API Producer Guidelines

These guidelines are provided to properly document, design and implement a REST API.

They are organized into 5 **themes** as point of views that will be the focus of distincts peoples.

Functional rules focus on **documentation** and **its accuracy**, to ensure that our APIs are understandable by most people and fulfills functional requirements.

Business Analysts and Functional Architects will have a focus on these rules.

> **Hint**
>
> Not all these guidelines apply to your perimeter !
>
> see SG Group API Guidelines and SGM API Wholesale Guidelines first.

Information¶

Properly declare your API in the Catalog and maintain this global information up-to-date

Declare in API Catalog¶

All SGM APIs, which aims to be used openly, must be declared into the API Catalog, so that everyone, IT or business, including those from the team building/maintaining APIs, is able to search or browse in the API Catalog, find the APIs related to their needs, easily understand the APIs from provided information and apply them quickly. So they don't need to do direct exchanges with the dev team.

You must declare into the API Catalog your API for each of its new **major versions**. A major version means that your API is upgraded with breaking changes. Otherwise, if your API is upgraded without impacting any existing client app, considered as a minor version, nothing needs to be updated within the Catalog apart from the swagger documentation.

When your API is a REST one secured with SG Connect, but aims to be used only within you own team (ex: to serve a SPA), you still declare it within the API Catalog, but explicit the usage perimeter of this API by setting its exposition to "Hidden".

When your API aims to be used primarily by a specific client (ex: a UI), and only partially by other clients : make explicit which operations are meant for the UI only and which are open to other clients. This distinction may be made using SG Connect client app rights (with a additional OAuth2 scope dedicated to the UI).

See also

- Detailed tutorial : Declare your API
- Warn consumers of major version shutdown

Ensure accurate and up-to-date information¶

In order to not mislead an user by wrong or deprecated information about your API, the API Manager, as declared in the API Catalog, must guarantee that all the information about your API is accurate and up to date.

The first information to be up-to-date is the name and contact of the **API Manager** itself!

Write information in US English¶

An API within SG may be consumed by worldwide developers, such as SG employees or SG clients/partners. They all have only US English as common language. So APIs must be exposed and documented on US English language.

This is especially true for the information you put in the API catalog since it is the entry point to discover or subscribe to APIs.

If your API is originated from **french specific** departments (ex: SGSS) and if the french language is strongly helpful for most people around the API, then you may use french by following principles:

1. API name must start in English, followed with 1 or 2 french keyword
2. description must start with few english words, followed by the original (more complete) french description

Examples

```
Get information of the specific french "PEA" product
---
Recupere des informations sur les PEA
```

See also Write API interface in US english

API-202 manual **Write catalog information in english**
Free text information set in the API catalog card is in english — Information

Provide an understandable API name¶

Your API will be first recognized and shared using its name only. Make your API name understandable to ensure that any user, including business and those far from the API team, can easily find it.

Name your API as : *"entity perimeter action (application)"*

- It must be functionally understandable as containing the main business entity (in singular) or process
- It must not be only an application or project code name
    - but you may put this information within parenthesis at the end
- It must not contain a verb, but only nouns, action can be expressed by using the noun
    - if there is no action noun, we consider that the API only exposes data
    - if action is : "Manager", "Management", "Catalog", it means that data can be modified
    - if action is : "Referential", it means that data is only exposed
- It must not contain fancy characters : only letters and digits
- It must not contain the word "API" or "Service"
- It must not have more than 65 characters.

Examples: *Embargo Country*, *OTC Equity Deal Booking*, *Market Data* , *Forex Pricer*, *Third Party Management*, *Third Party Manager*

Bad examples: *Heracles API for Socrate*

/

Note that initial 2017 recommendation was to use a construct as "*verb entity perimeter (application)*" with verb as *Get*, *Search*, *Manage*, *Compute* (eg. *Compute Regulatory Eligibility of Trades (RED)*). We will not enforce any changes on such API names, but please do not use it any more and change it when possible.

API-201 manual **Provide an understandable API name** Name has a form like "Entity Action Perimeter" (no action means GET), like: "Trade Booking in London", "Third Party"		Information

Define the API exposition accurately¶

Ensure your API is up-to-date and accurate on the value of Exposition.

Especially for the "Partner" and "Public" values that are only for API exposed to external clients/partners.

