IATA Open Air API Standards and Best Practices

Version 1.0

May 2020

Contents

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

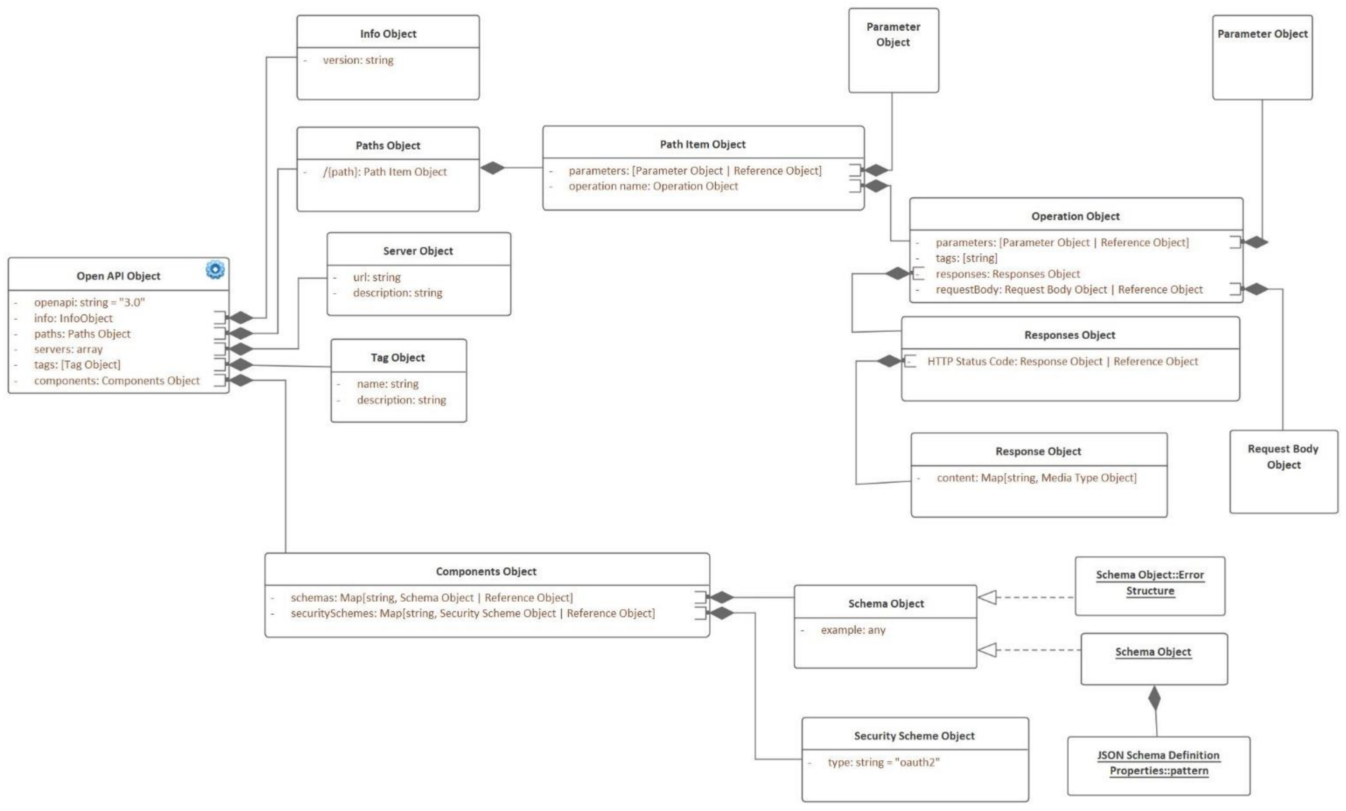
When describing the Best Practices and the Checklist, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2.3. 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
Parameter Object
Parameter Object
Paths Object
- /{path}: Path Item Object
Path Item Object
- parameters: [Parameter Object | Reference Object]
- operation name: Operation Object
Operation Object
- parameters: [Parameter Object | Reference Object]
- tags: [string]
- responses: Responses Object
- requestBody: Request Body Object | Reference Object
Server Object
- url: string
- description: string
Open API Object
- openapi: string = "3.0"
- info: InfoObject
- paths: Paths Object
- servers: array
- tags: [Tag Object]
- components: Components Object
Tag Object
- name: string
- description: string
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

2.3.1 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:
"openapi": "3.0.2"

2.3.2 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.

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

2.3.3 Server Object

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

2.3.3.1 URL

In order to make sure the consistency of url formatting and better readability:

1. All characters in url **SHOULD** be in lowercase.
2. A hyphen (-) **MUST** be used in url to separate multi-word phrases.

Examples:
```
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
```

3. 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.

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

2.3.4 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.

2.3.5 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 in order to construct the full URL of the resource.

