AsyncAPI 3.0 Cheat Sheet

bump.sh

Document Structure

An AsyncAPI document is a JSON or YAML file containing the following root elements:

```yaml
asyncapi: 3.0  # The spec version
info: {}       # API and document info, tags, ...
servers: {}    # List of available servers
channels: {}   # List of addressable channels
operations: {} # List of operations
components: {} # Reusable objects ($ref)
```

General Information

```yaml
info:
  title: Your Awesome API
  version: 1.2.14
  description: What our API does is...
  tags: {} # Define the logical grouping tags

servers:
  production:
    host: example.com
    pathname: /ws
    protocol: websocket
    security: []
  staging:
    ...
```

Security

Define the APIs Security Schemes, then apply them globally per server or per operation using the security keyword.

Define security schemes

```yaml
components:
  securitySchemes: # Define for use later
    ApiKey: # Arbitrary name
      type: http
      scheme: bearer
```

Apply security schemes (in server or operation objects)

```yaml
operations:
  onSignup: # Apply on this operation only
    action: send
    security:
      - $ref: '#/components/securitySchemes/ApiKey'
```

Allowed types

`userPassword`, `apiKey`, `X509`, `httpApiKey`, `http`, `oauth2`, `openIdConnect`, `scramSha256`, `scramSha512`, ...

Channels

Think of channels as data pipelines delivering the intended messages to the right participants.

```yaml
channels:
  userSignedUp:
    address: 'user.signedup'
    messages:
      userData:
        $ref: '#/components/messages/userData'
```

Operations

Relation between an action, a channel and the allowed messages.

```yaml
operations:
  onUserSignUp:
    title: User sign up
    description: It updates this and remove that...
    action: receive # Either send or receive
    messages:
      - $ref: '#/channels/userSignedUp/messages/userData'
    channel:
      $ref: '#/channels/userSignedUp'
```

Messages

Describe data structures that can be exchanged in your API, either at the channel or operation level.

```yaml
userData:
  description: This message is used to...
  payload: {} # schema or multiFormat schema object
  headers: {} # A map of key-value pairs schema
  correlationId: {} # Identifier for tracing
```

Schemas

Schemas define input and output data types. JSON Schema is the default format, but you can use other formats (Avro, RAML, ...).

JSON Schema

```yaml
type: object
title: User
properties:
  id:
    type: string
    format: uuid
```

Avro

```yaml
schemaFormat: ...avro;version=1.9
schema:
  type: record
  name: User
    namespace: example.avro
    fields:
      - name: id
        type: string
        logicalType: uuid
```

Protocol Bindings

Provide additional context and configuration options for the protocols used by your API. Depending on the protocol, bindings can be defined at server, channel, operation or message level.

```yaml
bindings:
  ws:
    headers:
      properties:
        Authorization:
          type: string
    method: GET
```

Full list: github.com/asyncapi/bindings

Reuse Elements

Avoid duplicating elements by defining reusable components:

```yaml
components:
  servers: {}
  channels: {}
  operations: {}
  messages: {}
  schemas: {}
  securitySchemes: {}
  ...
```

Use your components with the $ref keyword:

```yaml
channels:
  userSignedUp:
    messages:
      userData:
        $ref: '#/components/messages/userData'
```

Components can be reached:

internally: `#/components/schemas/User`
through a remote URL: `https://example.com/user.yml`
on file system: `./user.yml#/components/schemas/User`

Polymorphism

AsyncAPI traits

`traits: []` You can define traits to reuse specific properties across multiple messages and operations.

In JSON Schema

`oneOf:` Exactly one of the schemas (XOR)
`anyOf:` One or more of the schemas (OR)
`allOf:` All the schemas (AND)

In Avro

`type: []` One or more of the schemas (OR)