Applying the HTTP methods to the SDMX REST API for Structures

CREATE

Artefact (Structure)

Submitting SDMX Artefacts in bulk, either of the same or of different types, is achieved with a `POST` method.
Creating new Artefact(s) may be issued by:

- `POST` one or more Maintainable Artefacts under the proper resource type, e.g. for Codelists: `/structure/codelist/`
- `POST` one or more Maintainable Artefacts under the abstract structure resource type, e.g. `/structure/`
- `POST` one or more specific Items under a specific Item Scheme resource, e.g. `/structure/codelist/SDMX/CL_FREQ/1.0/`

In case one Artefact is to be created, `PUT` may be used, as well.
In that case, the following apply:

- `PUT` one fully identified Maintainable Artefact, e.g. for a Codelist: `/structure/codelist/SDMX/CL/1.0`
- `PUT` one fully identified Item , e.g. for Item `M`: `/structure/codelist/SDMX/CL_FREQ/1.0/M`

Client

In order to create Artefacts, the client:

- MAY set the `Accept` header to indicate the preferred response format;
- MUST set the `Content-type` header according to the format of the submitted Artefact;
- MUST include in the request body, one or more Maintainable Artefacts in the SDMX format indicated in the `Content-type` header and of the SDMX type indicated in the resource, i.e.:
  - For `POST`:
  - any set of Maintainable Artefacts under resource `/structure/`
  - a set of specific type of Maintainable Artefacts under the corresponding resource type, e.g. for Codelists: `/structure/codelist/`
  - a set of Items, within an Item Scheme, as identified by the resource url, e.g. for

Codelist `SDMX:CL_FREQ(1.0)` : e.g. `/structure/codelist/SDMX/CL_FREQ/1.0/`
  - For `PUT` :
  - one Maintainable Artefact under its fully identified resource, e.g. Codelist `SDMX:CL_FREQ(1.0)` under `/structure/codelist/SDMX/CL/1.0`
  - one Item in its Maintainable container under the fully identified resource, e.g. Item `SDMX:CL_FREQ(1.0):M` under `/structure/codelist/SDMX/CL_FREQ/1.0/M`

Server

In response to an Artefact creation, the server:

- MUST return `201` upon successful creation (or `207` for partial success);
- MUST return a `SubmitStructureResponse` message with the result of the action(s), according to the `Accept` header, or the default, if the `Accept` type is not supported (currently available only in SDMX-ML 2.1);
- MUST set the `Content-type` according to the returned format;
- MAY set the `Location` header to point to the created/primary resource/Artefact (only one instance is allowed);

```
RFC7231
(https://tools.ietf.org/html/rfc7231)

6.3.2. 201 Created
(https://tools.ietf.org/html/rfc7231#section-6.3.2)
[...]
   The primary resource created by the request is identified
   by either a Location header field in the response or, if no Location
   field is received, by the effective request URI.

4.3.3. POST
(https://tools.ietf.org/html/rfc7231#section-4.3.3)
  [...]
   If one or more resources has been created on the origin server as a
   result of successfully processing a POST request, the origin server
   SHOULD send a 201 (Created) response containing a Location header
   field that provides an identifier for the primary resource created
   (Section 7.1.2) and a representation that describes the status of the
   request while referring to the new resource(s).
  [...]
```

Response

The `SubmitStructureResponse` message must be returned in any case (success, partial success, failure).
In SDMX 2.1 this is defined as part of the `RegistryInterface` messages.

The details of the message are explained in section Response message, below.

Especially when different results occur on the Artefacts (e.g. partial success), the following should occur:

- Return a multi-status return code (like `207`);
- Return a `JSON` or `XML` message with the details of the result (currently available only in SDMX-ML 2.1);

Method	Exists	Is Final	Is Referenced	Refs exist/ provided	Response Code
POST/PUT	F	-	-	T	`201` (successful) or `207` (partially successful)
POST/PUT	F	-	-	F	`409` failed references

T: True, F: False, I: Irrelevant, -: Not applicable

UPDATE

Following the current SDMX practices, updating (replacing) means providing the new version of any Maintainable Artefact. In the case of Items, it means updating their details.
According to RFC7231, `PUT` is the proper way to update a resource, as identified by the URL, but `POST` may also be used in case more than one resources need to be updated.

