SDMX STANDARDS: SECTION 5

SDMX REGISTRY SPECIFICATION: LOGICAL FUNCTIONALITY AND LOGICAL INTERFACES

VERSION 3.1

May 2025

Revision History

Revision	Date	Contents
DRAFT 1.0	December 2024	Draft release updated for SDMX 3.1 for public consultation
1.0	May 2025	Public release for SDMX 3.1

1

Contents

1 Introduction

The business vision for SDMX envisages the promotion of a “data sharing” model to facilitate low-cost, high-quality statistical data and metadata exchange. Data sharing reduces the reporting burden of organisations by allowing them to publish data once and let their counterparties “pull” data and related metadata as required. The scenario is based on:

- the availability of an abstract information model capable of supporting time series and cross-sectional data, structural metadata, and reference metadata (SDMX-IM)
- standardised XML and JSON schemas for the SDMX-ML and SDMX-JSON formats derived from the model (XSD, JSON)
- the use of web-services technology (XML, JSON, Open API)

Such an architecture needs to be well organised, and the SDMX Registry/Repository (SDMX-RR) is tasked with providing structure, organisation, and maintenance and query interfaces for most of the SDMX components required to support the data sharing vision.

However, it is important to emphasise that the SDMX-RR provides support for the submission and retrieval of all SDMX structural metadata and provisioning metadata. Therefore, the Registry not only supports the data-sharing scenario, but this metadata is also vital in order to provide support for data and metadata reporting/collection, and dissemination scenarios.

Standard formats for the exchange of aggregated statistical data and metadata as prescribed in SDMX v3.1 are envisaged to bring benefits to the statistical community because data reporting and dissemination processes can be made more efficient.

As organisations migrate to SDMX enabled systems, many XML, JSON (and conventional) artefacts will be produced (e.g., Data Structure, Metadata Structure, Code List and Concept definitions – often collectively called structural metadata – XML schemas generated from data structure definitions, XSLT stylesheets for transformation and display of data and metadata, terminology references, etc.). The SDMX model supports interoperability, and it is important to be able to discover and share these artefacts between parties in a controlled and organized way.

This is the role of the registry.

With the fundamental SDMX standards in place, a set of architectural standards are needed to address some of the processes involved in statistical data and metadata exchange, with an emphasis on maintenance, retrieval and sharing of the structural metadata. In addition, the architectural standards support the registration and discovery of data and referential metadata.

These architectural standards address the ‘how’, rather than the ‘what’, and are aimed at enabling existing SDMX standards to achieve their mission. The architectural standards address registry services, which initially comprise:

- structural metadata repository

- data and metadata registration
- query

The registry services outlined in this specification are designed to help the SDMX community manage the proliferation of SDMX assets and to support data sharing for reporting and dissemination.

2 Scope and Normative Status

The scope of this document is to specify the logical interfaces for the SDMX registry in terms of the functions required and the data that may be present in the function call, and the behaviour expected of the registry.

In this document, functions and behaviours of the Registry Interfaces are described in four ways:

- in text
- with tables
- with UML diagrams excerpted from the SDMX Information Model (SDMX-IM)
- with UML diagrams that are not a part of the SDMX-IM but are included here for clarity and to aid implementations (these diagrams are clearly marked as “Logical Class Diagram ...”)

Whilst the introductory section contains some information on the role of the registry, it is assumed that the reader is familiar with the uses of a registry in providing shared metadata across a community of counterparties.

Note that chapters 5 and 6 below contain normative rules regarding the Registry Interface and the identification of registry objects. Further, the minimum standard for access to the registry is via a REST interface (HTTP or HTTPS), as described in the appropriate sections. The notification mechanism must support e-mail and HTTP/HTTPS protocols as described. Normative registry interfaces are specified in the SDMX-ML specification (Section 3 of the SDMX Standard). All other sections of this document are informative.

Note that although the term “authorised user” is used in this document, the SDMX standards do not define an access control mechanism. Such a mechanism, if required, must be chosen and implemented by the registry software provider.

3 Scope of the SDMX Registry/Repository

3.1 Objective

The objective of the SDMX registry/repository is, in broad terms, to allow organisations to publish statistical data and reference metadata in known formats such that interested third parties can discover these data and interpret them accurately and correctly. The mechanism for doing this is twofold:

1. To maintain and publish structural metadata that describes the structure and valid content of data and reference metadata sources such as databases, metadata repositories, data sets, metadata sets. This structural metadata enables software applications to understand and to interpret the data and reference metadata in these sources.

2. To enable applications, organisations, and individuals to share and to discover data and reference metadata. This facilitates data and reference metadata dissemination by implementing the data sharing vision of SDMX.

3.2 Structural Metadata

Setting up structural metadata and the exchange context (referred to as “data provisioning”) involves the following steps for maintenance agencies:

- agreeing and creating a specification of the structure of the data (called a Data Structure Definition or DSD in this document but also known as “key family”), which defines the dimensions, measures and attributes of a dataset and their valid value set;

- if required, defining a subset or view of a DSD which allows some restriction of content called a “dataflow definition”;

- agreeing and creating a specification of the structure of reference metadata (Metadata Structure Definition) which defines the metadata attributes and their presentational arrangement in a Metadataset or as part of a Dataset, and their valid values and content;

- if required, defining a subset or view of an MSD which allows some restriction of content called a “metadataflow”;

- defining which subject matter domains (specified as a Category Scheme) are related to the Dataflow and Metadataflow to enable browsing;

- defining one or more lists of Data and Metadata Providers;

- defining which Data/Metadata Providers have agreed to publish a given Dataflow/Metadataflow – this is called a Provision Agreement or Metadata Provision Agreement, respectively.

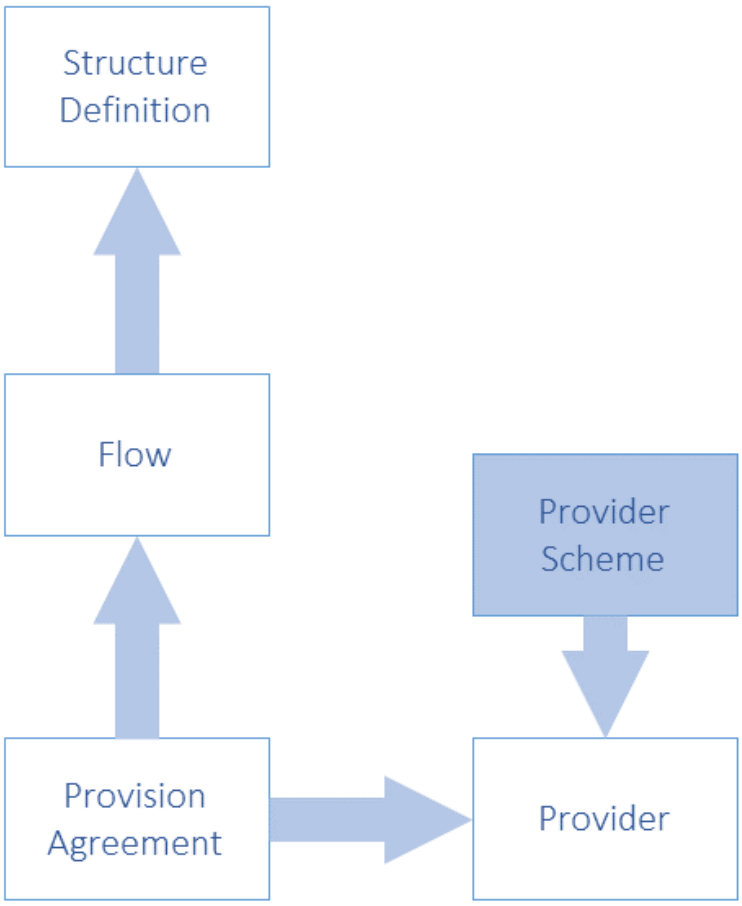


Figure 1: Schematic of the Basic Structural Artefacts in the SDMX-IM

Note that in Figure 1 (but also most of the relevant subsequent figures) terms that include both data and metadata have been used. For example:

- Structure Definition: refers to Data Structure Definition (DSD) and Metadata Structure Definition (MSD)
- Flow: refers to Dataflow and Metadataflow
- Provision Agreement: refers to Provision Agreement (for data) and Metadata Provision Agreement
- Provider Scheme: refers to Data Provider Scheme and Metadata Provider Scheme
- Provider: refers to Data Provider and Metadata Provider

In that context, the term “Metadata” refers to reference metadata.

3.3 Registration

Publishing the data and reference metadata involves the following steps for a Data/Metadata Provider:

- making the reference metadata and data available in SDMX-ML/JSON conformant data files or databases (which respond to an SDMX query with data). The data and reference metadata files or databases must be web accessible, and must conform to an agreed Dataflow or Metadataflow (Data Structure Definition or Metadata Structure Definition subset);

- registering the existence of published reference metadata and data files or databases with one or more SDMX registries.

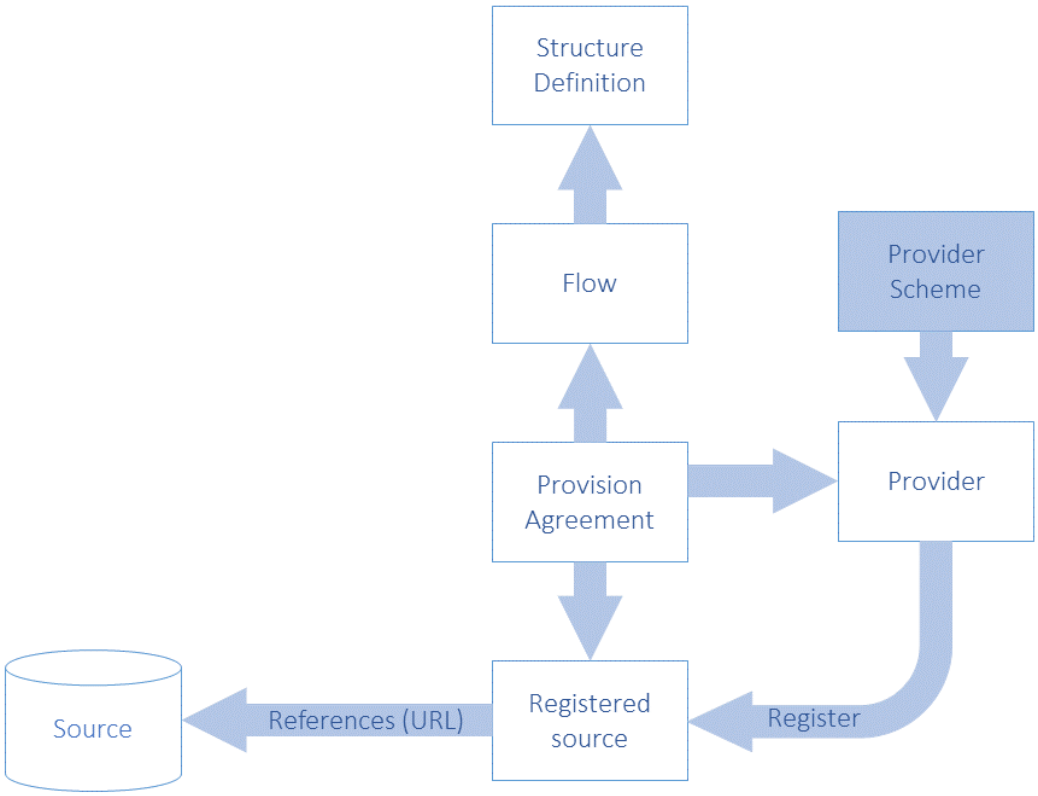


Figure 2: Schematic of Registered Data and Metadata Sources in the SDMX-IM

3.4 Notification

Notifying interested parties of newly published or re-published data, reference metadata or changes in structural metadata involves:

- registry support of a subscription-based notification service which sends an email or notifies an HTTP address announcing all published data that meets the criteria contained in the subscription request.

3.5 Discovery

Discovering published data and reference metadata involves interaction with the registry to fulfil the following logical steps that would be carried out by a user interacting with a service that itself interacts with the registry and an SDMX-enabled data or reference metadata resource:

- optionally browsing a subject matter domain category scheme to find Dataflows (and hence Data Structure Definitions) and Metadataflows which structure the type of data and/or reference metadata being sought;
- build a query, in terms of the selected Data Structure Definition or Metadata Structure Definition, which specifies what data are required and submitting this to a service that can query an SDMX registry which will return a list of (URLs of) data and reference metadata files and databases which satisfy the query;
- processing the query result set and retrieving data and/or reference metadata from the supplied URLs.

