

Package: orchestraManifest (via r-universe)

October 15, 2026

Title The Crop Analytics Orchestra Manifest and Refusal Contract

Version 0.2.0.9000

Description A versioned, integrity-hashed and provenance-tracked 'S7' contract for the results that crop-analytics components exchange. A manifest records what was inferred, which component emitted it, the method and seed behind it, the load-bearing payload, and the lineage of any upstream manifest it was derived from; a 'SHA-256' hash over version-stable serialised bytes lets any consumer verify that the payload has not been modified since emission. Consumers declare the manifest versions they accept and refuse anything else loudly rather than coercing silently across a schema boundary. Companion predicates recognise a structured refusal or abstention from any component regardless of its object system, so a gate that should stop on a declined result cannot sail through it, and grounding tokens propagate worst-case so that one unverified input taints a downstream verdict. Adapters lift a 'PESTO' ensemble manifest and a 'proxymix' Gaussian mixture fit into the contract.

License MIT + file LICENSE

URL <https://github.com/max578/orchestraManifest>

BugReports <https://github.com/max578/orchestraManifest/issues>

Encoding UTF-8

Language en-AU

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Depends R (>= 4.3.0)

Imports digest, S7, utils

Suggests ggplot2, knitr, openssl, PESTO, proxymix, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Remotes max578/PESTO, max578/proxymix
Config/roxygen2/version 8.1.0
Repository https://max578.r-universe.dev
Date/Publication 2026-10-15 22:55:35 UTC
RemoteUrl https://github.com/max578/orchestraManifest
RemoteRef main
RemoteSha 711950c5b8417bd0424e45cd359e697f576f6bc2

Contents

as_orchestra_manifest	2
as_orchestra_refusal	3
assert_obs_schema_correspondence	4
combine_grounding	6
consume_manifest	7
GROUNDING_GROUNDED	8
grounding_token	8
GROUNDING_UNVERIFIED	9
inferential_targets	10
is_orchestra_decline	11
is_orchestra_manifest	12
is_orchestra_refusal	13
manifest_data_hash	14
manifest_from_gmm	15
manifest_from_pesto	16
manifest_lineage	17
manifest_summary	18
manifest_version	20
orchestra_manifest	21
ORCHESTRA_REFUSAL_CLASS	23
verify_manifest	24
Index	26

as_orchestra_manifest	<i>Adapt a component's result into an orchestra manifest</i>
-----------------------	--

Description

The emitter generic. A component that produces its own result type gains contract membership by having a method registered here, which lifts that type into an [orchestra_manifest](#) without the component itself changing.

Usage

as_orchestra_manifest(x, ...)

Arguments

x	The component result to adapt.
...	Arguments passed to the method. See manifest_from_pesto() and manifest_from_gmm() for the two that ship here.

Details

Two methods ship with this package, for a PESTO ensemble manifest and a proxymix mixture fit, and each is registered only when its component is installed. The equivalent plain functions are [manifest_from_pesto\(\)](#) and [manifest_from_gmm\(\)](#).

Value

An [orchestra_manifest](#).

Author(s)

Max Moldovan

See Also

[manifest_from_pesto\(\)](#), [manifest_from_gmm\(\)](#).

Examples

```
# Dispatch requires an installed component that carries a method.
if (requireNamespace("proxymix", quietly = TRUE)) {
  print(class(as_orchestra_manifest))
}
```

as_orchestra_refusal *Stamp the shared refusal marker onto an object*

Description

Prepends [ORCHESTRA_REFUSAL_CLASS](#) to an object's class vector so that [is_orchestra_refusal\(\)](#) recognises it by the explicit marker rather than by the naming convention. Useful for seam-side code that mints a refusal of its own and has no component name to follow the convention with.

Usage

```
as_orchestra_refusal(x)
```

Arguments

x	The refusal object to mark.
---	-----------------------------

Details

The operation is idempotent: stamping an object that already carries the marker returns it unchanged, so a refusal can pass through several layers without accumulating duplicate classes.

Value

x, with `ORCHESTRA_REFUSAL_CLASS` prepended to its class vector if it was not already present.

Author(s)

Max Moldovan

See Also

`is_orchestra_refusal()`.

