IATA Open Air API Standards and Best Practices

Version 1.2

August 2021

Contents

Revision History

Version	Date	Description of change
1.0	May 2020	Initial version

1.1	February 2021	OAS Schema mapping with AIDM elements: 2.4.14 Schema Object;
		Resource naming: 2.4.3 URI, 2.4.4 Server Object, 2.4.6 Paths Object;
		Versioning: 2.4.2 Info Object, 3.2 Versioning.

1.2	August 2021	OAS Schema mapping with AIDM elements: 2.4.14 Schema Object; Description of industry standard APIs.

1. Introduction

1.1. Purpose

IATA's Open Air initiative was created to develop industry standards and best practices for the use of RESTful API technology in the airline industry, and an API ecosystem conformant to the standards.

The purpose of this document is to define a common technical approach to describing API Definitions so that industry parties can benefit from a shared understanding leading to efficiency of API development, understanding, implementation and use of conformant APIs.

1.2. Audience

This standard assumes the reader has an understanding of the OAS 3.0 specification, and AIDM methodology. This document is intended for:

- API developers in the airline industry, who have experience in RESTful API design and development;
- Enterprise Architects responsible for the coherent strategies of their companies' integration policies;
- Planners and Managers responsible for delivering business integration solutions.

1.3. Document Structure

The document covers each OAS Object and its fields or patterned fields where there is a variation with the OAS Standard, or where the Object is required but no variation is defined. The OAS nodes affected by this standard are shown in Figure 1 – Open API Standards Scope.

Variances in the standard for an Object that appears in more than one place, that is, more than one node in the OAS specification are detailed in a sub-section of the section covering the Object.

The sections detailing each object are ordered as they appear in the OAS Standard.

2. Open Air API Standard

2.1. Objective

The objective of this standard is to ensure API Documents are consistent in their structure, nomenclature and semantics and to enable the data structures in conformant messages to be validated; JSON Schema keywords for validation are utilized to achieve this.

2.2. Interpretation

When describing the Best Practices and the Checklist, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2.3. Standard APIs

There are 2 types of standard APIs supported in the Open Air program.

1. Compliant API

2. Conformant API

2.3.1 Compliant – Proprietary API

IATA recommends the API providers to follow the industry adopted Open Air API Standards and Best Practices in the proprietary API. The provider may design and develop the proprietary API to address any business scenario in support of any stages or any steps across the Airline Value Chain, which is shown in Figure 5.

An airline industry relevant API provider, who follows the standards and successfully delivers the API, may apply for Open Air compliance certification for the proprietary API. The certified API is recognized as a Compliant API and is listed in the Open Air Industry API Registry, providing visibility to the industry.

2.3.2 Conformant – Industry Standard API

A Conformant API is an API designed by the industry standard development bodies, for example, the business and technology working groups and boards under IATA PSC governance, to address a common business scenario faced by most of the applicable industry parties. The business scenario can be from any stages or steps across the Airline Value Chain capabilities shown in Figure 5.

A Conformant API specification MUST adhere to the Open Air standard and Best Practices. The delivery of IATA Conformant APIs MUST follow industry governance process under the Passenger Standards Conference (PSC). The business process and requirement MUST be adopted by corresponding business boards, for example, Plan Board, Shop-Order Board, Pay-Account Board, Travel Board, etc, and the API specification, as well as data model MUST be adopted by Architecture and Technology Strategy Board (ATSB).

2.4. API specification

This section defines the Open Air Standard and Best Practice for API specification. API specification is a reference manual on the API capability, meaning how the API behaves and what to expect from the API. A well-documented specification will help developers to understand and adopt the API.
An IATA Open Air API document **MUST** be RESTful, **MUST** adhere to OAS 3.0 standard, and **MUST** use HTTPS protocol. All data structures **MUST** be defined using JSON Schema with modifications as defined in the OAS 3.0 standard. Each Open Air API document **MUST** be available in JSON format, and **MAY** in addition be available in YAML format.

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs. The Open Air Standard leverages OAS 3.0 and covers the usage of the OAS 3.0 objects and fields shown in Figure 1 below. This document does not intend to describe the usage of all required or optional OAS objects and related fields. Any API defined using the Open Air Standard **MAY** make use of any other objects and fields of the OAS 3.0 Standard, but any such usage **MUST NOT** affect the meaning or behaviour of the objects and fields covered by this standard.