2.3.5.1 Resource Naming

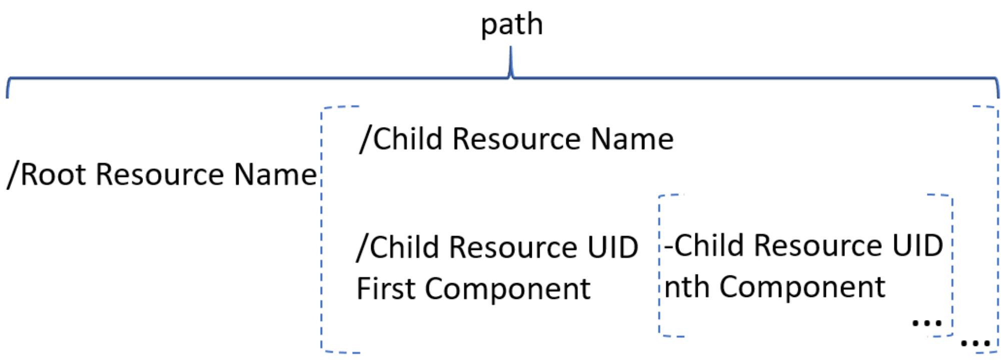
In REST, primary data representation is called Resource. Generally, a resource is a thing not an action and is identified by a noun. HTTP Verbs **MUST** be used to define the action to be performed on the Resource.

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

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
...
...

For example: "/customers/12345" for the URN "/customers/{customerId}".

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 ("/"). Resource name **SHOULD** be identified by a noun, unless the archetype is Controller. Resource name **MUST** be plural unless it is a singleton resource in which case a singular noun **MUST** be used.

The Child Resource UID **MUST** be the unique identifier components of the ABIE separated by a hyphen.
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).

2.3.6 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.

2.3.7 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.

2.3.8 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.

2.3.9 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.

2.3.10 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.

2.3.11 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.

2.3.12 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:

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

2.3.13 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 Specification Wright Draft 00". For more information about the properties, see "JSON Schema Core" and "JSON Schema Validation". Unless stated otherwise, the property definitions follow the JSON Schema.

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:

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

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

2.3.13.1 Schema Definition

This section defines the standard for describing shareable data structures that appear as Schema Objects in the Schemas section, 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.

As all data structures **MUST** be derived from an AIDM Logical Data Model. Table 2 defines how to represent that data element in an API schema. In addition, Table 3 defines how to construct the physical name of an element from the business name.

Element Type	AIDM Derivation	API Schema Template

Object	ABIE
	BDT
		"name": {
		      "type": "object"
		}

Object Mandatory Elements	BBIE CON SUP	"required": ["<mandatory element name>"...]

Property	BBIE
	CON
	SUP
		"properties": {
		      "<name>": {
		      },
		}

Association | Classification	ASBIE	"<role name | target ABIE name>": {
		 "type": "array",
		 "minitems": <minimum cardinality>,
		 "maxitems": <maximum cardinality >,
		 "items": {
		 "$ref": "#/components/schemas/<name of referenced object>"
		 }
		}

Enumerations	ENUM
	
	CodelistEntry
		"name": {
		      "type": "array",
		"items": {
		      "type": "<Restricted Primitive>",
		      "enum": ["<code list entry name>"...]
		 }
		}

		or for open enumerations (that is, enumerations with no CodelistEntry)
		"name": {
		"type": "<Restricted Primitive>",
		"pattern": "<Pattern Tag Value>"
		}

Primitives		{
		"type": "<primitive name>",
		"format": "<format string>"
		}

Association Mutual Exclusivity	XOR	"OrderOwner": {
		"oneOf": [
		{"$ref": "#/components/schemas/< objectA>"},
		{"$ref": "#/components/schemas//<objectB>"}
		]
		}

Table 2 - Implementing AIDM Logical Elements

Source / PIM Model Element	Element Name*	JSON Schema Type
ABIE	ABIE Name	Object
BBIE	BBIE Name	Property
BDT (with restriction of SUPS)	BDT Name	Object
BDT (without restriction of SUPS)	BDT Name	Property Type
CON	“Value”	Property
SUP	SUP Name	Property
ENUM	ENUM Name	Object
CodelistEntry	CodelistEntry Name	Enum Array Element
ASBIE	If present use the Source Role Name otherwise use the Source ABIE Name	Ref
PRIM	See Section 2.3.13.2	Primitive

Table 3 - Nomenclature

* Note that all Element Names **MUST** be in camel case and 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.

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	Derivation
minlength	BBIE, CON, SUP Tagged value “minimumLength”
maxlength	BBIE, CON, SUP Tagged value “maximumLength”
minimum	BBIE, CON, SUP Tagged value “minimumInclusive”
maximum	BBIE, CON, SUP Tagged value “maximumInclusive”
exclusiveMinimum	BBIE, CON, SUP Tagged value “minimumExclusive”

exclusiveMaximum	BBIE, CON, SUP Tagged value "maximumExclusive"
pattern	BBIE, CON, SUP, ENUM Tagged value "pattern"
minitem	ASBIE Lower Value of Source Cardinality
maxitems	ASBIE Higher Value of Source Cardinality

Table 4 - Allowable Keywords for Validation

2.3.13.2 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
Token	string	byte

Table 5 - Implementation of Primitive Data Types

Schema Object Example:

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

2.3.13.3 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.

