API Standards

API Standardisation & Governance in Autodesk

- Core Principles
- API Standards
- API Versioning
- HATEOAS Rest Principle
- Localization
- Filtering
- Caching
- Pagination
- Resource Naming
- Naming conventions

API Standards - Core Principles

OpenAPI Specification

- Every HTTP/1.1 REST API **MUST** be described using an API description format.
- The API description format used **MUST** be the OpenAPI Specification (formerly known as Swagger).

  Current APIs __MUST__ use [OpenAPI 2](https://swagger.io/specification/v2/) or [OpenAPI 3](https://swagger.io/specification/). OpenAPI 3 is recommended.
- Every API description **MUST** be published in the API design platform.

API Standards - Core Principles

Autodesk OpenAPI Tools

- + [Documentation Pipeline](https://wiki.autodesk.com/display/FCPA/Forge+Documentation+Pipeline+Onboarding) - Documentation generation for the Forge Developer Portal.

- + [Resiliency SDK](https://git.autodesk.com/cloudplatform-apim/forge-rsdk-codegen) - Generates client SDKs from a Swagger file in multiple languages with resiliency patterns built in.

API Standards - Core Principles

API Design Platform

- [Stoplight](https://platform.stoplight.autodesk.com) is our primary platform supporting the API first workflow. Stoplight **SHOULD** be used during API design.
- Every API description **MUST** be stored in a git repository. Stoplight or another API design platform **MAY** be used to edit the file, but it should be stored permanently in a repository.
- The Swagger files **MUST** be the single source of truth to learn about existing APIs within the organization. The Swagger file **MUST** contain documentation for each API which will be used to generate API reference documentation.

API Standards - Core Principles

NOTE: Stoplight supports API-first approach in multiple ways: They validate API description for correctness and automatically generate API documentation to drive the discussion between stakeholders. (No more emails with API description flying between stakeholders)
Autodesk-managed service is available at platform.stoplight.autodesk.com and it is accessible from all of the company's networks, including VPN. Because it is deployed behind the corporate firewall, it can access git.autodesk.com, which allows for data synchronization between the two systems. More detailed information about the service is available on the service [main page](https://platform.stoplight.autodesk.com).

API Standards - Core Principles

Contract

- Approved API Design, represented by its API Description, **MUST** represent the contract between API stakeholder, implementors, and consumers.
- Any change to an API **MUST** be accompanied by a related update to the contract (API Description).

Contract Validation

- Every API description (contract) using HTTP(S) protocol **MUST** be tested against its API implementation.
- In addition to local runs, the tests **MUST** be an integral part the API implementation's CI/CD pipeline.
- The CI/CD pipeline **MUST** be configured to run the test after every change of the API implementation.

API Standards - Core Principles

Design Maturity

API Design Maturity

> How to design an API

- When designing an API an [API First](API_First.md) process **SHOULD** be followed. This practice surfaces design problems earlier in the lifecycle when they are easier to correct.
- Every API design **MUST** be resource-centric ([Web API Design Maturity Model Level 2](http://amundsen.com/talks/2016-11-apistrat-wadm/2016-11-apistrat-wadm.pdf). That is an API design **MUST** revolve around Web-styled resources, relations between the resources and the actions the resources may afford.
- An API **MAY** choose to support Web Maturity Model Level 3 ([HATEOAS](HATEOAS.md)) if appropriate for the service.

API Standards - Core Principles

API Design Implementation Maturity

> How to implement the API design

- Every API design implementation using the HTTP 1.1 protocol **MUST** use the appropriate HTTP Request Method ([Richardson Maturity Model Level 2](https://martinfowler.com/articles/richardsonMaturityModel.html#level2)) to implement an action afforded by a resource.

Robustness

- Every API implementation and API consumer **MUST** follow Postel's law:
- > Be conservative in what you send, be liberal in what you accept.
- >– John Postel
- That is, send the necessary minimum and be tolerant as possible while consuming another service ([tolerant reader](https://martinfowler.com/bliki/TolerantReader.html)).

API Standards - Core Principles

Rules for Extending

- Any modification to an existing API **MUST** avoid breaking changes and **MUST** maintain backward compatibility.
- In particular, any change to an API **MUST** follow the following Rules for Extending:
- You **MUST NOT** take anything away (related: Minimal Surface Principle, Robustness Principle)
- You **MUST NOT** change processing rules
- You **MUST NOT** make optional things required
- Anything you add **MUST** be optional (related Robustness Principle)
- See [API Versioning](Version_Versioning.md) for more details.

NOTE: These rules cover also renaming and change to identifiers (URIs). Names and identifiers should be stable over the time including their semantics.

API Standards - Core Principles

Loose Coupling

- In addition to the robustness principle, API consumers (clients) **MUST** operate independently of API implementation internals.
- Similarly, the API consumers **MUST NOT** assume or rely on any knowledge of the API service internal implementation.

