by \$initialize().  
 proposals Initial proposal objects used by \$initialize().

**Method** set\_approved\_workflow(): Replace the approved workflow.

*Usage:*

```
WorkflowProposalState$set_approved_workflow(workflow)
```

*Arguments:*

workflow Approved workflow specification used by \$set\_approved\_workflow().

**Method** add\_proposal(): Store a proposal object.

*Usage:*

```
WorkflowProposalState$add_proposal(proposal)
```

*Arguments:*

proposal Proposal object used by \$add\_proposal().

**Method** get\_proposal(): Return a stored proposal by id.

*Usage:*

```
WorkflowProposalState$get_proposal(proposal_id)
```

*Arguments:*

proposal\_id Proposal identifier used by \$get\_proposal() and \$approve\_proposal().

**Method** list\_proposals(): Return proposal summary rows.

*Usage:*

```
WorkflowProposalState$list_proposals(status = NULL)
```

*Arguments:*

status Optional proposal status filter used by \$list\_proposals().

**Method** latest\_proposal(): Return the latest proposal or NULL.

*Usage:*

```
WorkflowProposalState$latest_proposal()
```

**Method** active\_proposals(): Return active proposals.

*Usage:*

```
WorkflowProposalState$active_proposals()
```

**Method** proposal\_history(): Return proposal history.

*Usage:*

```
WorkflowProposalState$proposal_history()
```

**Method** approve\_proposal(): Approve a stored proposal and supersede older active proposals.

*Usage:*

```
WorkflowProposalState$approve_proposal(proposal_id, timestamp = Sys.time())
```

*Arguments:*

proposal\_id Proposal identifier used by \$get\_proposal() and \$approve\_proposal().

timestamp Timestamp used by \$approve\_proposal().

