Package: ruleatlas (via r-universe)

November 2, 2026

Type Package

Title Syntactic Sport and Game Rules

Version 0.0.1

Description Provides native validation of sport and game states, permitted moves and observed events, deterministic rule consequences, and outcome calculation. Includes 212 profiles with explicit variant scopes and a bounded declarative interpreter for versioned user-defined rules. Does not simulate physical movement, generate strategies, or estimate winning probabilities.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports jsonlite, stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

URL https://github.com/rds0001/rulesatlas-r

BugReports https://github.com/rds0001/rulesatlas-r/issues

Author Dimitrios Geromichalos [cre], RiskDataScience GmbH [aut, cph]

Maintainer Dimitrios Geromichalos <riskdatascience@web.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-02 17:21:16 UTC

RemoteUrl https://github.com/cran/ruleatlas

RemoteRef HEAD

RemoteSha bed22cb7d8d6f07286f6fad4ae70f0abbff403ce

Contents

ruleatlas-package . . . 2
apply_action . . . 3
get_capabilities . . . 4
get_game . . . 5
legal_actions . . . 6
list_games . . . 7
new_state . . . 8
outcome . . . 9
register_rules . . . 11
ruleatlas-formats . . . 12
ruleatlas-profiles . . . 14
unregister_rules . . . 20
validate_rules . . . 22
validate_state . . . 23

Index **24**

`ruleatlas-package` *Native Syntactic Sport and Game Rules*

Description

Validate rule-relevant states, inspect permitted next moves or observed events, apply deterministic consequences and determine outcomes. Includes 212 retained profiles and a bounded declarative custom-rule interpreter. No simulation, movement model, strategy, solver or prediction is included.

Details

The public API is the same as the native Python library. R does not call Python or a remote service. Each profile implements one declared variant; inspect its scope, observations and limitations before use. Errors and returned objects follow the portable conventions in ruleatlas-formats. The profile index is in ruleatlas-profiles.

Workflow

Use list_games, get_game and get_capabilities to select a profile. Create a state with new_state; inspect it with validate_state, legal_actions and outcome. Submit one move or official event with apply_action. Its result is a new state; the input is unchanged. Custom rulepacks use validate_rules, register_rules and unregister_rules.

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")))
next_state <- apply_action(state, list(type = "place",
                                       params = list(cell = 0L)))
validate_state(next_state)$valid
outcome(next_state)$status
```

`apply_action` *Apply One Validated Rule Transition*

Description

Validate and apply one submitted move or observed event, including its rule consequences and postconditions.

Usage

```
apply_action(state, action)
```

Arguments

`state`	A named list containing game_id, ruleset_id, ruleset_version, variant, players and data, with optional metadata. Use new_state to create the envelope. See ruleatlas-formats.
`action`	A named list with type (one character string) and params (a named list). Optional actor is a participant ID; optional metadata is an object. Use a named empty list for parameterless actions. Profile-specific parameters are described by legal_actions.

Details

Terminal states reject further actions. Numeric indices in params are zero-based. An optional actor must identify a participant; this is not an authentication or private-information boundary. In the custom DSL, effect expressions read the old state and commit atomically after validation.

Value

A new validated state envelope of the same ruleset. The input state remains unchanged. Invalid actions raise a ruleatlas_error condition; no partial transition is committed.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")),
                   ruleset_version = "1.0.0")
action <- list(type = "place", params = list(cell = 0L))
after <- apply_action(state, action)
state$data$board[[1]]
after$data$board[[1]]
tryCatch(apply_action(after, action),
         ruleatlas_error = function(e) e$code)
```

`get_capabilities` *Inspect Rule-Checking Capabilities*

Description

Describe available operations and the scope of legality checks for a selected ruleset.

Usage

```
get_capabilities(game_id, ruleset_version = NULL, ruleset_id = NULL)
```

Arguments

`game_id`	One case-sensitive character string identifying a retained built-in profile or a registered custom game in the user. namespace. See ruleatlas-profiles.
`ruleset_version`	NULL or one character string selecting an exact version, for example "1.0.0". NULL succeeds only when selection is unambiguous. Pin versions when persisting states.
`ruleset_id`	NULL or one character string selecting a ruleset belonging to game_id. Unknown selections fail instead of falling back to another ruleset.