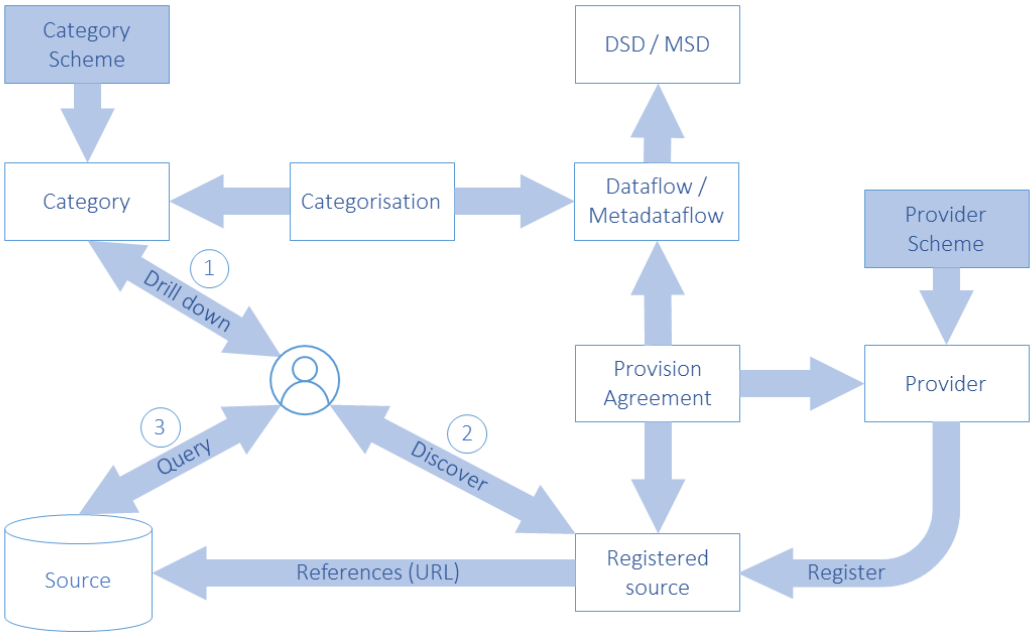


Figure 3: Schematic of Data and Metadata Discovery and Query in the SDMX-IM

4 SDMX Registry/Repository Architecture

4.1 Architectural Schematic

The architecture of the SDMX registry/repository is derived from the objectives stated above. It is a layered architecture that is founded by a structural metadata repository which supports a provisioning metadata repository which supports the registry services. These are all supported by the SDMX-ML schemas. Applications can be built on top of these services which support the reporting, storage, retrieval, and dissemination aspects of the statistical lifecycle as well as the maintenance of the structural metadata required to drive these applications.

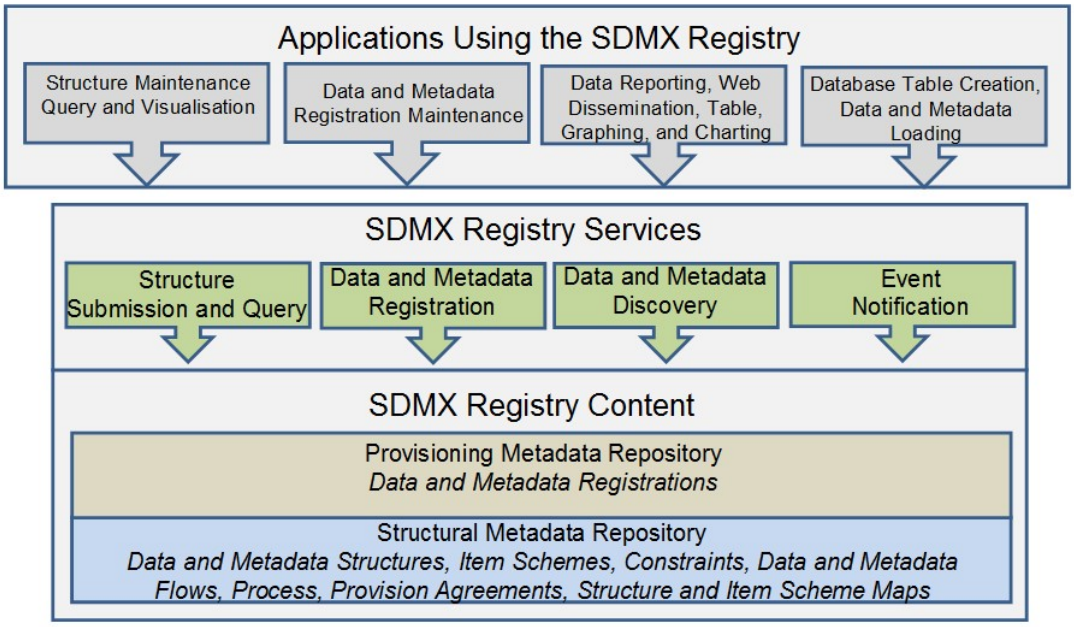


Figure 4: Schematic of the Registry Content and Services

4.2 Structural Metadata Repository

The basic layer is that of a structural metadata service which supports the lifecycle of SDMX structural metadata artefacts such as Maintenance Agencies, Data Structure Definitions, Metadata Structure Definitions, Provision Agreements, Processes etc. This layer is supported by the Structure Submission and Query Service.

Note that the SDMX REST API supports all of the SDMX structural artefacts. The only structural artefacts that are not yet supported are:

- Registration of data and metadata sources
- Subscription and Notification

As of the initial version of SDMX 3.0 no messages are defined to support these artefacts; hence, users may need to use SDMX 2.1 Registry Interface messages, instead.

4.3 Provisioning Metadata Repository

The function of this repository is to support the definition of the structural metadata that describes the various types of data-store which model SDMX-conformant databases or files, and to link to these data sources. These links can be specified for a data/metadata provider, for a specific data or metadata flow. In the SDMX model this is called the Provision or Metadata Provision Agreement.

This layer is supported by the Data and Metadata Registration Service.

5 Registry Interfaces and Services

5.1 Registry Interfaces

The Registry Interfaces are:

- Notify Registry Event
- Submit Subscription Request
- Submit Subscription Response
- Submit Registration Request
- Submit Registration Response
- Query Registration Request
- Query Registration Response
- Query Subscription Request
- Query Subscription Response

The registry interfaces are invoked in one of two ways:

1. The interface is the name of the root node of the SDMX-ML document
2. The interface is invoked as a child element of the `RegistryInterface` message where the `RegistryInterface` is the root node of the SDMX-ML document.

In addition to these interfaces the registry must support a mechanism for submitting and querying for structural metadata. This is detailed in sections 5.2.2 and 5.2.3.

All these interactions with the Registry – with the exception of `NotifyRegistryEvent` – are designed in pairs. The first document, the one which invokes the SDMX-RR interface, is a "Request" document. The message returned by the interface is a "Response" document.

It should be noted that all interactions are assumed to be synchronous, with the exception of Notify Registry Event. This document is sent by the SDMX-RR to all subscribers whenever an even occurs to which any users have subscribed. Thus, it does not conform to the request-response pattern, because it is inherently asynchronous.

5.2 Registry Services

5.2.1 Introduction

The services described in this section do not imply that each is implemented as a discrete web service.

5.2.2 Structure Submission Service

The registry must support a mechanism for submitting structural metadata. This mechanism can be the SDMX REST interface for structural metadata (this is defined in the corresponding GitHub project, dedicated to the SDMX REST API: https://github.com/sdmx-twg/sdmx-rest). In order for the architecture to be scalable, the finest-grained piece of structural metadata that can be processed by the SDMX-RR is a `MaintainableArtefact`, with the exception of Item Schemes, where changes at an Item level is also possible (see next section on the SDMX Information Model).

5.2.3 Structure Query Service

The registry must support a mechanism for querying for structural metadata. This mechanism can be the SDMX REST interface for structural metadata (this is defined in the corresponding GitHub project, dedicated to the SDMX REST API: https://github.com/sdmx-twg/sdmx-rest). The registry response to this query mechanism is the SDMX Structure message, which has as its root node:

- `Structure`

The SDMX structural artefacts that may be queried are defined in the SDMX structure web service specification which is compatible with this release of the standard

https://github.com/sdmx-twg/sdmx-rest/blob/master/doc/structures.md

5.2.4 Data and Reference Metadata Registration Service

This service must implement the following Registry Interfaces:

- `SubmitRegistrationRequest`
- `SubmitRegistrationResponse`
- `QueryRegistrationRequest`
- `QueryRegistrationResponse`

The Data Registration Service allows SDMX conformant files and web-accessible databases containing published data and reference metadata to be registered in the SDMX Registry. The registration process MAY validate the content of the datasets or metadata-sets, and MAY extract a concise representation of the contents in terms of concept values (e.g., values of the data attribute, dimension, metadata attribute), or entire keys, and storing this as a record in the registry to enable discovery of the original dataset or metadata-set. These are called Constraints in the SDMX-IM.

The Data Registration Service MAY validate the following, subject to the access control mechanism implemented in the Registry:

- that the data/metadata provider is allowed to register the dataset or metadataset;
- that the content of the dataset or metadataset meets the validation constraints. This is dependent upon such constraints being defined in the structural repository and which reference the relevant Dataflow, Metadataflow, Data Provider, Metadata Provider, Data Structure Definition, Metadata Structure Definition, Provision Agreement, Metadata Provision Agreement;
- that a queryable data source exists – this would necessitate the registration service querying the service to determine its existence;
- that a simple data source exists (i.e., a file accessible at a URL);
- that the correct Data Structure Definition is used by the registered data;
- that the components (Dimensions, Attributes, Measures) are consistent with the Data Structure Definition;
- that the valid representations of the concepts to which these components correspond conform to the definition in the Data Structure Definition.

The Registration has an action attribute which takes one of the following values:

Action Attribute Value	**Behaviour**
`Append`	Add this registration to the registry
`Replace`	Replace the existing Registration with this Registration identified by the id in the Registration of the Submit Registration Request
`Delete`	Delete the existing Registration identified by the id in the Registration of the Submit Registration Request

The Registration has three Boolean attributes which may be present to determine how an SDMX compliant dataset or metadataset indexing application must index the datasets or metadatasets upon registration. The indexing application behaviour is as follows:

Boolean Attribute	**Behaviour if Value is "true"**
`indexTimeSeries`	A compliant indexing application must index all the time series keys

`indexDataSet`	A compliant indexing application must index the range of actual (present) values for each dimension of the Dataset. Note that for data this requires much less storage than full key indexing, but this method cannot guarantee that a specific combination of Dimension values (the Key) is actually present in the Dataset
`indexReportingPeriod`	A compliant indexing application must index the time period range(s) for which data are present in the Dataset.

5.2.5 Data Discovery

The Data Discovery Service implements the following Registry Interfaces:

- `QueryRegistrationRequest`
- `QueryRegistrationResponse`

5.2.6 Subscription and Notification

The Subscription and Notification Service implements the following Registry Interfaces:

- `SubmitSubscriptionRequest`
- `SubmitSubscriptionResponse`
- `NotifyRegistryEvent`

The data sharing paradigm relies upon the consumers of data and metadata being able to pull information from data providers' dissemination systems. For this to work efficiently, a data consumer needs to know when to pull data, i.e., when something has changed in the registry (e.g., a dataset has been updated and re-registered). Additionally, SDMX systems may also want to know if a new Data Structure Definition, or Code List has been added. The Subscription and Notification Service comprises two parts: subscription management, and notification.

Subscription management involves a user submitting a subscription request which contains:

- a query or constraint expression in terms of a filter which defines the events for which the user is interested (e.g., new data for a specific dataflow, or for a domain category, or changes to a Data Structure Definition).
- a list of URIs or endpoints to which an XML notification message can be sent. Supported endpoint types will be email (mailto:) and HTTP POST (a normal http:// address);

- request for a list of submitted subscriptions;
- deletion of a subscription;

Notification requires that the structural metadata repository and the provisioning metadata repository monitor any event which is of interest to a user (the object of a subscription request query), and to issue an SDMX notification document to the endpoints specified in the relevant subscriptions.

5.2.7 Registry Behaviour

The following table defines the behaviour of the SDMX Registry for the various Registry Interface messages. It should be noted, though, that as of SDMX 3.0, an extended versioning scheme newly including semantic versioning is foreseen for all Maintainable Artefacts. Moreover, while the old versioning scheme is allowed, given there is no more a "final" flag, there is no way guaranteeing the consistency across version of a Maintainable, unless semantic versioning is used.

Given the above, the behaviour described in the following table concerns either draft Artefacts using semantic versioning or any Artefacts using the old versioning scheme. Nevertheless, in the case of semantic versioning the registry must respect the versioning rules when performing the actions below. For example, it is not possible to replace a non-draft Artefact that follows semantic versioning, unless a newer version is introduced according to the semantic versioning rules. Furthermore, even when draft Artefacts are submitted, the registry has to verify semantic versioning is respected against the previous non-draft versions. It is worth noting that the rules for semantic versioning and replacing or maintaining semantically versioned Artefacts applies to externally shared Artefacts. This means that any system may internally perform any change within a version of an Artefact, until the latter is shared outside of that system or becomes public. Then (as also explained in the SDMX Standards Section 6 “Technical Notes”) the Artefacts must adhere to the Semantic Versioning rules.

Interface	**Behaviour**
All	1) If the action is set to “replace” (or a maintainable Artefact is PUT or POSTed) then the entire contents of the existing maintainable object in the Registry MUST be replaced by the object submitted. 2) Cross referenced structures MUST exist in either the submitted document (in Structures or Structure Location) or in the registry to which the request is submitted. 3) If the action is set to “delete” (or a maintainable Artefact is DELETEd) then the Registry MUST verify that the object can be deleted. In order to qualify for deletion, the object must: a) Be a draft version.

Interface	Behaviour
	b) Not be explicitly[1] referenced from any other object in the Registry. 4) The semantic versioning rules in the SDMX documentation MUST be obeyed.
Structure submission	Structures are submitted at the level of the Maintainable Artefact and the behaviour in “All” above is therefore at the level of the Maintainable Artefact.
`SubmitRegistrationRequest`	If the datasource is a file (simple datasource) then the file MAY be retrieved and indexed according to the Boolean attributes set in the Registration. For a queryable datasource the Registry MAY validate that the source exists and can accept an SDMX data query.

[1] With semantic versioning, it is allowed to reference a range of artefacts, e.g., a DSD referencing a Codelist with version 1.2.3+ means all patch versions greater than 1.2.3. This means that deleting 1.2.4-draft does not break integrity of the aforementioned DSD.

6 Identification of SDMX Objects

6.1 Identification, Versioning, and Maintenance

All major classes of the SDMX Information model inherit from one of:

- ***IdentifiableArtefact*** – this gives an object the ability to be uniquely identified (see following section on identification), to have a user-defined URI, and to have multi-lingual annotations.
- ***NameableArtefact*** – this has all of the features of *IdentifiableArtefact* plus the ability to have a multi-lingual name and description.
- ***VersionableArtefact*** – this has all of the above features plus a version number, according to the SDMX versioning rules in SDMX Standards Section 6 “Technical Notes”, paragraph “4.3 Versioning”, and a validity period.
- ***MaintainableArtefact*** – this has all of the above features, plus registry and structure URIs, and an association to the maintenance organisation of the object.