Examples

```
r <- as_orchestra_refusal(structure(list(why = "no data"), class = "list"))
class(r)
identical(as_orchestra_refusal(r), r)
```

assert_obs_schema_correspondence

Assert that two manifests describe the same quantities the same way

Description

Compares the observation schemas of two manifests and refuses the pairing when a column they share by name disagrees on its quantity or its unit.

Usage

```
assert_obs_schema_correspondence(m1, m2, what = c("outputs", "params"))
```

Arguments

m1	The first manifest.
m2	The second manifest.
what	Which schema sides to compare. Any subset of "outputs" and "params"; both by default.

Details

The check is opt-in by presence. When either manifest lacks an `obs_schema` it is a graceful no-op, so the guard activates only once both emitters populate the field and no existing pipeline breaks on the day the guard lands. Columns present in only one schema are ignored; only the shared names are compared, and only on the fields that carry meaning for that side – quantity and unit for outputs, unit for params.

Refusing rather than warning is the deliberate choice. A unit mismatch between two manifests is not a degraded comparison, it is an incommensurable one, and any number produced from it is wrong by a factor nobody downstream can recover.

Value

NULL, invisibly, when the schemas agree or when either side carries no schema. Errors on a disagreement.

Author(s)

Max Moldovan

See Also

[orchestra_manifest](#) for the `obs_schema` property.

Examples

```
schema_of <- function(unit) {
  list(outputs = data.frame(name = c("o1", "o2"),
                            quantity = c("grain_yield", "grain_yield"),
                            unit = c(unit, "t/ha")))
}
build <- function(unit) {
  orchestra_manifest(
    emitter_package = "example",
    emitter_version = "0.1.0",
    inferential_target = "predictions",
    run_id = paste0("example-", unit),
    method = "example:demo",
    obs_schema = schema_of(unit),
    timestamp = Sys.time(),
    data_hash = manifest_data_hash(data.frame(), NULL, NULL, NULL,
                                   NA_integer_)
  )
}
assert_obs_schema_correspondence(build("t/ha"), build("t/ha"))
try(assert_obs_schema_correspondence(build("t/ha"), build("kg/ha")))
```

combine_grounding	<i>Combine grounding tokens worst-case</i>
-------------------	--

Description

Reduces the tokens of several input facts to the token of a verdict that rests on all of them. The combination is worst-case: the verdict is grounded only when every input is grounded.

Usage

```
combine_grounding(tokens)
```

Arguments

tokens	A character vector of grounding tokens, each of which should be GROUNDING_GROUNDED or GROUNDING_UNVERIFIED . The token emitted up to 0.1.0 and any unrecognised string are treated as not grounded, which is the safe reading. A zero-length vector is valid and means "the verdict rests on nothing checked".
--------	--

Details

Worst-case combination is what stops a grounding label from being laundered through a pipeline. Any other rule – a majority, a mean, an optimistic "at least one" – would let a verdict inherit confidence from the checked inputs while quietly resting on an unchecked one. As with [grounding_token](#), an empty set of inputs is unverified rather than vacuously grounded.

Value

A length-1 character vector: [GROUNDING_GROUNDED](#) when every token is grounded, [GROUNDING_UNVERIFIED](#) otherwise.

Author(s)

Max Moldovan

See Also

[grounding_token](#) for minting a token from verification dates.

Examples

```
combine_grounding(c(GROUNDING_GROUNDED, GROUNDING_GROUNDED))
combine_grounding(c(GROUNDING_GROUNDED, GROUNDING_UNVERIFIED))
combine_grounding(character(0))
```

consume_manifest	<i>Consume a manifest, or refuse it loudly</i>
------------------	--

Description

The consumer-side guard. Checks that the object is a manifest, that its schema version is one the caller declared it accepts, and that its payload still verifies, and errors informatively on any of the three rather than proceeding.

Usage

```
consume_manifest(m, accept = manifest_version())
```

Arguments

m	The manifest to consume.
accept	A character vector of accepted contract versions. Defaults to manifest_version() , the version this package implements. Widen it deliberately, per consumer, when a consumer has been shown to read an older schema correctly.

Details

The version check is the point. Coercing silently across a schema boundary is how a pipeline ends up reading a slot that means something else than it did when the consumer was written, and the failure surfaces far downstream as a wrong number rather than an error. Declaring the accepted set makes the boundary explicit and moves the failure to the seam where it belongs.