Details

Validation may be structural, local_rule_consistency or history_consistency. None of these implies a solver or a general proof of historical reachability. Read the profile limitations and inspect the current state with legal_actions().

Value

A named list containing new_state, validate_state, legal_actions, apply_action, outcome and validation_scope. Custom rulepacks also report max_enumeration = 512. legal_actions describes the capability; it is not a list of currently allowed actions.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
cap <- get_capabilities("G105")
cap$legal_actions
cap$validation_scope
```

`get_game` *Inspect a Profile and Its Declared Variant*

Description

Retrieve the supported variant, capabilities, scope and limitations before supplying a state or action.

Usage

```
get_game(game_id, ruleset_version = NULL, variant = NULL, ruleset_id = NULL)
```

Arguments

`game_id`	One case-sensitive character string identifying a retained built-in profile or a registered custom game in the user. namespace. See ruleatlas-profiles.
`ruleset_version`	NULL or one character string selecting an exact version, for example "1.0.0". NULL succeeds only when selection is unambiguous. Pin versions when persisting states.
`variant`	NULL or an empty named list. Nonempty variant overrides are currently rejected; the selected profile describes its fixed variant and any configuration actions.
`ruleset_id`	NULL or one character string selecting a ruleset belonging to game_id. Unknown selections fail instead of falling back to another ruleset.

Details

Returned built-in metadata is not an editable configuration object. Configuration, where supported, uses the actions declared by that profile. Modifying returned data does not modify the registry.

Value

For a built-in game, a named profile record with profile_id, default_ruleset_id, rulesets, capabilities, coverage_scope, known_limitations, provenance and evidence. For a custom game, the validated registered rulepack, including state, actions and outcomes.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
game <- get_game("G105", ruleset_version = "1.0.0")
game$name
game$coverage_scope
```

---

legal_actions *Inspect Currently Permitted Actions or Events*

Description

Return finite legal actions where supported, or typed event constraints for the current phase.

Usage

```
legal_actions(state)
```

Arguments

`state`	A named list containing game_id, ruleset_id, ruleset_version, variant, players and data, with optional metadata. Use new_state to create the envelope. See ruleatlas-formats.

Details

For custom rules, finite parameter combinations are enumerated only up to 512 combinations; larger or unbounded domains expose event types. A listed event type can still reject a particular parameter combination. External adjudication and missing information are explicit in profile capabilities and outcomes. Always submit an action through apply_action().

Value

A named list with mode, actions, action_types, constraints, completeness and reason. In enumerated mode, actions contains executable type/params objects. In event_types mode, action_types contains parameter schemas and preconditions; it is not an exhaustive list of valid parameter values. Terminal states return no actions.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")),
                   ruleset_version = "1.0.0")
legal <- legal_actions(state)
legal$mode
length(legal$actions)
legal$actions[[1]]
```

---

list_games *List Available Rule Profiles*

Description

List installed accepted profiles and session-local custom rulepacks. This does not run a game or search possible future states.

Usage

```
list_games(include_candidates = FALSE)
```

Arguments

`include_candidates`
: One logical value. If TRUE, also include catalog entries without accepted implementations. In this distribution all 212 catalog entries are implemented; removed profiles are not candidates.

Details

Custom versions may produce multiple records for one game ID. Use explicit ruleset selection for reproducible calls. Profile acceptance describes the declared implementation scope, not federation certification or legal clearance.

Value

A list of named lists. Built-in entries contain game_id, profile_id, name, implementation_status, rulesets, capabilities, coverage_scope, known_limitations, provenance and evidence. Custom summaries contain game_id, name, ruleset_id, ruleset_version and implementation_status. Candidate records, when present, identify their unsupported status.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
games <- list_games()
length(games)
head(vapply(games, function(x) x$game_id, character(1)))
```

---

new_state *Create an Initial Rule State*

Description

Create and validate an initial state for the selected built-in or registered custom ruleset.