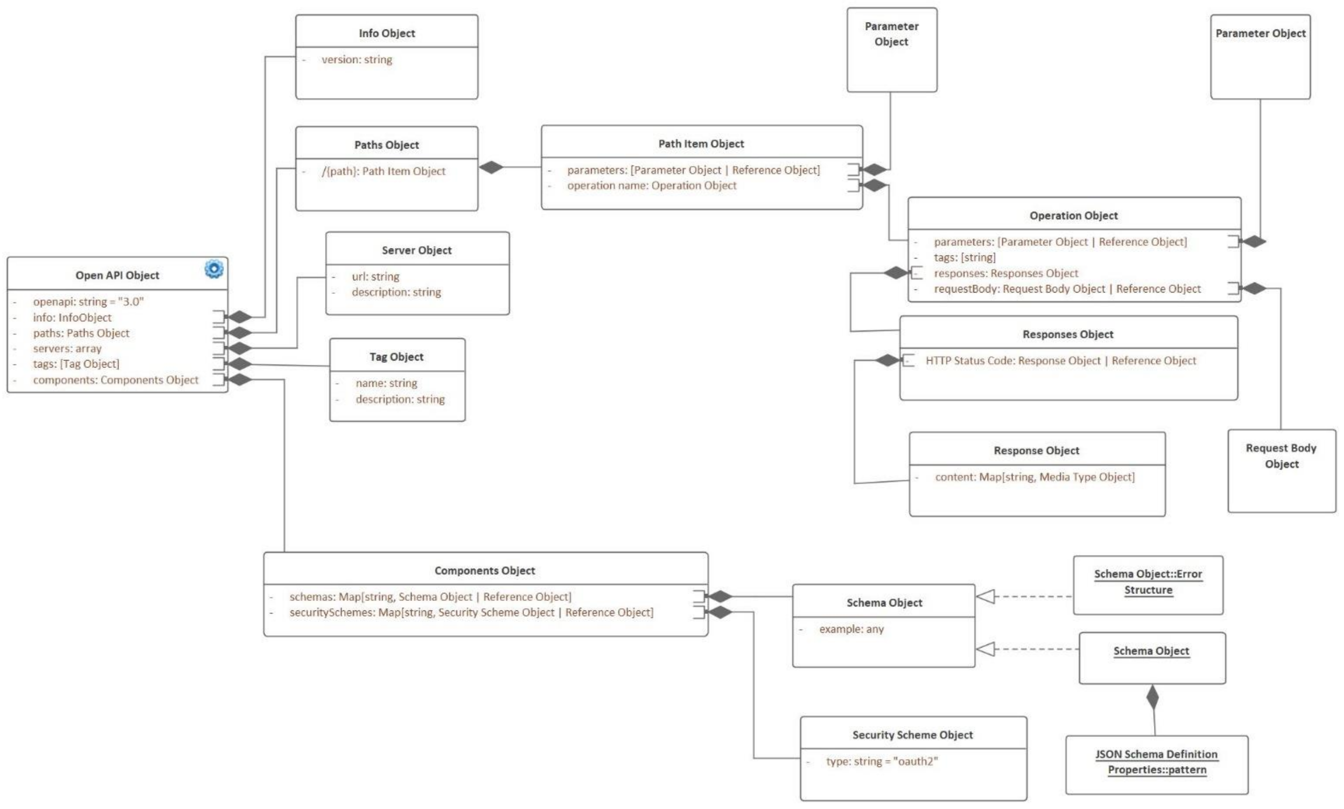

Info Object
- version: string
Paths Object
- /{path}: Path Item Object
Path Item Object
- parameters: [Parameter Object | Reference Object]
- operation name: Operation Object
Parameter Object
Parameter Object
Open API Object
- openapi: string = "3.0"
- info: InfoObject
- paths: Paths Object
- servers: array
- tags: [Tag Object]
- components: Components Object
Server Object
- url: string
- description: string
Tag Object
- name: string
- description: string
Operation Object
- parameters: [Parameter Object | Reference Object]
- tags: [string]
- responses: Responses Object
- requestBody: Request Body Object | Reference Object
Responses Object
- HTTP Status Code: Response Object | Reference Object
Response Object
- content: Map[string, Media Type Object]
Request Body Object
Components Object
- schemas: Map[string, Schema Object | Reference Object]
- securitySchemes: Map[string, Security Scheme Object | Reference Object]
Schema Object
- example: any
Schema Object::Error Structure
Schema Object
Security Scheme Object
- type: string = "oauth2"
JSON Schema Definition Properties::pattern

Figure 1 – Open API Standards Scope

2.4.1 OpenAPI Object

OpenAPI Object is the root document object of the OAS 3.0 specification.

In IATA standard API specification, the value of openapi attribute **MUST** be 3.0 or a minor version there-of. It means the API specification is documented using Open API Specification 3.0.

Example 1

"openapi": "3.0.2"

2.4.2 Info Object

Info Object provides metadata about the API.

Each specification (OAS) **MUST** have its own version. Version notation **MUST** follow Semantic Versioning 2.0.0. Section 3.2 describes the versioning best practice.

Example 2

```
"info": { "version": "1.0.1"
…
 }
```

2.4.3 URI

A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies a resource. URI Internet Standard Specification is defined in RFC3986.

As resource oriented design approach, it is critical to define a meaningful URI for the resource, in order to help developers to understand and use the RESTful API.

Figure 2 – URI components below is the example that shows the components in the URI.

```
  foo://example.com:8042/over/there?name=ferret#nose
  \_/   \______________/\_________/ \_________/ \__/
   |           |            |            |        |
scheme     authority       path        query   fragment
```

Figure 2 – URI components

The standard and best practice on URI is defined in 2.4.4 Server Object and 2.4.6 Paths Object.

Example 3 - URI in the API

HTTP POST
https://api.example.com/v1/inventory-management/managed-entities/{id}/install-script-location

URI component	Example	Related OAS 3 object/field
scheme	https	Server.url field
authority	api.example.com	Server.url field
path	/v1	Major version of API spec in either Server.url field or Paths Object
path	/inventory-management/managed-entities/{id}/install-script-location	Paths Object
query		Parameter Object

2.4.4 Server Object

Servers Object include an array of Server Objects. The Server Object provides connectivity information to a target server.

2.4.4.1 URL

To keep the consistency of URL formatting:

1. URL **SHOULD** follow URI specification RFC3986.
2. URL **SHOULD** define the scheme and authority components of URI specification.

3. All characters in URL **SHOULD** be in lowercase.
4. A hyphen (-) **MUST** be used in URL to separate multi-word phrases, except that the parameter name **MUST** follow camel case as naming convention as described in section 2.4.9 Parameter Object.

Example 4 – URL

http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable

http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation //Less readable

5. File extensions **MUST NOT** be included in the server URL.

   It does not add any value to use file extension and makes the URL longer. Instead of using file extensions, mime-type should be used to identify the type of data.

Description field **MUST** be defined for each Server Object structure.

Example 5 - Server Object

```
"servers": [{
   "url": "https://test.iata.org",
   "description": "User Acceptance Testing environment"
 }, {
   "url": "https://prod.iata.org",
   "description": "Production environment”
 } ]
```

2.4.5 Components Object

Holds a set of reusable objects for different aspects of the OAS. All objects defined within the components object will have no effect on the API unless they are explicitly referenced from properties outside the components object.

There are no variations to the OAS Standard defined for this object.

2.4.6 Paths Object

The Paths Object holds the relative paths to the individual endpoints and their operations. The path is appended to the URL from the Server Object to construct the full URL of the resource.

2.4.6.1 Resource Naming

In REST, primary data representation is called Resource. Generally, a resource is a thing not an action and is identified by a noun. HTTP Verbs **MUST** be used to define the action to be performed on the Resource.

Table 1 categories resources and defines the nature of the resource name for each category.