Value

m, invisibly, when all three checks pass. Errors otherwise.

Author(s)

Max Moldovan

See Also

[verify_manifest\(\)](#) for the non-erroring integrity check.

Examples

```
params <- data.frame(a = c(0.1, 0.2))
m <- orchestra_manifest(
  emitter_package = "example",
  emitter_version = "0.1.0",
  inferential_target = "parameters",
  run_id = "example-001",
  method = "example:demo",
```

```

    seed = 1L,
    params = params,
    timestamp = Sys.time(),
    data_hash = manifest_data_hash(params, NULL, NULL, NULL, 1L)
)
identical(consume_manifest(m), m)

# A consumer that only reads a future schema refuses this one.
try(consume_manifest(m, accept = "9.9.9"))

```

GROUNDING_GROUNDED	<i>The token marking a fact as grounded</i>
--------------------	---

Description

The canonical string stamped on a fact whose external referent has been checked against a source that did not also produce the claim. A fact is grounded if and only if it carries a non-missing verification date.

Usage

```
GROUNDING_GROUNDED
```

Format

A length-1 character vector.

See Also

[GROUNDING_UNVERIFIED](#) for its complement, [grounding_token](#) for the function that mints either.

grounding_token	<i>Mint a grounding token for a set of facts</i>
-----------------	--

Description

Reduces a vector of verification dates to a single grounding token. A fact is grounded only when its verification date is present, so an empty set and a set containing any missing date both yield [GROUNDING_UNVERIFIED](#).

Usage

```
grounding_token(verified_on)
```

Arguments

`verified_on` A vector of verification dates, one per fact. Any type for which `is.na()` is meaningful is accepted; only missingness is inspected, never the date values themselves. A zero-length vector is valid and means "no facts were checked".

Details

The asymmetry is deliberate and is the whole point of the vocabulary. An empty evidence set is not neutral – it is the absence of evidence – and a partially checked set is not partially trustworthy, because the consumer has no way to tell which element it may lean on. Both collapse to the unverified token so that a downstream gate errs towards abstention.

Value

A length-1 character vector: `GROUNDING_GROUNDED` when every element is present, `GROUNDING_UNVERIFIED` otherwise.

Author(s)

Max Moldovan

See Also

[combine_grounding](#) for combining tokens rather than dates.

Examples

```
grounding_token(as.Date(c("2026-07-05", "2026-07-06")))
grounding_token(as.Date(c("2026-07-05", NA)))
grounding_token(as.Date(character(0)))
```

`GROUNDING_UNVERIFIED` *The token marking a fact as unverified*

Description

The canonical string stamped on a fact whose external referent has not been checked, or whose evidence set contains at least one unchecked element. It is the safe default: a fact is unverified until something grounds it, never the other way round.

Usage

`GROUNDING_UNVERIFIED`

Format

A length-1 character vector.

Details

The value is "ungrounded". [combine_grounding\(\)](#) also reads the token that releases up to 0.1.0 emitted as not grounded.

See Also

[GROUNDING_GROUNDED](#) for its complement, [combine_grounding](#) for how one unverified input taints a set.

inferential_targets	<i>The permitted inferential targets</i>
---------------------	--

Description

Reports the closed vocabulary a manifest's `inferential_target` property must draw from. The vocabulary tells a consumer what kind of thing the manifest holds before it inspects any payload, so a consumer that only understands parameter posteriors can decline a decision manifest without guessing from the shape of params.

Usage

```
inferential_targets()
```

Details

The set grows additively. A new target is a minor, non-breaking change to the contract: manifests already in circulation stay valid, and consumers that do not recognise the new value simply refuse it.

Value

A character vector of the permitted `inferential_target` values.

Author(s)

Max Moldovan

See Also

[orchestra_manifest](#) for the property this vocabulary constrains.

Examples

```
inferential_targets()
```

is_orchestra_decline	<i>Did any orchestra component decline to stand behind a result?</i>
----------------------	--

Description

The gate-facing union. Returns TRUE for a refusal, where a component will not produce the result at all, and also for an abstention, where a component produces a typed "assumptions not met" object instead of a number.