Usage

```
new_state(game_id, players = NULL, variant = NULL, ruleset_version = NULL,
          ruleset_id = NULL)
```

Arguments

`game_id`	One case-sensitive character string identifying a retained built-in profile or a registered custom game in the user. namespace. See ruleatlas-profiles.
`players`	A list of participant records, each a named list with a distinct, nonempty character id. The number must satisfy the selected profile. Although the default is NULL, participant records are required.

`variant` NULL or an empty named list. Nonempty variant overrides are currently rejected; the selected profile describes its fixed variant and any configuration actions.

`ruleset_version` NULL or one character string selecting an exact version, for example "1.0.0". NULL succeeds only when selection is unambiguous. Pin versions when persisting states.

`ruleset_id` NULL or one character string selecting a ruleset belonging to game_id. Unknown selections fail instead of falling back to another ruleset.

Details

Player IDs identify results and actors. Numeric player and position indices in rule data and action parameters are zero-based even though R list indexing is one-based. Random draws, official decisions and observations are supplied by the caller. The function does not start clocks or generate random events.

Value

A named state-envelope list with game_id, ruleset_id, ruleset_version, variant, players and profile-specific data. The initial phase may require explicit setup actions before play.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")),
                   ruleset_version = "1.0.0")
state$data
validate_state(state)$valid
```

`outcome` *Determine the Current Rule Outcome*

Description

Determine whether the state is ongoing, terminal or requires further information or official adjudication under the selected ruleset.

Usage

```
outcome(state)
```

Arguments

state	A named list containing game_id, ruleset_id, ruleset_version, variant, players and data, with optional metadata. Use new_state to create the envelope. See ruleatlas-formats.

Details

An ongoing result is not a prediction. Ties, rankings, forfeits and adjudicated results depend on the declared variant. Invalid states raise an error instead of producing a guessed outcome. The custom DSL rejects states in which multiple outcome predicates hold simultaneously.

Value

A named list containing status, result_kind, winners, losers, placements, reason, evidence_required and rule_id. winners and losers are lists of participant IDs, not numeric player indices. placements and evidence_required are profile-specific lists. rule_id can be NULL when no result rule applies.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")),
                   ruleset_version = "1.0.0")
for (cell in c(0L, 3L, 1L, 4L, 2L)) {
  state <- apply_action(state, list(type = "place",
                                    params = list(cell = cell)))
}
result <- outcome(state)
result$status
result$winners
length(legal_actions(state)$actions)
```

`register_rules` *Register an Immutable Custom Ruleset Version*

Description

Validate and register a user-defined rulepack in the current R session so it can use the same state/action/outcome API as built-in games.

Usage

```
register_rules(rules_or_path, overwrite = FALSE)
```

Arguments

`rules_or_path`	A named list describing a custom rulepack, or one character string naming a regular local JSON file. Symbolic links, directories, URLs and files larger than 1 MiB are rejected. Use jsonlite::fromJSON(..., simplifyVector = FALSE) when parsing externally.
`overwrite`	One logical value retained for API compatibility. TRUE does not allow replacing an existing version with different contents; register a new version instead.

Details

Registration is process-local and does not write files or change the installed package. Built-in IDs are reserved. Multiple versions may coexist; select explicitly when more than one matches. Changing a version requires a new version identifier. Returned list modifications do not replace the registry entry.