Single Artefact (Structure)

In this case, the `PUT` method may be used. This is performed:

- Under a fully identified Maintainable Artefact, e.g. `/structure/codelist/SDMX/CL_FREQ/1.0`,
- Under a fully identified Item, e.g. `/structure/codelist/SDMX/CL_FREQ/1.0/M`.
  In both cases, the identification of the contained SDMX-ML Artefact must match the resource identification of the URL; otherwise, an error is thrown.
  The result in this case is completely replacing the identified Artefact.

Client

In order to update an Artefact, the client:

- MAY set the `Accept` header to indicate the preferred response format;
- MUST set the `Content-type` header according to the format of the submitted Artefact;
- MUST include in the request body, one Maintainable Artefact, or one Item in the SDMX format indicated in the `Content-type` header and of the SDMX type indicated in the resource.

Server

In response to an Artefact update, the server:

- MUST respond with `200` in case of successful update;
- MUST return a `SubmitStructureResponse` message with the result of the action, according to the `Accept` header, or the default, if the `Accept` type is not supported (currently available only in SDMX-ML 2.1);
- MUST set the `Content-type` according to the returned format;
- MUST respond with `422` in case of resource type mismatch, i.e. the artefact identified in the URL does not match either to the resource type or identification of the included SDMX Artefact.

```
6.5.13.  415 Unsupported Media Type
(https://tools.ietf.org/html/rfc7231#section-6.5.13)
   The 415 (Unsupported Media Type) status code indicates that the
   origin server is refusing to service the request because the payload
   is in a format not supported by this method on the target resource.
   The format problem might be due to the request's indicated
   Content-Type or Content-Encoding, or as a result of inspecting the
   data directly.

11.2.  422 Unprocessable Entity
(https://tools.ietf.org/html/rfc4918#section-11.2)
   The 422 (Unprocessable Entity) status code means the server
   understands the content type of the request entity (hence a
   415(Unsupported Media Type) status code is inappropriate), and the
   syntax of the request entity is correct (thus a 400 (Bad Request)
   status code is inappropriate) but was unable to process the contained
   instructions.  For example, this error condition may occur if an XML
   request body contains well-formed (i.e., syntactically correct), but
   semantically erroneous, XML instructions.
```

Response

The `SubmitStructureResponse` message must be returned in any case (success, failure).
In SDMX 2.1 this is defined as part of the `RegistryInterface` messages.

The details of the message are explained in section Response message, below.

The following matrix summarises the returned `HTTP` response codes.

Method	Exists	Is Final	Is Referenced	Refs exist/ provided	Response Code
PUT	T	F	F	T	`200`
PUT	T	F	F	F	`409`
PUT	T	F	T	T	`200` if update is possible - `409` if references would break
PUT	T	T	-	-	`409` (only for structural updates)
PUT	-	-	-	-	`422` see section Server

T: True, F: False, I: Irrelevant, -: Not applicable

Multiple Artefacts (structures)

In this case, the `POST` method must be used.
It shall be performed:

- under `/structure` for different types of Maintainable Artefacts, i.e. an SDMX Structure message;
- under `/structure/{maintainable}` for Maintainable Artefacts of type `{maintainable}`, i.e. an SDMX Structure message including only a specific type of Structures;
- under `/structure/{itemscheme}/{itemschemeidentifier}` for Items of the Item Scheme identified by `{itemschemeidentifier}`, i.e. an SDMX Structure message with an Item Scheme and the Items to be updated.