Usage

```
is_orchestra_decline(x)
```

Arguments

x	Any object.
---	-------------

Details

Gate logic almost always wants this rather than [is_orchestra_refusal\(\)](#), because the two outcomes mean the same thing to the gate: no usable result from this lens. Enumerating component classes at the gate is the alternative, and it silently stops recognising declines the moment a new component joins.

Value

A length-1 logical.

Author(s)

Max Moldovan

See Also

[is_orchestra_refusal\(\)](#) for the narrower refusal-only test.

Examples

```
is_orchestra_decline(structure(list(), class = "example_refusal"))
is_orchestra_decline(structure(list(), class = "example_abstention"))
is_orchestra_decline(42)
```

is_orchestra_manifest *Is this object an orchestra manifest?*

Description

Tests contract identity tolerantly, accepting both the package-qualified class this package defines and the bare class minted by emitters that implement the contract without depending on this package.

Usage

```
is_orchestra_manifest(x)
```

Arguments

x	Any object.
---	-------------

Details

A strict `S7::S7_inherits()` check against the class object defined here would reject every manifest built anywhere else, including all of those already in circulation, because `S7` records the defining package in the class identity. Since the contract is deliberately implementable without a dependency on this package, identity has to be tested on the class name, which is the part the contract actually fixes. The object must still be an `S7` object, so an ordinary list carrying a lookalike class attribute does not pass.

Value

A length-1 logical, `TRUE` when `x` is an `S7` object whose class vector contains an `orchestra_manifest` entry in either the bare or the package-qualified form.

Author(s)

Max Moldovan

See Also

[orchestra_manifest](#) for the class, and its `Class` identity across homes section for why the tolerance is needed.

Examples

```
m <- orchestra_manifest(
  emitter_package = "example",
  emitter_version = "0.1.0",
  inferential_target = "parameters",
  run_id = "example-001",
  method = "example:demo",
  timestamp = Sys.time(),
  data_hash = "sha256:example"
```

```
)  
is_orchestra_manifest(m)  
is_orchestra_manifest(list(a = 1))
```

is_orchestra_refusal *Is this a refusal from any orchestra component?*

Description

The producer-agnostic replacement for a bare `is_refusal()` at a seam between components. Returns TRUE for the explicit [ORCHESTRA_REFUSAL_CLASS](#) marker, or for any object whose class follows the `<component>_refusal` naming convention.

Usage

```
is_orchestra_refusal(x)
```

Arguments

x Any object.

Details

Testing the class vector rather than dispatching on a known type is what makes this work across object systems and without attaching, or even depending on, the producing component's namespace. An S3 refusal has class "example_refusal" and an S7 one `c("example::example_refusal", "S7_object")`; both are recognised.

Value

A length-1 logical.

Author(s)

Max Moldovan

See Also

[is_orchestra_decline\(\)](#) for the wider gate-facing union that also covers abstentions.

Examples

```
is_orchestra_refusal(structure(list(), class = "example_refusal"))  
is_orchestra_refusal(structure(list(), class = "example_abstention"))  
is_orchestra_refusal(42)
```

manifest_data_hash	<i>Hash a manifest's load-bearing payload</i>
--------------------	---

Description

Computes the digest that goes in a manifest's `data_hash` property, and that `verify_manifest()` recomputes and compares against. An emitter calls this once, on the payload it is about to seal into a manifest.

Usage

```
manifest_data_hash(params, outputs, weights, obs_target, seed, summary = NULL)
```

Arguments

<code>params</code>	The parameter draws, typically a data frame. Pass the value destined for the manifest's <code>params</code> property, including its default empty data frame when there are no draws.
<code>outputs</code>	The model-predicted outputs, or <code>NULL</code> when the manifest holds no ensemble payload.
<code>weights</code>	The observation weights, or <code>NULL</code> .
<code>obs_target</code>	The observed targets, or <code>NULL</code> .
<code>seed</code>	The integer RNG seed behind the result, or <code>NA_integer_</code> .
<code>summary</code>	The typed verdict from <code>manifest_summary()</code> , or <code>NULL</code> .

Details

The hash covers the load-bearing data slots only. The schema version, the emitter identity and the timestamp are metadata and are deliberately outside it, so re-tagging a manifest does not invalidate a payload that has not changed, while any edit to the payload is detected.