6.1.1 Identification, Naming, Versioning, and Maintenance Model

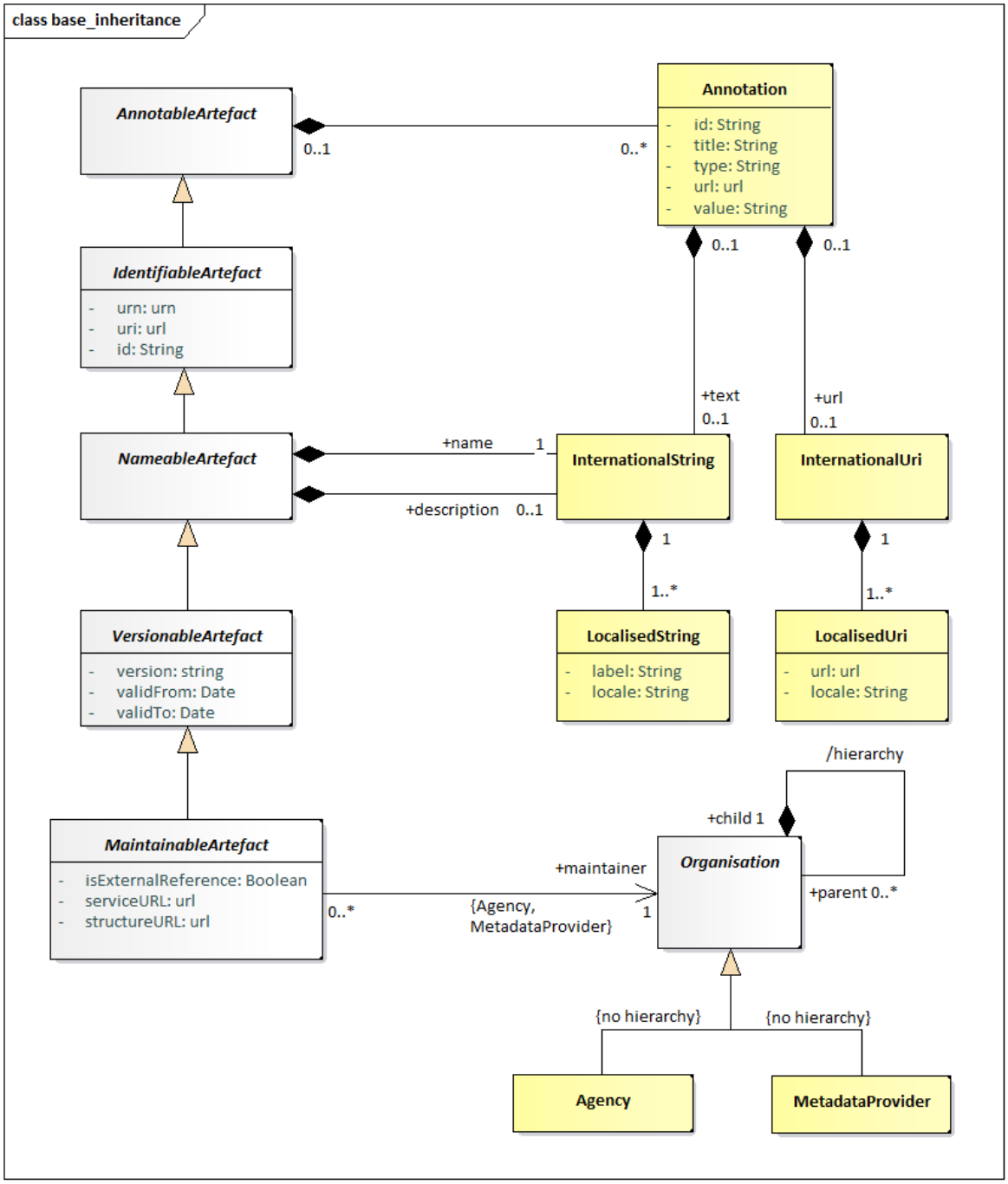


Figure 5: Class diagram of fundamental artefacts in the SDMX-IM

The table below shows the identification and related data attributes to be stored in a registry for objects that are one of:

- *Annotable*
- *Identifiable*

- *Nameable*
- *Versionable*
- *Maintainable*

Object Type	Data Attributes	Status	Data type	Notes
Annotable	AnnotationTitle	C	string	
	AnnotationType	C	string	
	AnnotationURN	C	string	
	AnnotationText in the form of InternationalString	C		This can have language-specific variants
Identifiable	All content as for *Annotable* plus			
	id	M	string	
	uri	C	string	
	urn	C	string	Although the urn is computable and therefore may not be submitted or stored physically, the Registry must return the urn for each object, and must be able to service a query on an object referenced solely by its urn.
Nameable	All content as for *Identifiable* plus			
	Name in the form of InternationalString	M	string	This can have language specific variants.
	Description in the form of InternationalString	C	string	This can have language specific variants.
Versionable	All content as for *Identifiable* plus			
	version	M	string	This is the version number according to SDMX versioning rules.
	validFrom	C	Date/time	
	validTo	C	Date/time	

Maintainable	All content as for *Versionable* plus			
	isExternalReference	C	boolean	Value of “true” indicates that the actual resource is held outside of this registry. The actual reference is given in the registry URI or the structureURL, each of which must return a valid SDMX-ML file.
	serviceURL	C	string	The url of the service that can be queried for this resource.
	structureURL	C	string	The url of the resource.
	(Maintenance) organisationId	M	string	The object must be linked to a maintenance organisation, i.e., Agency or Metadata Provider.

Table 1: Common Attributes of Object Types

6.2 Unique identification of SDMX objects

6.2.1 Agencies and Metadata Providers

The Maintenance Agency in SDMX is maintained in an Agency Scheme which itself is a sub class of Organisation Scheme – this is shown in the class diagram below.

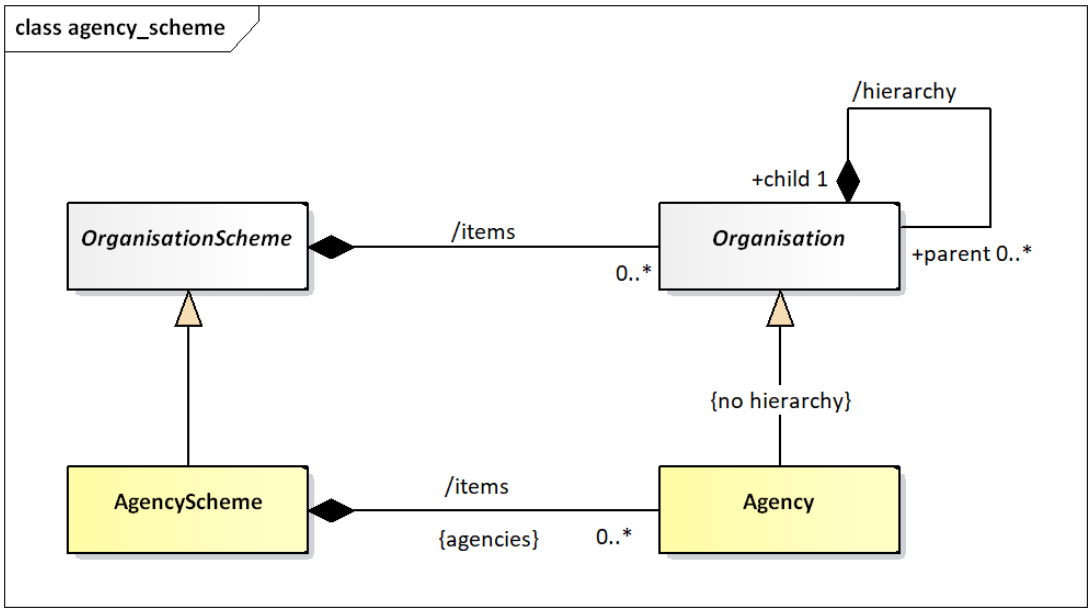


Figure 6: Agency Scheme Model

The Agency in SDMX is extremely important. The Agency Id system used in SDMX is an n-level structure. The top level of this structure is maintained by SDMX. Any Agency in this top level can declare sub agencies and any sub agency can also declare sub agencies. The Agency Scheme has a fixed id and version (version '1.0') and is never declared explicitly in the SDMX object identification mechanism.

In order to achieve this SDMX adopts the following rules:

- Agencies are maintained in an Agency Scheme (which is a sub class of Organisation Scheme).
- The agency of the Agency Scheme must also be declared in a (different) Agency Scheme.
- The "top-level" agency is SDMX and maintains the "top-level" Agency Scheme.
- Agencies registered in the top-level scheme can themselves maintain a single Agency Scheme. Agencies in these second-tier schemes can themselves maintain a single Agency Scheme and so on.
- The `AgencyScheme` has a fixed version, i.e., '1.0', hence it is an exception from the Semantic Versioning that other Artefacts follow.
- There can be only one `AgencyScheme` maintained by any one Agency. It has a fixed `id` of `AGENCIES`.

- The `/hierarchy` of Organisation is not inherited by Maintenance Agency – thus each Agency Scheme is a flat list of Maintenance Agencies.

- The format of the agency identifier is `agencyID.agencyID` etc. The top-level agency in this identification mechanism is the agency registered in the SDMX agency scheme. In other words, SDMX is not a part of the hierarchical ID structure for agencies. However, SDMX is, itself, a maintenance agency and is contained in the top-level Agency Scheme.

This supports a hierarchical structure of `agencyID`.

An example is shown below.

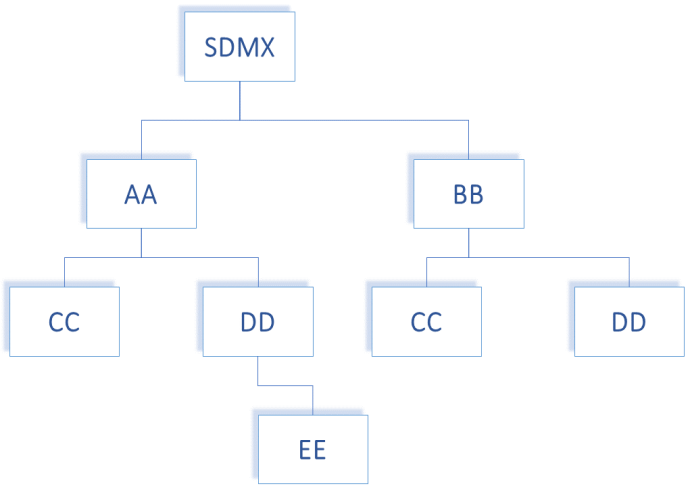


Figure 7: Example of Hierarchic Structure of Agencies

The following organizations maintain an Agency Scheme.

- SDMX – contains Agencies AA, BB
- AA – contains Agencies CC, DD
- BB – contains Agencies CC, DD
- DD – Contains Agency EE

Each agency is identified by its full hierarchy excluding SDMX.

e.g., the id of EE as an `agencyID` is AA.DD.EE

An example of this is shown in the XML snippet below:

```
<str:Codelists>
    <str:Codelist id="CL_FREQ" agencyID="SDMX" version="1.0.0">
        <com:Name xml:lang="en">Standard frequency Codelist</com:Name>
    </str:Codelist>
```

```
<str:Codelist id="CL_FREQ" agencyID="AA" version="1.0.0">
    <com:Name xml:lang="en">Codelist maintained by agency AA</com:Name>
</str:Codelist>
<str:Codelist id="CL_FREQ" agencyID="AA.CC" version="1.0.0">
    <com:Name xml:lang="en">Codelist maintained by the AA unit CC</com:Name>
</str:Codelist>
<str:Codelist id="CL_FREQ" agencyID="BB.CC" version="1.0.0">
    <com:Name xml:lang="en">Codelist maintained by the BB unit CC</com:Name>
</str:Codelist>
```

Figure 8: Example Showing Use of Agency Identifiers

Each of these maintenance agencies has an identical Code list with the Id CL_BOP. However, each is uniquely identified by means of the hierarchic agency structure.

Following the same principles, the Metadata Provider is the maintenance organisation for a special subset of Maintainable Artefacts, i.e., the Metadatasets; the latter are the containers of reference metadata combined with a target that those metadata refer to.

6.2.2 Universal Resource Name (URN)

6.2.2.1 Introduction

To provide interoperability between SDMX Registry/Repositories in a distributed network environment, it is important to have a scheme for uniquely identifying (and thus accessing) all first-class (Identifiable) SDMX-IM objects. Most of these unique identifiers are composite (containing maintenance agency, or parent object identifiers), and there is a need to be able to construct a unique reference as a single string. This is achieved by having a globally unique identifier called a universal resource name (URN) which is generated from the actual identification components in the SDMX-RR APIs. In other words, the URN for any Identifiable Artefact is constructed from its component identifiers (agency, id, version etc.).

6.2.2.2 URN Structure

Case Rules for URN

For the URN, all parts of the string are case sensitive. The generic structure of the URN is as follows:

```
SDMXprefix.SDMX-IM-package-name.class-name=agencyid:maintainedobject-
id(maintainedobject-version).*containerobject-id.object-id
```

* this can repeat and may not be present (see explanation below)

Note that in the SDMX Information Model there are no concrete Versionable Artefacts that are not a Maintainable Artefact. For this reason, the only version information that is allowed is for the maintainable object.

The Maintenance agency identifier is separated from the maintainable artefact identifier by a colon ':'. All other identifiers in the SDMX URN syntax are separated by a period '.'. The version

information is encapsulated in parentheses '()' and adheres to the SDMX versioning rules, as explained in SDMX Standards Section 6 "Technical Notes", paragraph "4.3 Versioning.