The semantics of `POST` are different to that of `PUT`. While `PUT` is used to fully replace the identified Artefact, `POST` is meant to update it, at the level of the resource. This means that for a submitted Maintainable Artefact, properties that exist are replaced, while those that do not exist are added. By properties, we mean Names, Descriptions, Annotations at Maintainable Artefact level, as well as their containing Identifiable Artefacts, i.e.:

- Categorisation (Source, Target)
- AttachmentConstraint (ConstraintAttachment, DataKeySet | MetadataKeySet)
- ContentConstraint (type?, ConstraintAttachment, DataKeySet | MetadataKeySet | CubeRegion | MetadataTargetRegion, ReleaseCalendar, ReferencePeriod)
- DSD (DataStructureComponent)
- ItemScheme (Item)
- MSD (MetadataTarget and ReportStructure)
- HierarchicalCodelist (IncludedCodelist, Hierarchy)
- Dataflow, Metadataflow (Structure)
- ProvisionAgreement (StructureUsage, DataProvider)
- Process (ProcessStep)
- ReportingTaxonomy (ReportingCategory)
- CategoryScheme (Category)
- StructureSet (RelatedStructure, OrganisationSchemeMap | CategorySchemeMap | CodelistMap | ConceptSchemeMap | ReportingTaxonomyMap | HybridCodelistMap | StructureMap)

The rules for `POST` are the following:
For any of the following properties submitted within a Maintainable Artefact, the property will be updated if it exists, otherwise it will be added:

- Any of the XML attributes: `validFrom`, `validTo`, `uri`, `isExternalReference`, `serviceURL`, `structureURL`, `isFinal`
- Name of a specific language, e.g.:

```
<com:Name lang="en">An English name</com:Name>
```

- Description of a specific language, e.g.:

```
<com:Description lang="en">An English description</com:Description>
```

- Annotation of a specific `id` (or `AnnotationType`?), e.g.:

```
<com:Annotation id="identifier">
  <com:AnnotationType>TYPE</com:AnnotationType>
</com:Annotation>
```

The same holds for all content described in the Maintainable Artefacts, above. For any content

that is not identified within the Maintainable, e.g. ConstraintAttachments in a ContentConstraint, or Source and Target in a Categorisation, new instances will replace all the existing ones. On the other hand, when they can be identified, only existing instances will be replaced, e.g. Components in a DSD.
The following table illustrates the above two cases per Maintainable Artefact:

Maintainable Artefact	Identifiable content	Non-identifiable content
Categorisation	N/A	Source, Target
AttachmentConstraint	N/A	ConstraintAttachment, DataKeySet, MetadataKeySet
ContentConstraint	N/A	type(?), ConstraintAttachment, DataKeySet, MetadataKeySet, CubeRegion, MetadataTargetRegion, ReleaseCalendar, ReferencePeriod
DataStructure	DataStructureComponent	N/A
ItemScheme	Item	N/A
MetadataStructure	MetadataTarget, ReportStructure	N/A
HierarchicalCodelist	Hierarchy	IncludedCodelist
Dataflow, Metadataflow	N/A	Structure
ProvisionAgreement	N/A	StructureUsage, DataProvider
Process	ProcessStep (see Nested Items)	N/A
ReportingTaxonomy	ReportingCategory (see Nested Items)	N/A
CategoryScheme	Category (see Nested Items)	N/A

StructureSet	OrganisationSchemeMap, CategorySchemeMap, CodelistMap, ConceptSchemeMap, ReportingTaxonomyMap, HybridCodelistMap, StructureMap (*Maps may become Maintainable Artefacts in SDMX 3.0*)	RelatedStructure

This means that:

- When using `PUT` to submit an Artefact, the resulting Artefact MUST be exactly the same to the submitted one;
- When using `POST` to submit an Artefact, the resulting Artefact MUST follow the rules for updating, as stated above.

