




# DCSA API Design Principles 1.1

September 2021

# Table of contents



Change history	4
Terms, acronyms, and abbreviations	4
1 Introduction	5
1.1 Purpose and scope	5
1.2 API characteristics	5
1.3 Conventions	6
2 Suitable	6
3 Maintainable	6
3.1 JSON	6
3.2 URLs	6
3.3 Collections	6
3.4 Sorting	6
3.5 Pagination	7
3.6 Property names	8
3.7 Enum values	9
3.8 Arrays	9
3.9 Date and Time properties	9
3.10 UTF-8	9
3.11 Query parameters	10
3.12 Custom headers	10
3.13 Binary data	11
3.14 Error handling	11
3.15 Subscription model	13
4 Stable	13
4.1 Versioning	13
4.2 Backward compatibility	13

4.3 Deprecation	14
5 Secure	14
5.1 Endpoints	15
6 Performant	15
7 Well-documented	15
7.1 General documentation	15
7.2 HATEOAS	15

## Tables

Table 1: API quality characteristics	5
Table 2: Examples of sorting	7
Table 3: Pagination	8
Table 4: Examples of time format	9
Table 5: Query parameters naming conventions	10
Table 6: Custom headers Binary data	11

## Change history

Rev	Issue	Contributors	Description
1.0	September 2020	DCSA	First release
1.1	September 2021	DCSA	Subscription API

## Terms, acronyms, and abbreviations

Term	Definition
API	Application programming interface
camelCase	In this document camelCase is defined as <b>lower camel case</b> . Definition of lower camel case is that initial letter is lower and every subsequent word is Capitalized.
CRUD	Create, Read, Update and Delete (equivalent to POST, GET, PUT/PATCH and DELETE rest operations)
FK	Foreign Key
HATEOAS	Hypermedia as the Engine of Application State
kebab-case	kebab-case combines words by replacing each space with a dash (-). All letters are lower case. For example, this-is-a-kebab-case-example.
PK	Primary Key
UUID	Universally unique identifier

## 1 Introduction

This chapter describes the purpose, structure and conventions used in this document.

### 1.1 Purpose and scope

The purpose of this document is to establish guidelines on how APIs should be designed, developed and implemented in the digital container shipping context. The guidelines describe the requirements that DCSA-compliant APIs should fulfil.

### 1.2 API characteristics

As described in the table below, a good API should have the following quality characteristics or have a balance of these quality characteristics.

Index	Characteristics	Descriptions
1	Suitable	APIs provide a well-defined service that optimizes value for consumers by having clearly defined responsibilities and an appropriate level of abstraction.
2	Maintainable	It is easy to understand how to interact with the API and development of maintainable code is facilitated.
3	Stable	The occurrence and impact of major changes are minimized.
4	Secure	Measures and practices are implemented to prevent abuse.
5	Performant	The performance consequences of API design are considered, monitored and optimized for common interactions.
6	Well-documented	Up-to-date, complete and easy-to-read documentation is available to facilitate adoption.

Table 1: API quality characteristics

The structure of this documents follows the order of the API quality characteristics mentioned in the table above.

### 1.3 Conventions

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](https://www.ietf.org/rfc/rfc2119.txt) (<https://www.ietf.org/rfc/rfc2119.txt>).

## 2 Suitable

Good APIs are suitable regarding their functionality and responsibilities. Suitable APIs provide a well-defined service that optimizes value for consumers by having clearly defined responsibilities and an appropriate level of abstraction.

To achieve this objective, two requirements are identified below:

- The API is constructed with the consumer in mind and **SHOULD** be based on User Stories.
- The API is constructed according to a minimalistic approach – less is more.

## 3 Maintainable

APIs need to be maintainable. This means it should be easy to understand how to interact with the APIs, and development of maintainable code should be facilitated. The requirements to achieve good maintainability are described in this chapter.

### 3.1 JSON

Requests and responses **MUST** be in [The application/json Media Type for JavaScript Object Notation \(JSON\)](#).

Requests and responses **MAY** be in other formats as well – e.g. [XML Media Types](#).

### 3.2 URLs

**MUST** point to a resource (entity, process).

**MUST** consist of nouns and **MUST NOT** be actions. HTTP verbs (GET, PUT, PATCH, POST, DELETE) **MUST** be used for actions.

"/" **MUST** be used to indicate hierarchical relations.

Kebab-case **MUST** be used in URLs. Path parameters **MUST** be consistent with property names. Path parameters referring to property names **MUST** use camelCase.

Query parameters **MUST** use camelCase.

### 3.3 Collections

Collection items **MUST NOT** consist of composite keys. A unique key **MUST** identify a collection element. Collections **SHOULD** be pluralized.