API Standards - Versioning

 # API Versioning

> The fundamental principle is that you can’t break existing clients, because you don’t know what they implement, and you don’t control them.

> In doing so, you need to turn a backwards-incompatible change into a compatible one.

>> – [Mark Nottingham](https://www.mnot.net/blog/2011/10/25/web_api_versioning_smackdown)

Any modification to an existing API **MUST** maintain backward compatibility. Any change that does not maintain backward compatibility must use a new major version for that API, while maintaining support for the old version for a given period of time.

API Standards - Versioning

 # API Versioning

In particular, any change to an existing API **MUST** follow the following **Rules for Extending**:

- You **MUST NOT** take anything away
- You **MUST NOT** change processing rules
- You **MUST NOT** make optional things required
- Anything you add **MUST** be optional

All the rules above apply for:

- **Resource identifier** (resource name / URI) including any **query parameters** and their semantics
- **Resource metadata** (_e.g._ HTTP headers)
- **Actions** the resource affords (_e.g._ available HTTP Methods)
- **Relation** to other resources (_e.g._ Links)
- **Representation format** (_e.g._ HTTP request and response bodies)

API Standards - Versioning

 # API Versioning

URI-based versioning

The major version of a resource must include a version identifier in the path.

- Resource URL before a breaking change:

  https://developer.api.autodesk.com/data/v1/banners

- Resource URL after a breaking change:

  https://developer.api.autodesk.com/data/v2/banners

The resource version path should only include the major version number, like `/v1` or `/v3`. If your system uses semantic versioning, then the minor and patch version numbers __MUST NOT__ be included in the version path.

The old version __MUST__ remain available for a specified period of time. The exact period depends on business and technical decisions.

API Standards - Versioning

All new resources __MUST__ use a `/v1` (or the most recent [Global Version](API_Versioning.md#global-vs-per-resource-versioning)) version identifier in its path, following the service name, and preceding all other parts of the path.

API Standards - Versioning

 # API Versioning

API Versions for Changelogs

When an API or service is updated, typically the changes are are published publicly to let people know what changed. Changelogs should identify versions using dates

A service __MAY__ choose to use semantic version numbers internally, but the changes should be documented to clients publicly using dates, not the semantic version number.

Recommended Reading

+ [API Versioning Has No "Right Way"](https://blog.apisyouwonthate.com/api-versioning-has-no-right-way-f3c75457c0b7)

API Standards - Filtering

 # Filtering

A search (filter) operation on a collection resource __SHOULD__ be defined as safe, idempotent and cacheable, therefore using the GET HTTP request method.

Every search parameter __SHOULD__ be provided in the form of a query parameter. In the case of search parameters being mutually exclusive or requiring the presence of another parameter, the explanation **MUST** be part of operation's description.

Filters are specified in the query string using the name of a field, and a value or range of values to test against. The syntax in the query string is:

?filter[<fieldName>]=<valueToFilterOn>

Filters **MAY** supply a range of values rather than a specific value, using the `..` notation as shown in the examples below. This can be used for numeric ranges and date ranges.

API Standards - Filtering

 # Filtering

Examples

```
    GET /books?filter[title]=Great Expectations

Will return all books with the title `Great Expectations`.

    GET /books?filter[price]=10..20

Will return all books with a price in the range of 10 to 20, inclusive.

    GET /books?filter[price]=..50

Returns all the books with a price up to 50 dollars.

    GET /books?filter[price]=10..&filter[title]=The Bible

Will return all books with a price over 10, and the title `The Bible`
```

API Standards - Pagination

 # Pagination

Access to lists of data items __MAY__ support pagination for best client-side batch processing and iteration experience. This holds true for all lists that are (potentially) larger than just a few hundred entries.

There are two pagination techniques:

- Offset/Limit-based pagination: numeric offset identifies the first entry in the page

- Cursor-based — aka key-based — pagination: a unique key element identifies the first entry in the page.

Offset/Limit-based Pagination (Preferred Option)

An offset-based pagination __MUST__ accept the following query parameters:

- offset: Offset from the start of the collection to the first entry in the page. Zero based.

- limit: Determines the maximum number of objects that __MAY__ be returned. A query __MAY__ return fewer than the value of limit due to filtering or other reasons.

API Standards - Pagination

 # Pagination

The response of the offset based pagination MUST contain a pagination object with the following fields:

- offset: The offset value supplied in the request.

- limit: The limit value used by the server in the response. Note that this will be different from request in case of server overrides.

- totalResults: Optional. The total number of results that match the query irrespective of limit.

- nextUrl: [Link](Links.md) that will return the next page of data. If not included, this is the last page of data. It is allowed that a page may be empty but contain a next paging link. Consumers should stop paging only when the next link no longer appears.