Resource Category	Name Style
Collection	Plural Noun

Document	Singular Noun or Unique Identifier
Controller (such as business process resource)	A controller resource models a procedural concept. Use "verb" to define a directive action to be performed by a Resource. *e.g. http://api.example.com/cart-management/users/{id}/cart/checkout*

Table 1 - Resource Naming Conventions

Figure 3 - Resource naming structure shows the general pattern of a Resource Name being:

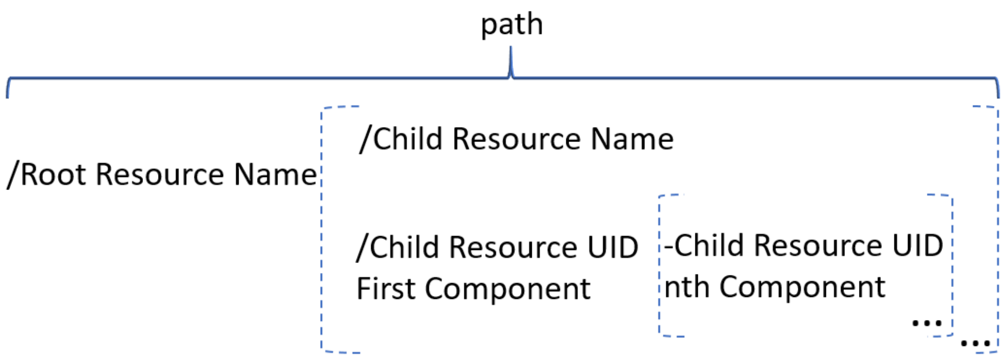

path
/Root Resource Name
/Child Resource Name
/Child Resource UID First Component
-Child Resource UID nth Component
...
...

Figure 3 - Resource naming structure

In order to keep the consistency in Resource Naming:

1. The Root and Child Resource Name **MUST** be the names of ABIEs or ASBIE Roles optionally preceded with a Status separated by a forward slash ("/").
2. Resource name **SHOULD** be identified by a noun, unless the archetype is Controller.
3. Resource name **MUST** be plural unless it is a singleton resource in which case a singular noun **MUST** be used.
4. The Child Resource UID **MUST** be the unique identifier components of the ABIE separated by a hyphen.

5. The hierarchical structure of a Resource Name **MUST** be constructed by traversing through the Integrated Data Model in the AIDM moving from an ABIE (Resource) to a child ABIE (Resource) via an ASBIE (Hierarchical Link).
6. Controller Resource name **MAY** be a verb, and **MAY** be a verb with a suffix, for example "-jobs", so that the resource could be accessed via proper UID, for example job ID.

2.4.7 Paths Item Object

Describes the operations available on a single path. A Path Item **MAY** be empty, due to ACL constraints. The path itself is still exposed to the documentation viewer but they will not know which operations and parameters are available.

There are no variations to the OAS Standard defined for this object.

2.4.8 Operation Object

Operation Object defines the HTTP methods can be used to access a path. A unique operation is a combination of a path and an HTTP method. A single path can support multiple operations.

OAS 3.0 supports HTTP methods of get, post, put, patch, delete, head, options, and trace. In Operation Objects, the specification **MUST** follow all HTTP methods definition and guidance in RFC7231.

2.4.9 Parameter Object

Parameter Object describes a single operation parameter. A unique parameter is defined by a combination of a name and location.

There are four possible parameter locations specified by the “in” field:

1. **path** - Used together with Path Templating, where the parameter value is actually part of the operation's URL. The path parameter is required in all cases to access the API.
2. **query** - Parameters that are appended to the URL. e.g /users?role=admin
3. **header** - Custom headers that are expected as part of the request. Refer to RFC7230 for more information
4. **cookie** - Used to pass a specific cookie value to the API. Refer to RFC6265 for more information

Path parameter **SHOULD** be used to identify a specific resource via UID, e.g get /users/{id}.

Query parameter **SHOULD** be used to filter or sort the sources.

The parameter name **MUST** follow camel case as naming convention.

2.4.10 Request Body Object

The Request Body Object describes a single request body, which is used to send data via REST API.

A Request Body **SHOULD** be used to send resource information, that is, content, in order to create or update the resource, in POST or PUT operations respectively.

2.4.11 Responses Object

A container for the expected responses of an operation. The container maps a HTTP response code to the expected response.

There are no variations to the OAS Standard defined for this object.

2.4.12 Response Object

Response Object describes an expected response of an operation. A response is defined by its HTTP status code and the data returned in the response body and/or headers.
In IATA standard API specification:

1. Response **MUST** be defined for HTTP status codes of 2xx (successful), 4xx (Client Error), and 5xx (Server Error). Refer to RFC7231 for the available status code and definition.
2. The media type “application/json” **MUST** be used by default.

2.4.13 Tag Object

The Tag Object adds metadata, including name and description, to a single tag which can be used for logical grouping of operations for the specific resource. In OpenAPI Object, the tags fields **MUST** be denoted to declare the list of Tag Object used in Operation Object in the specification.

In Operation Object, tags field contains a list of tag names of Tag Objects defined within OpenAPI Object. The operation.tags field **MUST** include a list of Airline Value Chain business capabilities key words (refer to Appendix 3.1 for more information).

For example, “Flight Status” API has “Communication Management” as Business Capability.

Example 6 – Tags

```
{
 "openapi": "3.0.0",
 "tags": [ {
  "name": "communication-management",
 }],
 "paths": {
  "/v1/flights": {
   "get": {
         "tags": ["communication-management"]
…
}
```

2.4.14 Schema Object

The Schema Object allows the definition of input and output data types. These types can be objects, but also primitives and arrays. This object is an extended subset of the JSON Schema published Draft 5 which includes the Core "draft-wright-json-schema-00" and the Validation "draft-wright-json-schema-validation-00". Unless stated otherwise, the property definitions follow this standard.

In IATA standard API specification:

1. If the data type is object, Schema object **MUST** be defined within Components Object, which can be referenced from other objects in the specification
2. If the data type is primitives, schema object **MAY** be defined in line with the parent structure
3. Regular expressions **SHOULD** be defined using the JSON Schema keyword "pattern"; for data validation purposes.

Example 7

```
"parameters": [ {
     "name": "agencyCode",
     "in": "path",
     "required": true,
     "schema": {
      "pattern": "^[0-9]{8}$|^[0-9]{7}$",
      "type": "string"
     }
    } ]
```

4. An Examples structure **MUST** be defined for all schema objects in the API specification.

2.4.14.1 Schema Definition

This section defines the standard for describing reusable data structures that appear as Schema Objects within the Components section.