### 3.4 Sorting

Sorting **SHOULD** be limited to specific fields. The parameter **MUST** be restricted to the values ASC (ascending order) and DESC (descending order) describing the sort direction. The direction **MAY** be omitted. If omitted ascending order **MUST** be used. A colon : is used to separate the field to sort by and the direction. Multiple fields can be used, in which case each field is separated by a comma,. The following table shows some examples:

Query Parameter	Explanation
sort=name	Sort by the name field in ascending (default) order.
sort=name:desc	Sort by the name field in descending order.
sort=name:asc	Sort by the name field in ascending order.
sort=name,age:desc	Sort first by the name field in ascending order. If two equal names exist, then sort those names by age in descending order.

Table 2: Examples of sorting

The sort parameter is optional and if not specified, the order is set by the implementer.

It is up to the implementer to find a suitable Collator to use when sorting on String values.

### 3.5 Pagination

GET requests on collection results **SHOULD** implement pagination. Links to current, previous, next, first and last page **SHOULD** be available in the response headers; more links **MAY** be present.

Link	Can Be Null	Explanation	Example
First-Page	Yes	A link to the first page. First-Page header link <b>MAY</b> be omitted if current page is the first page.	<https://api.dcsa.org/events?limit=20&cursor=xxx>; rel="First-Page"
Previous-Page	Yes	A link to the previous page. Previous-Page header link <b>MAY</b> be omitted if the current page is the first page.	<https://api.dcsa.org/events?limit=20&cursor=xxx>; rel="Previous-Page"

Link	Can Be Null	Explanation	Example
Next-Page	Yes	A link to the next page. Next-Page header link <b>MAY</b> be omitted if the current page is the last page.	<code>&lt;https://api.dcsa.org/events?limit=20&amp;cursor=xxx&gt;;rel="Next-Page"</code>
Last-Page	Yes	A link to the last page. Last-Page header link <b>MAY</b> be omitted if the current page is the last page.	<code>&lt;https://api.dcsa.org/events?limit=20&amp;cursor=xxx&gt;;rel="Last-Page"</code>
Current-Page	No	A link to the current page.	<code>&lt;https://api.dcsa.org/events?limit=20&amp;cursor=xxx&gt;;rel="Current-Page"</code>

Table 3: Pagination

The default page size **SHOULD** be 100, if it is not specified on the endpoint. The default page size **MAY** be changed per endpoint. If the response payload is large, the page size **SHOULD** be changed to a smaller, more suitable number. The consumer **SHOULD** be able to override the default page size via the limit-parameter.

Keyset-based pagination **MUST** be used. The server generates all links to pages relative to the current page requested. It is not possible for the consumer to request a specific page – the consumer **MUST** select a page provided by the server. The server **MUST** provide links to the previous and next page (if possible) and **SHOULD** provide links to first and last page.

If the filter and/or sorting used when getting a collection result is changed, the pagination **MUST** be reset to first page.

It is important to note that the pagination represents a cut of the data at a specific moment in time. If somebody else triggers the same search a minute later, they might get a different data set for the same filtering/sorting criteria.

### 3.6 Property names

Property names on objects **MUST** be in camelCase.

Property names containing arrays **SHOULD** be plural.

Property names **MUST NOT** include FK (Foreign Key) or PK (Primary Key), as this exposes database design.

Boolean properties **MUST** be prefixed by either **is** or **has**.



### 3.7 Enum values

Enum values **SHOULD** be declared using UPPER\_SNAKE\_CASE.

### 3.8 Arrays

Empty arrays **MUST NOT** be represented with null – but **MUST** be empty lists [].

### 3.9 Date and Time properties

Date properties **MUST** be suffixed with “Date”. A Date property only contains a date in YYYY-MM-DD format.

Time properties **MUST** be suffixed with “Time”. A Time property only contains a specific time of a day in 24-hour-format using the following format: hh:mm. Time format **MAY** include seconds in which case the following is added :ss.

DateTime properties **MUST** be suffixed with “DateTime”. A DateTime property contains both a date and time.

Date, Time and DateTime **MUST** be presented using the ISO 8601 format.

Example	Explanation
2020-07-13	13 <sup>th</sup> of July 2020
The following 5 timestamps represent the same time period in different Time Zones:	
2020-07-05T02:15-04:00	5 <sup>th</sup> of July 2020, 02:15 in New York (EDT Eastern Daylight Time)
2020-07-05T08:15+02:00	5 <sup>th</sup> of July 2020, 08.15 in Rotterdam, Netherlands (CEST Central European Summer Time)
2020-07-05T11:45+05:30	5 <sup>th</sup> of July 2020, 11.45 in New Delhi, India (IST India Standard Time)
2020-07-05T14:15+08:00	5 <sup>th</sup> of July 2020, 14.15 Shanghai, China (CST China Standard Time)
2020-07-04T23:15-07:00	4 <sup>th</sup> of July 2020, 23.15 San Francisco (PDT Pacific Daylight Time)

Table 4: Examples of time format

### 3.10 UTF-8

Encoding **MUST** be UTF-8.

