Jentic

Beyond OpenAPI: Workflows That AI Agents Can *Actually* Execute

Date:
3rd June, 2026

Presented by:
Frank Kilcommins

I'm Frank Kilcommins

› Head of Architecture @ Jentic

› Engineer and Architect (♥ APIs & Developer/Agent Experience)

› Governance Board member @ OpenAPI Initiative (OAI)

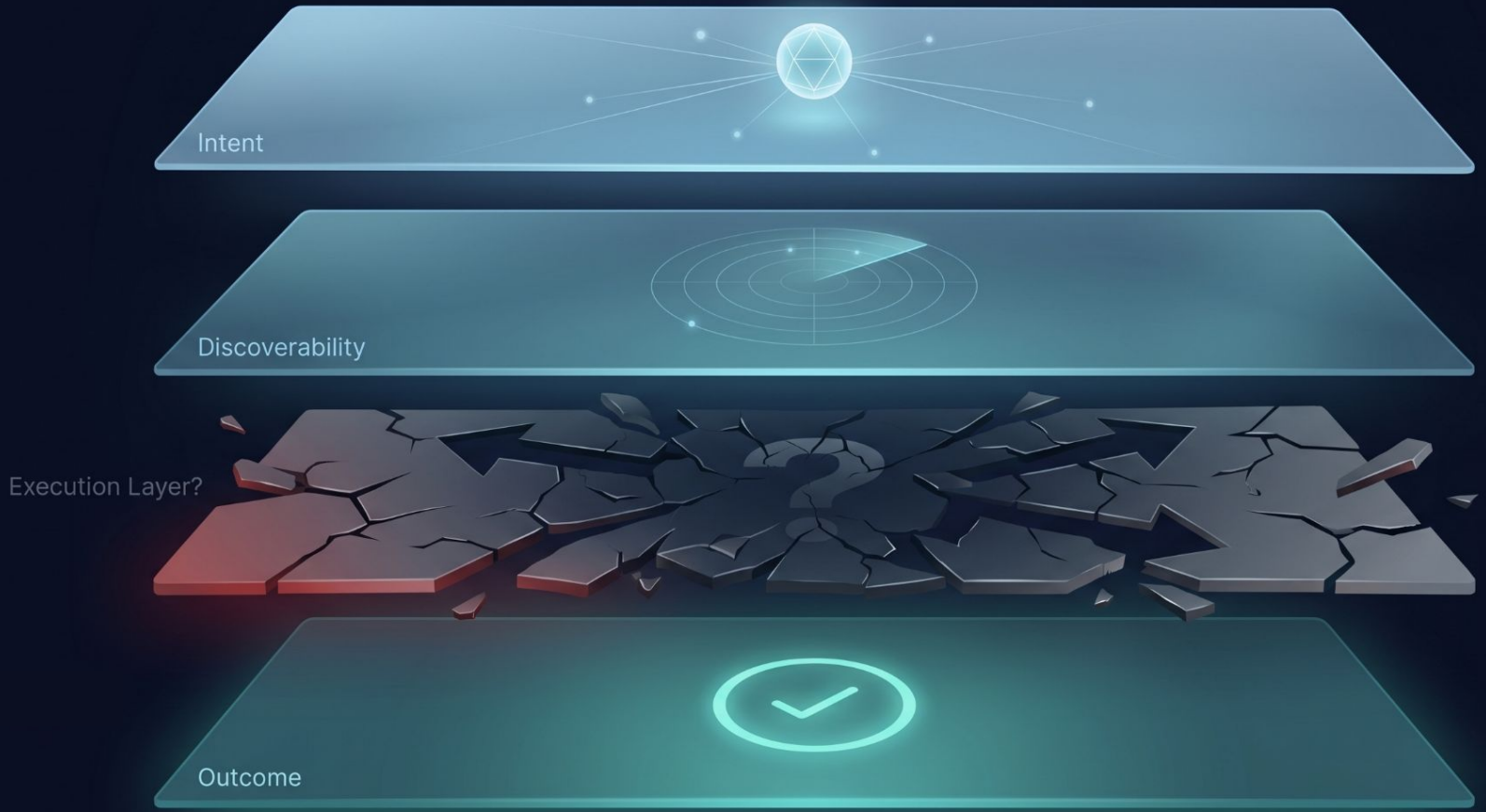
› Co-Author / Maintainer of the Arazzo Specification

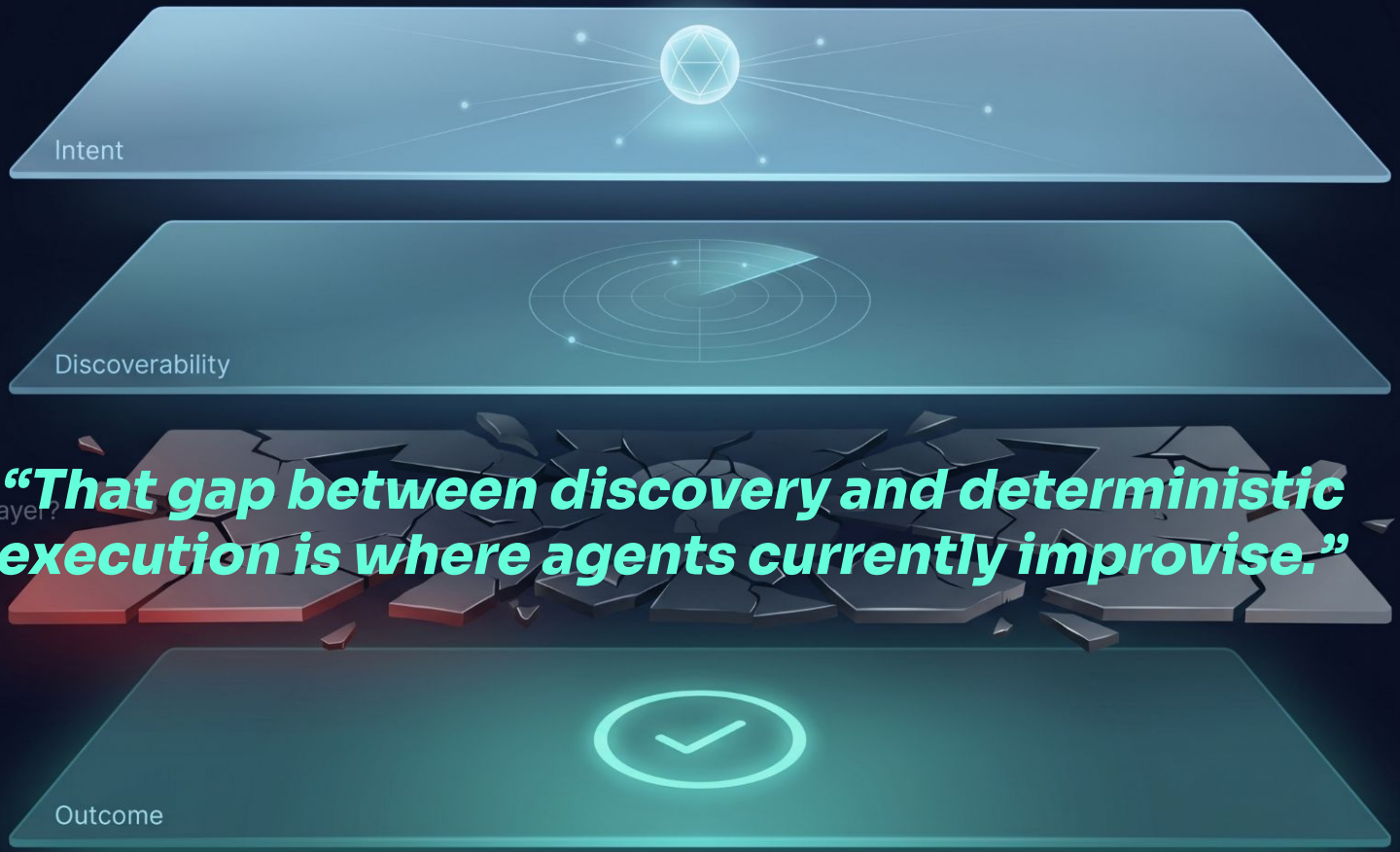
Connect:

@fkilcommins.bsky.social

@frank-kilcommins

frank@jentic.com


Intent
Discoverability
Execution Layer?
Outcome



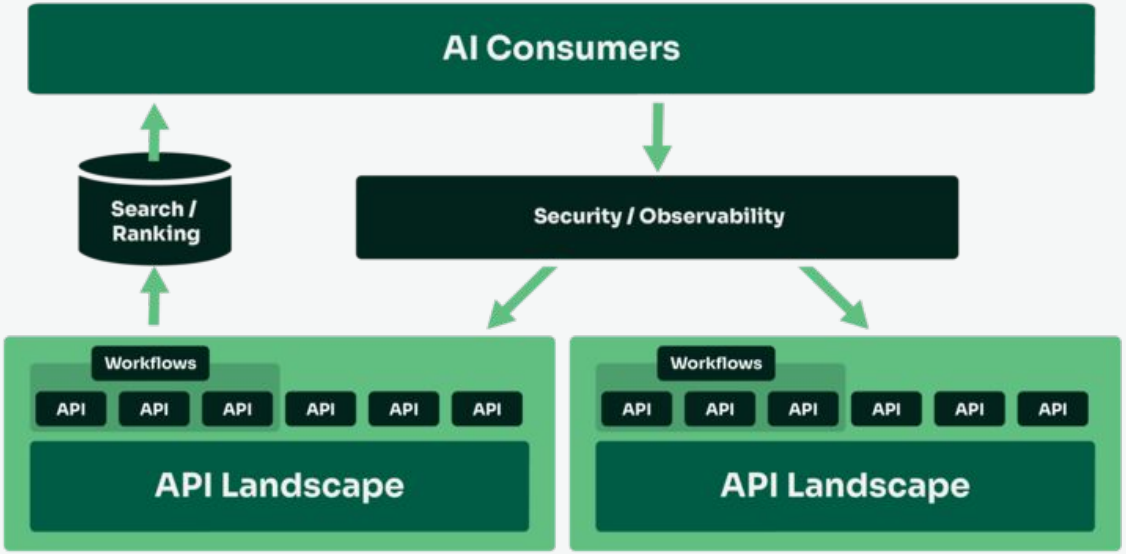
“That gap between discovery and deterministic execution is where agents currently improvise.”

Enabling Agents to Use APIs

	Prompting	MCP	Agent Skills	Workflows / Capabilities
Execution	LLM	Tool engine	LLM (guided)	Workflow engine
LLM role	Full	Discovery & invocation	Within constraints	Discovery & invocation
Reproducibility	Low	Medium	Medium	High
Maintainability	None	Medium (servers, code)	Low - medium	Low (declarative)

Skills v Workflow distinction: A skill tells the LLM *how to think* about a task (flexible path). A workflow tells the *system what to do*, step by step, without asking the LLM (fixed path). They compose: skills discover and run workflows

The API “Context Era”: Endpoints ≠ Use Cases



*“**Endpoints != Use-cases. Think about the workflows**” - Adam DuVander*

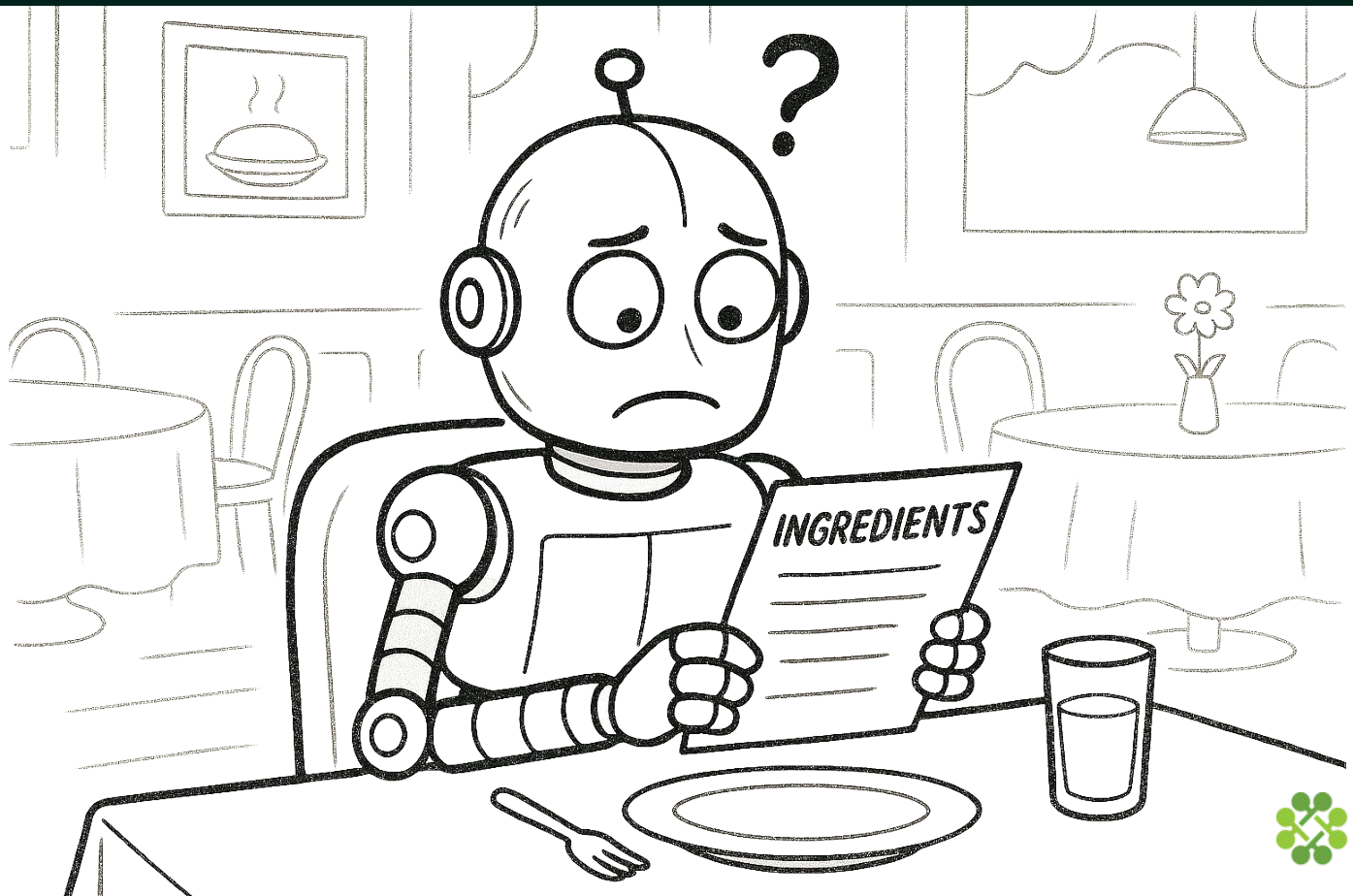
Why Workflows Matter

An agent's job isn't to invent business processes - it's to use them!