Value

The validated registered rulepack as a named list. Registering identical contents for the same game_id/ruleset_id/ruleset_version is idempotent. Conflicting contents raise RULESET_CONFLICT.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
path <- system.file("extdata", "examples", "first_to_three.json",
                    package = "ruleatlas", mustWork = TRUE)
local({
  registered <- register_rules(path)
  on.exit(unregister_rules(registered$game_id,
                           ruleset_version = registered$ruleset_version))
  state <- new_state(registered$game_id,
                     players = list(list(id = "alice"), list(id = "bob")),
                     ruleset_version = "1.0.0")
  state <- apply_action(state, list(type = "score",
                                    params = list(player = 0L)))
  print(state$data$scores)
})
```

`ruleatlas-formats` *State, Action, Outcome and Custom Rulepack Formats*

Description

Portable data conventions, error handling and the bounded custom rule language shared by the native R and Python implementations.

Details

R representation. JSON objects are named lists; arrays are unnamed lists, including arrays of scalar numbers. Scalars are length-one character, numeric or logical values. JSON null is R NULL. Use `structure(list(), names = character())` for an empty object and `list()` for an empty array. Do not substitute data frames, factors, NA, NaN or infinite numbers. When reading JSON externally use `jsonlite::fromJSON(path, simplifyVector = FALSE)`. Numeric values are bounded by 2^53 - 1 in absolute value. Integer fields require whole values. Array and player indices in public rule data are zero-based; R list access is one-based. Results identify players by their character IDs.

State envelopes. A state is a named list with `game_id`, `ruleset_id`, `ruleset_version`, `variant`, `players` and `data`, with optional `metadata`. Players are a list of records with distinct nonempty `id` strings. Data depend on the selected profile. Start with `new_state()` and submit setup or play actions through `apply_action()`. Some profiles need an external deal, course, lexicon, measurements or official decisions. These are caller inputs, never guessed. Imported states receive only the checks described by the profile; they do not receive a general historical-reachability proof.

Actions and legality. An action contains `type` and `params`, with optional `actor` and `metadata`. An actor identifies a participant but does not implement authentication. `legal_actions()` returns `mode`, `actions`, `action_types`, `constraints`, `completeness` and `reason`. Use complete action objects in enumerated mode. For typed events, follow the parameter schema and preconditions; enumeration is not promised and a particular submission can still fail. The state is authoritative and may contain hidden cards or secrets. Applications must implement their own private views.

Results. `outcome()` returns `status`, `result_kind`, `winners`, `losers`, `placements`, `reason`, `evidence_required` and `rule_id`. Status can be ongoing, terminal, indeterminate or awaiting_adjudication. An official observation required by a profile is not inferred from physical movement. Read `reason` and `evidence_required` before treating a result as final. Rankings and ties follow the selected profile. No probabilities or strategies are calculated.

Validation. `validate_state()` and `validate_rules()` return `valid`, `scope` and `issues`. Every issue has `code`, `path`, `message` and `severity`. Other operations may raise a `ruleatlas_error` condition with `code`, `message` and optional `path`. Stable codes include UNKNOWN_GAME, UNKNOWN_RULESET, UNSUPPORTED_CAPABILITY, INVALID_STATE, ILLEGAL_ACTION, MISSING_INFORMATION, RULESET_CONFLICT, INVALID_RULEPACK, RESOURCE_LIMIT and INTERNAL_ERROR. Catch the condition class instead of matching human-readable messages. Rejected actions do not partially modify the state.

Custom DSL version 1.0

A custom rulepack has `schema_version = "1.0"`, a `game_id` starting with `user.`, `name`, `ruleset_id`, `ruleset_version`, `family`, `participants`, `state`, `actions`, `outcomes` and `provenance`. Optional description and capabilities are metadata. Nonempty variants are unsupported. A ruleset ID starts with the game ID followed by a dot. Participants declare integer min and max from 1 to 512. Family is finite_turn, event_scoring, measured_rank, adjudicated or custom. Provenance supplies basis and verification_status strings and source references.

State declares a named object of field schemas and an initial state. Supported field types are integer, number, boolean, string, enum, array and object. Numeric bounds use minimum/maximum, strings minLength/maxLength and arrays minItems/maxItems. Enum uses a values array; array uses items; object uses properties and required. Nullable fields permit null. All top-level state fields are required by the runtime.

Each action declares type, a named params schema, a boolean when expression and an effects array. Expressions are scalar constants, an object with ref such as `list(ref = "state.turn")`, or an op/args object. References address only state or params; numeric path components are zero-based array indices. Allowed operators are eq, ne, lt, lte, gt, gte, not, len, add, sub, mul, mod, in, and, or. Boolean operators require booleans; arithmetic requires numbers. Modulo requires integers and a positive divisor. Expressions are data, not R code.

Effects are set with value, inc with by, append with value, or remove_at with index. Each has a slash-separated path starting at the state data, such as `"/scores/0"`. A path segment `"\{player\}"` can use a nonnegative integer parameter as an array index. Overlapping target paths are rejected. All effect expressions read the old state; results commit atomically after postcondition checks. File paths, shell commands and executable R are not DSL operations.

An outcome declares when, result_kind and a winners array of zero-based player indices or numeric expressions, with optional reason. More than one matching outcome is invalid. No match means ongoing. Terminal states reject actions. Registration is session-local and version-immutable. The same version and same contents can be registered repeatedly; changed contents require a new version.

Rulepacks are bounded to 1 MiB, JSON depth 32, 50,000 value nodes, 512 actions, 512 outcomes, 100 effects per action and 2,000 expression-check nodes. Runtime expression evaluation also uses bounded budgets. Custom legal-action enumeration is limited to 512 parameter combinations; otherwise typed event schemas are returned. Structural validation does not prove fairness or eventual termination.

Bundled schemas and example

The extdata/contracts directory contains the language-neutral JSON schemas for states, actions, results, validation, profile records and rulepacks. The original extdata/examples/first_to_three.json rulepack is a small runnable template. It uses alternating zero-based player indices and a three-point win threshold.

See Also

new_state, legal_actions, outcome, validate_rules, register_rules, ruleatlas-profiles

Examples