Argument order is part of the contract, not an implementation detail. The six slots are hashed as a single list in exactly this order, over the version-stable byte representation described in the class documentation. A component that emits manifests without depending on this package must reproduce that order and that representation exactly, or its manifests will not verify here.

Value

A length-1 character vector of the form "sha256: <hex>", ready to be stored as a manifest's `data_hash`.

Author(s)

Max Moldovan

See Also

`verify_manifest()`, which recomputes this and compares.

Examples

```
params <- data.frame(a = c(0.1, 0.2), b = c(1.5, 1.4))
manifest_data_hash(params, NULL, NULL, NULL, 1L)
```

manifest_from_gmm	<i>Adapt a proxymix mixture fit</i>
-------------------	-------------------------------------

Description

Lifts a fitted Gaussian-mixture proxy into the contract. In orchestra terms such a fit is an approximation to a posterior over parameters, so it emits a manifest with `inferential_target = "parameters"`, the mixture as the payload, and the manifold it approximated recorded as lineage.

Usage

```
manifest_from_gmm(
  x,
  ...,
  seed = NA_integer_,
  consumed = list(),
  run_id = NULL,
  emitter_version = NULL
)
```

Arguments

<code>x</code>	A <code>proxymix::gmm_fit</code> .
<code>...</code>	Unused, present for method compatibility.
<code>seed</code>	The integer RNG seed behind the fit, or <code>NA_integer_</code> .
<code>consumed</code>	A list of upstream manifests this fit approximated. A non-empty list marks the emitted manifest as derived.
<code>run_id</code>	A run identifier. Defaults to "proxymix-" followed by the first twelve hexadecimal digits of a digest of the payload, which makes it stable under re-emission of the same fit.
<code>emitter_version</code>	The emitting component's version. Defaults to the installed proxymix version.

Details

Passing `consumed` is what marks the result derived, and the class validator then requires the lineage to be non-empty. That is the rule that stops an approximation from quietly losing the provenance of the thing it approximated – a proxy of a calibration ensemble must be traceable back to that ensemble, or a reader has no way to tell an approximation from a fit.

Value

An [orchestra_manifest](#) with `emitter_package` set to "proxymix".

Author(s)

Max Moldovan

See Also

[as_orchestra_manifest\(\)](#) for the generic this mirrors, [manifest_lineage\(\)](#) for the handle recorded in `consumed_manifests`.

Examples

```
set.seed(1)
draws <- matrix(stats::rnorm(200), 100, 2, dimnames = list(NULL,
                                                             c("a", "b")))
fit <- proxymix::fit_proxymix(proxymix::gmm_target_from_samples(draws),
                             N = 2L, regime = "sample", max_iter = 50L)
m <- manifest_from_gmm(fit, seed = 1L)
verify_manifest(m)$ok
```

manifest_from_pesto	<i>Adapt a PESTO ensemble manifest</i>
---------------------	--

Description

Wraps a `pesto_ensemble_manifest` into the generic contract. The wrap is deliberate: PESTO is untouched, its own class and its own hash scheme stay exactly as they are, and the wrapper re-hashes the same payload under the contract's scheme so the result is self-verifying here.

Usage

```
manifest_from_pesto(x, ..., inferential_target = "parameters")
```

Arguments

<code>x</code>	A <code>PESTO::pesto_ensemble_manifest</code> .
<code>...</code>	Unused, present for method compatibility.
<code>inferential_target</code>	What the wrapped manifest is about, one of inferential_targets() . Defaults to "parameters", which is what a calibration ensemble carries.

Details

PEST0's original digest is preserved in `metadata$pesto_data_hash`, which makes a cross-scheme audit possible: the two hashes cover the same slots by different rules, so a payload that verifies under one and not the other localises the disagreement to the hashing, not to the data.

The observation schema is carried through when the source manifest has one, guarded so that a manifest emitted before the schema slot existed still adapts cleanly.

Value

An `orchestra_manifest` with `emitter_package` set to "PEST0".

Author(s)

Max Moldovan

See Also

`as_orchestra_manifest()` for the generic this mirrors.