Client

In order to update Artefacts, the client:

- MAY set the `Accept` header to indicate the preferred response format;
- MUST set the `Content-type` header according to the format of the submitted Artefact;
- MUST include in the request body, one or more Maintainable Artefacts in the SDMX format indicated in the `Content-type` header and of the SDMX type indicated in the resource, i.e.:
  - any set of Maintainable Artefacts under resource `/structure/`
  - a set of specific type of Maintainable Artefacts under the corresponding resource type, e.g. for Codelists: `/structure/codelist/`
  - a set of Items, within an Item Scheme, as identified by the resource url, e.g. for Codelist `SDMX:CL_FREQ(1.0)`: e.g. `/structure/codelist/SDMX/CL_FREQ/1.0/`

Server

In response to Artefact(s) update, the server:

- MUST respond with `200` in case of successful update;
- MUST return a `SubmitStructureResponse` message with the result of the action(s), according to the `Accept` header, or the default, if the `Accept` type is not supported (currently available only in SDMX-ML 2.1);
- MUST set the `Content-type` according to the returned format;

- MUST respond with `422` in case of resource type mismatch, i.e. the resource type identified in the URL does not match to the Artefact type(s) of the included SDMX Artefact(s).

Response

The `SubmitStructureResponse` message must be returned in any case (success, failure).
In SDMX 2.1 this is defined as part of the `RegistryInterface` messages.
The details of the message are explained in section Response message, below.

The following matrix summarises the returned `HTTP` response codes.

Method	Exists	Is Final	Is Referenced	References exist/provided	Return Code
POST	T	F	F	T	`201` (successful) or `207` (partially successful)
POST	T	F	F	F	`409`
POST	T	F	T	T	`200` if update is possible - `409` if references would break
POST	T	T	-	-	`409` (only for structural updates)
POST	-	-	-	-	`422` see section Server

T: True, F: False, I: Irrelevant, -: Not applicable

DELETE

Always concerns one Maintainable Artefact or one Item.
For example:

- A fully identified Maintainable Artefact, e.g. `/structure/codelist/SDMX/CL_FREQ/1.0`
- A fully identified Item, e.g. `/structure/codelist/SDMX/CL_FREQ/1.0/M`

Client

In order to delete an Artefact, the client:

- MAY set the `Accept` header to indicate the preferred response format;
- MUST fully identify exactly one Maintainable Artefact or one Item, by means of the proper URL;

Server

In response to an Artefact deletion, the server:

- MUST respond with `200` in case of successful deletion;
- MUST return a `SubmitStructureResponse` message with the result of the action, according to the `Accept` header, or the default, if the `Accept` type is not supported (currently available only in SDMX-ML 2.1);
- MUST respond with `404` if the resource was not found;

Response

The `SubmitStructureResponse` message must be returned in any case (success, failure).
In SDMX 2.1 this is defined as part of the `RegistryInterface` messages.
The details of the message are explained in section Response message, below.

The following matrix summarises the returned `HTTP` response codes.

Method	Exists	Is Final	Is Referenced	Refs exist/ provided	Response Code
DELETE	F	-	-	-	`404`
DELETE	T	F	F	I	`200`
DELETE	T	T	I	I	`409`
DELETE	T	F	T	I	`409`

T: True, F: False, I: Irrelevant, -: Not applicable

Response message

The `SubmitStructureResponse` message must be returned in any case (success, partial success,

failure).
In SDMX 2.1 this is defined as part of the `RegistryInterface` messages.
This message includes the following information per submitted Artefact:

- The `action`, e.g. `Append`, `Replace` or `Delete` (`Information` is also available)
- A reference to a specific Maintainable Artefact
- A status message with the result of the action, which contains:
    - The status, e.g. `Success`, `Failure` or `Warning`
    - One or more message texts to explain the result (we need only one per Artefact), which in turn contains:
    - A code (could be the HTTP code)
    - A multilingual text message

An example is shown below:

```
<reg:SubmissionResult>
  <reg:SubmittedStructure action="Append"> <!-- Append|Delete|Replace -->
    <reg:MaintainableObject>
      <Ref agencyID="SDMX" id="CL_FREQ" version="1.0"
        package="codelist" class="Codelist" />
    </reg:MaintainableObject>
  </reg:SubmittedStructure>
  <reg:StatusMessage status="Success"> <!-- Success|Failure|Warning -->
    <reg:MessageText code="204"> <!-- Could be the HTTP code 200, 201, 204, ... -->
      <com:Text xml:lang="en">Codelist successfully deleted</com:Text>
      <com:Text xml:lang="en">Codelist supprimé avec succès</com:Text>
    </reg:MessageText>
  </reg:StatusMessage>
</reg:SubmissionResult>
```