- Many tasks follow predictable, repeatable sequences and/or processes
- Agents should not be **forced to re-discover** the same steps
- Workflow-level tools reduce tool sprawl and context challenges
- Fewer tools = less context confusion = better, faster results

Workflows are recipes for *success*

Jentic



Arazzo
OPENAPI INITIATIVE

The open standard for workflows

Jentic

What is the Arazzo Specification?

It describes an approach to document **use-case oriented workflows** in a programmatically readable format

A workflow is a **series of API calls** which when woven together accomplish some business objective

The human-readable and machine-readable nature simultaneously improves **developer experience** (DX) and **agent experience** (AX)

Dominant Use Cases for Arazzo

Deterministic *context aware* recipes for API use (for **humans** and **AI-agents)**
- Make sense of large unwieldy API descriptions
- Bridge the *gap* where flows span more than a single API

Discoverable *living documentation* for agents and humans

Doubles as a **test specification** to assert workflow promises throughout API evolution

Targeted **code & SDK** (and MCP server) generation driven by **use-cases**

Open standard for codifying your **intellectual property API-oriented capabilities**

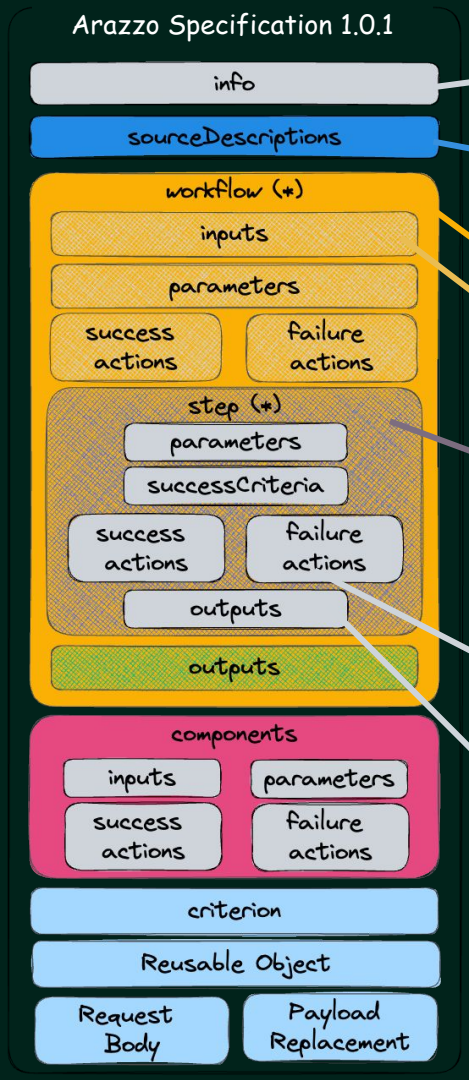
Example Travel Workflow Use-Case

A concierge-grade workflow for rebooking disrupted flights for *Platinum* and *Ultimate* passengers, rebooking hotels, and arranging transportation

Steps:

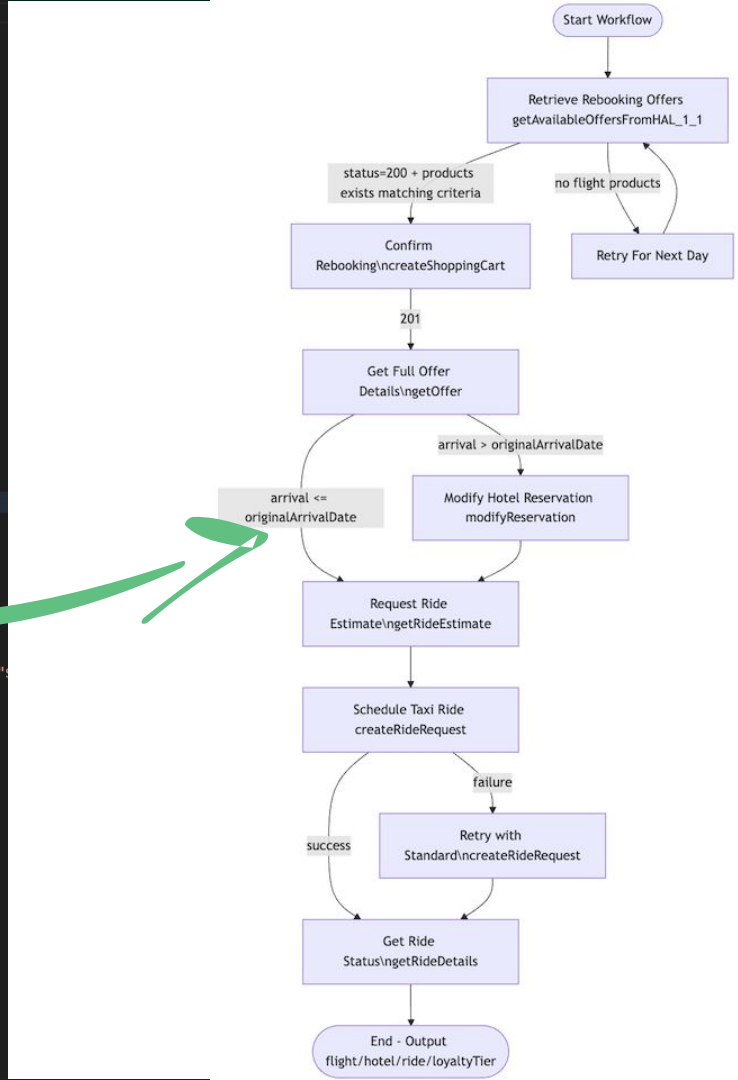
1. Retrieve rebooking options based on loyalty profile
2. Confirm selected offer and create shopping cart entry
3. Fetch full confirmed offer details
4. Modify hotel reservation (if arrival date not honoured)
5. Resolve hotel location - *Overture Maps via PlaceId*
6. Request luxury ride pickup for 30 mins after arrival
7. Confirm ride details and pickup ETA for customer support
8. Return consolidated itinerary