Examples

```
set.seed(1)
design <- matrix(stats::rnorm(6), 3, 2)
prior <- matrix(stats::rnorm(60), 30, 2, dimnames = list(NULL, c("a", "b")))
obs <- as.numeric(design %*% c(0.5, -1))
names(obs) <- paste0("o", 1:3)
fit <- PEST0::pesto_ies_callback(function(t) t %*% t(design), prior, obs,
                                0.05, noptmax = 2, verbose = FALSE)
m <- manifest_from_pesto(PEST0::as_manifest(fit, seed = 1L))
verify_manifest(m)$ok
```

manifest_lineage

Take a provenance handle from an upstream manifest

Description

Reduces a manifest to the four fields a downstream manifest records about it in `consumed_manifests`.

Usage

```
manifest_lineage(cm)
```

Arguments

`cm` The upstream manifest that was consumed.

Details

Recording the upstream data_hash is what makes the lineage load-bearing rather than decorative. A run identifier alone says only which run was consumed; the hash says which payload that run had when it was consumed, so a later re-emission that changed the numbers cannot be passed off as the same input.

The check on the argument is deliberately the tolerant `is_orchestra_manifest()`, not a strict class test, because the upstream result routinely comes from a component that implements the contract without depending on this package.

Value

A named list with run_id, data_hash, emitter_package and emitter_version.

Author(s)

Max Moldovan

See Also

[orchestra_manifest](#) for the consumed_manifests property this fills, and the lineage rule its validator enforces.

Examples

```
upstream <- orchestra_manifest(
  emitter_package = "example",
  emitter_version = "0.1.0",
  inferential_target = "parameters",
  run_id = "example-001",
  method = "example:demo",
  timestamp = Sys.time(),
  data_hash = manifest_data_hash(data.frame(), NULL, NULL, NULL,
                                NA_integer_)
)
manifest_lineage(upstream)
```

manifest_summary

A manifest's headline verdict

Description

Builds the typed, classed list that goes in a manifest's summary property: the headline label of a non-ensemble inference, whether the emitter abstained, why it abstained, and any key metrics behind the call.

Usage

```
manifest_summary(  
  headline,  
  abstained = FALSE,  
  metrics = list(),  
  abstain_reason = NA_character_  
)
```

Arguments

headline	A single non-empty string stating the conclusion, for example "effect detected" or "insufficient evidence".
abstained	A single logical. TRUE when the emitter declined to stand behind a result. Coerced with <code>isTRUE()</code> on the way in, so a missing value is recorded as FALSE.
metrics	A named list of the key numbers behind the headline, for example an effect size, a p-value, or an effective sample size. Keep it small; it is a summary, not a results table.
abstain_reason	A single string explaining an abstention, or NA_character_ when the emitter did not abstain.

Details

The slot exists because a verdict and an ensemble are different shapes of result. A causal test or a decision produces one labelled conclusion, not a matrix of draws, and the alternative – smuggling it into params as a single-row table – leaves consumers unable to distinguish a verdict from a degenerate posterior. Because summary is inside the integrity hash, a tampered verdict is detected exactly like a tampered payload.

An abstention is a first-class outcome here, not a failure. A component that will not stand behind a number reports `abstained = TRUE` with a reason, and a downstream gate treats that as "no usable result from this lens" rather than reading a number that was never meant to be trusted.

Value

A list of class `manifest_summary` with elements `headline`, `abstained`, `abstain_reason` and `metrics`.

Author(s)

Max Moldovan

See Also

[orchestra_manifest](#) for the property this fills.

Examples

```
manifest_summary("effect detected", metrics = list(estimate = 0.42))

manifest_summary(
  "insufficient evidence",
  abstained = TRUE,
  abstain_reason = "effective sample size below the floor"
)
```

manifest_version	<i>The contract version this package emits and accepts</i>
------------------	--

Description

Reports the semantic version of the manifest contract implemented here. Emitters stamp it onto every manifest they build, and [consume_manifest](#) uses it as the default accepted-version set.

Usage

```
manifest_version()
```

Details

The version is a deliberate constant rather than the package version. The contract and its reference implementation move at different speeds: a patch release of this package must not look to a consumer like a new schema, and a schema change must be visible even when the package version happens not to have moved.

Value

A length-1 character vector, the contract's semantic version.