Especially when different results occur on the Artefacts (e.g. partial success), the following should occur:

- Return a multi-status return code (like `207`);
- Return a JSON/XML message with the results details (currently available only in SDMX-ML 2.1);

In the case of a multi-status response, the `SubmitStructureResponse` message will include the corresponding code per Artefact, e.g.:

```
<reg:SubmissionResult>
  <reg:SubmittedStructure action="Append">
    <reg:MaintainableObject>
      <Ref agencyID="SDMX" id="CODELIST" version="1.0"
```

```xml
        package="codelist" class="Codelist"/>
      </reg:MaintainableObject>
    </reg:SubmittedStructure>
    <reg:StatusMessage status="Warning">
      <reg:MessageText code="201">
        <com:Text xml:lang="en">Successfully created Codelist</com:Text>
      </reg:MessageText>
    </reg:StatusMessage>
  </reg:SubmissionResult>

  <reg:SubmissionResult>
    <reg:SubmittedStructure action="Delete"> <!-- Append|Delete|Replace -->
      <reg:MaintainableObject>
        <Ref agencyID="SDMX" id="CL_FREQ" version="1.0"
          package="codelist" class="Codelist" />
      </reg:MaintainableObject>
    </reg:SubmittedStructure>
    <reg:StatusMessage status="Success"> <!-- Success|Failure|Warning -->
      <reg:MessageText code="204"> <!-- Could be the HTTP code 200, 201, 204, ... -->
        <com:Text xml:lang="en">Codelist successfully deleted</com:Text>
        <com:Text xml:lang="en">Codelist supprimé avec succès</com:Text>
      </reg:MessageText>
    </reg:StatusMessage>
  </reg:SubmissionResult>
```

Nested Items

This section aims at explaining the particularities of nested Items for a subset of the available Item Schemes, namely:

- Category Scheme (Category)
- Process (ProcessStep)
- ReportingTaxonomy (ReportingCategory)

In all the above cases, Items may contain other Items in a tree-like hierarchy. As a result, the resource for an Item within such an hierarchy need to inlcude the full path of that Item in order to exactly identify it.
For example, for the following Category Scheme (excerpt of SDMX:STAT_SUBJECT_MATTER(1.0) from the Global SDMX Registry):

```xml
  <str:CategoryScheme agencyID="SDMX" id="STAT_SUBJECT_MATTER" version="1.0">
      <com:Name xml:lang="en">SDMX Statistical Subject-Matter Domains</com:Name>
      <str:Category id="DEMO_SOCIAL_STAT">
          <com:Name xml:lang="en">Demographic and social statistics</com:Name>
      </str:Category>
      <str:Category id="ECO_STAT">
          <com:Name xml:lang="en">Economic statistics</com:Name>
```

```
            <str:Category id="MACROECO_STAT">
                <com:Name xml:lang="en">Macroeconomic statistics</com:Name>
            </str:Category>
            <str:Category id="SECTORAL_STAT">
                <com:Name xml:lang="en">Sectoral statistics</com:Name>
                <str:Category id="AGRI_FOREST_FISH">
                    <com:Name xml:lang="en">Agriculture, forestry, fisheries</com:Name>
                </str:Category>
                <str:Category id="ENERGY">
                    <com:Name xml:lang="en">Energy</com:Name>
                </str:Category>
            </str:Category>
            <str:Category id="GOV_FINANCE_PUBLIC_SECTOR">
                <com:Name xml:lang="en">Government finance, fiscal and public sector
            statistics</com:Name>
            </str:Category>
        </str:Category>
        <str:Category id="ENVIRONMENT_MULTIDOMAIN_STAT">
            <com:Name xml:lang="en">Environment and multi-domain statistics</com:Name>
        </str:Category>
    </str:CategoryScheme>
```

In order to get Item `ENERGY` we need to request the following resource:
`categoryscheme/SDMX/CAT/1.0/ECO_STAT.SECTORAL_STAT.ENERGY`
Instead of the identifier of the Item, the full path of identifiers that lead to that Item are required, i.e.: `ECO_STAT` -> `SECTORAL_STAT` -> `ENERGY`.