The organization of data structures in this standard is similar to the Venetian Blind concept in that all object definitions are defined globally and may be reused by other objects in the Schemas section.

All data structures in IATA standard APIs **MUST** be derived from the AIDM, with the exception of Experimental Content.

Experimental Content is any schema data structure in an API specification, which has not been derived from the AIDM. Developers may want to extend the standard schema object to fulfill the business requirements in the proprietary API, so Experimental Content offers flexibility of extension. Experimental Content MUST NOT appear in any Conformant API specifications (Industry Standard Open API specifications published by IATA).

Table 2 defines how to represent the data element from AIDM in an OAS schema object. All the fields in OAS Schema template described in Table 2 are mandatory, except where it is indicated otherwise.

Figure 4 is the meta-model diagram for the elements in AIDM integrated data model.

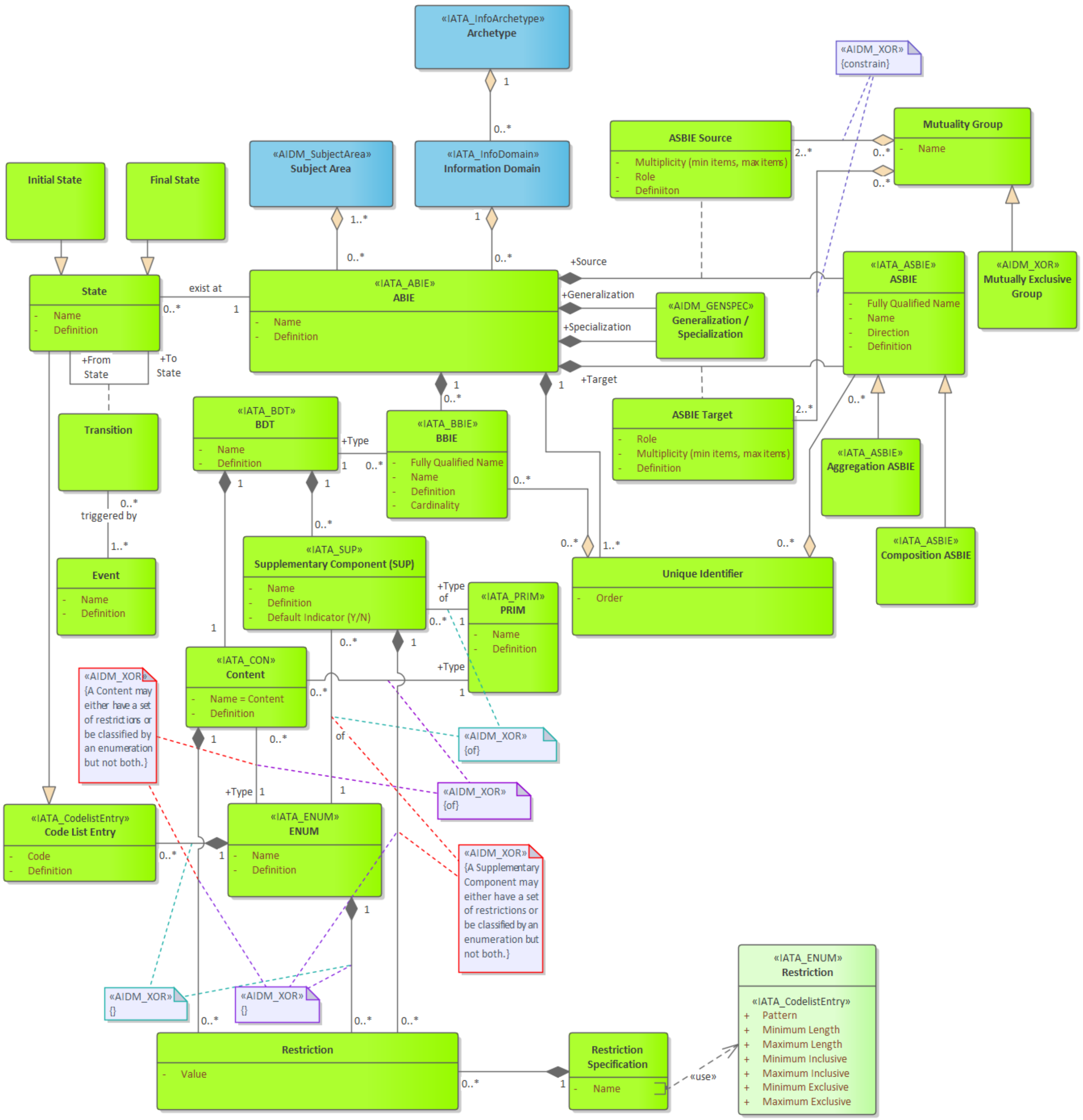


Figure 4 Integrated Model Meta-Model elements

Element Type	AIDM Derivation	OAS Schema Template

Object	ABIE	"<name>": {
		"title": "<Fully Qualified Name of data element>",
		"description": "<Description of data element>",
		"type": "object"
		}

Object	BDT with SUP with Default	"<name>": { "title": "<Fully Qualified Name of data element>", "description": "<Description of data element>",