```
path <- system.file("extdata", "examples", "first_to_three.json",
                    package = "ruleatlas", mustWork = TRUE)
validate_rules(path)$valid
schema <- system.file("extdata", "contracts", "rulepack.schema.json",
                      package = "ruleatlas", mustWork = TRUE)
file.exists(schema)
tryCatch(get_game("user.not_registered"),
         ruleatlas_error = function(e) e$code)
```

---

ruleatlas-profiles *Index of the 212 Retained Profiles*

Description

Stable profile identifiers for the 151 sports and 61 games included in this distribution. Gaps are intentional; removed profiles are unavailable.

Details

Each identifier selects a declared variant, not every edition of a game. Use `get_game(id)` for its coverage, known limitations, provenance and capabilities. Setup requirements and current actions are exposed by `legal_actions(state)`. No certification of a federation or legal clearance is claimed.

`G001` Orthodox chess

`G002` 9x9 area Go

`G003` Standard shogi with agreed twenty-four-point impasse

`G004` Xiangqi with adjudicated repetition disputes

`G005` Casual Janggi with bikjang and two-pass draw

`G006` English draughts

`G007` Reversi

`G008` Backgammon single game with doubling cube

`G009` Nine Men's Morris

`G011` Freestyle Gomoku
`G012` Hex
`G019` Oware Abapa
`G020` Toguz Kumalak
`G021` Fanorona riatra
`G022` Seven-by-seven Brandubh-style tafl
`G023` Two-player Chinese Checkers
`G035` Dominoes
`G036` Mahjong unscored structural single hand
`G043` Skat pre-agreed suit hand game at eighteen
`G044` Doppelkopf forty-eight-card normal hand without announcements
`G045` Schafkopf pre-agreed suit-solo single hand
`G046` Single-board duplicate contract bridge
`G047` Hearts
`G048` Partnership Spades
`G049` Partnership Euchre
`G050` Canasta four-player partnership match
`G051` Two-player basic Rummy
`G052` Two-player Mau-Mau, thirty-two cards
`G055` Two-player non-transferable Durak
`G056` Three-player President, single deal
`G057` Sixty-Six twenty-four-card match to seven
`G060` Tarock pre-agreed three-player contract
`G061` Trump-only partnership Schieber Jass
`G062` Two-player Briscola
`G063` Two-player Scopa
`G064` Partnership Belote to one thousand
`G065` Two-player cribbage to one hundred twenty-one
`G066` Three-player Thirty-One / Schwimmen
`G068` Heads-up no-limit Texas Hold'em
`G069` Heads-up pot-limit Omaha
`G070` Heads-up fixed-limit Seven Card Stud
`G071` Heads-up no-limit Five Card Draw
`G072` Single-deck blackjack
`G073` Punto Banco Baccarat
`G075` Pai Gow Poker, ace-low-wheel variant
`G076` Single-zero Roulette

`G077` Craps Pass and Dont Pass contract with 3-4-5x odds

`G078` Sic Bo

`G079` 80-number Keno

`G080` 75-ball Line Bingo

`G083` Non-wild Liar's Dice

`G084` Three-player Maexchen

`G085` Shut the Box

`G088` Six-player Vanilla Mafia

`G099` Hangman

`G100` Categories (City, Country, River)

`G101` Sudoku

`G102` Kakuro

`G103` Monochrome Nonogram

`G104` Minesweeper

`G105` Tic-Tac-Toe

`S001` Association Football

`S002` Futsal match with rolling substitutions and accumulated fouls

`S003` American football IFAF 2023 completed-down protocol

`S004` Flag football five-a-side 2023 down protocol

`S005` Rugby union regulation official-event match

`S006` Rugby league regulation official-event match

`S007` Australian football classic eighteen-a-side regulation

`S008` Gaelic football traditional regulation match

`S009` Basketball four-quarter official-event match

`S010` Netball traditional seven-position regulation match

`S011` Korfball 2024 match protocol

`S012` Indoor handball regulation league match

`S013` Volleyball best-of-five with fixed six-player rosters

`S014` Fistball best-of-five, 2019 format

`S015` Baseball completed-play league format

`S016` Fast-pitch softball completed-play league format

`S017` Cricket twenty-over regulation league match

`S018` Field hockey four-quarter regulation match

`S019` Floorball regulation league match

`S020` Lacrosse Sixes 2021 official-event match

`S021` Ultimate seven-a-side uncapped game to fifteen

`S022` Dodgeball six-player untimed elimination set

`S023` Sepak takraw ISTAF 2016 single regu
`S024` Kabaddi IKF 2.1.0 mens regulation league
`S025` Kho Kho traditional nine-minute league protocol
`S026` Gaelic hurling traditional regulation match
`S027` Seven-a-side Tchoukball
`S028` Roundnet doubles single game to twenty-one
`S029` Tennis
`S030` Table Tennis
`S031` Badminton
`S032` Squash
`S033` Traditional singles racquetball, side-out scoring
`S034` Singles side-out pickleball
`S035` Doubles padel, advantage games and tiebreak sets
`S036` Singles beach tennis, no-ad best of three sets
`S037` Singles Crossminton
`S038` Basque hand-pelota singles on a 36-metre fronton
`S039` Real tennis singles with ordered chases
`S040` Indoor Tamburello international thirteen-game match
`S043` Golf scratch eighteen-hole stroke-play round
`S044` Recreational minigolf, eighteen holes
`S045` Disc golf eighteen-hole stroke-play group
`S046` Darts
`S047` Traditional nine-ball pool, single rack