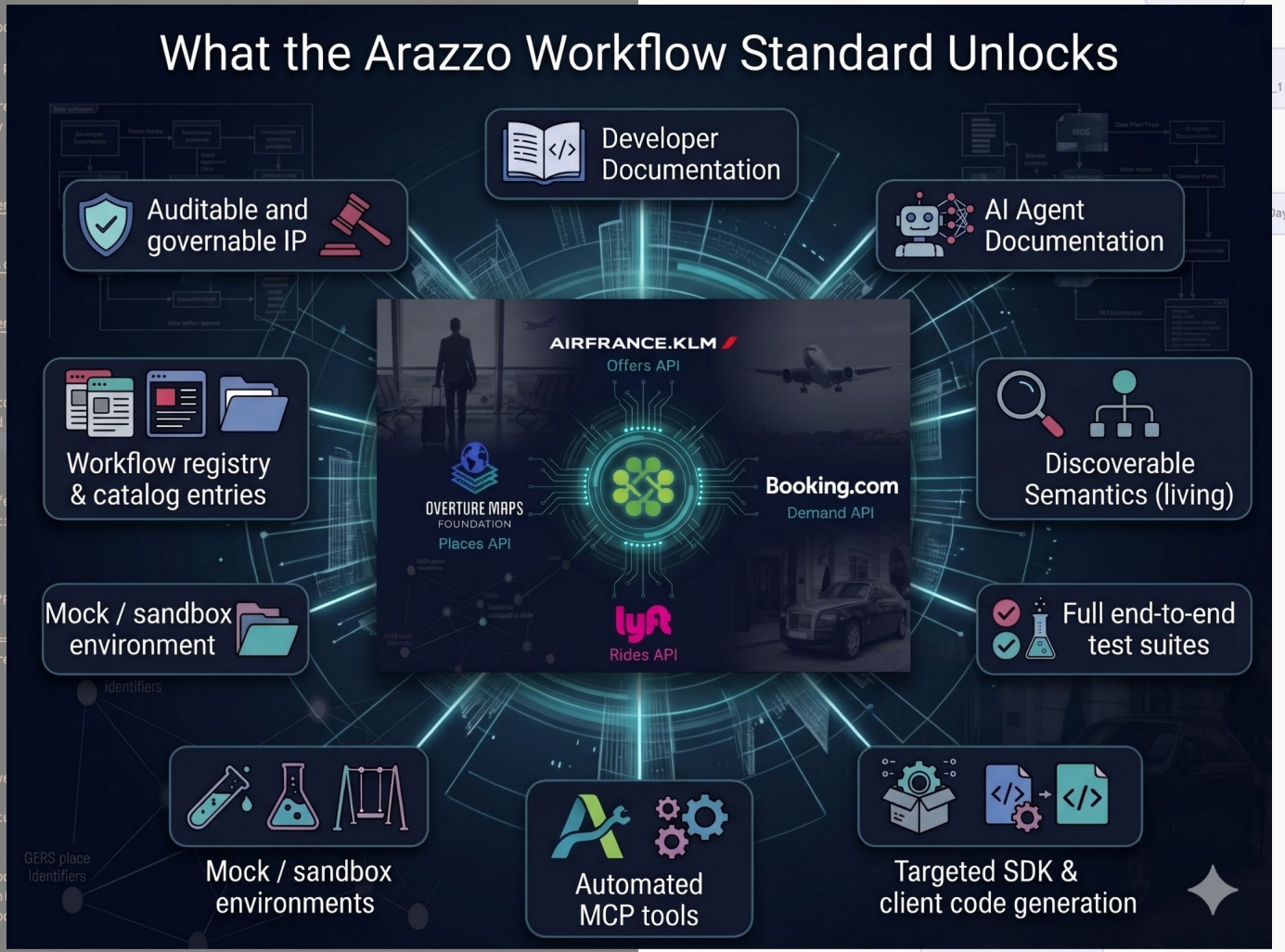
Multiple APIs and Endpoints Required




```yaml
arazzo: 1.0.1
info:
  title: Premium Concierge Reaccommodation Service
  version: "1.0.0"
  summary: Reaccommodate high-value passengers after travel disruption.
  description: >
    A concierge-grade workflow for rebooking disrupted flights for Platinum and Ultimate Blue passengers.
    Includes hotel update and luxury ride scheduling.

sourceDescriptions:
  - name: airfrance_klm
    url: https://raw.githubusercontent.com/zbloss/airfrance-klm-api/main/openapi.json
    type: openapi
  - name: booking
    url: https://developers.booking.com/_spec/demand/docs/open-api/demand-api.yaml?download
    type: openapi
  - name: lyft
    url: https://raw.githubusercontent.com/APIs-guru/openapi-directory/refs/heads/main/APIs/lyft.com/1.0.0/swagger.yaml
    type: openapi

workflows:
  - workflowId: premium-travel-reaccommodation
    summary: Reaccommodate disrupted high-tier loyalty members
    inputs: …

    steps:
      - stepId: retrieveRebookingOffers
        operationId: $sourceDescriptions.airfrance_klm.getAvailableOffersFromHAL_1_1
        parameters:
          - name: d
            in: query
            value: '{"bookingFlow":"REWARD","commercialCabins":[{"$value":"$inputs.preferredCabin"}],"customer":{"profileId":"
        successCriteria:
          - condition: $statusCode == 200
          - condition: $[?length(@.recommendations.flightProducts) > 0]
            context: $response.body
            type: JSONPath
        onFailure:
          - name: retryForTomorrow
            type: goto
            workflowId: premium-travel-reaccommodation
            parameters:
              - name: fallbackDepartureDate
                value: "2025-12-12"
            criteria:
              - condition: $statusCode == 200
              - condition: $[?length(@.recommendations.flightProducts) == 0]
                context: $response.body
                type: JSONPath
        outputs:
          matchedOffer: $response.body#/recommendations/0/flightProducts/0
```

```yaml
arazzo: 1.0.1
info:
  title: Premium Concierge Reaccommodation Service
  version: "1.0.0"
  summary: Reaccommodate high-value passengers after travel disruption.
  description: >
    A concierge-grade workflow for rebooking disrupted flights for Platinum and Ultimate Blue passengers.
    Includes hotel update and luxury ride scheduling.

sourceDescriptions:
  - name: airfrance_klm
    url: https://raw.githubusercontent.com/zbloss/airfrance-klm-api/main/openapi.json
    type: openapi
  - name: booking
    url: https://developers.booking.com/_spec/demand/docs/open-api/demand-api.yaml?download
    type: openapi
  - name: lyft
    url: https://raw.githubusercontent.com/APIs-guru/openapi-directory/refs/heads/main/APIs/lyft.com/1.0.0/swagger.yaml
    type: openapi

workflows:
  - workflowId: premium-travel-reaccommodation
    summary: Reaccommodate disrupted high-tier loyalty members
    inputs: …

    steps:
      - stepId: retrieveRebookingOffers
        operationId: $sourceDescriptions.airfrance_klm.getAvailableOffersFromHAL_1_1
        parameters:
          - name: d
            in: query
            value: '{"bookingFlow":"REWARD","commercialCabins":[{"$value":"$inputs.preferredCabin"}],"customer":{"profileId":"
        successCriteria:
          - condition: $statusCode == 200
          - condition: $[?length(@.recommendations.flightProducts) > 0]
            context: $response.body
            type: JSONPath
        onFailure:
          - name: retryForTomorrow
            type: goto
            workflowId: premium-travel-reaccommodation
            parameters:
              - name: fallbackDepartureDate
                value: "2025-12-12"
            criteria:
              - condition: $statusCode == 200
              - condition: $[?length(@.recommendations.flightProducts) == 0]
                context: $response.body
                type: JSONPath
        outputs:
          matchedOffer: $response.body#/recommendations/0/flightProducts/0
```




What the Arazzo Workflow Standard Unlocks
Developer Documentation
Auditable and governable IP
AI Agent Documentation
AIRFRANCE.KLM
Offers API
Workflow registry & catalog entries
Discoverable Semantics (living)
OVERTURE MAPS FOUNDATION
Places API
Booking.com
Demand API
Mock / sandbox environment
Full end-to-end test suites
lyft
Rides API
identifiers
Mock / sandbox environments
Automated MCP tools
Targeted SDK & client code generation
GERS place identifiers

Specification Structure

Metadata about the defined Arazzo Document

Arazzo Specification - Info Object Example

```yaml
arazzo: 1.0.0
info:
  title: A pet adoption workflow
  summary: Search for and adopt a pet from the PetCo
  description: This workflow walks you through the steps of `searching` for, `selecting`,
and `adopting` an available pet.
  version: 1.1.0
```

Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Jentic

Arazzo Specification - Source Descriptions Example

```yaml
sourcesDescriptions:
- name: petsAPI
  url: https://api.swaggerhub.com/apis/frank-kilcommins/Pets-API/1.0.0/swagger.json
  type: openapi
- name: adoptionsAPI
  url: https://api.swaggerhub.com/apis/frank-kilcommins/Adoptions-API/1.0.0/swagger.json
  type: openapi
```

Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

Jentic

Arazzo Specification -Workflows Example

```yaml
workflows:
- workflowId: FindAndAdoptPet
  summary: This workflow lays out the steps and API calls needed to search for and adopt a Pet
  Description: This workflow lays out the steps needed to adopt a pet.
  inputs: {}
  parameters: []
  successActions: []
  failureActions: []
  steps: []
  dependsOn: []
  outputs: []
```

Metadata about the defined Arazzo Document

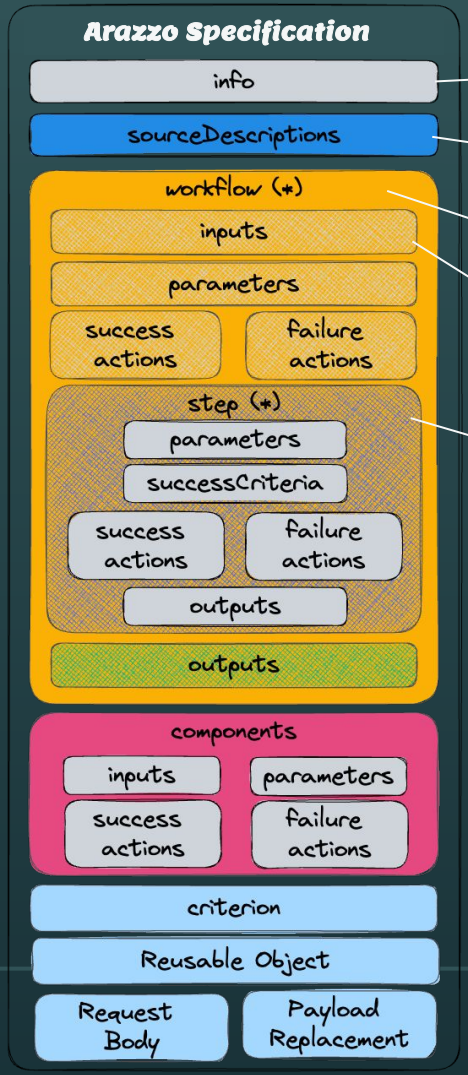
Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

A JSON Schema object representing the inputs used by this workflow

Arazzo Specification -Workflow Inputs Example

```yaml
inputs:
  type: object
  properties:
    category:
      type: string
    breed:
      type: string
    location:
      type: string
    apiKey:
      type: string
  required:
  - apiKey
  - category
  - breed
```



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

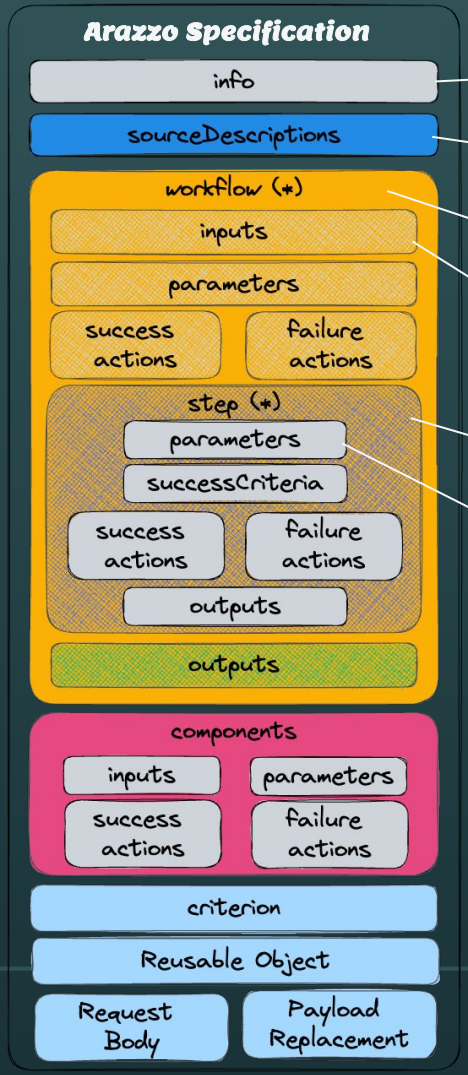
A JSON Schema object representing the inputs used by this workflow

The defined workflow steps, each representing a call to an API operation (or another workflow)

Arazzo Specification - Steps Example

```yaml
steps:
- stepId: searchPets
  description: This step demonstrates how to search pets matching the criteria
  operationId: petsAPI.getPets
  parameters: []
  successCriteria:
  - condition: $statusCode == 200
  - condition: $[?length(@.pets) > 0]
    context: $response.body
    type: JSONPath
  onSuccess: []
  onFailure: []
  outputs: []
```

Jentic



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

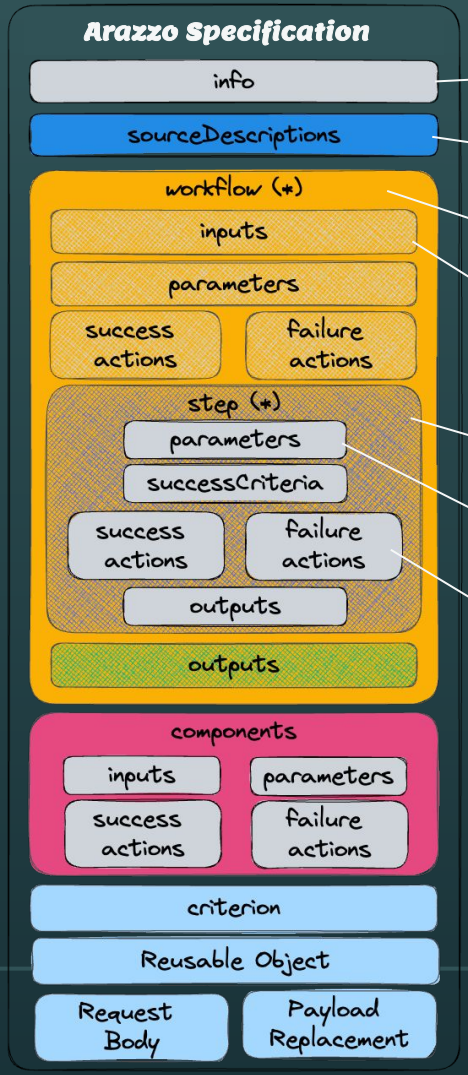
A JSON Schema object representing the inputs used by this workflow

The defined workflow steps, each representing a call to an API operation (or another workflow)

A list of parameter objects, representing parameters to pass to an operation or workflow

Arazzo Specification - Steps Parameters Example

```yaml
parameters:
- name: category
  in: query
  value: $inputs.category
- name: breed
  in: query
  value: $inputs.breed
- name: location
  in: query
  value: $inputs.location
- name: status
  in: query
  value: available
```



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

A JSON Schema object representing the inputs used by this workflow

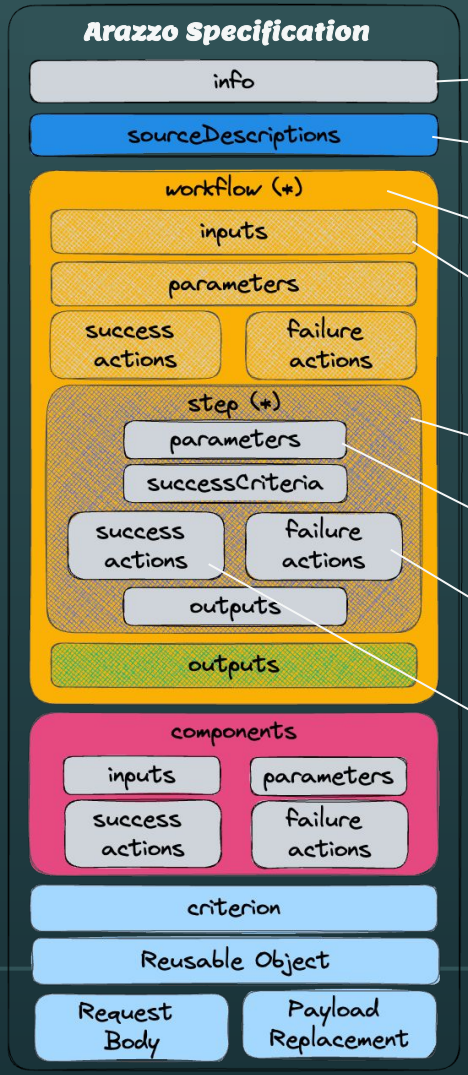
The defined workflow steps, each representing a call to an API operation (or another workflow)