- previousUrl: Optional. [Link](Links.md) that will return the previous page of data. If the previous link is supported, then the absence indicates the first page of results.

API Standards - Pagination

 # Pagination

Example

A response of offset-based paginated collection

```
{
   "pagination": {
    "limit": 20,
    "offset": 0,
    "totalResults": 21,
    "nextUrl": "https://developer.api.autodesk.com/data/v1/folder/contents?limit=20&offset=20"
   }, "results": [
    {...},
        ]
 }
```

API Standards - Resource Names

 # Resource Names

In resource-oriented APIs, *resources* are named entities, and *resource names* are their identifiers. Each resource **MUST** have its own unique resource name.

The resource name is made up of the ID of the resource itself, the IDs of any parent resources, and its API service name. We'll look at resource IDs and how a resource name is constructed below.

A *collection* is a special kind of resource that contains a list of sub-resources of identical type. For example, a directory is a collection of file resources. The resource ID for a collection is called collection ID.

The resource name is organized hierarchically using collection IDs and resource IDs, separated by forward slashes. If a resource contains a sub-resource, the sub-resource's name is formed by specifying the parent resource name followed by the sub-resource's ID - again, separated by forward slashes.

API Standards - Resource Names

Example 1: A storage service has a collection of buckets, where each bucket has a collection of objects

```
| API Service Name| Collection ID | Resource ID| Collection ID | Resource ID |
|-------------|----------|--------|----------|---------|
|storage.googleapis.com | `/buckets` | `/{bucket-id}` | `/objects` | `/{object-id}` |
```

API Standards - Resource Names

 # Resource Names

Example 2: An email service has a collection of users

Each user has a settings sub-resource, and the settings sub-resource has a number of other sub-resources, including customFrom:

```
| API Service Name| Collection ID | Resource ID| Collection ID | Resource ID |
|-----------|----------|--------|----------|---------|
| mail.googleapis.com | `/users` | `/name@example.com` | `/settings` | `/customFrom` |
```

An API producer can choose any acceptable value for resource and collection IDs as long as they are unique within the resource hierarchy.

API Standards - Resource Names

 # Resource Names

Prefix placement

The prefix is usually placed in the resource path in front of the specific resource name but it can be part of the API service domain name. Services defining their own domain can put part of the prefix in the domain name:

- https://developer.api.autodesk.com/sales/pricing/v1/items
- https://sales.api.autodesk.com/pricing/v1/items
- https://enterprise.api.autodesk.com/order/v3/orders

API Standards - Resource Names

 # Resource Names

Core APIs

Core APIs, consist of core Forge services, such as authentication and Forge Data Management, should follow this pattern:

/core/<category>/<service>/<version>/<resource path>

Examples:

```
/core/data/oss/v2/buckets

/core/data/asset-graph/v1/collections
```

API Standards - Resource Names

 # Resource Names

Customer Data APIs

Customer Data APIs consist of APIs for particular products, like BIM360. These APIs should use the following template:

<industry>/<service>/<version>/<resource path>

Examples:

```
/construction/issues/v2/projects/{projectId}/issues/{issueId}/attachments

/production/components/v3/entities
```

API Standards - Resource Names

 # Resource Names

Enterprise/Business APIs

Enterprise/Business APIs consists of B2B or B2C APIs for various domains like subscriptions, orders, channel partners, etc. These should use the following template:

<domain>/<version>/<resource path>

Examples:

```
order/v3/order-details/{id}
sales/v2/offerings/{id}
account/v2/accounts/{csn}
```

API Standards - Resource Names

 # Resource Names

Generic Pattern

APIs not covered by the sections above should still use an API prefix. This is the suggested form, but others could be considered, as appropriate.

<category>/<service>/<version>/<resource path>

Example:

```
/sales/pricing/v1/items
```

https://nordicapis.com/10-best-practices-for-naming-api-endpoints/

API Standards - Error Responses

 # Error Responses

An error response is intended for use with the HTTP status codes 4xx and 5xx. Error response **MUST NOT** be used with 2xx status code responses.

At the minimum, any error response **MUST** have the 'title' and 'detail' fields.

Example

```
{
   "title": "Authentication required",
   "detail": "Missing authentication credentials for the Greeting resource."
 }
```

API Standards - Error Responses

Error Responses

Optional Fields

It **SHOULD** have the 'type' field with the identifier of the error.

```
{
    "type": "https://developer.api.autodesk.com/problems/scv/unauthorized",
    "title": "Authentication required",
    "detail": "Missing authentication credentials for the Greeting resource."
  }
```

> **NOTE:** The type field is an identifier, and as such it **MAY** be used to denote additional error codes.

Additional Fields

If needed, the error response **MAY** include additional fields.