Author(s)

Max Moldovan

See Also

[consume_manifest](#) for the guard that uses it.

Examples

```
manifest_version()
```

orchestra_manifest *The orchestra manifest*

Description

A versioned, integrity-hashed, provenance-tracked record of one inference. It is the single result type components exchange: an emitter fills it in once, and every downstream consumer reads the same properties in the same places whatever produced them.

Usage

```
orchestra_manifest(
  manifest_version = "2.0.0-draft",
  emitter_package = character(0),
  emitter_version = character(0),
  inferential_target = character(0),
  run_id = character(0),
  method = character(0),
  seed = NA_integer_,
  params = data.frame(),
  outputs = NULL,
  weights = NULL,
  obs_target = NULL,
  obs_schema = NULL,
  summary = NULL,
  consumed_manifests = list(),
  metadata = list(),
  timestamp = .POSIXct(double(), tz = ""),
  data_hash = character(0)
)
```

Arguments

manifest_version	The contract version this object conforms to. Defaults to manifest_version() . A consumer refuses a version it does not accept rather than coercing across a schema boundary.
emitter_package	The name of the component that emitted this manifest, as a single string.
emitter_version	The version of the emitting component, as a single string.
inferential_target	What the manifest is about. Must be one of inferential_targets() .
run_id	A single non-empty string identifying this run. Emitters commonly derive it from a truncated payload digest so that it is stable under re-emission of the same result.

method	A single string naming the method that produced the result, conventionally qualified by its component, as in "PEST0:ies_callback".
seed	The integer RNG seed behind the result, or NA_integer_ when the result is deterministic or the seed is unknown.
params	A data frame of parameter draws, one row per realisation. Defaults to an empty data frame, which suits a verdict manifest that carries its content in summary.
outputs	Model-predicted outputs at each parameter draw, or NULL when the manifest carries no ensemble payload.
weights	Observation weights, or NULL.
obs_target	The observed targets the model was compared against, or NULL.
obs_schema	A semantic descriptor of the observation columns, giving each one a quantity and a unit, or NULL. Populating it activates the correspondence guard in assert_obs_schema_correspondence() .
summary	A manifest_summary() holding the headline verdict, or NULL for an ensemble manifest that has no verdict to state.
consumed_manifests	A list of provenance handles, each as returned by manifest_lineage() , for the upstream manifests this result was derived from. Required to be non-empty when metadata\$derived is TRUE.
metadata	A named list of caller-supplied metadata. The element derived is load-bearing: setting it to TRUE turns on the lineage requirement.
timestamp	A POSIXct emission time.
data_hash	The payload digest, as a single "sha256:<hex>" string. Emitters compute it over the payload slots; verify_manifest() recomputes and compares.

Details

The properties fall into four groups. Identity (manifest_version, emitter_package, emitter_version, run_id, method, timestamp) says what this is and what made it. Semantics (inferential_target, obs_schema) says what it is about, in a closed vocabulary a consumer can dispatch on. Payload (params, outputs, weights, obs_target, summary, seed) carries the load-bearing content, and is exactly the set covered by data_hash. Provenance (consumed_manifests, metadata) records what this result was derived from.

Two rules are enforced by the validator rather than left to convention. A manifest whose metadata\$derived is TRUE must record at least one entry in consumed_manifests: an approximation may never silently drop the lineage of the thing it approximated. And a summary, when set, must be a [manifest_summary\(\)](#) object, so a verdict has a typed home instead of riding in params as a single-row pseudo-parameter table.

The data_hash deliberately covers the payload only. The schema version, the emitter identity and the timestamp are metadata, so re-tagging a manifest does not invalidate a payload that has not changed, while any edit to the payload is detected by [verify_manifest\(\)](#).

Value

An orchestra_manifest object.

Class identity across homes

The contract began as a script sourced into a workspace, where the class was created outside any package and its class vector reads `c("orchestra_manifest", "S7_object")`. Defined inside this package the same class is package-qualified and reads `c("orchestraManifest::orchestra_manifest", "S7_object")`, and components that emit a manifest without depending on this package still mint the bare form. Both are the same contract. Consumers in this package therefore test identity with `is_orchestra_manifest()`, which accepts either, rather than with a strict class check that would reject every manifest emitted outside this package.

