Description of **basic principles** for SMDG Webservice usage and
implementation / design guideline.
This includes a recommendation of how to handle **security and authentication.***🡺 Action on first draft: Peter, Daniel and Olaf*

**Why is API design essential?**

While currently this is just a working group to find ideas for common web services there will be a

Following the trend on the Internet, electronic communications will continue to evolve from known and currently used EDI protocols to web services. These APIs will continue to spread and as there are many formats for various EDI use cases today, there will also be countless APIs one day to solve similar problems.

A uniform interface design will therefore ensure the software can be reused. At the same time, the simpler an interface is, the more clients will use it. The fewer hurdles an API has, the easier it is to use and the more likely it will become accepted.

It therefore makes sense to use a uniform guideline right at the beginning of the interface design. The parameters SHOULD be implemented by all web services to make it easier for developers to consume the API.

**Changes in API design**

You can differentiate between breaking and non-breaking changes. Non-breaking changes allow further use of the API without customization of the client software. Breaking changes, however, require a change in the consuming software.

Once an interface has been defined, deviating from it is no longer allowed – not even non-breaking.

Deviation from an established interface would lead to a drifting apart of the different implementations, which in turn would lead to many different but similar services in the long run. These services would be incompatible at some point. Any deviation from any of the interface descriptions would automatically generate a new API service of a new API version.

If the still to be established API guideline has weaknesses, the guideline must be amended. Services or service versions that already exist and have therefore been created with older versions of the guideline will not be customized. A once specified interface will not be adapted later.

**Resources and Security**

While some parts of an API can be described very accurately, others in any implementation are freely designable. Client software must be able to adapt to the particular circumstances of an implementation.

These free-form parts include

* Resource base paths
* Security

Each API provider can specify an arbitrary *base path* for an API. In the Swagger / OpenAPI standard, the field "basepath" is intended for this purpose. The assignment of this field is not specified within this guideline. Each API implementation can set the value freely. Only the "paths" defined in the Swagger / OpenAPI are fixed and must not be changed by an implementation.

A client for a particular API must be able to use any "basepath".

Security cannot be set generically. Different companies may have different policies and views in this area. At the same time every use case is different and has to be considered separately. An API interface specification will therefore not include security policies. Each API implementation will therefore apply its own guidelines - as part of the Swagger / OpenAPI specification.

An API client must therefore be able to use any Swagger / OpenAPI-specified security settings.

*Note: Swagger / OpenAPI 2 defines exactly three ways to protect an API:*

*- Basic Auth*

*- API key*

*- OAuth2*

*These options (and combinations thereof) must be supported by every client.*

**URL patterns**

A URL pattern MUST look like this:

https://{host}:{port}/{base-path}/v{version-number}/{list-resource}/{primary-resource}/{sub-resource}

* {base-path} will be automatically created within your project. All corresponding API will use the same context-root. This part can be set freely by every implementation.
* {version-number} is a simple number that is increased every time you implementing a breaking change in your API
* {list-resource} MUST be a noun in plural. It allows access to a whole set of resources.
* {primary-resource} is an ID pointing to one specific resource. Do NOT use implementation details as public IDs (e.g. the database sequence).
* {sub-resource} is a resource that has a life cycle that is bound to its primary-resource
* query-parameters can be used to filter or sort resources.

You MUST NOT use verbs or other method names in a URL.

**Naming**

API names MUST be written in English.

The different parts of the API MUST use lowercase and follow the naming conventions:

|  |  |  |  |
| --- | --- | --- | --- |
| URLs/ Resources | Kebab case | /delivery-orders/ | Compound words MUST be separated with hyphens |
| Query parameter | Snake case | ?order\_id=foo | Compound words MUST be separated with underscores |
| Properties (in JSON) | Snake case | zip\_code | Compound words MUST be separated with underscores |