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
$atatusCode ...
```

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!