A list of parameter objects, representing parameters to pass to an operation or workflow

An array of failure action objects that specify what to do on step failure

Arazzo Specification - FailureAction Object Example

```yaml
name: retryStep
type: retry
retryAfter: 1
retryLimit: 5
criteria:
    # assertions to determine if this action should be executed
    - condition: $statusCode == 503
```



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

A JSON Schema object representing the inputs used by this workflow

The defined workflow steps, each representing a call to an API operation (or another workflow)

A list of parameter objects, representing parameters to pass to an operation or workflow

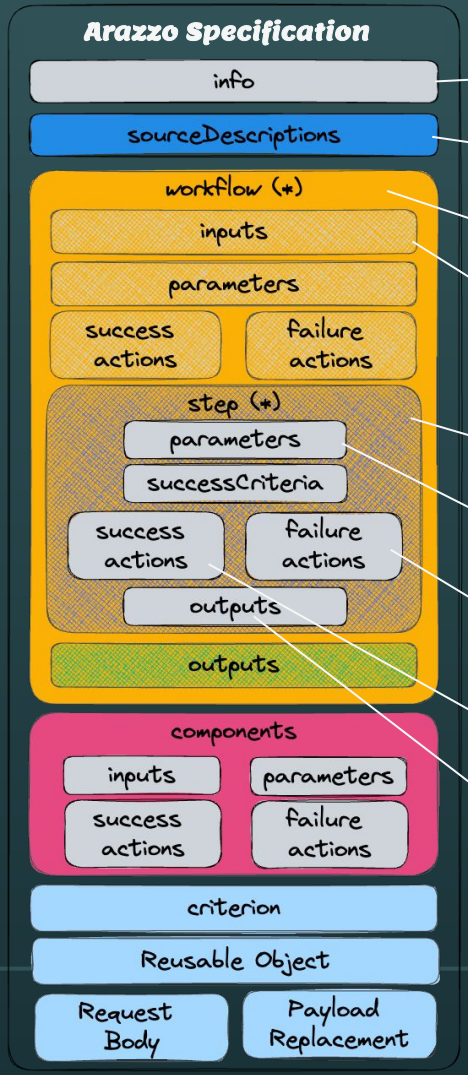
An array of failure action objects that specify what to do on step failure

An array of success action objects that specify what to do upon step success

Jentic

Arazzo Specification - SuccessAction Object Example

```yaml
name: JoinWaitingList
type: goto
stepId: joinWaitingListStep
criteria:
    # assertions to determine if this success action should be executed
    - context: $response.body
      condition: $[?count(@.pets) > 0]
      type: JSONPath
```



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

A JSON Schema object representing the inputs used by this workflow

The defined workflow steps, each representing a call to an API operation (or another workflow)

A list of parameter objects, representing parameters to pass to an operation or workflow

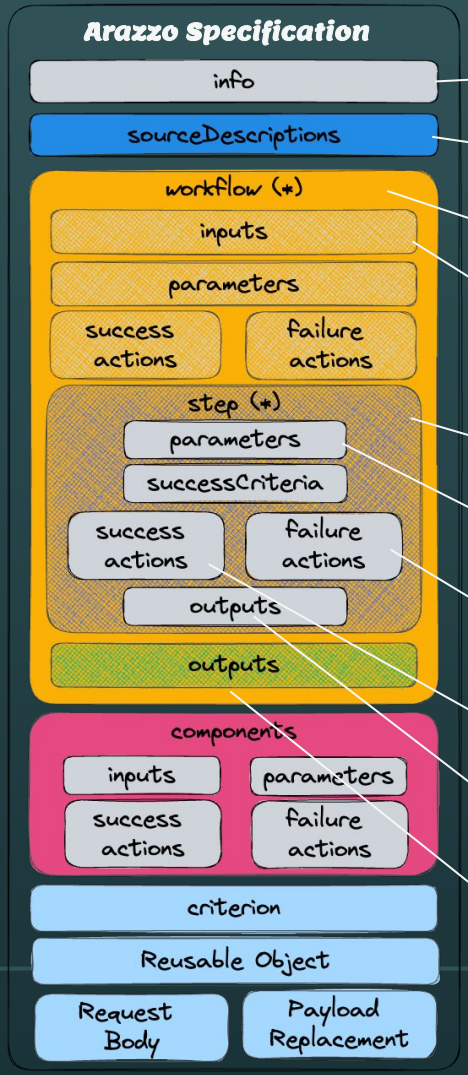
An array of failure action objects that specify what to do on step failure

An array of success action objects that specify what to do upon step success

A map between a friendly name and a dynamic output value for a step

Arazzo Specification - Step Outputs Example

```yaml
outputs:
  petId: $response.body.pets[0].id
```



Metadata about the defined Arazzo Document

Lists source descriptions (e.g., OpenAPI description) that can be referenced by one or more workflows

Describes the workflows to be taken across one or more APIs to achieve an objective/outcome.

A JSON Schema object representing the inputs used by this workflow

The defined workflow steps, each representing a call to an API operation (or another workflow)

A list of parameter objects, representing parameters to pass to an operation or workflow

An array of failure action objects that specify what to do on step failure

An array of success action objects that specify what to do upon step success

A map between a friendly name and a dynamic output value for a step

A map between a friendly name and a dynamic output value for a workflow

Jentic

Arazzo Specification - Workflow Outputs Example

```yaml
outputs:
  petId: $steps.searchPets.outputs.petId
  petName: $steps.searchPets.outpust.petName
  status: $steps.approveAdoption.outputs.status
```

Reusable (referenceable) components and objects

Just Released – “Arazzo Specification 1.1”

TABLE OF CONTENTS

The Arazzo Specification v1.1.0

Version 1.1.0

Published 17 May 2026

▼ More details about this document

Latest published version:
https://spec.openapis.org/arazzo/latest.html

Latest editor's draft:
https://github.com/OAI/Arazzo-Specification/

Editors:
Frank Kilcommins
Nick Denny
Kevin Duffey

Participate
GitHub OAI/Arazzo-Specification
File a bug
Commit history
Pull requests

What is the Arazzo Specification?

The Arazzo Specification provides a mechanism that can define sequences of calls and their dependencies to be woven together and expressed in the context of delivering a particular outcome or set of outcomes when dealing with API descriptions (such as OpenAPI descriptions).

What’s New:

- AsyncAPI Support
- Chained Workflow Execution
- Advanced JSONPath / XPath Selector Support
- OpenAPI 3.2 Alignment
- Precision Improvements:
  - Full ABNF grammar
  - Truthy/Falsy condition semantics
  - And more!