`S048` Snooker fifteen-red official-stroke frame
`S049` Three-cushion carom, forty points with equal innings
`S050` Nine-pin individual 120-delivery total-pin contest
`S051` Ten-pin Bowling
`S052` Singles petanque to thirteen
`S053` Boccia BC2 individual four-end match
`S054` Singles Cornhole
`S055` Table shuffleboard singles to twenty-one
`S056` Kubb traditional six-baton game
`S058` Golf Croquet thirteen-point level singles
`S059` Recurve individual set match
`S060` Ten-metre air pistol sixty-shot qualification
`S061` Trap twenty-five-target round
`S062` Professional boxing, twelve three-minute rounds

`S064` WMC 2020 Muay Thai bout-state variant

`S065` Taekwondo cumulative scoring, 2019 format

`S066` Classic adult kumite, three minutes without first-score advantage

`S067` Adult judo, four minutes, two waza-ari and no yuko

`S068` Adult freestyle wrestling without video challenges

`S069` Adult black-belt BJJ, ten-minute points and advantages

`S070` Sport Sambo adult bout score sheet

`S071` Sumo Bout

`S072` Professional MMA, three five-minute rounds

`S073` Epee five-touch individual pool bout

`S074` Kendo individual sanbon-shobu with one encho and hantei

`S075` Adult Sanda with five sideline judges

`S076` Savate assaut with double-category scorecards

`S077` Armwrestling right-arm best-of-three match

`S078` Ice hockey regulation league protocol 2023/24

`S079` Four-player team curling with five-rock protection

`S080` Icestock six-end team match, 2022 stock scoring

`S081` Long-track speed skating five-hundred-metre pair

`S082` Figure skating men’s short-program final

`S083` Alpine slalom two-run final

`S084` Cross-country skiing interval-start freestyle race

`S085` Biathlon classic individual with four shooting bouts

`S086` Ski jumping single-round K90 fixed-gate competition

`S087` Nordic combined K90 single-jump Gundersen pursuit

`S088` Snowboard giant slalom two-run time trial

`S089` Freestyle skiing aerials single-jump final

`S090` Bobsleigh four-heat final

`S091` Singles luge four-run final

`S092` Skeleton four-heat final

`S093` Ski mountaineering individual vertical race

`S094` 50-metre Freestyle Swimming Final

`S095` Three-metre springboard six-dive final

`S096` Artistic swimming duet free-routine final

`S097` Water polo FINA 2019-2021 regulation league

`S098` Surfing four-surfer priority final heat

`S099` Windsurfing fleet series

`S100` Kiteboarding fleet series

`S101` Sailing five-race low-point series
`S102` Rowing single-final crew classification
`S103` Canoe sprint single-final boat classification
`S104` Canoe slalom twenty-gate single-run final
`S105` Canoe polo ICF 2023 official match
`S106` Dragon boat 500-metre final
`S107` Waterski slalom classification round
`S108` Underwater hockey CMAS 2025 match
`S109` Underwater rugby 2024 regulation league
`S110` 100-metre Sprint Final
`S111` 110-metre Hurdles Final
`S112` 4x100-metre Relay Final
`S113` Race walking final, three-red-card disqualification
`S114` High Jump Qualification
`S115` Pole Vault Qualification
`S116` Long Jump Qualification
`S117` Triple Jump Qualification
`S118` Shot Put Qualification
`S119` Discus Throw Qualification
`S120` Javelin Throw Qualification
`S121` Hammer Throw Qualification
`S122` Athletics men's decathlon official-result competition
`S123` Road cycling individual time trial with course controls
`S124` Track cycling four-kilometre individual pursuit qualification
`S125` Mountainbike downhill eight-rider final
`S126` BMX individual time-trial seeding run
`S127` Individual triathlon with ordered transitions
`S128` Point-to-Point Orienteering
`S129` Single-stage trail race with checkpoint cutoffs
`S130` Artistic gymnastics floor final with five execution judges
`S131` Rhythmic gymnastics individual hoop final
`S132` Trampoline ten-element five-judge E-plus-D final
`S133` Acrobatic gymnastics mixed-pair combined-routine final
`S134` Aerobic gymnastics individual routine final
`S135` Parkour speed final with mandatory course gates
`S136` Bouldering Qualification Round
`S137` Weightlifting two-lift total

`S138` Powerlifting three-lift total

`S139` Bodybuilding single-class comparison final

`S140` Breaking individual best-of-three nine-judge battle

`S141` DanceSport single-dance skating final

`S142` Cheerleading all-girl routine final

`S143` Rope skipping thirty-second single-rope speed trial

`S144` Circuit motorsport official-lap local sprint

`S145` Three-stage single-leg rally with time controls

`S146` Karting official-lap local sprint

`S147` Motorcycle circuit sprint local classification

`S148` Motocross timed plus two laps local classification

`S149` Drone racing individual multi-lap time trial

`S150` Show jumping twelve-single-fence timed round

`S151` Dressage five-judge prescribed test

`S152` Eventing three-phase individual classification

`S153` Flat horse race official classification

`S154` Polo HPA 2022 outdoor match

`S155` Agility fixed twenty-obstacle course

See Also

list_games, get_game

Examples