	Indicator = “Y” exists	"type": "object" }
Object	BDT without SUP with Default Indicator = “Y” exists	"<name>": { "title": "<Fully Qualified Name of data element>", "description": "<Description of data element>", }
Object Mandatory Elements	BBIE SUP ASBIE XOR	"required": ["<mandatory element name>"…]
Property	BBIE	"properties": { "<name>": { "title": "<Fully Qualified Name of data element>", "description": "<Description of data element>", "$ref": "#/components/schemas/<name of BDT>" } }
Property	CON for a BDT where SUP with Default Indicator = “Y” exists	if the property is governed by a primitive "properties": { "value": { "title": "<Fully Qualified Name of data element>", "description": "<Description of data element>", "type": "<primitive name>", "format": "<format string>", *(optional)* "pattern": "<Pattern Tag Value>", *(optional)* "maxLength": "<Maximum Length Tag Value>", *(optional)* "minLength": "<Minimum Length Tag Value>", *(optional)* "maximum": "<Maximum Inclusive Tag Value>", *(optional)* "minimum": "<Minimum Inclusive Tag Value>" *(optional)* "exclusiveMaximum": "<Maximum Exclusive Tag Value>", *(optional)* "exclusiveMinimum": "<Minimum Exclusive Tag Value>" *(optional)* } } or if the property is governed by an enumeration "properties": { "value": { "title": "<Fully Qualified Name of data element>", "description": "<Description of data element>", "$ref": "#/components/schemas/<name of enumeration>" } }
Property	CON for a BDT where SUP with Default Indicator = “Y” does not exists	if the property is governed by a primitive "type": "<primitive name>", "format": "<format string>", *(optional)* "pattern": "<Pattern Tag Value>", *(optional)* "maxLength": "<Maximum Length Tag Value>", *(optional)* "minLength": "<Minimum Length Tag Value>", *(optional)* "maximum": "<Maximum Inclusive Tag Value>", *(optional)* "minimum": "<Minimum Inclusive Tag Value>" *(optional)* "exclusiveMaximum": "<Maximum Exclusive Tag Value>", *(optional)* "exclusiveMinimum": "<Minimum Exclusive Tag Value>" *(optional)* or if the property is governed by an enumeration "$ref": "#/components/schemas/<name of enumeration>"

Property	SUP with Default Indicator = “Y”	if the property is governed by a primitive
		"properties": {
		"<name>": {
		"title": "<Fully Qualified Name of data element>",
		"description": "<Description of data element>",
		"type": "<primitive name>",
		"format": "<format string>", *(optional)*
		"pattern": "<Pattern Tag Value>", *(optional)*
		"maxLength": "<Maximum Length Tag Value>", *(optional)*
		"minLength": "<Minimum Length Tag Value>", *(optional)*
		"maximum": "<Maximum Inclusive Tag Value>", *(optional)*
		"minimum": "<Minimum Inclusive Tag Value>" *(optional)*
		"exclusiveMaximum": "<Maximum Exclusive Tag Value>", *(optional)*
		"exclusiveMinimum": "<Minimum Exclusive Tag Value>" *(optional)*
		}
		}
		or if the property is governed by an enumeration
		"properties": {
		"<name>": {
		"title": "<Fully Qualified Name of data element>",
		"description": "<Description of data element>",
		"$ref": "#/components/schemas/<name of enumeration>"
		}
		}

Association [maximum cardinality> 1]	ASBIE	"< target or source role name | target or source ABIE name>": {
		"title": "<Fully Qualified Name of data element>",
		"Description": "<Description of data element>",
		"type": "array",
		"minitems": <minimum cardinality>,
		"maxitems": <maximum cardinality >,
		"items": {
		"$ref": "#/components/schemas/<name of referenced object>"
		}
		}

Association [maximum cardinality= 1]	ASBIE	"< target or source role name | target or source ABIE name>": {
		“title”: "<Fully Qualified Name of data element>",
		“description”: "<Description of data element>",
		"$ref": "#/components/schemas/<name of referenced object>"
		}

Enumerations	ENUM
	
	Code List Entry
		"<name>": {
		“title”: "<Fully Qualified Name of data element>",
		“description”: "<Description of data element>.
		Valid Values: <code list entry name n> - <Description of code list entry n>; …",
		"type": "<Restricted Primitive>",
		"enum": ["<code list entry name>"…]
		
		}
		or for open enumerations (that is, enumerations with no Code List Entry)
		"name": {
		“title”: "<Fully Qualified Name of data element>",
		“Description”: "<Description of data element>",
		"type": "<Restricted Primitive>",

		"pattern": "<Pattern Tag Value>" *(optional)* }

Association Mutual Exclusivity (Association [maximum cardinality> 1])	XOR	"<XOR Name>": {
		"oneOf": [
		{
		"title": "<Fully Qualified Name of data element>",
		"Description": "<Description of data element>",
		"type": "array",
		"minitems": <minimum cardinality>,
		"maxitems": <maximum cardinality >,
		"items": {
		"$ref": "#/components/schemas/<name of referenced object>"
		}
		}
		]
		}

Association Mutual Exclusivity (Association [maximum cardinality= 1])	XOR	"<XOR Name>": {
		"oneOf": [
		{
		"title": "<Fully Qualified Name of data element>",
		"description": "<Description of data element>",
		"$ref": "#/components/schemas/<name of referenced object>"
		}
		]
		}

Table 2 - Implementing AIDM Logical Elements

In addition, Table 3 defines how to construct the physical name of an element from the business name.

Source / PIM Element	Element Name*	JSON Schema Type
ABIE	ABIE Name	Object
BBIE	BBIE Name	Property
BDT	BDT Name	Object
CON where SUP with Default Indicator = "Y" exists	"Value"	Property
CON where SUP with Default Indicator = "Y" does not exist		Not Used
SUP where Default Indicator = "Y"	SUP Name	Property
SUP where Default Indicator = "N"		Not used
ENUM	ENUM Name + "Enum"	Object
Code List Entry	Code List Entry Name	Enum Array Element
ASBIE	If present, ASBIE Source or Target Role Name otherwise ASBIE Source or Target ABIE Name	Reference
PRIM	See Section 2.4.14.2 Primitive Data Types	Primitive

Table 3 - Nomenclature

* Note that Element Names **MUST** be in camel case except for Schema Object names which **MUST** be in Pascal case.

All element name MUST have AIDM abbreviations applied in order of length starting with the longest. For example, if there is an abbreviation of PO for Post Office and an abbreviation of PST for Post, if you apply PST abbreviation first, the outcome will be PST Office, whereas applying the longest phrase to be substituted will result in the outcome of PO; which is the desired result. In addition, the name of a structured SUP attributes must have the period punctuation mark (.), used as a grouping mechanism, removed.

An objective of this standard is the validation of the data content of instances of OAS Documents. Keywords, as well as data structure, are used to address this objective. Table 4 identifies the allowable keywords for validating the content of data and the cardinality of arrays.