See https://spec.openapis.org/arazzo/v1.1.0.html

Arazzo 1.1 – AsyncAPI Support

Include AsyncAPI Documents in Source Descriptions

```yaml
sourceDescriptions:
  - name: PaymentsEventBus
    url: ./user-payment-events.asyncapi.yaml
    type: asyncapi
```

Source Descriptions Object now supports `openapi`, `arazzo`, and `asyncapi` documents.

Arazzo 1.1 – AsyncAPI Support

Async steps define an `action` (“send” or “receive”) indicating message flow intent.

Step Object references operations (or channels) described in AsyncAPI Document using `operationId` or `channelPath`.

Async Recieve Step with specified timeout

```yaml
- stepId: WaitForUserAck
  operationId: $sourceDescriptions.AsyncAPIEventBus.receiveUserAck
  action: receive
  correlationId: $steps.GetUser.outputs.userCorrelationId
  timeout: 10000
  successCriteria:
    - condition: $message.payload.status == "ok"
```

A `correlationId` in AsyncAPI links a request with its response, and must be in-sync with the AsyncAPI Document

A `timeout` sets the maximum number of milliseconds to wait for the step to complete before aborting and failing the step.

Arazzo 1.1 – AsyncAPI Support

Async Send Step with specified dependsOn

```yaml
- stepId: SendPaymentRequest
  operationId: $sourceDescriptions.AsyncAPIEventBus.sendPaymentRequest
  action: send
  dependsOn:
    - WaitForUserAck
  requestBody:
    contentType: application/json
    payload: |
      {
        "userId": "{$steps.GetUser.outputs.userId}",
        "amount": "{$inputs.amount}"
      }
  parameters:
    - name: correlationId
      in: header
      value: "{$inputs.orderId}"
```

Step Object now has the ability to specify explicit dependencies.

These are a list of steps that MUST be completed before the current step can be executed.

`dependsOn` only establishes a prerequisite relationship for the current step and does not trigger execution of the referenced steps.

Arazzo 1.1 – Chaining Workflows

Chaining Workflows with Action Objects

```yaml
name: retryStep
type: retry
retryAfter: 1
retryLimit: 3
workflowId: refreshTokenFlow
parameters:
  - name: refreshToken
    value: $inputs.refreshToken
  - name: clientId
    value: $inputs.clientId
criteria:
  - condition: $statusCode == 401
  - context: $response.header.WWW-Authenticate
    condition: '^.*[,\s]error="invalid_token".*$'
    type: regex
```

Action Objects now fully support chaining workflows, and mapping workflow inputs using the `parameters` fixed field.

Arazzo 1.1 – Advanced Selector Support

New **Selector Object** which enables fine-grained traversal and precise data selection from structured data such as JSON or XML, using a defined selector syntax such as `jsonpath`, `xpath`, or `jsonpointer`.

Selector Object Usage Example

```yaml
outputs:
  userEmail:
    context: $response.body
    selector: $.user.profile.email
    type: jsonpath
```

The `context` sets the root node for the selector, the `selector` specifies the expression, while the `type` indicates the expression type.

The Selector Object is usable across workflow outputs, step outputs, requestbody payloads, parameter values, payload replacement values.

Arazzo 1.1 – Selector Expression Types

Overriding the default Selector Expression versions via the **Expression Type Object** (renamed from Criterion Expression Type Object)

Selector Obj combined with Expression Type Obj

```yaml
requestBody:
  contentType: application/json
  payload:
    invoiceId:
      context: $steps.fetchXml.outputs.invoiceXml
      selector: /Invoice/Header/InvoiceNumber
      type:
        type: xpath
        version: xpath-30
```

Specifying the `type` and `version` for the selector expression.

Arazzo 1.1 – Selector Expression Types

Ov
(re

The supported expression selector types and versions are as follows:

Type	Allowed Versions	Default
`jsonpath`	`rfc9535`, `draft-goessner-dispatch-jsonpath-00`	`rfc9535`
`xpath`	`xpath-31`, `xpath-30`, `xpath-20`, `xpath-10`	`xpath-31`
`jsonpointer`	`rfc6901` (added for completeness)	`rfc6901`

```
type:
  type: xpath
  version: xpath-30
```

Arazzo 1.1 – OpenAPI 3.2 Alignment

The Parameter Object has a new `querystring` option for the `in` fixed field, which treats the entire URL query string as a single value.

Mapping to an API Operation's "querystring" parameter

```yaml
# Querystring Example (application/x-www-form-urlencoded)
- name: searchParams
  in: querystring
  value: "filter=active&sort=desc&limit=50"

# Querystring with Runtime Expressions (application/x-www-form-urlencoded)
- name: fullQuery
  in: querystring
  value: "category={$inputs.category}&minPrice={$inputs.minPrice}&inStock=true"

# Querystring Example (application/json)
- name: filterParams
  in: querystring
  value: '{"filter":"active","sort":"desc","limit":50}'
```

Arazzo 1.1 – Identity-based Referencing

To ensure interoperability, when referencing an Arazzo Description by URI, references MUST use the target document's `$self` if present.

$self for identity-based document referencing

```yaml
arazzo: 1.1.0
$self: https://api.example.com/workflows/pet-purchase.arazzo.yaml
info:
  title: A pet purchasing workflow
  summary: A workflow for how to purchase a pet through a
sequence of API calls
  description: |
      Walks you through the workflow and steps of `searching`
for, `selecting`, and `purchasing` an available pet.
  version: 1.0.0
```

A URI-reference for the Arazzo Description ensuring portable and unambiguous reference resolution.

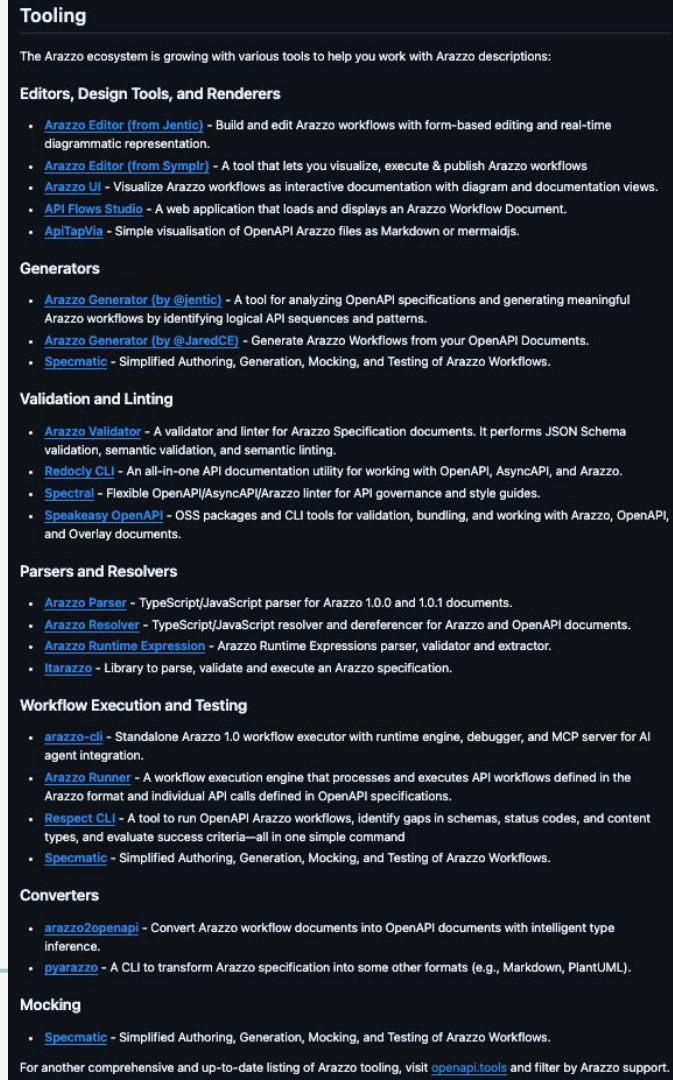
Arazzo 1.1 – Housekeeping

ABNF Expression

- Complete grammar
- Clarity on string embedding
- Source Description resolution ordering

Truthy/Falsy evaluation semantics for conditions

Tool listing in ReadMe



Tooling

The Arazzo ecosystem is growing with various tools to help you work with Arazzo descriptions:

Editors, Design Tools, and Renderers

- Arazzo Editor (from Jentic) - Build and edit Arazzo workflows with form-based editing and real-time diagrammatic representation.
- Arazzo Editor (from Symplr) - A tool that lets you visualize, execute & publish Arazzo workflows
- Arazzo UI - Visualize Arazzo workflows as interactive documentation with diagram and documentation views.
- API Flows Studio - A web application that loads and displays an Arazzo Workflow Document.
- ApiTapVia - Simple visualisation of OpenAPI Arazzo files as Markdown or mermaidjs.