See also Ensure accurate and up-to-date information

API-209 manual **Define the API exposition accurately** Exposition declared in catalog reflects the reality of usage of the API		Information

Provide up-to-date Lifecycle status¶

Ensure your API is up-to-date and accurate on the value of Lifecycle Status

Take care of setting it to *"Released"* as soon as your API get to production, and so completing few more information.

It is specifically important for this field to be up-to-date as various quality/security/governance processes rely on it.

See also Ensure accurate and up-to-date information

API-208 manual **Provide up-to-date Lifecycle status** Lifecycle statuts field reflects the reality of the API, especially for the status "Released" (note that some fields become mandatory when Released)		Information

Provide meaningful keywords in declaration¶

An API declaration must contain the keywords that client will naturally use to find this API.

API keywords should be mainly functional ones, but can also be related to the organization, the project, from the API team or from its surrounding ecosystem.

Putting keywords as tags will also enforce the visibility and meaning of your API. Tags are used in the API search feature so using meaningful keywords will help people find your API.

Weight of keywords presence in field for search result sorting is, in order :

1. name
2. summary
3. tags
4. description

API-204 manual **Provide useful tags** Tags are meaningfull so people will mostly find your API, and tags also help to categorize it		Information

Ensure API contacts are valid¶

Provide valid contacts for your API as described in People

This will help people that have inquiries:

- potential users of your API
- Architects or domain owners for governance.

In particular:

- the API manager and its backups must be valid addresses matching a member of the *Producer unit* field in API declaration;
- the Product Owner must be a valid address matching a member of the *Owner unit* field in API declaration.

> API-402 manual **Complete or update people and orga fields**
> Fields: Manager, Backups, Product owner, Support contact, Region, Developer unit are set with up-to-date values Information

Declare API organization accurately¶

Provide up-to-date organization information about your API as described in Organization

---

Provide understandable description¶

When people find your API in the developer portal they will first see the description page.

The description should help people decide if they can/want to use your API. In particular it should describe the functional perimeter of the API and its business value. This will let people decide if it matches their use case.

It should also describe any specifics you may have regarding the subscription process. If you ask some questions during an on-boarding process before validating subscriptions then the API description is a good place to mention them.

See also : field-description

> API-205 manual **Provide understandable description**
> Content of Summary and Description fields is understandable by any user, it explains theInformation global functionality and some important other points

Reference your swagger file in Catalog¶

All the operations/datas of your API is documented from within your API entry of API Catalog.

A developer expecting to use your API needs a clear and understandable documentation to properly call your operations and react on the result.

A business user expects to be able to understand the data/operations provided by your API by looking at this documentation, without contacting the API team.

API Catalog requires the API team to provide a link to (or upload) the Swagger (OAS) file for each environment used by the API.

When an API is Released, it must have at least a “prod” environment with a swagger/oas file.

Such file, once API starts to be implemented, is usually generated from the code to ensure to match it.

Some quality checks will be automatically performed on this specification file.

When exposing your API on internet using SG gateways, they will use this specification file to filter the HTTP calls.

A Swagger 2 or OpenAPI 3 file can be specified, in json or yaml format.

API-210 auto-swagger-crawl **Provide a specification file (OpenAPI/swagger), displaying direct documentation in Documentation tab** A Swagger/OAS contract is always available from API Catalog on all environments (imported or crawled from url)	Documentation
API-211 auto-swagger-crawl **Swagger documentation is up-to-date** OpenAPI Specification file was accessed by API catalog tool less than 15 days ago	Documentation

Define the API domain accurately¶

The domain is used as a search filter to help people focus on a functional perimeter when having too many results.

The domain and associated domains allow to identify re-use needs and to bring consistency across strongly related APIs.

For each domain a set of persons is responsible to qualify and steer the APIs:

- “Domain Architect” : an IT person from a transversal architecture team
- “Domain Manager” : an IT person from a specific program/project/application
- “Domain Owner” : a Business person

They each have right to:

- Reassign the API to another Domain
- Set associated domains if needed
- Set the “Representative Architect” field, this person will then have same rights

When it is difficult to associate an API to a single domain it might be an indicator that the API handles to many concerns (See rule Ensure proper granularity and consistency). If you think it is a legit use case - or until you segregate concerns - you can define *Associated domains* on the API to keep trace of potential impact of this API to these other domains.