Keyword	Derived from **Restriction Specification Name**
minlength	Minimum Length
maxlength	Maximum Length
minimum	Minimum Inclusive
maximum	Maximum Inclusive
exclusiveMinimum	Minimum Exclusive
exclusiveMaximum	Maximum Exclusive
pattern	Pattern
	Derived from **ASBIE Source or Target Multiplicity**
minitem	Min Items
maxitems	Max Items

Table 4 - Allowable Keywords for Validation

2.4.14.2 Primitive Data Types

In the AIDM a Content Component and a Supplementary Component may be classified by an enumeration or by a primitive data type. If an attribute is classified by an enumeration the Primitive Data Type can be found in a tagged value called “restrictedPrimitive”; otherwise the classifier is the primitive data type. In either case the primitive data type must be transformed into a platform dependent data type as defined in Table 5 below.

Reference to PRIM in AIDM	Reference to standard OAS data type	Format
AnyURI	string	uri
Binary	string	binary
Boolean	boolean	
DatePoint	string	date
Decimal	number	double
Double	number	double
Float	number	float
Integer	integer	int32
NormalizedString	string	
String	string	
TimeDuration	string	duration
TimePoint	string	date-time
TimeOfDay	string	hh:mm:ss
Token	string	byte

Table 5 - Implementation of Primitive Data Types

Example 8 - Schema object

```
"components": {
  "schemas": {
   "customer": {
     "type": "object",
     "properties": {
       "active": {
         "type": "boolean"
       },
       "code": {
         "pattern": "^[0-9]{8}$",
             "type": "string"
       },
     },
     "example": {
       "code": "98210019",
       "active": true
     }
   }}}
```

2.4.14.3 Traceability

No traceability from AIDM to API data elements is demanded. In future the keyword “$comments” may be used to describe the derivation of a data element from the AIDM. Including but not limited to the node path and name of the diagram used to generate the data structure and the date and time it was generated, and the navigation path and GUID of the source data element.

2.4.14.4 Error Structure

IATA standard API **SHOULD** use the below general error object structure within the Component Object to handle the general error message as part of an API response.

Field Name	Date Type	Description	Optionality
id	string	A unique identifier for this specific instance of the error.	Optional
status	string	The HTTP status code applicable to the error.	Mandatory
code	string	an application-specific error code.	Optional
title	string	A short, human-readable summary of the problem that **SHOULD NOT** change from occurrence to occurrence of the error, except for purposes of localization.	Optional
language	string	The code of the language used in the error message. Not required when the language is a variant of English.	Optional
detail	string	a human-readable explanation specific to this occurrence of the issue.	Optional

url	string	A link to an on-line description of the error where one **COULD** find statements pertaining to the consequences of the error and indications as to actions that might be taken and actions that should or must not be taken.	Optional

Table 6 - Components of the Standard Error Structure

Example 9 - Error Structure

```
"components": {
  "schemas": {
   "error": {
    "type": "object",
    "required": ["status"],
      "properties": {
        "id": {"type": "string"},
        "status": {"type": "string"},
        "code": {"type": "string"},
        "title": {"type": "string"},
        "language": {"type": "string"},
        "detail": {"type": "string"},
        "url": {"type": "string"}
     }
    }
   "errors" : {
    "type" : "object",
    "properties" : {
     "errors" : {
      "type" : "array",
      "items" : {
       "$ref" : "#/components/schemas/error"
      }
     }
    }
   }
   }
}
...
   "404": {
    "content": {
     "application/json": {
      "schema": {
       "type": "object",
       "items": {
         "$ref": "#/components/schemas/errors"
       }}}}}
```

2.4.15 Security Scheme Object

The Security Scheme Object defines a security scheme that can be used by the operations.
In IATA standard API specification:

1. APIs **SHOULD** apply OAuth 2.0 as security mechanism.
2. APIs **SHOULD** apply OWASP best practices. Additional information can be found at www.owasp.org.

3. Appendix

3.1. Airline Value Chain

Figure 5 below shows a chain of primary activities and their process areas that a firm is operating in the airline industry performs in order to deliver a valuable product or service to the market.