### 3.11 Query parameters

The following table shows naming conventions to be used:

Query Parameter	Description	Example
limit	Limits the number of objects returned in a collection-response. In responses that contain a lot of data per object, this might be lower.  Any number > 0 can be used.  <i>Default is 100 (unless otherwise specified).</i>	limit=20
cursor	A server-generated pointer to a page in a collection-response. This is never generated by the consumer.  In collection-responses, predefined header links to next and previous page <b>MUST</b> be added and links to first and last <b>SHOULD</b> be added. These links are populated with a <i>cursor</i> query parameter. The <i>cursor</i> <b>MUST NOT</b> be modified by the consumer.  If no <i>cursor</i> is set – then first page is returned.	cursor=ABC123
sort	A comma-separated list of field names to define the sort order. Field names <b>SHOULD</b> be suffixed by a colon (:) followed by either the keyword ASC (for ascending order) or DESC (for descending order) to specify direction. :ASC <b>MAY</b> be omitted, in which case ascending order will be used.	sort=name sort=name:ASC sort=name:DESC

Table 5: Query parameters naming conventions

### 3.12 Custom headers

Custom headers **MUST NOT** use the prefix *x-*. Please see [Deprecating the "X-" Prefix and Similar Constructs in Application Protocols](#).

The following table contains the agreed DCSA header:

Header	Description
API-Version	<p><i>In requests:</i> used to specify the version of the contract (API version) requested by the consumer.</p> <p><i>In responses:</i> used to indicate the version of the contract (API version) returned by the implementer.</p>

Table 6: Custom headers Binary data

### 3.13 Binary data

When the payload is only binary data, it **MUST NOT** be encoded. The format returned will be made available during content negotiation.

When binary data is part of the payload (in a property), it **MUST** be Base64 encoded.

### 3.14 Error handling

Standard error codes **MUST** be used. The table below extends the standard http error codes with extra information. The extended error codes **SHOULD** be used together with standard error codes for easier bug finding during development and maintenance. This is not an exhaustive list; other codes may be used in implementations.

HTTP code	Error code	Error message
400 – <i>Bad Request</i>	invalidParameter	An invalid parameter or parameter value was supplied in the request.
400 – <i>Bad Request</i>	missingParameter	The API request is missing a required parameter.
400 – <i>Bad Request</i>	invalidQuery	The request query was invalid. Check the documentation to ensure that the supplied parameters are supported, and check if the request contains an invalid combination of parameters, or an invalid value.
400 – <i>Bad Request</i>	invalidUri	The requested URI is not represented on the server.

HTTP code	Error code	Error message
<i>401 – Unauthorized</i>	missingCredentials	The user is not authorized to make the request.
<i>401 – Unauthorized</i>	invalidCredentials	The supplied authorization credentials for the request are invalid.
<i>401 – Unauthorized</i>	expiredAccessToken	The supplied Access Token has expired.
<i>403 – Forbidden</i>	accessDenied	The requested operation is forbidden.
<i>403 – Forbidden</i>	insufficientPermissions	The authenticated user is not permitted to execute this request.
<i>404 – Not found</i>	notFound	The requested resource could not be found.  404 Not found <b>MAY</b> be used to mask sensitive information for both 400, 401 and 403.
<i>405 – Method not allowed</i>	httpMethodNotAllowed	The HTTP method for the request is not supported.
<i>409 - Conflict</i>	ResourceAlreadyExists	The resource already exists.  Used when a “Unique constrain violation” occurs.
<i>429 – Too many requests</i>	rateLimitExceeded	Too many requests have been sent recently.  A Retry-After header <b>SHOULD</b> be added to notify consumer when to try again.
<i>500 – Internal server error</i>	internalError	The request failed due to an internal error.

HTTP code	Error code	Error message
503 – Service Unavailable		Used for maintenance purposes  A Retry-After header <b>SHOULD</b> be added to notify consumer when to try again.

Table 6: Error handling

404 Not Found **MAY** be returned instead of 400 Bad Request, 401 Unauthorized or 403 Forbidden for sensitive information. This can be used when the implementer of the API does not want to verify if restricted information is available.

Requests rejected before reaching the API (rejected at Gateway level) **MAY** omit the error object.

### 3.15 Subscription model

Many of DCSA’s APIs include or rely on subscription features where one party can subscribe for updates. DCSA maintains a separate publication, **DCSA Subscription Callback API**, that documents the subscription model provided in the DCSA standardized APIs.