API-213 manual **Define the API Domain accurately** API Domain is correct and associated domains are set if needed, if unsure let the functional architect check this review point	Information

Define the SG group category¶

The SG Group Category must be properly specified as a mandatory SG Group governance requirement.

API-537 auto-catalog-check **Set the SG Group Category** The SG Group Category must be properly specified as a mandatory SG Group governance requirement	Information

Documentation¶

Document your API in/out data and operations, by using OpenAPI Specification (swagger).

To make it easier to understand your API, for functional or business people and for developers that are less experienced in using such API, you need to expose enough information on the various items composing your API.

Provide valid OpenAPI Specification¶

We use the OpenAPI standard to define API specifications (see Reference your swagger file in Catalog).

We encourage to use OpenAPI 3.0, but still support Swagger 2. Most libraries (including our SDK) generate Swagger 2 format.

Take care to publish a valid OpenAPI file if it is managed manually or as some swagger file generation tools may be too much permissive.

Validity is important to ensure proper documentation rendering, client code generation, automated quality reviews, integration in API Gateways, try-out features

You can use a SG hosted Swagger Editor to check and correct the validity of the specification file.

API-242 auto-swagger-check **Provide a valid Specification file (OpenAPI/swagger), requires API-210 to be GREEN** The Swagger/OAS format must be fully respected to allow the automatic controls : API-243, ...	Documentation

Write API interface in US english¶

An API within SG may be produced and consumed by **worldwide developers**, such as SG employees or SG clients/partners, who all have only US english as common language.

The API's fields name, urls and descriptions, coming from the OpenAPI Specification, must use the US english language.

Note: There is a distinction between US english (en-us : color, dialog, catalog, analyze) and UK english (en-gb : colour, dialogue, catalogue, analyse).

See also Write information in US English

API-240 manual **Write API interface and documentation in english** Operations and fields are named and described using english	Documentation

Provide usage overview¶

In complement to the description of each unitary operation of the API, a helpful documentation typically describes first the following global aspects:

- API purpose, and usual use cases
- concrete examples of API usage
- security and rights handling on all calls
- error handling, generic limits and thresholds
- pagination, sorting, filtering, fields selection mecanisms

Although the OpenAPI Specification allows to expose a global description on the whole API, sometimes this placeholder is not practical to use.

So it is suggested to provide, aside the OpenAPI Specification itself, an API developer manual as an aside file in markdown format.

Such file should be located in same source code repository to ensure it will be always up-to-date.

The API Catalog allows you to associate such markdown content with your API and to show it to end user.

Examples of such usage overview / developer manual :

- https://developers.jivesoftware.com/api/v3/cloud/rest/index.html

- https://developer.github.com/v3/

API-241 manual **Provide usage overview in documentation** Generic behaviors and basic usage scenarios are described on the start of the documentation using markdown, or else say "No needs" in review comment	Documentation

Set a functional name on resources/operations¶

Name the resource and operation using functional words so that it can be displayed in table of content of documentation, and it can be understood by functional people.

> **Note**
>
> In OpenAPI documentation the `tag` describes the *resource name* and the `summary` describes the *operation name*

The resource name being a code (no space allowed), use PascalCase.

```yaml
tags:
  - name: Apis
paths:
  /apis/{apiCode}/versions/{version}:
    get:
      tags: [ "Apis" ]
      summary: "Get a versioned api by its code and major version number"
```

See also Describe resources/operations

API-243 auto-swagger-check **Give a name on operations** Each operation has an explicit name in swagger specification, will stay Pending if API-242 is not green (API-242 must be GREEN, else this rule may stay Pending)	Documentation
API-244 manual **Give a human friendly name on operations** Each operation has a human friendly name as a short imperative sentence, like: Seach Third Parties with multiple criteria	Documentation
API-401 manual **Give a human friendly name on resources** Each resource has a human friendly name, like "Third Parties"	Documentation

Describe resources/operations¶

Following Set a functional name on resources/operations and when the name only is not sufficient, provide a more complete description for each resource/operation.