```
get_game("S001")$name
get_game("G001")$capabilities
```

---

unregister_rules *Remove a Session-Local Custom Ruleset*

Description

Remove one matching custom ruleset from the session registry. Built-in profiles cannot be unregistered.

Usage

```
unregister_rules(game_id, ruleset_version = NULL, ruleset_id = NULL)
```

Arguments

game_id	One case-sensitive character string identifying a retained built-in profile or a registered custom game in the user. namespace. See ruleatlas-profiles.
ruleset_version	NULL or one character string selecting an exact version, for example "1.0.0". NULL succeeds only when selection is unambiguous. Pin versions when persisting states.
ruleset_id	NULL or one character string selecting a ruleset belonging to game_id. Unknown selections fail instead of falling back to another ruleset.

Details

This function does not delete a rulepack file. States referring to a removed custom version cannot be processed until that version is registered again. Pin both identifiers where necessary to remove only the intended version.

Value

TRUE when one custom ruleset was removed, or FALSE if the game or selected version was not registered. An ambiguous selection or attempted built-in removal raises RULESET_CONFLICT.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
path <- system.file("extdata", "examples", "first_to_three.json",
                    package = "ruleatlas", mustWork = TRUE)
local({
  registered <- register_rules(path)
  on.exit(unregister_rules(registered$game_id,
                           ruleset_version = "1.0.0"))
  print(unregister_rules(registered$game_id,
                         ruleset_version = "1.0.0"))
  print(unregister_rules(registered$game_id,
                         ruleset_version = "1.0.0"))
})
```