## 4 Stable

APIs need to be stable over time, which means the occurrence and impact of major changes are minimized. The requirements to achieve stability are described in this chapter.

### 4.1 Versioning

Semantic Versioning 2.0 **MUST** be used to specify version information restricted to the format *<MAJOR>.<MINOR>.<PATCH>*.

- *MAJOR* to be increased when backward compatibility is broken.
- *MINOR* to be increased when new functionality (e.g. new operations, new optional fields) is added in a backward-compatible manner
- *PATCH* to be increased when backward-compatible bug fixes are made. PATCH does not include new functionality.

**Pre-releases** ([rule 9](#) from Semantic Versioning) and **build metadata** ([rule 10](#)) **MUST NOT** be used in API version information.

URI versioning **MUST** be used. Only *<MAJOR>* version is included – example: */v2/events*. *First version SHOULD include /v/*. An *API-Version* custom header **MAY** be added to the request; if added it **MUST** only contain *<MAJOR>* version. *API-Version* header **MUST** be aligned with the URI version.

A custom header called *API-Version* **MUST** be added in the response-specifying version. The *API-Version* includes all three formats – example: *API-Version: 2.0.0*.

### 4.2 Backward compatibility

Backward compatibility **MUST NOT** be broken within a major release.

When adding new features, the following rules **SHOULD** be followed:

- Preferably add optional and not mandatory fields
- Never change the semantics of a field
- Enum range can be reduced (server needs to be able to handle old values)
- Enum range can be expanded but only when used for input parameters

Major versioning **SHOULD** be avoided:

- Create a new resource (a variant of the old) in addition to the old
- Create a new service endpoint (Duplicate and Deprecate: add Deprecation header to old endpoint; eventually add Sunset headers)
- Create a new API version in parallel with the old

Clients of the API **SHOULD** be [robust](#):

- Be conservative with API requests and data passed as input
- Be tolerant with unknown fields in the payload, but do not eliminate them from payload if needed for subsequent PUT requests

Subscriber to the API **SHOULD** be [robust](#) when receiving a Message (Subscriber and Message as defined in the Subscription model):

- Be conservative with API requests and data passed as input
- Be tolerant with unknown fields in the payload, but do not eliminate them from payload if needed for subsequent PUT requests
- Explicitly use subscription filters on enum values OR cope with unknown enum values in notifications.

Implementors of the API **MUST NOT** be more than 1 major version behind the latest version. No more than 3 parallel major versions **SHOULD** be available.

#### 4.3 Deprecation

Deprecated endpoints **MUST** be documented in OpenAPI specification using the "Deprecated" property introduced in OpenAPI 3.0.

Deprecated endpoints **SHOULD** add Deprecation and Sunset headers to responses. See [The Deprecation Header Field](#) and [The Sunset http Response Header Field](#).

A Link header providing additional information **SHOULD** be added in combination with the Deprecation header. If added, the link provided **MUST** point to the documentation for additional information.

Communication **SHOULD** be sent to consumers of deprecated endpoints where possible.

## 5 Secure

APIs need to be secure, which means measures and practices are implemented to prevent abuse. The requirements to achieve this objective are described in this chapter.

## 5.1 Endpoints

All endpoints **MUST** be secured. Security used **SHOULD** be OAuth2. Other security schemes **MAY** be used. HTTPS **MUST** be used.

Note that the security of the **Receiver's** endpoint ("Callback URL") in the Subscription model is exempt from this section as it uses a different security scheme, which is documented in the **DCSA Subscription Callback API** publication.

## 6 Performant

The performance consequences of API design need to be considered, monitored, and optimized for common interactions.

To achieve this objective, two requirements are identified below:

- Circuit Breaker Pattern **SHOULD** be used for fast fail. If the implementer of the API has internal errors or a database that is not responding - it would be a good idea to add a Circuit Breaker Pattern in order for the endpoint to respond quickly, instead of waiting for Timeout.
- Rate Limiting **SHOULD** be used to prevent too many requests.

## 7 Well-documented

Up-to-date, complete, and easy-to-read documentation needs to be available to facilitate adoption. The requirements to achieve this objective are described in this chapter.

### 7.1 General documentation

Open API 3.0.x **MUST** be used for documenting endpoints. British English **MUST** be used.

SwaggerHub **MUST** contain the latest specification for the API:

- DCSA OpenAPI specification for Operational Vessel Schedules [v1.0.0](#)
- DCSA OpenAPI [specification for Track & Trace v2.0.0](#)

GitHub **MUST** contain the latest documents and links to the latest documents:

<https://github.com/dcsaorg/DCSA-OpenAPI>

### 7.2 HATEOAS

Where possible, header links **SHOULD** be added to discover relations to the resource requested.