6.2.2.3 Explanation of the generic structure

In the explanation below the actual object that is the target of the URN is called the **actual object**.

`SDMXPrefix`: `urn:sdmx:org`

`SDMX-IM-package-name`: `sdmx.infomodel.package=`

The packages are:

`base`

`codelist`

`conceptscheme`

`datastructure`

`categoryscheme`

`registry`

`metadatastructure`

`process`

`structuremapping`

`transformation`

`maintainable-object-id` is the identifier of the maintainable object. This will always be present as all identifiable objects are either a maintainable object or contained in a maintainable object.

`maintainable-object-version` is the version, according to the SDMX versioning rules, of the maintainable object and is enclosed in parentheses '()', which are always present.

`container-object-id` is the identifier of an intermediary object that contains the actual object which the URN is identifying. It is not mandatory as many actual objects do not have an intermediary container object. For instance, a `Code` is in a maintained object (`Codelist`) and has no intermediary container object, whereas a `MetadataAttribute` has an intermediary container object (`MetadataAttributeDescriptor`) and may have an intermediary container object, which is its parent `MetadataAttribute`. For this reason, the container object id may repeat, with each repetition identifying the object at the next-lower level in its hierarchy. Note that if there is only a single containing object in the model then it is NOT

included in the URN structure. This applies to `AttributeDescriptor`, `DimensionDescriptor`, and `MeasureDescriptor` where there can be only one such object and this object has a fixed id. Therefore, whilst each of these has a URN, the id of the `AttributeDescriptor`, `DimensionDescriptor`, and `MeasureDescriptor` is not included when the actual object is a `DataAttribute` or a `Dimension`/ `TimeDimension`, or a `Measure`.

Note that although a `Code` can have a parent `Code` and a `Concept` can have a parent `Concept` these are maintained in a flat structure and therefore do not have a `container-object-id`.

For example, the sequence is `agency:DSDid(version).DimensionId` and not `agency:DSDid(version).DimensionDescriptorId.DimensionId`.

`object-id` is the identifier of the actual object unless the actual object is a *`Maintainable`* object. If present it is always the last id and is not followed by any other character.

Generic Examples of the URN Structure

Actual object is a maintainable

```
SDMXPrefix.SDMX-IM-package-name.classname=agencyid:maintained-object-id(version)
```

Actual object is contained in a maintained object with no intermediate containing object

```
SDMXPrefix.SDMX-IM-package-name.classname=agencyid:maintained-object-id(version).object-id
```

Actual object is contained in a maintained object with an intermediate containing object

```
SDMXPrefix.SDMX-IM-package-name.classname=agencyid:maintained-object-id(version).contained-object-id.object-id
```

Actual object is contained in a maintained object with no intermediate containing object but the object type itself is hierarchical

In this case the object id may not be unique in itself but only within the context of the hierarchy. In the general syntax of the URN all intermediary objects in the structure (with the exception, of course, of the maintained object) are shown as a contained object. An example here would be a `Category` in a `CategoryScheme`. The `Category` is hierarchical, and all intermediate `Categories` are shown as a contained object. The example below shows the generic structure for `CategoryScheme`/ `Category`/ `Category`.

```
SDMXPrefix.SDMX-IM-package-name.classname=agencyid:maintained-object-id(version).contained-object-id.object-id
```

Actual object is contained in a maintained object with an intermediate containing object and the object type itself is hierarchical

In this case the generic syntax is the same as for the example above as the parent object is regarded as a containing object, even if it is of the same type. An example here is a `MetadataAttribute` where the contained objects are `MetadataAttributeDescriptor` (first contained object id) and `MetadataAttribute` (subsequent contained object ids). The example below shows the generic structure for `MSD/ MetadataAttributeDescriptor/ MetadataAttribute/ MetadataAttribute`

```
SDMXPrefix.SDMX-IM-package-name.classname=agencyid:maintained-object-id(version).contained-object-id.contained-object-id contained-object-id.object-id
```

Concrete Examples of the URN Structure

The Data Structure Definition CRED_EXT_DEBT of legacy version 2.1 maintained by the top-level Agency TFFS would have the URN:

```
urn:sdmx:org.sdmx.infomodel.datastructure.DataStucture=TFFS:CRED_EXT_DEBT(2.1)
```

The URN for a code for Argentina maintained by ISO in the code list CL_3166A2 of semantic version 1.0.0 would be:

```
urn:sdmx:org.sdmx.infomodel.codelist.Code=ISO:CL_3166A2(1.0.0).AR
```

The URN for a category (id of 1) which has parent category (id of 2) maintained by SDMX in the category scheme SUBJECT_MATTER_DOMAINS of the semantic extended version 1.0.0-draft would be:

```
urn:sdmx:org.sdmx.infomodel.categoryscheme.Category=SDMX:SUBJECT_MATTER_DOMAINS(1.0.0-draft).1.2
```

The URN for a Metadata Attribute maintained by SDMX in the MSD CONTACT_METADATA of semantic version 1.0.0 where the hierarchy of the Metadata Attribute is CONTACT_DETAILS/CONTACT_NAME would be:

```
urn:sdmx:org.sdmx.infomodel.metadatastructure.MetadataAttribute=SDMX:CONTACT_METADATA(1.0.0).CONTACT_DETAILS.CONTACT_NAME
```

The TFFS defines ABC as a sub-Agency of TFFS then the URN of a Dataflow maintained by ABC and identified as EXTERNAL_DEBT of semantic version 1.0.0 would be:

```
urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=TFFS.ABC:EXTERNAL_DEBT(1.0.0)
```

The SDMX-RR MUST support this globally unique identification scheme. The SDMX-RR MUST be able to create the URN from the individual identification attributes submitted and to transform the URN to these identification attributes. The identification attributes are:

- **Identifiable and Nameable Artefacts**: id (in some cases this id may be hierarchic)

- **Maintainable Artefacts**: id, version, agencyId

The SDMX-RR MUST be able to resolve the unique identifier of an SDMX artefact and to produce an SDMX-ML rendering of that artefact if it is located in the Registry.

6.2.3 Table of SDMX-IM Packages and Classes

The table below lists all of the packages in the SDMX-IM together with the concrete classes that are in these packages and whose objects have a URN.

Package	URN class name (model class name where this is different)
base	Agency
	AgencyScheme
	DataConsumer
	DataConsumerScheme
	DataProvider
	DataProviderScheme
	MetadataProvider
	MetadataProviderScheme
	OrganisationUnit
	OrganisationUnitScheme
datastructure	AttributeDescriptor
	DataAttribute
	Dataflow
	DataStructure (DataStructureDefinition)
	Dimension
	DimensionDescriptor
	GroupDimensionDescriptor
	Measure
	MeasureDescriptor
	TimeDimension
metadatastructure	MetadataAttribute
	MetadataAttributeDescriptor
	MetadataStructure (MetadataStructureDefinition)

Package	URN class name (model class name where this is different)
	Metadataflow
	MetadataSet
process	Process
	ProcessStep
	Transition
registry	DataConstraint
	MetadataConstraint
	MetadataProvisionAgreement
	ProvisionAgreement
	Subscription
structuremapping	CategorySchemeMap
	ConceptSchemeMap
	OrganisationSchemeMap
	ReportingTaxonomyMap
	RepresentationMap
	StructureMap
codelist	Code
	Codelist
	HierarchicalCode
	Hierarchy
	HierarchyAssociation
	Level
	ValueList
categoryscheme	Categorisation
	Category
	CategoryScheme
	ReportingCategory
	ReportingTaxonomy

Package	URN class name (model class name where this is different)
conceptscheme	Concept
	ConceptScheme
transformation	CustomType
	CustomTypeScheme
	NamePersonalisation
	NamePersonalisationScheme
	Ruleset
	RulesetScheme
	Transformation
	TransformationScheme
	UserDefinedOperator
	UserDefinedOperatorScheme
	VtlCodelistMapping
	VtlConceptMapping
	VtlDataflowMapping
	VtlMappingScheme

514 **Table 2: SDMX-IM Packages and Contained Classes**

6.2.4 URN Identification components of SDMX objects

The table below describes the identification components for all SDMX object types that have identification. Note the actual attributes are all 'id' but have been prefixed by their class name or multiple class names to show navigation, e.g., 'conceptSchemeAgencyId' is really the 'Id' attribute of the Agency class that is associated to the ConceptScheme.

Note that for brevity the URN examples omit the prefix (classnames in italics indicate maintainable objects, keywords in bold indicate fixed value) All URNs have the prefix:

`urn:sdmx.org.sdmx.infomodel.{package}.{classname}=`

Classname	Ending URN pattern	Example
Agency[2]	agencySchemeAgencyId:**AGENCIES**(**1.0**).agencyId	ECB:**AGENCIES**(**1.0**).AA
AgencyScheme	agencySchemeAgencyId:**AGENCIES**(**1.0**)	ECB:**AGENCIES**(**1.0**)
Categorisation	categorisationAgencyId:categorisationId(version)	IMF:cat001(1.0.0)
Category	categorySchemeAgencyId:categorySchemeId(version).categoryId.categoryId.categoryId etc.	IMF:SDDS(1.0.0):level_1_category.level_2_category …
CategoryScheme	categorySchemeAgencyId:categorySchemeId(version)	IMF:SDDS(1.0.0)

[2] The identification of an Agency in the URN structure for the maintainable object is by means of the agencyId. The AgencyScheme is not identified as SDMX has a mechanism for identifying an Agency uniquely by its Id. Note that this Id may be hierarchical. For example, a sub-agency of IMF is referred like this: IMF.SubAgency1

Classname	Ending URN pattern	Example
CategorySchemeMap	catSchemeMapAgencyId:catSchemeMapId(version)	SDMX:EUROSTAT_SUBJECT_DOMAIN(1.0.0)
Code	codeListAgencyId:codelistId(version).codeId	SDMX:CL_FREQ(1.0.0).Q
Codelist	codeListAgencyId:codeListId(version)	SDMX:CL_FREQ(1.0.0)
ComponentMap	structureMapAgencyId:structureMap(version).componentMapId	SDMX:BOP_STRUCTURES(1.0.0).REF_AREA_TO_COUNTRY
Concept	conceptSchemeAgencyId:conceptSchemeId(version).conceptId	SDMX:CROSS_DOMAIN_CONCEPTS(1.0.0).FREQ
ConceptScheme	conceptSchemeAgencyId:conceptSchemeId(version)	SDMX:CROSS_DOMAIN_CONCEPTS(1.0.0)
ConceptSchemeMap	conceptSchemeMapAgencyId:conceptSchemeMapId(version)	SDMX:CONCEPT_MAP(1.0.0)
CustomType	customTypeSchemeAgencyId customTypeSchemeId(version) customTypeId	ECB: CUSTOM_TYPE_SCHEME(1.0.0).CUSTOM_TYPE_1
CustomTypeScheme	customTypeSchemeAgencyId customTypeSchemeId(version)	ECB:CUSTOM_TYPE_SCHEME(1.0.0)
DataAttrribute	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).dataAttributeId	TFFS:EXT_DEBT(1.0.0).OBS_STATUS
DataConstraint	dataConstraintAgencyId:dataConstraintId(version)	TFFS:CREDITOR_DATA_CONTENT(1.0.0)

Classname	Ending URN pattern	Example
DataConsumer	dataConsumerSchemeAgencyId:**DATA_CONSUMERS**(**1.0**).dataConsumerId	SDMX:**DATA_CONSUMERS**(**1.0**).CONSUMER_1
DataConsumerScheme	dataConsumerSchemeAgencyId:**DATA_CONSUMERS**(**1.0**)	SDMX:**DATA_CONSUMERS**(**1.0**)
Dataflow	dataflowAgencyId:dataflowId(version)	TFFS:CRED_EXT_DEBT(1.0.0)
DataProvider	dataProviderSchemeAgencyId:**DATA_PROVIDERS**(**1.0**).dataProviderId	SDMX:**DATA_PROVIDERS**(**1.0**).PROVIDER_1
DataProviderScheme	dataProviderSchemeAgencyId:**DATA_PROVIDERS**(**1.0**)	SDMX:**DATA_PROVIDERS**(**1.0**)
DataStructure	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version)	TFFS:EXT_DEBT(1.0.0)
Dimension	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).dimensionId	TFFS:EXT_DEBT(1.0.0).FREQ
DimensionDescriptor MeasureDescriptor AttributeDescriptor	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).componentListId where the componentListId is the name of the class (there is only one occurrence of each in the Data Structure Definition)	TFFS:EXT_DEBT(1.0.0).DimensionDescriptor TFFS:EXT_DEBT(1.0.0).MeasureDescriptor TFFS:EXT_DEBT(1.0.0).AttributeDescriptor
GroupDimensionDescriptor	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).groupDimensionDescriptorId	TFFS:EXT_DEBT(1.0.0).SIBLING
HierarchicalCode	hierarchyAgencyId:hierarchyId(version).hierarchicalCode.hierarchicalCode	UNESCO:H-C-GOV(1.0.0).GOV_CODE1.GOV_CODE1_1