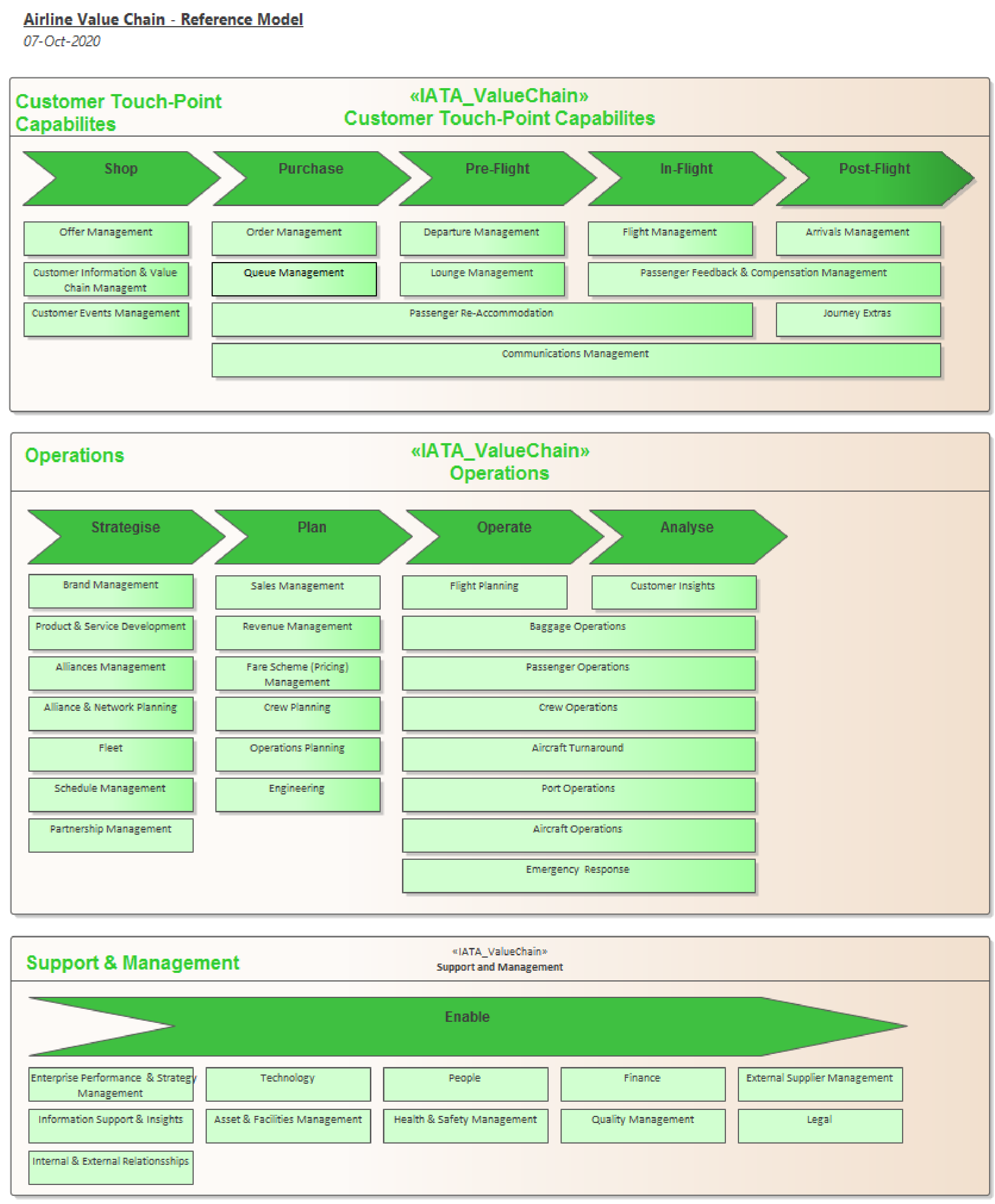


Figure 5 - Airline Value Chain Reference Model

Customer Touch-Point Capabilities

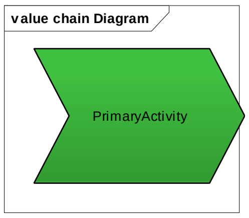
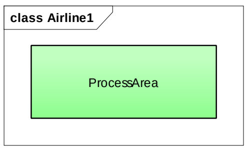
Business capabilities needed to deliver the core product to the customer. Organized by the lifecycle of product delivery.

Operations

Business capabilities needed to enable the running of “passenger airline” business. Organized by typical business planning cycle.

Support & Management
Business capabilities needed to enable the running of “any” business.

3.1.1 Artifacts

Artifact	Description
Business Primary Activity 	Entry point of the Business Activities
Business Support Capability 	It describes Business Processes capability details for a specific business purpose.

3.1.2 Business capability Mapping

The provider should identify the relevant Business Support Capabilities of the API, and specify those capabilities in the Tag Object of the OAS specification. It helps the API consumers to better understand the business purpose and benefits of the API.

3.2. Versioning

This main purpose of this section is to describe the API versioning guideline and best practice. To cover the basic context, it also gives an overview of IATA release, deliverables in the release package and versioning guidelines of each artifact.

3.2.1 Release Overview

An IATA Release is the fact to package and publish IATA Standards deliverables at the planned schedule. For example, standard XML schemas generated from the AIDM; OpenAPI specification of standard APIs; export of complete AIDM model are grouped in one package, after ATSB adoption, etc.

The release number identifies each IATA Release. The format of release number is “IATAYYYY.S”, with “YYYY” as the year and “S” as release sequence within the year. Example of release number: IATA2018.1 - the first Release in year 2018, called also 18.1.

Figure 6 Release Overview describe the present and planned deliverables within an IATA release package, the version example of each deliverable, and where the versioning guideline for that deliverable is defined.