Example:

```yaml
tags:
  - name: Apis
    description: Apis management
paths:
  /apis/{apiCode}/versions/{version}:
    get:
      tags: [ "Apis" ]
      description: "Give full information on the API, including the fine grained rights of current user"
```

API-245 manual **Describe meaningfully each operation when needed** All operations have human friendly implementation notes or are self-described by its name	Documentation
API-246 auto-swagger-check **Set a description on each operation when needed**	Documentation

Each operation has an implementation notes in the description field, a reviewer may override this automatic rule as "Not Applicable" (API-242 must be GREEN, else this rule may stay Pending)

Document operations parameters¶

For each path or query parameter:

- Describe its meaning if its name is not self-explaining
- Provide a sample value to simplify testing and understanding
- Explicit if there is some default value on optional params (eg. default pageSize)
- Explicit its specialized format (on date and float/double)

Note that OAS3 has a specific “example” field to put a sample value, with Swagger2 just add it within “description”

Examples of description with sample value

```yaml
parameters:
  - name: apiCode
    in: path
    # describe it more and provide a sample value
    description: "the unique api code as minor case letters in with dash (eg. service-catalog)"
    required: true
    type: string
  - name: version
    in: path
    # describe it more and provide a sample value
    description: "the major version of the api as a string but usually being an integer (eg. 2)"
    required: true
    type: string
```

Examples of format and default value

```yaml
parameters:
  - name: releasedSince
    in: query
    required: false
    description: "get only products released on or after this date (eg. 2019-05-01)"
    type: string
    # this string being an ISO date must be made explicit :
    format: date
  - name: pageSize
    in: query
    required: false
    description: "max number of items returned (eg. 50)"
    type: integer
    # default value of pageSize must be made explicit :
    default: 100
```

API-247 manual **Document operations parameters**
Parameters are described unless name is sufficient, specific format is present (when date, float), sample value is present when needed, considered default value is mentionned when no value has no meaning Documentation

Accuracy¶

Ensure the API is functionally accurate, properly used and aligned with business needs and strategy.

Expose data only when official source¶

This guideline rule is meant to identify APIs that are a duplicate of official ones, totally or just partially.

You should not expose nor share through your API data or computational work for which your API is not the official source.

For instance if you create an API to expose data that is replicated from the official golden source it should stay private to your team.

Same goes for computation: if you replicate a process (might be for performance reasons on local/not too important checks) then it should not be available to consumers outside of your team.

More generally think Inner Source (improve existing behavior instead of duplicating it).

API-268 manual **Expose data/operation when you are the official source** API is official source of data or operator of compute/change for the perimeter		Principles

Ensure proper granularity and consistency¶

An API with the proper granularity :

- only contains business objects (resources) closey related to each others
    - usually an API is about a main business object (the main resource)
    - and contains some closely associated or childs business objects (sister or child resources)
- fully and properly handles a business process
- is not too big to keep it manageable and evolutive
    - if the API contains more than 10 root resources or more than 100 operations, its granularity should be explicitely evaluated
- doesn't mix non directly related concerns together
    - there should not be a set of clients that always use a part of the API, and another set using only the other part.

API-269 manual **Ensure proper granularity and consistency** API contains only data and operations that are closely related to each other and centered on a main business object, aka: API granularity		REST design

Ensure subscriptions are legitimate¶

Activated subscriptions are to be legitimate as per :

- the strategy defined on the perimeter regarding "target APIs" to use
- the legitimacy of the use case
- the provided applicative rights (as subscription scopes)
    - eg: if a client application is only supposed to read data on an API that also has write operations, it must not have the write scopes
- the volume and frequency of calls supported by the SLA

API-270 manual **Ensure subscriptions are legitimate** API Manager's accepted subscriptions are confirmed as legitimate as per strategy, use case, provided rights, SLA		Information

Testing¶

Ensure the API is functionally tested.

Test your APIs¶

While we do not promote or provide a testing tool yet there are several tools available to test your API and communicate around it. Please contact us for more details on such tools.

You should test your API on a level allowing your clients to be confident when using it. You should therefore include functional testing on your APIs. It can be based on BDD tools or use Consumer-driven Contract Testing tool (such as Pact).

API-280 manual **Test your API at interface level** Functional tests are defined and continuously run by CI/CD against your API endpoints	Testing
API-281 manual **Expose testable usage scenarii** Tests scenarios are publicly accessible	Testing