**Method** as\_list(): Return a serializable state snapshot.

*Usage:*

```
WorkflowProposalState$as_list()
```

**Method** clone(): The objects of this class are cloneable with this method.

*Usage:*

```
WorkflowProposalState$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

# Index

add\_child\_task\_node, 5  
AEConfig, 6  
AffectiveConfig, 7  
AffectiveState, 9, 11, 12, 112  
AgentCore, 11, 112, 122, 123  
agentr\_workspace\_paths, 13  
AgentScaffoldState, 13  
AgentSpec, 15, 21, 22, 24, 27, 34, 36, 54, 58, 71, 113  
append\_decision\_trace, 18  
append\_reflection\_trace, 19  
apply\_design\_feedback, 19  
apply\_initial\_spec\_message, 20  
apply\_knowledge\_message, 20  
apply\_memory\_message, 21  
apply\_node\_detail\_message, 21  
apply\_revision\_message, 22  
apply\_scaffolder\_message, 23  
apply\_scaffolder\_message(), 43  
approve\_workspace\_proposal, 23  
article\_workflow\_specs\_from\_json, 24  
  
backup\_agent, 25  
build\_agent\_design\_prompt, 25  
build\_article\_workflow\_extraction\_prompt, 26  
build\_article\_workflow\_extraction\_prompt(), 24  
build\_design\_review\_data, 27  
build\_design\_review\_data(), 50, 53  
build\_implementation\_prompt, 28  
build\_initial\_spec\_prompt, 29  
build\_knowledge\_conflict\_check\_prompt, 29  
build\_knowledge\_design\_prompt, 30  
build\_knowledge\_elicitation\_prompt, 30  
build\_knowledge\_normalization\_prompt, 31  
build\_memory\_revision\_prompt, 31  
build\_memory\_schema\_prompt, 32  
  
build\_node\_detail\_prompt, 33  
build\_revision\_prompt, 33  
build\_scaffolder\_prompt, 34  
build\_workflow\_extraction\_prompt, 35  
build\_workflow\_extraction\_prompt(), 151  
build\_workspace\_implementation\_prompt, 36  
  
child\_task\_node, 37  
CognitiveConfig, 38  
CognitiveState, 11, 12, 40, 112  
collect\_scaffolder\_questions, 43  
combine\_emotions, 43  
combine\_emotions(), 44, 48  
compute\_blended\_emotions, 44  
create\_decision\_trace, 44  
create\_reflection\_trace, 45  
  
decay\_emotion\_state, 46  
default\_emotion\_state, 46  
default\_emotion\_state(), 9, 46, 48  
define\_random\_emotion\_state, 47  
define\_random\_emotion\_state(), 46, 48  
describe\_emotional\_state, 47  
design\_feedback\_item, 48  
design\_review\_html, 49  
design\_review\_html(), 53, 54  
DesignReviewSpec, 27, 50, 50, 53, 72, 92, 114, 141  
discover\_task\_specs, 52  
  
export\_design\_review\_html, 53  
export\_design\_review\_html(), 108  
export\_workspace\_design\_review, 53  
  
IACConfig, 54  
import\_extracted\_workflow, 56  
inferencer\_available, 57  
inferencer\_integration, 57

- init\_agentr\_proposal\_states, 58
- init\_agentr\_workspace, 58
- IntelligentAgent, 27, 59
- jsonlite::fromJSON(), 73
- knowledge\_action\_methods, 61
- knowledge\_graph\_data, 61
- knowledge\_graph\_data(), 62
- knowledge\_graph\_from\_spec, 62
- KnowledgeProposal, 62, 64, 73, 114
- KnowledgeProposalState, 20, 27, 30, 64, 100
- KnowledgeSpec, 27, 30, 61, 62, 64, 66, 74, 75, 98, 104, 105, 115, 116, 143, 146
- LAConfig, 68
- list\_workspace\_proposals, 70
- load\_agent, 71
- load\_agent\_spec, 71
- load\_design\_feedback, 72
- load\_design\_review\_spec, 72
- load\_json\_file, 73
- load\_knowledge\_proposal, 73
- load\_knowledge\_spec, 74
- load\_knowledge\_spec\_json, 74
- load\_knowledge\_spec\_yaml, 75
- load\_memory\_spec, 75
- load\_memory\_spec\_json, 76
- load\_memory\_spec\_yaml, 76
- load\_subsystem\_spec, 77
- load\_task\_specs, 77
- load\_workflow\_proposal, 78
- load\_workflow\_spec, 78
- load\_workflow\_spec\_json, 79
- load\_workflow\_spec\_yaml, 79
- load\_yaml\_file, 80
- mark\_node\_agent\_owned, 80
- mark\_node\_human\_owned, 81
- memory\_action\_methods, 81
- memory\_field, 82
- memory\_persistence\_policies, 83
- memory\_schema\_graph\_data, 83
- memory\_types, 84
- MemoryProposal, 84, 87, 88
- MemoryProposalState, 21, 27, 32, 87, 100
- MemorySpec, 27, 32, 61, 62, 75, 76, 83–85, 87, 88, 89, 98, 104–106, 116, 117, 144
- new\_design\_review\_spec, 92
- new\_task\_family\_workflow, 92
- new\_workflow\_spec, 93
- normalize\_subsystem\_key, 94
- parse\_design\_feedback\_json, 94
- parse\_knowledge\_message, 95
- parse\_memory\_message, 95
- parse\_scaffolder\_message, 96
- PGConfig, 96
- plot\_knowledge\_graph, 98
- plot\_workflow\_graph, 98
- preview\_design\_feedback, 99
- preview\_knowledge\_message, 100
- preview\_memory\_message, 100
- preview\_scaffolder\_message, 101
- print\_agentr\_workflow\_proposal, 102
- print\_agentr\_workflow\_spec, 102
- read\_decision\_traces, 103
- read\_reflection\_traces, 103
- reject\_workspace\_proposal, 104
- render\_knowledge\_graphviz, 104
- render\_markdown\_terminal, 105
- render\_memory\_schema\_graphviz, 106
- render\_schema\_shape\_graphviz, 107
- render\_task\_preview, 108
- render\_task\_preview(), 109
- render\_task\_previews, 109
- render\_workflow\_graphviz, 109
- RWMConfig, 110
- save\_agent, 112
- save\_agent\_spec, 113
- save\_design\_feedback, 113
- save\_design\_review\_spec, 114
- save\_knowledge\_proposal, 114
- save\_knowledge\_spec, 115
- save\_knowledge\_spec\_json, 115
- save\_knowledge\_spec\_yaml, 116
- save\_memory\_spec, 116
- save\_memory\_spec\_json, 117
- save\_memory\_spec\_yaml, 117
- save\_subsystem\_spec, 118
- save\_workflow\_proposal, 118
- save\_workflow\_spec, 119
- save\_workflow\_spec\_json, 119
- save\_workflow\_spec\_yaml, 120

Scaffolder, [11](#), [12](#), [19](#), [23](#), [25](#), [27](#), [28](#), [34](#), [43](#),  
[56](#), [99](#), [101](#), [110](#), [112](#), [120](#), [139](#), [140](#),  
[148](#), [150](#)

scaffolder\_action\_methods, [132](#)

schema\_shape\_graph\_data, [133](#)

set\_workflow\_node\_automation\_status,  
[133](#)

set\_workflow\_node\_owner, [134](#)

SubsystemSpec, [77](#), [118](#), [135](#)

task\_family\_metadata, [137](#)

task\_spec\_paths, [138](#)

terminal\_ask\_node\_complete, [138](#)

terminal\_ask\_node\_rule, [139](#)

terminal\_ask\_workflow\_changes, [139](#)

terminal\_discuss\_task, [140](#)

terminal\_scaffold\_input, [140](#)

validate\_design\_feedback, [141](#)

validate\_design\_review\_spec, [141](#)

validate\_knowledge\_item, [142](#)

validate\_knowledge\_proposal, [142](#)

validate\_knowledge\_spec, [143](#)

validate\_memory\_field, [143](#)

validate\_memory\_proposal, [144](#)

validate\_memory\_spec, [144](#)

validate\_scaffolder\_message, [145](#)

validate\_task\_specs, [145](#)

validate\_workflow\_proposal, [146](#)

validate\_workflow\_spec, [146](#)

workflow\_edge, [147](#)

workflow\_graph\_data, [148](#)

workflow\_node, [148](#)

workflow\_proposal\_graph\_data, [150](#)

workflow\_spec\_from\_json, [151](#)

workflow\_spec\_from\_yaml, [151](#)

WorkflowProposal, [27](#), [152](#)

WorkflowProposalState, [27](#), [155](#)