validate_rules *Validate a Local Custom Rulepack*

Description

Check a custom rulepack against the bundled JSON schema and bounded interpreter restrictions without registering it.

Usage

```
validate_rules(rules_or_path)
```

Arguments

rules_or_path	A named list describing a custom rulepack, or one character string naming a regular local JSON file. Symbolic links, directories, URLs and files larger than 1 MiB are rejected. Use jsonlite::fromJSON(..., simplifyVector = FALSE) when parsing externally.

Details

Validation checks supported expression operators, references, parameter and state types, effects and resource bounds. It is not a proof of termination, fairness or reachability. No rulepack code is executed and no network resources are retrieved. See ruleatlas-formats for the DSL and bundled schema.

Value

A named validation result with valid, scope = "structural" and issues. Each issue contains code, path, message and severity. Invalid files or rulepacks return valid = FALSE instead of being registered.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
path <- system.file("extdata", "examples", "first_to_three.json",
                    package = "ruleatlas", mustWork = TRUE)
validate_rules(path)$valid
bad <- jsonlite::fromJSON(path, simplifyVector = FALSE)
bad$game_id <- "invalid_namespace"
validate_rules(bad)$valid
```

`validate_state` *Check a Rule State*

Description

Check the state envelope, typed fields and the selected profile's consistency and outcome constraints without changing the input.

Usage

```
validate_state(state)
```

Arguments

`state`	A named list containing game_id, ruleset_id, ruleset_version, variant, players and data, with optional metadata. Use new_state to create the envelope. See ruleatlas-formats.

Details

Errors encountered during state validation are converted to validation results. This function does not establish facts omitted from the state, infer hidden observations or reconstruct an arbitrary game history. Other operations require a valid state and may raise an error condition.

Value

A named list with valid (logical), scope (character) and issues (a list). Each issue contains code, path, message and severity. A successful result has valid = TRUE and an empty issues list. A failed result has valid = FALSE and a diagnostic issue.

Errors

Except for validation results as documented above, failures use a `ruleatlas_error` condition with `message`, `code` and optional `path`. Catch this class with `tryCatch`; see ruleatlas-formats for stable error codes.

See Also

ruleatlas-package, ruleatlas-formats, ruleatlas-profiles

Examples

```
state <- new_state("G105", players = list(list(id = "alice"),
                                          list(id = "bob")),
                   ruleset_version = "1.0.0")
validate_state(state)$valid
bad <- state
bad$data$board[[1]] <- 99L
validate_state(bad)$valid
validate_state(state)$valid
```

Index

∗ **package**
ruleatlas-package, 2

apply_action, *2*, 3

get_capabilities, *2*, 4
get_game, *2*, 5, *20*

legal_actions, *2, 3*, 6, *14*
list_games, *2*, 7, *20*

new_state, *2, 3*, *6*, 8, *10*, *14*, *23*

outcome, *2*, 9, *14*

register_rules, *2*, 11, *14*
ruleatlas *(*ruleatlas-package*)*, 2
ruleatlas-formats, *2, 3*, *5–11*, 12, *21–23*
ruleatlas-package, 2, *3*, *5–11*, *21–23*
ruleatlas-profiles, *2–11*, 14, *14*, *21–23*

unregister_rules, *2*, 20

validate_rules, *2*, *14*, 22
validate_state, *2*, 23