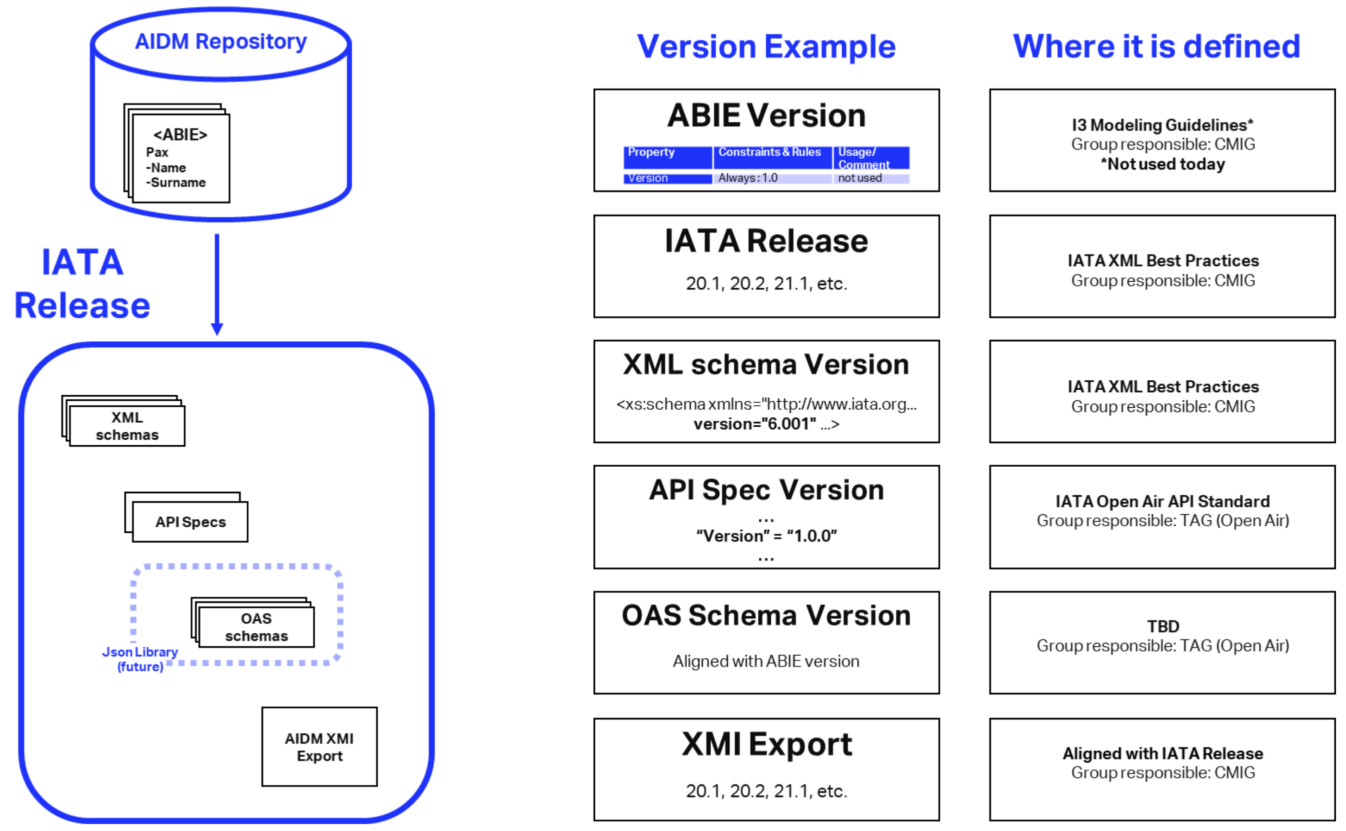


Figure 6 Release Overview

3.2.2 Release Deliverable Versioning

3.2.2.1 AIDM entity (ABIE) version

There is version property of ABIE (Aggregated Business Information Entity) in AIDM data model. The ABIE version property is not used as of today, and it is always set as 1.0 by default. Refer to "5.3 ABIE properties" in "AIDM Guidelines - I3 Info Logical Model v2.2.docx" as part of Ref 7 - AIDM Guidelines.

Open Air group recommends to defining ABIE version in AIDM, to track the evolution of the entities.

3.2.2.2 XML Schema version

Each standard XML schema has its own version defined. The guideline of XML schema versioning is defined in the section "11 Versioning and schema identification" of "IATA XML Best Practice Document ver1.6.1.doc" as part of Ref 7 - AIDM Guidelines.

3.2.2.3 API versioning guideline

Each Standard API **MUST** has its own version defined in the OAS spec, to track the evolution of the API.

The major version of API **MUST** be included in either Server.url field or Paths Object in the OAS spec. The major version **MUST** be consistent for all endpoints in one API spec.

The developer **SHOULD NOT** use different versions of same JSON schema in one API spec.

Table 7 - API versioning describes the versioning guideline, focused on evolution of major and minor version (as defined in Semantic Versioning 2.0.0) in the OAS spec. An API is Backwards Compatible when a client program written to consume the last version of that API will continue to work the same way against its current version.

Changes in OAS object	Evolution	Info.version (M)ajor.(m)inor.(p)atch	Major version in Paths Object or Server.url
Response body	Backward Compatible	v M.{m+1}.p	/vM
	Non-Backward Compatible	v {M+1}.0	/v{M+1}
Path Parameter	Backward Compatible	v M.{m+1}	/vM
	Non-Backward Compatible	v {M+1).0	/v{M+1}
Query parameter	Backward Compatible	v M.{m+1}	/vM
	Non-Backward Compatible	v {M+1}.0	/v{M+1}
Request Body	Backward Compatible	v M.{m+1}	/vM
	Non-Backward Compatible	v {M+1).0	/v{M+1}

Table 7 - API versioning

Example 10 – Backward compatible API

1. Add a path item, or response
2. Add a path parameter
3. Add an optional query parameter
4. Resource model changes
   a) Adding an optional field
   b) Changing Mandatory field to Optional field
   c) Changing a field pattern in less restrictive way

Example 11 – Non-backward compatible API

1. Add a path parameter
2. Delete path, required parameters or operation
3. Rename parameter/path, which means adding a new one and deleting the old one at the same time
4. Add a constraint on a query parameter (like isRequired)
5. Resource model changes
   a) Adding a mandatory field
   b) Changing Optional field to Mandatory field
   c) Changing a field pattern in more restrictive way

3.2.2.4 XMI Export

The version XMI export of the complete AIDM model is aligned with the IATA release number.

3.2.2.5 OAS Schema version

Open Air group propose to transform the standard entities from AIDM integrated data model and generate the entities in Schema Object format as defined in Ref 1 - Open API 3.0 Specification. Schema object of OpenAPI 3.0 is an extended subset of JSON Schema Specification Wright Draft 00.

Open Air JSON common library, which includes all the OAS schema objects generated from AIDM, plans to be published in the future IATA release. The library will be open for the industry to reuse during in API design phase and possibly in the development phase as well, for the purpose of standardization to achieve Open Air certification, or just for API provider to build the proprietary APIs.

The OAS Schema version **SHOULD** be aligned with the version of corresponding ABIE in AIDM.

4. Glossary

Term	Meaning
Camel Case	A method of creating a label from a word, acronym, or phrase by capitalizing the first letter of all words or acronyms except the first and removing all white spaces and hyphens. All other letters must be in lower case.
AIDM	Airline Industry Data Model
ABIE - Aggregated Business Information Entity	An ABIE is a collection of related pieces of information in AIDM that together convey a distinct meaning. An ABIE is the representation of an entity/object class, contains attributes/properties, and may participate in associations with other ABIEs.
BBIE - Basic Business Information Entities	A BBIE represents an attribute of an ABIE.
ASBIE - Association Business Information Entity	An ASBIE defines an association between one ABIE (the "associating" ABIE) and another ABIE (the "associated" ABIE). ASBIEs are UML associations of AggregationKind either "shared" or "composite".
BDT - Business Data Type	A business data type defines the value domain – set of valid values – that can be used for a particular BBIE. It represents a complex element, as a BDT has one content component and any number of supplementary components.
PRIM - Primitive Data Type	A PRIM represents basic building blocks for defining value domains of content and supplementary components. UN/CEFACT has defined a finite set of PRIMs. A PRIM may have a set of facets restricting the value domain.
ENUM – Enumeration Type	An ENUM is a collection of items that is a complete, ordered listing of all of the items in that collection.

5. References

Name	Location
Ref 1 - Open API 3.0 Specification	https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md
Ref 2 - JSONAPI Specification and Best Practice	https://jsonapi.org/
Ref 3 - REST Resource Naming Guide	https://restfulapi.net/resource-naming/
Ref 4 - OAuth2 specification	https://oauth.net/2/
Ref 5 - Swagger.io	https://swagger.io/docs/specification/about/
Ref 6 - JSON Schema	https://json-schema.org/
Ref 7 - AIDM Guidelines	https://guides.developer.iata.org/docs/aidm-guidelines