At the time of this writing there is an open issue of how an SDMX Web Service should behave when requeting such an Item, i.e. being nested in a tree-like hierarchy (https://github.com/sdmx-twg/sdmx-rest/issues/92). The question is whether all container and containing (ancestors & descendants) of the Item must be returned.

Similarly, it has to be clarified what happens when such an Item is submitted for updating. It is even a bit more complex, since when submitting a specific Item (using `PUT` or `POST`) while targeting one Category, the body shall include all ancestors (maybe also its descendants) of this Item within its hierarchy. In that case, is it required to update the ancestors (and descendants) if they are different, or should the Web Service deal only with the details of the specific Item?

Summary of HTTP response codes

Method	Exists	Is Final	Is Referenced	Refs exist/ provided	Response Code

POST/PUT	F	-	-	T	`201` (successful) or `207` (partially successful)
POST/PUT	F	-	-	F	`409` failed references
PUT	T	F	F	T	`200`
POST	T	F	F	T	`201` (successful) or `207` (partially successful)
POST/PUT	T	F	F	F	`409`
POST/PUT	T	F	T	T	`200` if update is possible - `409` if references would break
POST/PUT	T	T	-	-	`409` (only for structural updates)
POST/PUT	-	-	-	-	`422` see sections Server
DELETE	F	-	-	-	`404`
DELETE	T	F	F	I	`200`
DELETE	T	T	I	I	`409`
DELETE	T	F	T	I	`409`

T: True, F: False, I: Irrelevant, -: Not applicable

Examples

Difference between `PUT` and `POST`

To explain the updating semantics, the difference between using `PUT` and `POST` for the same SDMX Artefact shall be illustrated in the following example.
For the sake of simplicity, a Codelist will be utilised, i.e. let's assume:

```xml
<str:Codelist agencyID="SDMX" id="CL_DECIMALS" version="1.0">
   <com:Name>Code list for Decimals (DECIMALS)</com:Name>
   <com:Description xml:lang="en">It provides a list of values showing the
     number of decimal digits used in the data.</com:Description>
   <str:Code id="0">
      <com:Name>Zero</com:Name>
   </str:Code>
   <str:Code id="1">
      <com:Name>One</com:Name>
   </str:Code>
   <str:Code id="2">
      <com:Name>Two</com:Name>
   </str:Code>
</str:Codelist>
```

Let's assume that we `PUT` the following Codelist, under `/codelist/SDMX/CL_DECIMALS/1.0`:

```xml
<str:Codelist agencyID="SDMX" id="CL_DECIMALS" version="1.0">
   <com:Name>Code list for Decimals (DECIMALS)</com:Name>
   <com:Description xml:lang="en">It provides a list of values showing the
     number of decimal digits used in the data.</com:Description>
   <str:Code id="0">
      <com:Name>No decimal</com:Name>
   </str:Code>
   <str:Code id="1">
      <com:Name>One</com:Name>
   </str:Code>
 </str:Codelist>
```

This will result into replacing the original Codelist with the one submitted. Hence, in the new Codelist only two Codes will exist, the first one (Code `0`) with an updated name.

If, instead, we used `POST` for the above Codelist, under `/codelist`, then the result would be a bit different. The new Codelist will still have three Codes, but the first one (Code `0`) would have an updated name, i.e.:

```xml
<str:Codelist agencyID="SDMX" id="CL_DECIMALS" version="1.0">
   <com:Name>Code list for Decimals (DECIMALS)</com:Name>
   <com:Description xml:lang="en">It provides a list of values showing the
     number of decimal digits used in the data.</com:Description>
   <str:Code id="0">
      <com:Name>No decimal</com:Name>
   </str:Code>
   <str:Code id="1">
      <com:Name>One</com:Name>
   </str:Code>
```

```xml
    <str:Code id="2">
      <com:Name>Two</com:Name>
    </str:Code>
  </str:Codelist>
```