2.3.13.4 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

Error Structure Example:

```
"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"
            }}}}}
```

2.3.14 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 2 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.

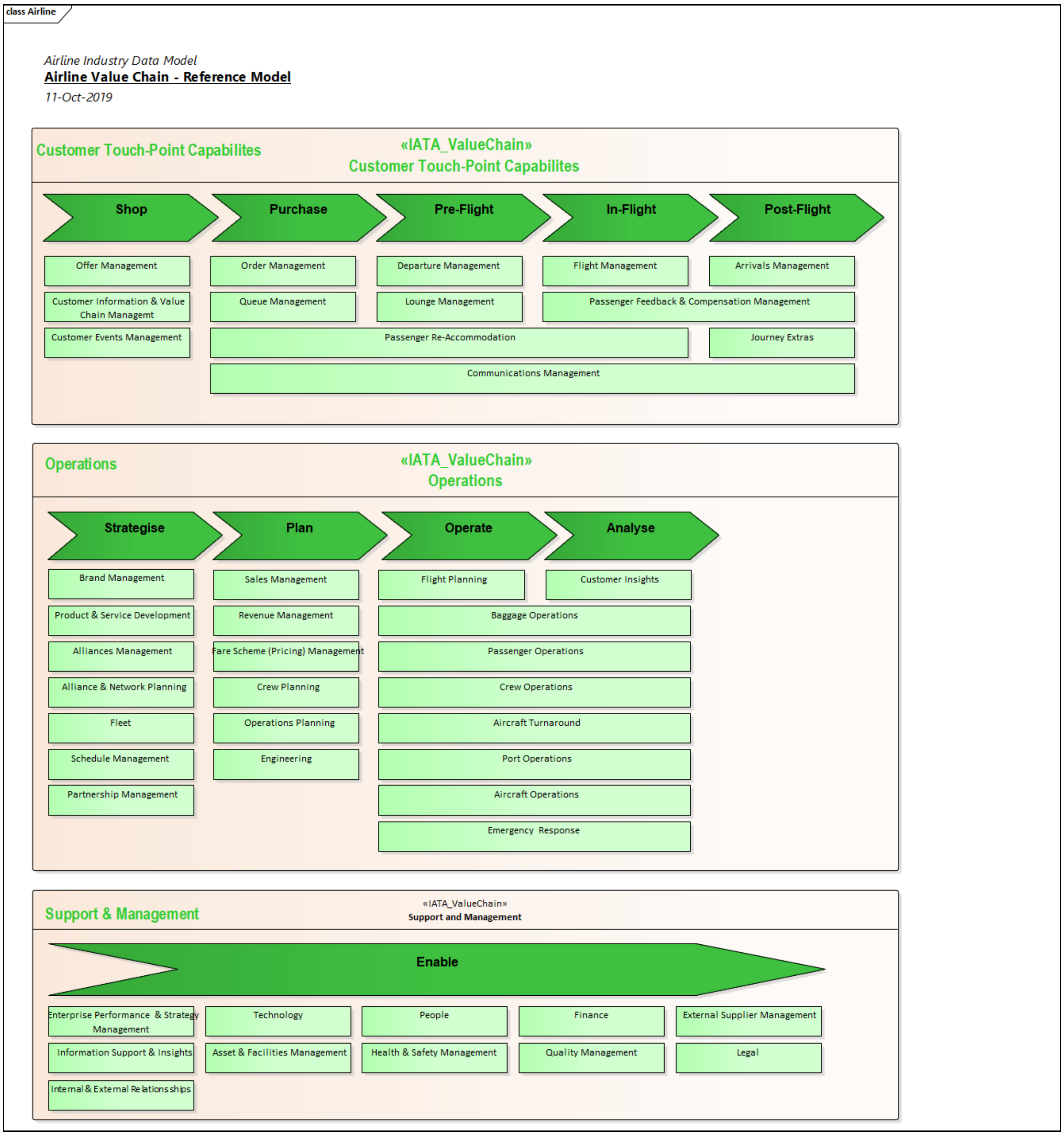


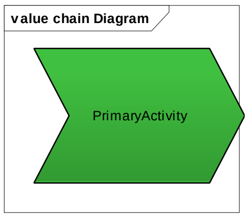
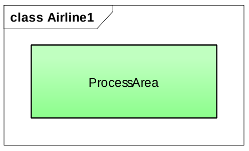
Figure 2 - Airline Value Chain Reference Model

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 and Properties

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 main Business Support Capabilities supported by the API should be identified by the provider. This will enable the API consumer to identity which areas of their business are most likely to benefit from the API.

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

Open API Specification 3.0.2	https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md
JSONAPI Specification and Best Practice	https://jsonapi.org/
REST Resource Naming Guide	https://restfulapi.net/resource-naming/
OAuth2 specification	https://oauth.net/2/
Swagger.io	https://swagger.io/docs/specification/about/
JSON Schema	https://json-schema.org/
AIDM Guidelines	https://airtechzone.iata.org/docs/AIDM%20Guidelines.zip