Classname	Ending URN pattern	Example
Hierarchy	hierarchyAgencyId:hierarchyId(version)	UNESCO:H-C-GOV(1.0.0)
HierarchyAssociation	hierarchyAssociationAgencyId:hierarchyAssociationId(version)	UNESCO:CL_EXP_SOURCE(1.0.0)
Level	hierarchyAgencyId:hierarchyId(version).level	UNESCO:H-C-GOV(1.0.0).LVL1
Measure	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).measureId	TFFS:EXT_DEBT(1.0.0).OBS_VALUE
MetadataAttribute	msdAgencyId:msdId(version).metadataAttributeId.metadataAttributeId	IMF:SDDS_MSD(1.0.0).COMPILATION.METHOD
MetadataAttributeDescriptor	msdAgencyId:msdId(version).metadataAttributeDescriptorId	IMF:SDDS_MSD(1.0.0).MetadataAttributeDescriptor
MetadataConstraint	metadataConstraintAgencyId:metadataConstraintId(version)	TFFS:CREDITOR_METADATA_CONTENT(1.0.0)
Metadataflow	metadataflowAgencyId:metadataflowId(version)	IMF:SDDS_MDF(1.0.0)
MetadataProvider	metadataProviderSchemeAgencyId:**METADATA_PROVIDERS**(**1.0**).metadataProviderId	SDMX:**METADATA_PROVIDERS**(**1.0**).MD_PROVIDER_1
MetadataProviderScheme	metadataProviderSchemeAgencyId:**METADATA_PROVIDERS**(**1.0**)	SDMX:**METADATA_PROVIDERS**(**1.0**)
MetadataProvisionAgreement	metadataProvisionAgreementAgencyId:metadataProvisionAgreementId(version)	IMF:SDDS_MDF_AB(1.0.0)
MetadataSet	metadataProviderId:metadataSetId(version)	MD_PROVIDER:METADATASET(1.0.0)
MetadataStructure	msdAgencyId:msdId(version)	IMF:SDDS_MSD(1.0.0)

Classname	Ending URN pattern	Example
NamePersonalisation	namePersonalisationSchemeAgencyId namePersonalisationSchemeId(version) namePersonalisationId	ECB:PSN_SCHEME(1.0.0).PSN1234
NamePersonalisationScheme	namePersonalisationSchemeAgencyId namePersonalisationSchemeId(version)	ECB:PSN_SCHEME(1.0.0)
OrganisationSchemeMap	orgSchemeMapAgencyId:orgSchemeMapId(version)	SDMX:AGENCIES_PROVIDERS(1.0.0)
OrganisationUnit	organisationUnitSchemeAgencyId:organisationUnitSchemeId(version).organisationUnitId	ECB:ORGANISATIONS(1.0.0).1F
OrganisationUnitScheme	organisationUnitSchemeAgencyId:organisationUnitSchemeId(version)	ECB:ORGANISATIONS(1.0.0)
Process	processAgencyId:processId{version)	BIS:PROCESS1(1.0.0)
ProcessStep	processAgencyId:processId(version).processStepId.processStepId	BIS:PROCESS1(1.0.0).STEP1.STEP1_1
ProvisionAgreement	provisionAgreementAgencyId:provisionAgreementId(version)	TFFS:CRED_EXT_DEBT_AB(1.0.0)
ReportingCategory	reportingTaxonomyAgencyId: reportingTaxonomyId(version).reportingCategoryId.reportingCategoryId	IMF:REP_1(1.0.0):LVL1_REP_CAT.LVL2_REP_CAT
ReportingTaxonomy	reportingTaxonomyAgencyId:reportingTaxonomyId(version)	IMF:REP_1(1.0.0)
ReportingTaxonomyMap	repTaxonomyAgencyId:repTaxonomyId(version)	SDMX:RT_MAP(1.0.0)

Classname	Ending URN pattern	Example
RepresentationMap	repMapAgencyId:repMapId(version)	SDMX:REF_AREA_MAPPING(1.0.0)
Ruleset	rulesetSchemeAgencyId rulesetSchemeId(version) rulesetId	ECB:RULESET_23(1.0.0).SET111
RulesetScheme	rulesetSchemeAgencyId rulesetSchemeId(version)	ECB:RULESET_23(1.0.0)
StructureMap	structureMapAgencyId:structureMap(version)	SDMX:BOP_STRUCTURES(1.0.0)
Subscription	The Subscription is not itself an Identifiable Artefact and therefore it does not follow the rules for URN structure. The name of the URN is registryURN There is no pre-determined format.	This cannot be generated by a common mechanism as subscriptions, although maintainable in the sense that they can be submitted and deleted, are not mandated to be created by a maintenance agency and have no versioning mechanism. It is therefore the responsibility of the target registry to generate a unique Id for the Subscription, and for the application creating the subscription to store the registry URN that is returned from the registry in the subscription response message.
TimeDimension	dataStructureDefinitionAgencyId:dataStructureDefinitionId(version).timeDimensionId	TFFS:EXT_DEBT(1.0.0).TIME_PERIOD
Transformation	transformationSchemeAgencyId transformationSchemeId(version) transformationId	ECB:TRANSFORMATION_SCHEME(1.0.0).TRANS_1
TransformationScheme	transformationSchemeAgencyId transformationSchemeId(version)	ECB: TRANSFORMATION_SCHEME(1.0.0)

Classname	Ending URN pattern	Example
Transition	processAgencyId:processId(version).processStepId.transitionId	BIS:PROCESS1(1.0.0).STEP1.TRANSITION1
UserDefinedOperator	userDefinedOperatorSchemeAgencyId userDefinedOperatorSchemeId(version) usserDefinedOperatorId	ECB:OS_CALC(1.2.0).OS267
UserDefinedOperatorScheme	userDefinedOperatorSchemeAgencyId userDefinedOperatorSchemeId(version)	ECB:OS_CALC(1.2.0)
ValueList	valuelistAgencyId:valuelistId(version)	SDMX:VLIST(1.0.0)
VtlCodelistMapping	vtlMappingSchemeAgencyId vtlMappingSchemeId(version) vtlCodelistMappingId	ECB:CLIST_MP(2.0.0).ABZ
VtlConceptMapping	vtlMappingSchemeAgencyId vtlMappingSchemeId(version) vtlConceptMappingId	ECB:CLIST_MP(1.0.0).XYA
VtlDataflowMapping	vtlMappingSchemeAgencyId vtlMappingSchemeId(version) vtlDataflowMappingId	ECB:CLIST_MP(1.0.0).MOQ
VtlMappingScheme	vtlMappingSchemeAgencyId VtlMappingSchemeId(version)	ECB:CLIST_MP(2.0.0)

522 **Table 3: Table of identification components for SDMX Identifiable Artefacts**

7 Implementation Notes

7.1 Structural Definition Metadata

7.1.1 Introduction

The SDMX Registry must have the ability to support agencies in their role of defining and disseminating structural metadata artefacts. These artefacts include data structure definitions, code lists, concepts etc. and are fully defined in the SDMX-IM. An authenticated agency may submit valid structural metadata definitions which must be stored in the registry. Note that the term “structural metadata” refers as a general term to all structural components (Data Structure Definitions, Metadata Structure Definitions, Code Lists, Concept Schemes, etc.)

At a minimum, structural metadata definitions may be submitted to and queried from the registry via an HTTP/HTTPS POST in the form of one of the SDMX-ML messages for structural metadata and the SDMX RESTful API for structure queries. The message may contain all structural metadata items for the whole registry, structural metadata items for one maintenance agency, or individual structural metadata items.

Structural metadata items

- may only be modified by the maintenance agency which created them;
- may only be deleted by the agency which created them;
- may not be deleted if they are referenced from other constructs in the Registry.

The level of granularity for the maintenance of SDMX Structural Metadata objects in the registry is the Maintainable Artefact. Especially for Item Schemes, though, partial maintenance may be performed, i.e., at the level of the Item, by submitting an Item Scheme with the 'isPartial' flag set and a reduced set of Items.

The following table lists the Maintainable Artefacts.

Maintainable Artefacts		**Content**
Abstract Class	**Concrete Class**	
Item Scheme	Codelist	Code
	Concept Scheme	Concept
	Category Scheme	Category
	Organisation Unit Scheme	Organisation Unit
	Agency Scheme	Agency
	Data Provider Scheme	Data Provider
	Metadata Provider Scheme	Metadata Provider

	Data Consumer Scheme	Data Consumer
	Reporting Taxonomy	Reporting Category
	Transformation Scheme	Transformation
	Custom Type Scheme	Custom Type
	Name Personalisation Scheme	Name Personalisation
	Vtl Mapping Scheme	Vtl Codelist Mapping Vtl Concept Mapping
	Ruleset Scheme	Ruleset
	User Defined Operator Scheme	User Defined Operator
Enumerated List	ValueList	Value Item
Structure	Data Structure Definition	Dimension Descriptor Group Dimension Descriptor Dimension Time Dimension Attribute Descriptor Data Attribute Measure Descriptor Measure
	Metadata Structure Definition	Metadata Attribute Descriptor Metadata Attribute
Structure Usage	Dataflow	
	Metadataflow	
None	Process	Process Step
None	Structure Map	Component Map Epoch Map Date Pattern Map
None	Representation Map	Representation Mapping
Item Scheme Map	Organisation Scheme Map	Item Map
	Concept Scheme Map	Item Map
	Category Scheme Map	Item Map
	Reporting Taxonomy Map	Item Map
None	Provision Agreement	
None	Metadata Provision Agreement	
None	Hierarchy	Hierarchical Code
None	Hierarchy Association	
None	Categorisation	
Constraint	Data Constraint	DataKeySet

		CubeRegion
Constraint	Metadata Constraint	MetadataTargetRegion

Table 4: Table of Maintainable Artefacts for Structural Definition Metadata

7.1.2 Item Scheme, Structure

The artefacts included in the structural definitions are:

- All types of Item Scheme (Codelist, Concept Scheme, Category Scheme, Organisation Scheme, Agency Scheme, Data Provider Scheme, Metadata Provider Scheme, Data Consumer Scheme, Organisation Unit Scheme, Transformation Scheme, Name Personalisation Scheme, Custom Type Scheme, Vtl Mapping Scheme, Ruleset Scheme, User Defined Operator Scheme)
- All types of Enumerated List (ValueList)[3]
- All types of Structure (Data Structure Definition, Metadata Structure Definition)
- All types of Structure Usage (Dataflow, Metadataflow)

7.1.3 Structure Usage

7.1.3.1 Structure Usage: Basic Concepts

The Structure Usage defines, in its concrete classes of Dataflow and Metadataflow, which flows of data and metadata use which specific Structure, and importantly for the support of data and metadata discovery, the Structure Usage can be linked to one or more Category in one or more Category Scheme using the Categorisation mechanism. This gives the ability for an application to discover data and metadata by “drilling down” the Category Schemes.

[3] Note that Codelist is also an EnumeratedList.

7.1.3.2 Structure Usage Schematic

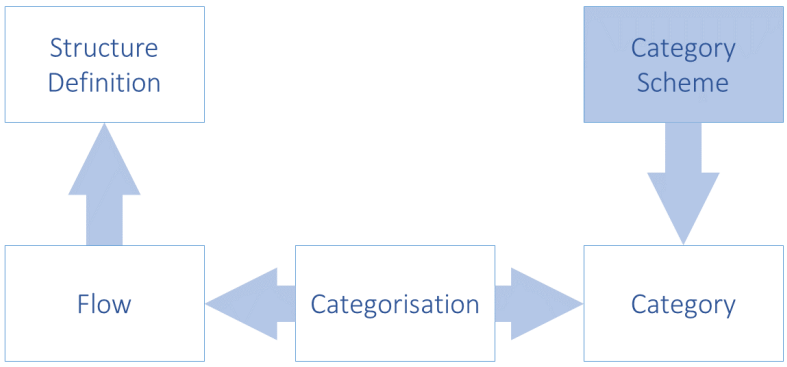


Figure 9: Schematic of Linking the Data and Metadata Flows to Categories and Structure Definitions

7.1.3.3 Structure Usage Model

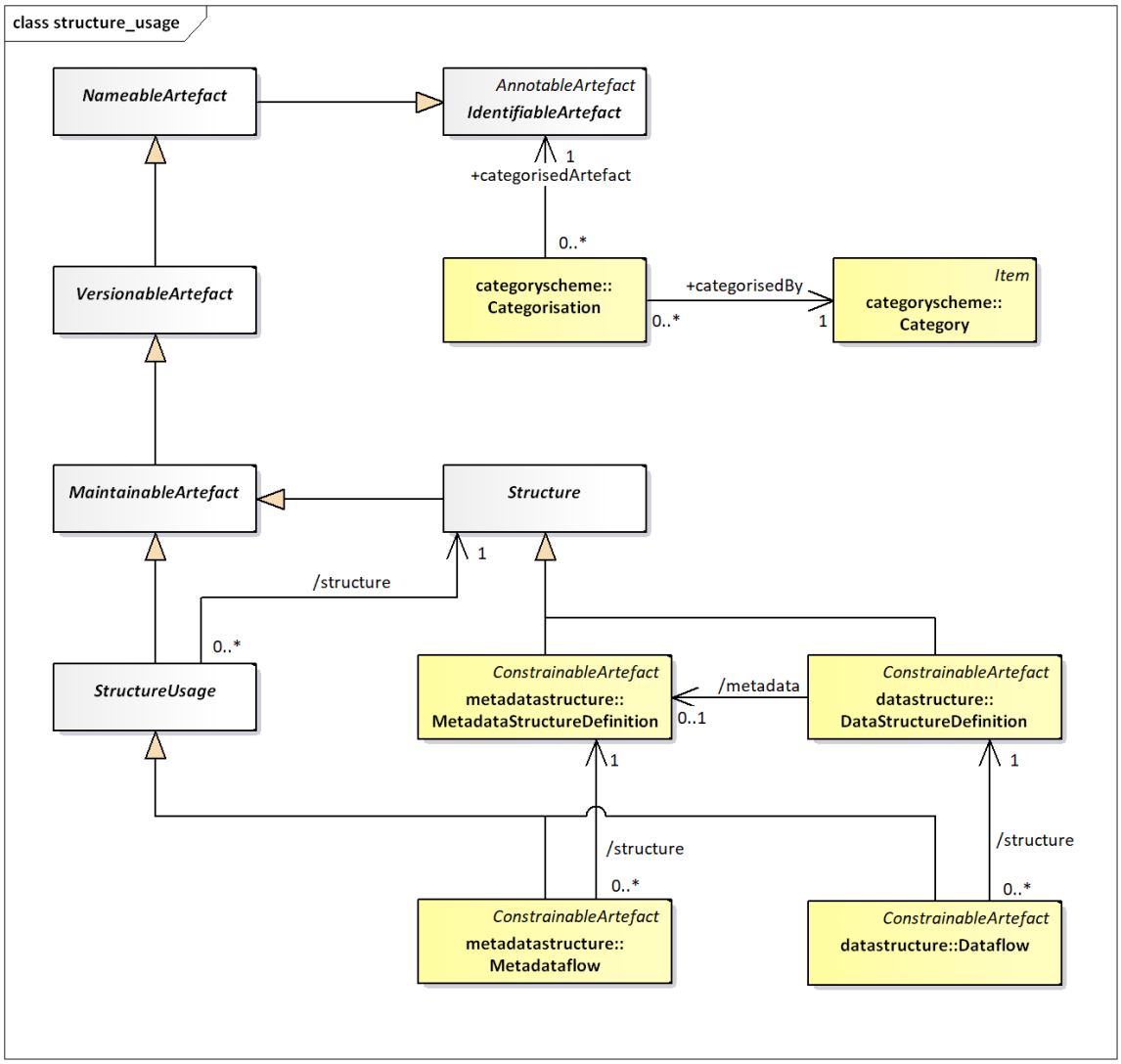


Figure 10: SDMX-IM of links from Structure Usage to Category

In addition to the maintenance of the Dataflow and the Metadataflow, the following links must be maintained in the registry:

- Dataflow to Data Structure Definition
- Metadataflow to Metadata Structure Definition

The following links may be created by means of a Categorisation

- Categorisation to Dataflow and Category
- Categorisation to Metadataflow and Category

7.2 Data and Metadata Provisioning

7.2.1 Provisioning Agreement: Basic concepts

Data/Metadata provisioning defines a framework in which the provision of different types of statistical data and metadata by various data/metadata providers can be specified and controlled. This framework is the basis on which the existence of data can be made known to the SDMX-enabled community and hence the basis on which data can subsequently be discovered. Such a framework can be used to regulate the data content to facilitate the building of intelligent applications. It can also be used to facilitate the processing implied by service level agreements, or other provisioning agreements in those scenarios that are based on legal directives. Additionally, quality and timeliness metadata can be supported by this framework which makes it practical to implement information supply chain monitoring.

Note that the term “data provisioning” here includes both the provisioning of data and metadata.

Although the Provision Agreement directly supports the data-sharing “pull” model, it is also useful in “push” exchanges (bilateral and gateway scenarios), or in a dissemination environment. It should be noted, too, that in any exchange scenario, the registry functions as a repository of structural metadata.

7.2.2 Provisioning Agreement Model – pull use case

An organisation which publishes statistical data or reference metadata and wishes to make it available to an SDMX enabled community is called a Data Provider. In terms of the SDMX Information Model, the Data Provider is maintained in a Data Provider Scheme.

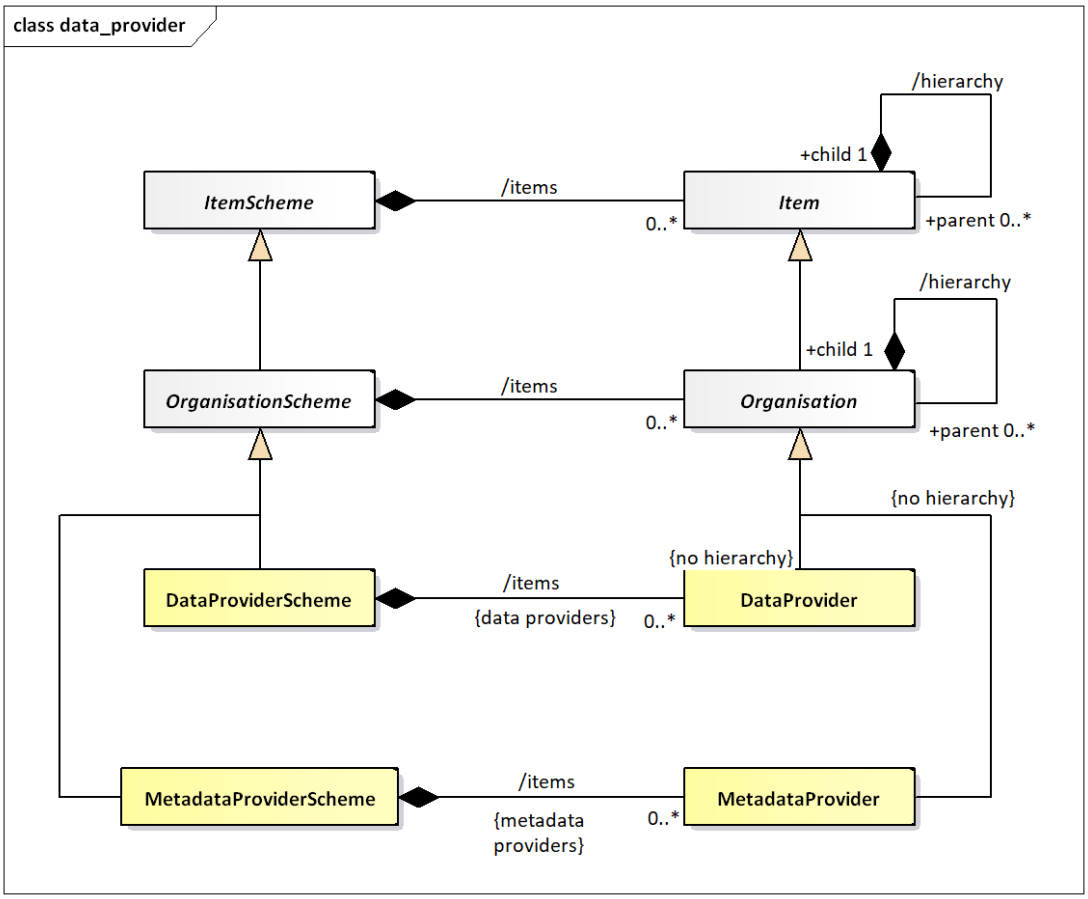
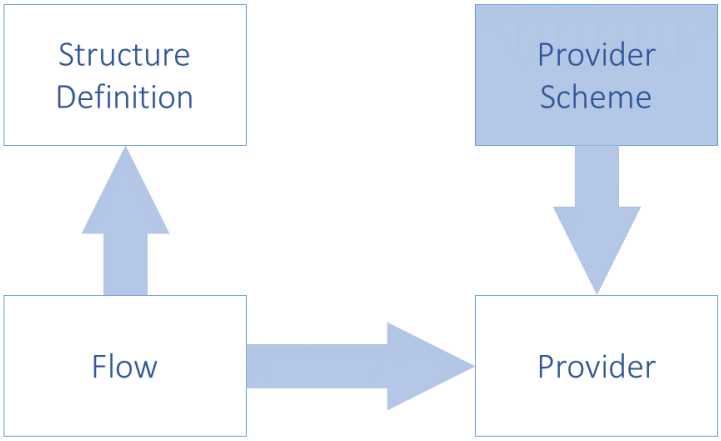


Figure 11: SDMX-IM of the Data Provider

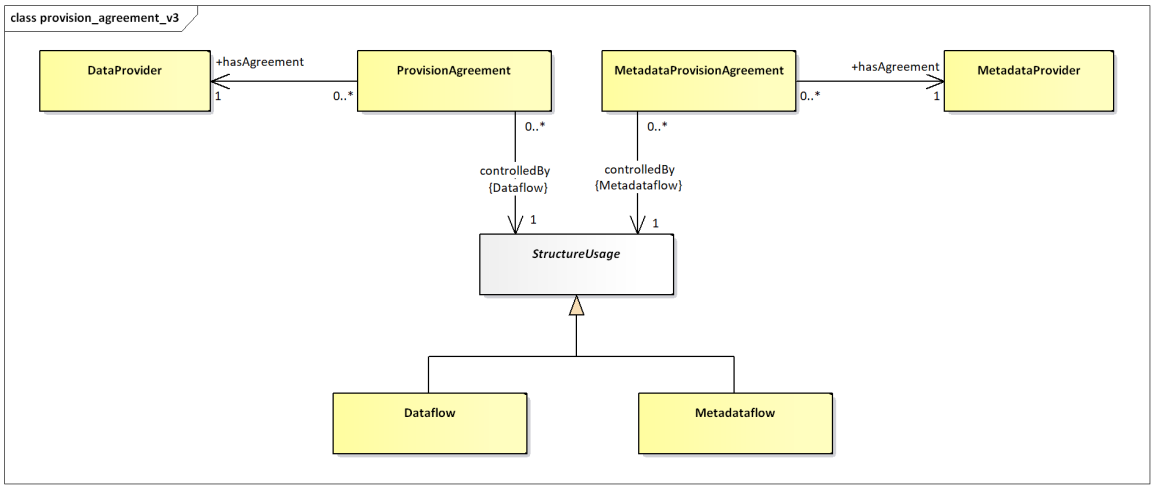
Note that the Data Provider does not inherit the hierarchy association. The diagram below shows a logical schematic of the data model classes required to maintain provision agreements.



608

609 **Figure 12: Schematic of the Provision Agreement**

610 The diagram below is a logical representation of the data required in order to maintain
611 Provision Agreements.



612

613 **Figure 13: Logical class diagram of the information contained in the Provision Agreement**

614 A Provision Agreement is structural metadata. Each Provision Agreement must reference
615 a Data Provider or Metadata Provider and a Dataflow or Metadataflow Definition. The
616 Data/Metadata Provider and the Dataflow/Metadataflow must exist already in order to set
617 up a Metadata Provision or Provision Agreement.

7.3 Data and Metadata Constraints

7.3.1 Data and Metadata Constraints: Basic Concepts

Constraints are, effectively, lists of the valid or actual content of data and metadata. Constraints can be used to specify a subset of the theoretical content of data set or metadata set which can be derived from the specification of the DSD or MSD. A Constraint can comprise a list of keys or a list of content (usually code values) of a specific component such as a dimension or attribute.

Constraints comprise the specification of subsets of key or attribute values that are to be provided for a Dataflow or Metadataflow, or directly attached to a Data Structure Definition or Metadata Structure Definition. This is important metadata because, for example, the full range of possibilities which is implied by the Data Structure Definition (e.g., the complete set of valid keys is the Cartesian product of all the values in the code lists for each of the Dimensions) is often more than is intended to be supplied according to a specific Dataflow.

Often a Data Provider will not be able to provide data for all key combinations, either because the combination itself is not meaningful, or simply because the provider does not have the data for that combination. In this case the Data Provider could constrain the data source (at the level of the Provision Agreement or the Data Provider) by supplying metadata that defines the key combinations or cube regions that are available. This is done by means of a Constraint. The Constraint is also used to define a code list subset which is used to populate a partial code list, and in generating a schema for data reporters to validate their datasets against.

Furthermore, it is often useful to define subsets or views of the Data Structure Definition which restrict values in some code lists, especially where many such subsets restrict the same Data Structure Definition. Such a view is called a Dataflow, and there can be one or more defined for any Data Structure Definition.

Whenever data is published or made available by a Data Provider, it must conform to a Dataflow (and hence to a Data Structure Definition). The Dataflow is thus a means of enabling content based processing.

In addition, DataAvailabilityConstraints can be extremely useful in a data visualisation system, such as dissemination of statistics on a website. In such a system a Cube Region can be used to specify the Dimension codes that actually exist in a data source (these can be used to build relevant selection tables).