Generators

- Arazzo Generator (by @jentic) - A tool for analyzing OpenAPI specifications and generating meaningful Arazzo workflows by identifying logical API sequences and patterns.
- Arazzo Generator (by @JaredCE) - Generate Arazzo Workflows from your OpenAPI Documents.
- Specmatic - Simplified Authoring, Generation, Mocking, and Testing of Arazzo Workflows.

Validation and Linting

- Arazzo Validator - A validator and linter for Arazzo Specification documents. It performs JSON Schema validation, semantic validation, and semantic linting.
- Redocly CLI - An all-in-one API documentation utility for working with OpenAPI, AsyncAPI, and Arazzo.
- Spectral - Flexible OpenAPI/AsyncAPI/Arazzo linter for API governance and style guides.
- Speakeasy OpenAPI - OSS packages and CLI tools for validation, bundling, and working with Arazzo, OpenAPI, and Overlay documents.

Parsers and Resolvers

- Arazzo Parser - TypeScript/JavaScript parser for Arazzo 1.0.0 and 1.0.1 documents.
- Arazzo Resolver - TypeScript/JavaScript resolver and dereferencer for Arazzo and OpenAPI documents.
- Arazzo Runtime Expression - Arazzo Runtime Expressions parser, validator and extractor.
- Itarazzo - Library to parse, validate and execute an Arazzo specification.

Workflow Execution and Testing

- arazzo-cli - Standalone Arazzo 1.0 workflow executor with runtime engine, debugger, and MCP server for AI agent integration.
- Arazzo Runner - A workflow execution engine that processes and executes API workflows defined in the Arazzo format and individual API calls defined in OpenAPI specifications.
- Respect CLI - A tool to run OpenAPI Arazzo workflows, identify gaps in schemas, status codes, and content types, and evaluate success criteria—all in one simple command
- Specmatic - Simplified Authoring, Generation, Mocking, and Testing of Arazzo Workflows.

Converters

- arazzo2openapi - Convert Arazzo workflow documents into OpenAPI documents with intelligent type inference.
- pyarazzo - A CLI to transform Arazzo specification into some other formats (e.g., Markdown, PlantUML).

Mocking

- Specmatic - Simplified Authoring, Generation, Mocking, and Testing of Arazzo Workflows.

For another comprehensive and up-to-date listing of Arazzo tooling, visit openapi.tools and filter by Arazzo support.

Arazzo: Looking Ahead to 1.2, 1.3, … , 2.0

- gRPC, GraphQL, SOAP, MCP, A2A step support
- Actor-in-loop support (human or agent)
- Transformer / function support
- Loops

Arazzo Specification 1.0 - Cheat Sheet

"A Tapestry for API Workflows"

Jentic

Document Structure

An Arazzo Document is a JSON or YAML file containing the following root elements:

```
arazzo 1.0.1            # The arazzo spec version
info: {}                # Workflow & document info, summary, ...
sourceDescriptions: {}  # APIs / workflows available for use
workflows: {}           # List of workflows to achieve use-cases
components: {}          # Reusable objects (referenceable)
```

General Information

```
info:
  title: Your Epic Workflow
  summary: Helps you achieve Epic outcomes
  description: Our Epic Workflow allows for ...
  version: 1.5.10
```

Source Descriptions

Defines the APIs and Workflows (OpenAPI/Arazzo) referenced by workflows authored in the current document (acts like 'imports' and/or 'namespaces' in programming languages).

```
sourceDescriptions:
  name: exampleAPI
  url: https://example.com/apis/someapi.openapi.json
  type: openapi
  name: AuthFlows
  url: https://example.com/apis/authflow.arazzo.json
  type: arazzo
```

Reusable Components

Reusable parameters, inputs, actions across workflows and steps.

```
components:
  parameters:
    apiKey:
      name: apiKey
      in: header
      value: $inputs.apiKey
```

Success Criteria

A list of assertions to determine the success of the step. All must be satisfied for the step to be deemed successful.

```
successCriteria:
  # assertions to determine step was successful
  condition: $statusCode == 200
  condition: $[?length(@.stuff) > 0]
  context: $response.body
  type: jsonpath
```

Branching & Conditionals

Success and Failure actions can control conditional navigation between steps or retry logic.

```
onFailure:
  name: retryGet
  type: retry # other types 'goto', 'end', ...
  retryAfter: 1
  retryLimit: 3
  criteria:
    condition: $statusCode == 503
```

Runtime Expressions

Success and Failure actions can control conditional navigation between steps and/or retry logic.

```
$inputs.userId
$steps.getPet.outputs.petId
$response.body#/status
$statusCode ...
```

Referencing or Chaining Workflows

Use 'workflowId' in a step to call another workflow.

```
workflowId: main-workflow
steps:
  stepId: sub-flow
  workflowId: another-workflow-runs-here
  parameters: [...]
```

Workflows

Describes the steps to be taken across one or more APIs to achieve an objective.

```
workflows:
  workflowId: EpicWorkflowId
  summary: Helps you achieve epic outcomes
  description: Achieve it by combining API calls and ...
  inputs: {}              # The inputs needed to start (JSON Schema)
  parameters: []          # Common params across all steps
  successActions: []      # Common actions across each successful step
  failureActions: []      # Common actions across each failed step
  steps: []               # The steps needed to achieve the workflow
  dependsOn: []           # Don't run this workflow until others complete
  outputs: []             # What to return at the end of the workflow
```

Steps

Each step represents a call to an API operation or to another workflow (e.g., workflow chaining).

```
steps:
  stepId: epicStepId
  description: This step helps delivery epic outcomes by...
  operationId: $sourceDescriptions.exampleApi.getEpics
  (operationPath):    # a JSON Pointer to an operation if no id exists
  (workflowId):       # a reference to a workflow rather than an API
  parameters: []      # map params into API operation or workflows
  successCriteria: [] # what determines success of a step
  requestBody: {}     # how to send a request body payload to API
  onSuccess: []       # things to do once success (log, branch, ...)
  onFailure: []       # things to do upon failure (retry, end, goto)
  outputs: []         # what to make available for nexts workflow steps
```

Outputs

Dynamic named values from steps and workflows.

```
outputs:
  someName: $steps.step1.outputs.output1
  statusCode: $statusCode
```

Scan here for a digital copy!

“It’s not about the agent; it’s about the workflow”

– McKinsey

Thank you!

Connect:

@fkilcommins.bsky.social

@frank-kilcommins

frank@jentic.com

Jentic