Author(s)

Max Moldovan

See Also

`verify_manifest()` and `consume_manifest()` for the integrity and schema guards, `as_orchestra_manifest()` for the emitter generic.

Examples

```
m <- orchestra_manifest(
  emitter_package = "example",
  emitter_version = "0.1.0",
  inferential_target = "parameters",
  run_id = "example-001",
  method = "example:demo",
  seed = 1L,
  params = data.frame(a = c(0.1, 0.2), b = c(1.5, 1.4)),
  timestamp = as.POSIXct("2026-07-05 09:00:00", tz = "UTC"),
  data_hash = "sha256:not-yet-computed"
)
m@emitter_package
```

ORCHESTRA_REFUSAL_CLASS

The shared refusal marker class

Description

The class a component may prepend to its refusal to declare conformance to the refusal contract explicitly, rather than relying on the `_refusal` naming convention. The predicates honour both routes, so opting in by marker and opting in by name are equally valid.

Usage

ORCHESTRA_REFUSAL_CLASS

Format

A length-1 character vector.

See Also

[as_orchestra_refusal\(\)](#) for stamping it, [is_orchestra_refusal\(\)](#) for the predicate that honours it.

verify_manifest	<i>Verify a manifest's payload against its stored hash</i>
-----------------	--

Description

Recomputes the payload digest from the manifest's own load-bearing slots and compares it with the data_hash the emitter stored. A mismatch means the payload was modified after emission.

Usage

```
verify_manifest(m)
```

Arguments

m	The manifest to verify. Anything that is not a manifest yields a failed verdict rather than an error.
---	---

Details

The check is deliberately total rather than throwing: a consumer often wants to record that a manifest failed verification and carry on with the other lenses it has, so the verdict is returned as data. The loud version is [consume_manifest\(\)](#), which is what a gate should call.

Because the hash covers the payload only, re-tagging a manifest with a different schema version, emitter or timestamp leaves verification intact while any edit to params, outputs, weights, obs_target, seed or summary breaks it.

Value

A list with ok, a length-1 logical, and message, a length-1 character vector describing the outcome.

Author(s)

Max Moldovan

See Also

[consume_manifest\(\)](#) for the erroring guard that calls this.

Examples

```
params <- data.frame(a = c(0.1, 0.2))
m <- orchestra_manifest(
  emitter_package = "example",
  emitter_version = "0.1.0",
  inferential_target = "parameters",
  run_id = "example-001",
  method = "example:demo",
  seed = 1L,
  params = params,
  timestamp = Sys.time(),
  data_hash = manifest_data_hash(params, NULL, NULL, NULL, 1L)
)
verify_manifest(m)$ok

m@params$a[1] <- 0.3
verify_manifest(m)$ok
```

Index

as_orchestra_manifest, 2
as_orchestra_manifest(), 16, 17, 23
as_orchestra_refusal, 3
as_orchestra_refusal(), 24
assert_obs_schema_correspondence, 4
assert_obs_schema_correspondence(), 22

combine_grounding, 6, 9, 10
combine_grounding(), 10
consume_manifest, 7, 20
consume_manifest(), 23, 24

GROUNDING_GROUNDED, 6, 8, 9, 10
grounding_token, 6, 8, 8
GROUNDING_UNVERIFIED, 6, 8, 9, 9

inferential_targets, 10
inferential_targets(), 16, 21
is.na(), 9
is_orchestra_decline, 11
is_orchestra_decline(), 13
is_orchestra_manifest, 12
is_orchestra_manifest(), 18, 23
is_orchestra_refusal, 13
is_orchestra_refusal(), 3, 4, 11, 24
isTRUE(), 19

manifest_data_hash, 14
manifest_from_gmm, 15
manifest_from_gmm(), 3
manifest_from_pesto, 16
manifest_from_pesto(), 3
manifest_lineage, 17
manifest_lineage(), 16, 22
manifest_summary, 18
manifest_summary(), 14, 22
manifest_version, 20
manifest_version(), 7, 21

orchestra_manifest, 2, 3, 5, 10, 12, 16–19,
21

ORCHESTRA_REFUSAL_CLASS, 3, 4, 13, 23

verify_manifest, 24
verify_manifest(), 7, 14, 22, 23