7.3.2 Data and Metadata Constraints: Schematic

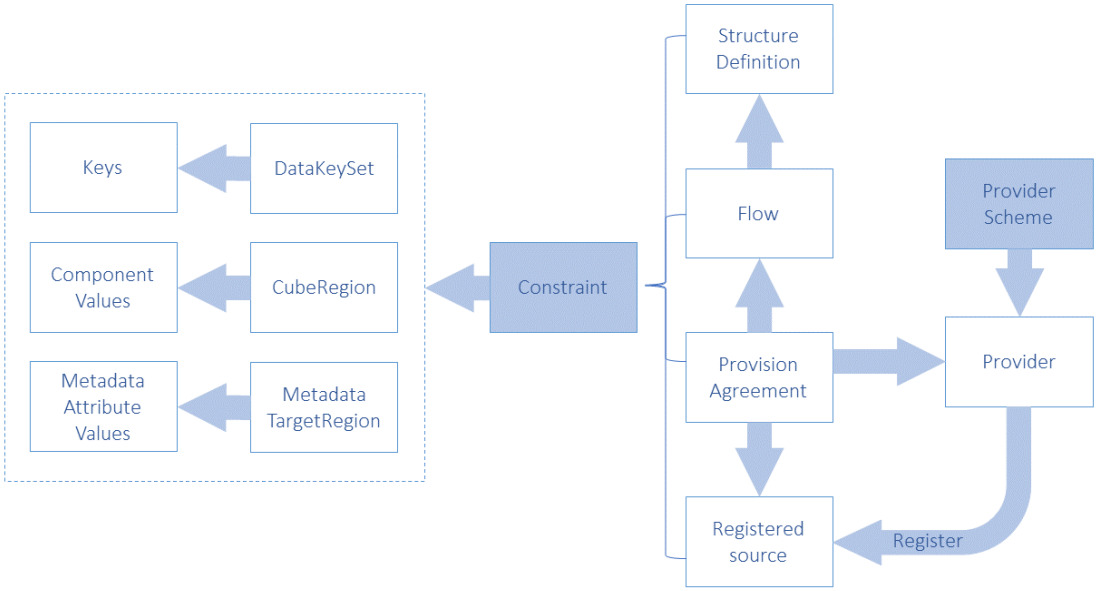


Figure 14: Schematic of the Constraint and the Artefacts that can be constrained

7.3.3 Data and Metadata Constraints: Model

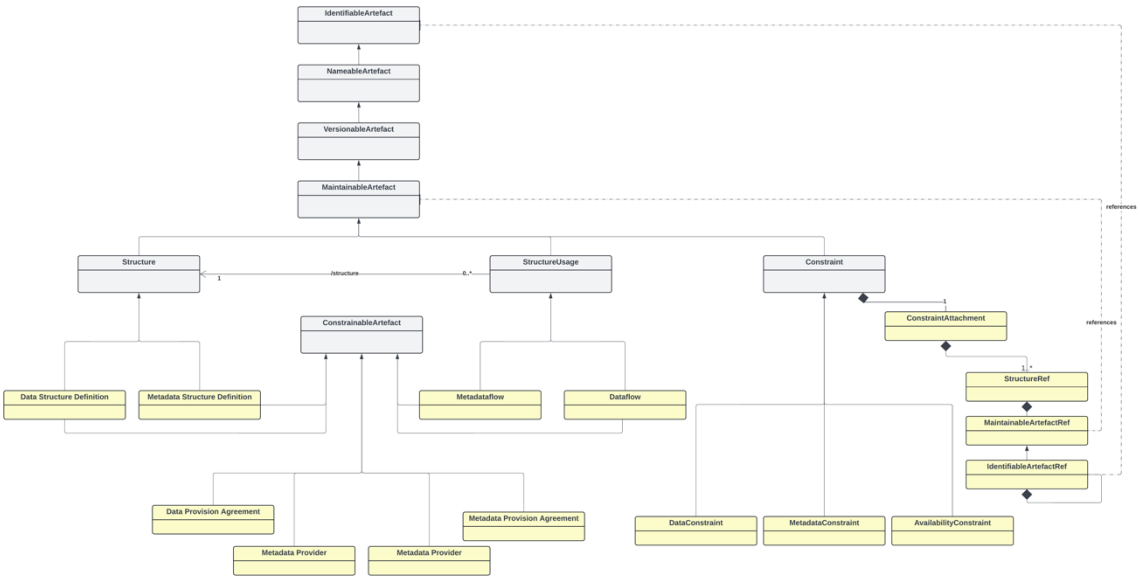


Figure 15: Logical class diagram showing inheritance between and reference to constrainable artefacts

Logical class diagram showing inheritance between and reference to constrainable artefacts

The class diagram above shows that Data Provider, Metadata Provider, Dataflow, Metadataflow, Provision Agreement, Metadata Provision Agreement, Data Structure Definition, Metadata Structure Definition are all concrete sub-classes of Constrainable Artefact and can therefore have Data Constraints specified. Note that the actual Constraint as submitted is associated to the reference classes defines in the Constraint Attachment. these are used to refer to the classes to which the Constraint applies.

The content of the Constraint can be found in the SDMX Information Model document.

7.4 Data Registration

7.4.1 Basic Concepts

A Data Provider has published a new dataset conforming to an existing Dataflow (and hence Data Structure Definition). This is implemented as either a web-accessible SDMX-ML file, or in a database which has a web-services interface capable of responding to an SDMX RESTful query with an SDMX-ML data stream.

The Data Provider wishes to make this new data available to one or more data collectors in a “pull” scenario, or to make the data available to data consumers. To do this, the Data Provider registers the new dataset with one or more SDMX conformant registries that have been configured with structural and provisioning metadata. In other words, the registry “knows” the Data Provider and “knows” what data flows the data provider has agreed to make available.

The same mechanism can be used to report or make available a metadata set.

SDMX-RR supports dataset registration via the Registration Request, which can be created by the Data Provider (giving the Data Provider maximum control). The registry responds to the registration request with a registration response which indicates if the registration was successful. In the event of an error, the error messages are returned as a registry exception within the response.

7.4.2 The Registration Request

7.4.2.1 Registration Request Schematic

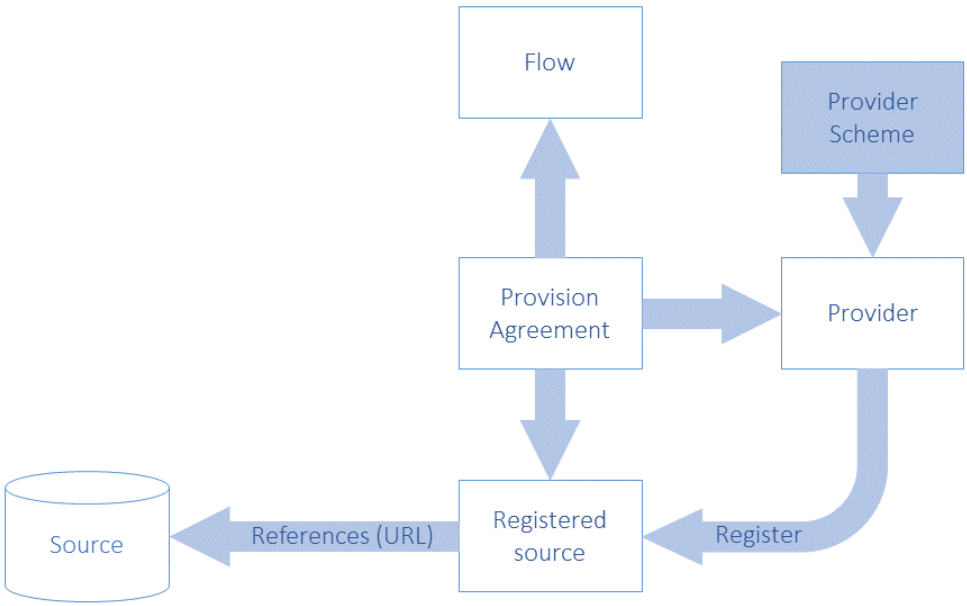


Figure 16: Schematic of the Objects Concerned with Registration

7.4.2.2 Registration Request Model

The following UML diagram shows the composition of the registration request. Each request is made up of one or more Registrations, one per dataset to be registered. The Registration can optionally have information, which has been extracted from the Registration:

- `validFrom`
- `validTo`
- `lastUpdated`

The last updated date is useful during the discovery process to make sure the client knows which data is freshest.

The Registration has an action attribute which takes one of the following values:

Action Attribute Value	Behaviour

`Append`	Add this Registration to the registry
`Replace`	Replace the existing Registration with identified by the id in the Registration of the `SubmitRegistrationRequest`
`Delete`	Delete the existing Registration identified by the id in the Registration of the `SubmitRegistrationRequest`

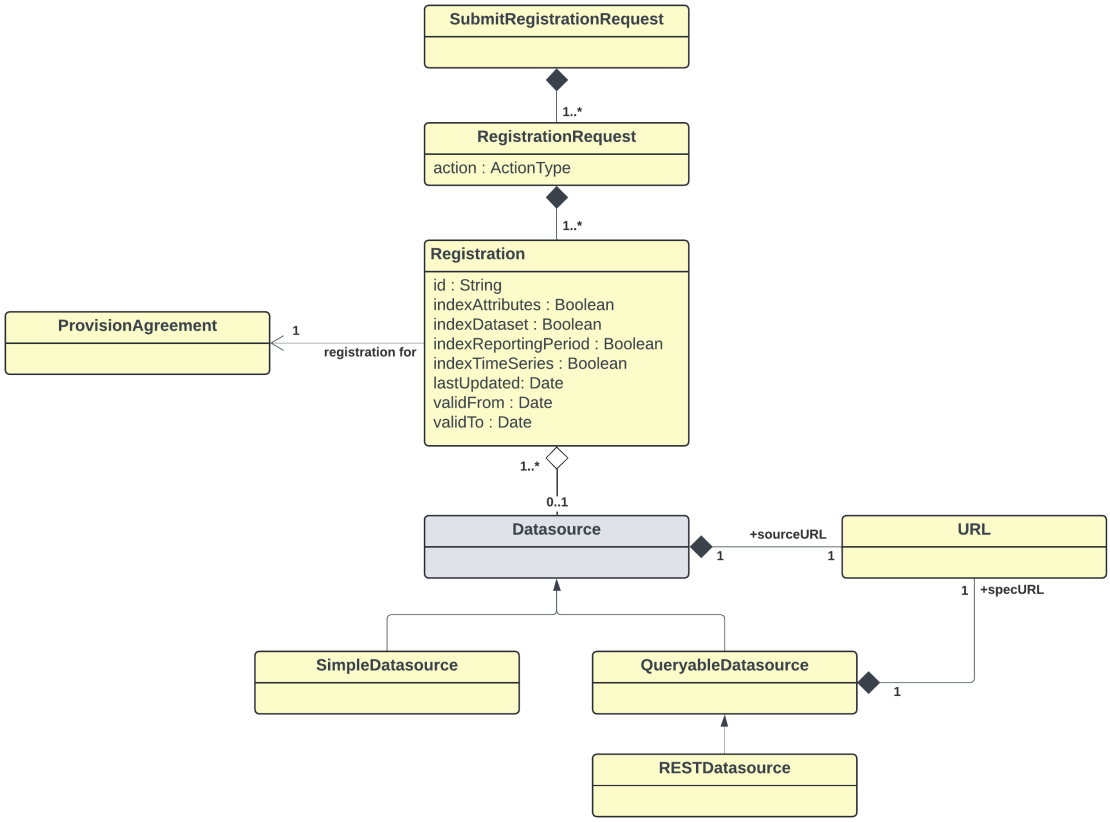


Figure 17: Logical Class Diagram of Registration of Data and Metadata

The *`QueryDatasource`* is an abstract class that represents a data source, which can understand an API query (i.e., a RESTful query – `RESTDatasource`) and respond appropriately. Each data source inherits the `dataURL` from *`Datasource`*, and the *`QueryDatasource`* has an additional URL to locate the specification of the service (`specURL`) to describe how to access it. All other supported protocols are assumed to use the `SimpleDatasource` URL.

A `SimpleDatasource` is used to reference a physical SDMX-ML file that is available at a URL.

The `RegistrationRequest` has an `action` attribute which defines whether this is a new (append) or updated (replace) `Registration`, or that the `Registration` is to be

714 deleted (delete). The `id` is only provided for the replace and delete actions, as the Registry
715 will allocate the unique `id` of the (new) `Registration`.

716 The `Registration` includes attributes that state how a `SimpleDatasource` is to be
717 indexed when registered. The Registry registration process must act as follows:

718 Information in the dataset is extracted and made available via the availability REST API as
719 documented here:

720 https://github.com/sdmx-twg/sdmx-rest/blob/master/doc/availability.md

721

Indexing Required	**Registration Process Activity**
`indexTimeSeries`	Extract all the series keys and create a KeySet(s) Constraint.
`indexDataSet`	Extract all the codes and other content of the Key value of the Series Key in a Data Set and create one or more Cube Regions containing Member Selections of Dimension Components of the Constraints model in the SDMX-IM, and the associated Selection Value.
`indexReportingPeriod`	This applies only to a registered dataset. Extract the Reporting Begin and Reporting End from the Header of the Message containing the data set, and create a Reference Period constraint.
`indexAttributes`	**Data Set** Extract the content of the Attribute Values in a Data Set and create one or more Cube Regions containing Member Selections of Data Attribute Components of the Constraints model in the SDMXIM, and the associated Selection Value **Metadata Set** Indicate the presence of a Reported Attribute by creating one or more Cube Regions containing Member Selections of Metadata Attribute Components of the Constraints model in the

Indexing Required	Registration Process Activity
	SDMX-IM. Note that the content is not stored in the Selection Value.

722

723 Constraints that specify the contents of a *QueryDatasource* are submitted to the
724 Registry via the structure submission service (i.e., the RESTful API).

725 The Registration must reference the ProvisionAgreement to which it relates.

726 ### 7.4.3 Registration Response

727 After a registration request has been submitted to the registry, a response is returned to
728 the submitter indicating success or failure. Given that a registration request can hold many
729 Registrations, then there must be a registration status for each Registration. The
730 SubmitRegistration class has a status field, which is either set to "Success",
731 "Warning" or "Failure".

732 If the registration has succeeded, a Registration will be returned – this holds the
733 Registry-allocated Id of the newly registered *Datasource* plus a *Datasource* holding
734 the URL to access the dataset or query service.

735 The RegistrationResponse returns set of registration status (one for each registration
736 submitted) in terms of a StatusMessage (this is common to all Registry responses) that
737 indicates success or failure. In the event of registration failure, a set of MessageText are
738 returned, giving the error messages that occurred during registration. It is entirely possible
739 when registering a batch of datasets, that the response will contain some successful and
740 some failed statuses. The logical model for the RegistrationResponse is shown below:

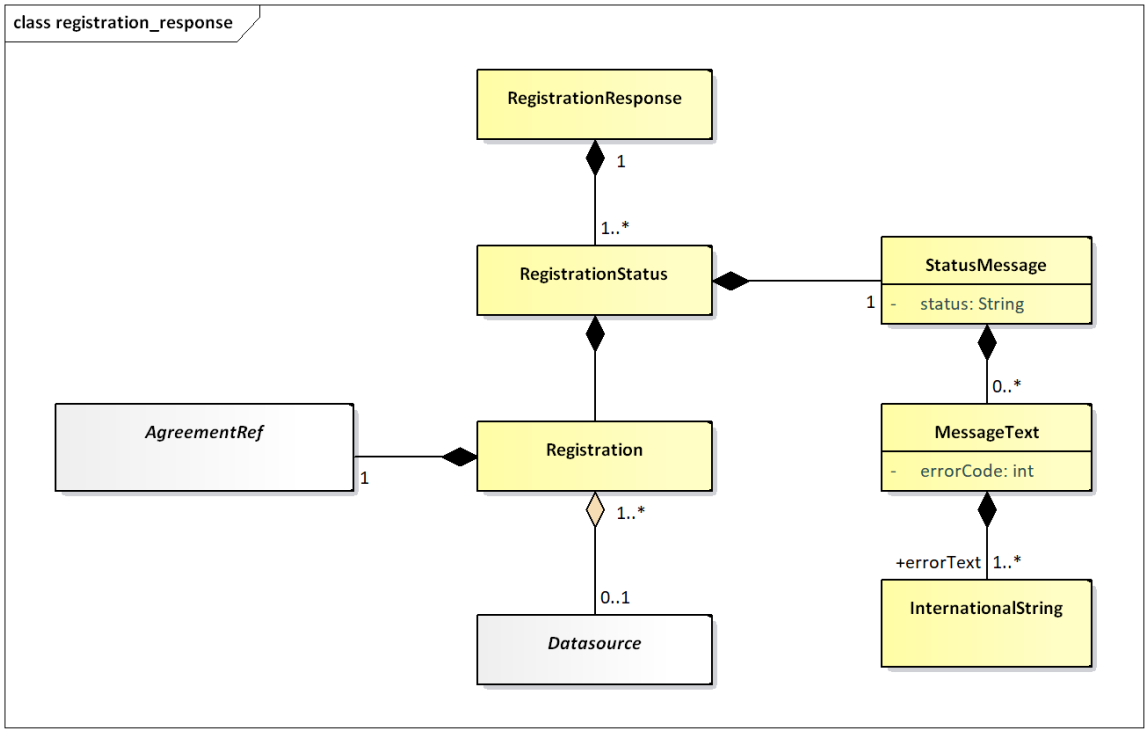


Figure 18: Logical class diagram showing the registration response

7.5 Subscription and Notification Service

The contents of the SDMX Registry/Repository will change regularly: new code lists and key families will be published and new datasets and metadata-sets will be registered. To obviate the need for users to repeatedly query the registry to see when new information is available, a mechanism is provided to allow users to be notified when these events happen.

A user can submit a subscription in the registry that defines which events are of interest, and either an email and/or an HTTP address to which a notification of qualifying events will be delivered. The subscription will be identified in the registry by a URN, which is returned to the user when the subscription is created. If the user wants to delete the subscription at a later point, the subscription URN is used as identification. Subscriptions have a validity period expressed as a date range (startDate, endDate) and the registry may delete any expired subscriptions, and will notify the subscriber on expiry.

When a registry/repository artefact is modified, any subscriptions which are observing the object are activated, and either an email or HTTP POST is instigated to report details of the changes to the user specified in the subscription. This is called a “notification”.

7.5.1 Subscription Logical Class Diagram

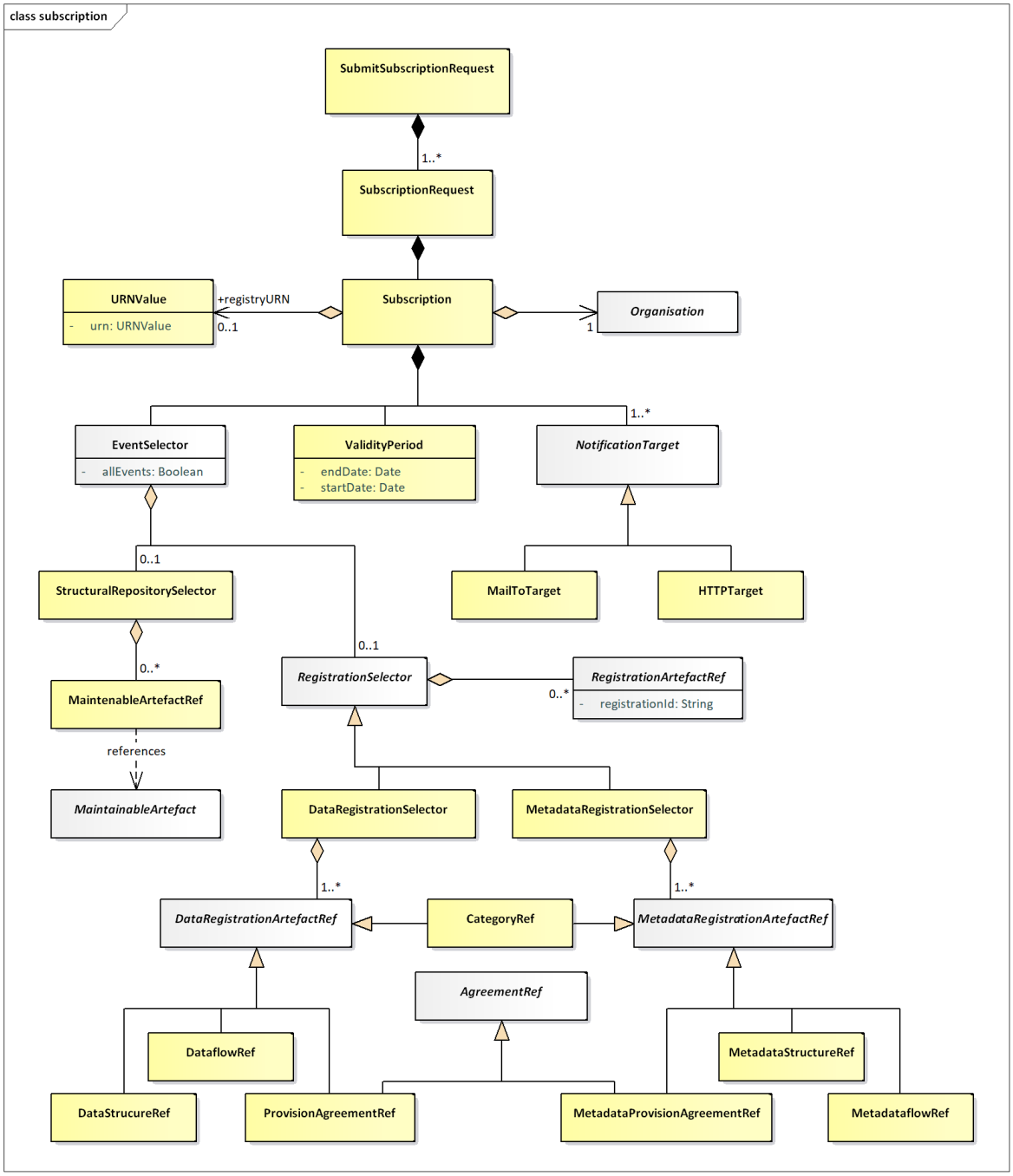


Figure 19: Logical Class Diagram of the Subscription

7.5.2 Subscription Information

Regardless of the type of registry/repository events being observed, a subscription always contains:

1. A set of URIs describing the end-points to which notifications must be sent if the subscription is activated. The URIs can be either mailto: or http: protocol. In the former case an email notification is sent; in the latter an HTTP POST notification is sent.

2. A user-defined identifier, which is returned in the response to the subscription request. This helps with asynchronous processing and is NOT stored in the Registry.

3. A validity period which defines both when the subscription becomes active and expires. The subscriber may be sent a notification on expiration of the subscription.

4. A selector which specifies which type of events are of interest. The set of event types is:

Event Type	Comment
`STRUCTURAL_REPOSITORY_EVENTS`	Life-cycle changes to Maintainable Artefacts in the structural metadata repository.
`DATA_REGISTRATION_EVENTS`	Whenever a published dataset is registered. This can be either a SDMXML data file or an SDMX conformant database.
`METADATA_REGISTRATION_EVENTS`	Whenever a published metadataset is registered. This can be either a SDMXML reference metadata file or an SDMX conformant database.
`ALL_EVENTS`	All events of the specified EventType

7.5.3 Wildcard Facility

Subscription notification supports wildcarded identifier components URNs, which are identifiers which have some or all of their component parts replaced by the wildcard character `*`. Identifier components comprise:

- `agencyID`
- `id`
- `version`

Examples of wildcarded identifier components for an identified object type of `Codelist` are shown below:

```
AgencyID = *
Id = *
```

785 `Version = *`

786 This subscribes to all `Codelist`s of all versions for all agencies.

787

788 `AgencyID = AGENCY1`

789 `Id = CODELIST1`

790 `Version = *`

791 This subscribes to all versions of `Codelist CODELIST1` maintained by the agency
792 `AGENCY1`.

793

794 `AgencyID = AGENCY1`

795 `Id = *`

796 `Version = *`

797 This subscribes to all versions of all `Codelist` objects maintained by the agency
798 `AGENCY1`.

799

800 `AgencyID = *`

801 `Id = CODELIST1`

802 `Version = *`

803 This subscribes to all versions of `Codelist CODELIST1` maintained by any agency.

804 Note that if the subscription is to the latest stable version then this can be achieved by the
805 + character, i.e.:

806 `Version = +`

807 A subscription to the latest version (whether stable, draft or non-versioned) can be
808 achieved by the ~ character, i.e.:

809 `Version = ~`

810 A subscription to the latest stable version within major version 2 starting with version 2.3.1
811 can be achieved by adding the + character after the minor version number, i.e.:

812 `Version = 2.3+.1`

The complete SDMX versioning syntax can be found in the SDMX Standards Section 6 "Technical Notes", paragraph "4.3 Versioning".

7.5.4 Structural Repository Events

Whenever a maintainable artefact (data structure definition, concept scheme, codelist, metadata structure definition, category scheme, etc.) is added to, deleted from, or modified in the structural metadata repository, a structural metadata event is triggered. Subscriptions may be set up to monitor all such events, or focus on specific artefacts such as a Data Structure Definition.

7.5.5 Registration Events

Whenever a dataset or metadata-set is registered a registration event is created. A subscription may be observing all data or metadata registrations, or it may focus on specific registrations as shown in the table below:

Selector	**Comment**
`DataProvider`	Any datasets registered by the specified dataprovider will activate the notification.
`ProvisionAgreement`	Any datasets for the agreement will activate the notification.
`Dataflow`	Any datasets for the specified dataflow will activate the notification.
`DataStructureDefinition`	Any datasets for those dataflows that are based on the specified Data Structure Definition will activate the notification
`Category`	Any datasets registered for those dataflows, provision agreements that are categorised by the category.

The event will also capture the semantic of the registration: deletion or replacement of an existing registration or a new registration.

7.6 Notification

7.6.1 Logical Class Diagram

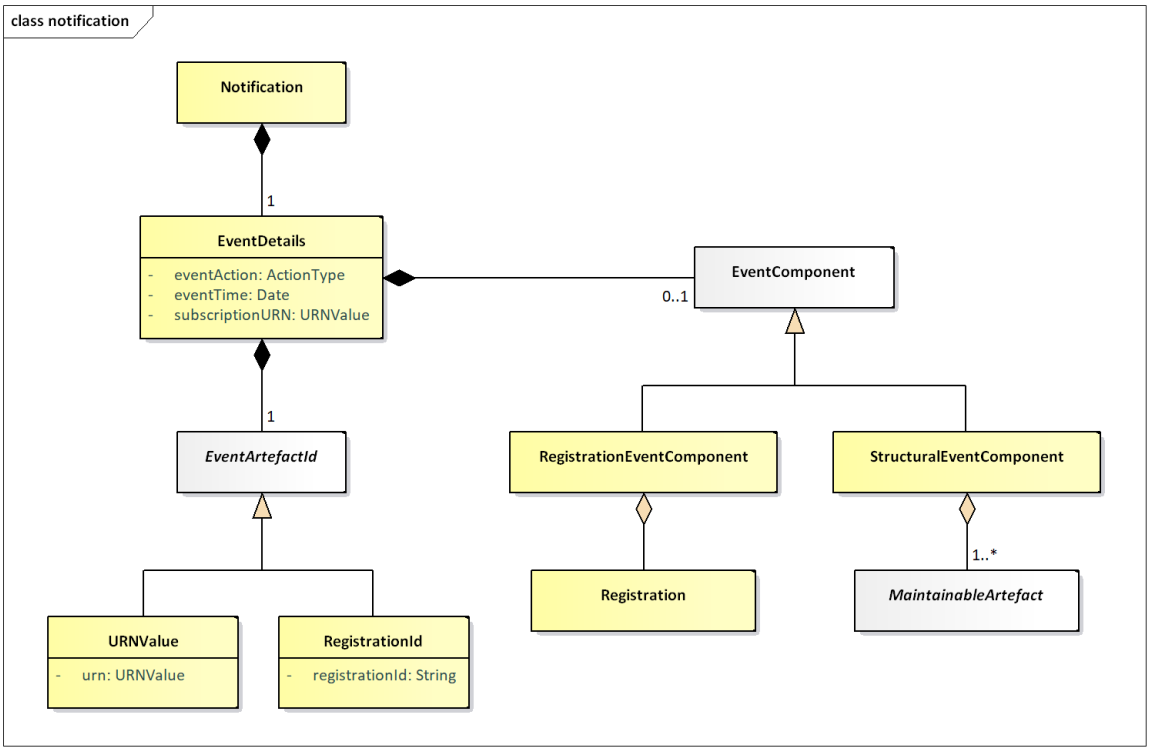


Figure 20: Logical Class Diagram of the Notification

A notification is an XML document that is sent to a user via email or http POST whenever a subscription is activated. It is an asynchronous one-way message.

Regardless of the registry component that caused the event to be triggered, the following common information is in the message:

- Date and time that the event occurred
- The URN of the artefact that caused the event
- The URN of the Subscription that produced the notification
- Event Action: Add, Replace, or Delete.

Additionally, supplementary information may be contained in the notification as detailed below.

7.6.2 Structural Event Component

The notification will contain the `MaintainableArtefact` that triggered the event in a form similar to the SDMX-ML structural message (using elements from that namespace).

7.6.3 Registration Event Component

The notification will contain the Registration.