

SDMX STANDARDS PART 5

**SDMX REGISTRY SPECIFICATION:
LOGICAL FUNCTIONALITY AND
LOGICAL INTERFACES**

VERSION 2.1

July 2011

Table of Contents

1	Introduction	2
2	Scope and Normative Status.....	3
3	Scope of the SDMX Registry/Repository	3
3.1	Objective	3
3.2	Structural Metadata.....	4
3.3	Registration	5
3.4	Notification.....	5
3.5	Discovery.....	5
4	SDMX Registry/Repository Architecture.....	6
4.1	Architectural Schematic.....	6
4.2	Structural Metadata Repository.....	7
4.3	Provisioning Metadata Repository.....	7
5	Registry Interfaces and Services	8
5.1	Registry Interfaces	8
5.2	Registry Services	8
5.2.1	Introduction	8
5.2.2	Structure Submission and Query Service.....	8
5.2.3	Structure Query Service	9
5.2.4	Data and Reference Metadata Registration Service	10
5.2.5	Data and Reference Metadata Discovery.....	11
5.2.6	Subscription and Notification	11
5.2.7	Registry Behaviour	12
6	Identification of SDMX Objects	13
6.1	Identification, Versioning, and Maintenance.....	13
6.1.1	Identification, Naming, Versioning, and Maintenance Model.....	14
6.2	Unique identification of SDMX objects	16

6.2.1	Agencies	16
6.2.2	Universal Resource Name (URN).....	18
6.2.3	Table of SDMX-IM Packages and Classes.....	22
6.2.4	URN Identification components of SDMX objects	24
7	Implementation Notes.....	31
7.1	Structural Definition Metadata	31
7.1.1	Introduction	31
7.1.2	Item Scheme, Structure	32
7.1.3	Structure Usage.....	33
7.2	Data and Metadata Provisioning	35
7.2.1	Provisioning Agreement: Basic concepts	35
7.2.2	Provisioning Agreement Model – pull use case.....	35
7.3	Data and Metadata Constraints.....	37
7.3.1	Data and Metadata Constraints: Basic Concepts	37
7.3.2	Data and Metadata Constraints: Schematic	38
7.3.3	Data and Metadata Constraints: Model	39
7.4	Data and Metadata Registration.....	40
7.4.1	Basic Concepts.....	40
7.4.2	The Registration Request	40
7.4.3	Registration Response	43
7.5	Subscription and Notification Service.....	43
7.5.1	Subscription Logical Class Diagram	45
7.5.2	Subscription Information	46
7.5.3	Wildcard Facility.....	46
7.5.4	Structural Repository Events	47
7.5.5	Registration Events.....	47
7.6	Notification.....	48

7.6.1	Logical Class Diagram.....	48
7.6.2	Structural Event Component.....	48
7.6.3	Registration Event Component.....	49

1 **Corrigendum**

2 The following problems with the specification dated April 2011 have been rectified as
3 described below.

4 **1. Problem**

5 Figure 17 - Logical Class Diagram of Registration of Data and Metadata –
6 shows the Provision Agreement as it was identified in version 2.0, and not as
7 it is identified in version 2.1.

8 **Rectification**

9 Provision Agreement is a Maintainable Artefact at version 2.1 and so the
10 relationship is shown directly to the Provision Agreement class and not
11 indirectly to the Provision Agreement via a ProvisionAgreementRef class.

12 **2. Problem**

13 Figure 17 - Logical Class Diagram of Registration of Data and Metadata –
14 shows the Registration class without the indexAttributes attribute.

15 **Rectification**

16 The attribute indexAttribute attribute is added to the Registration class and a
17 description of its purpose is given in the table at line 916.

18 **3. Problem**

19 Lines 437 and 648 of the April 2011 document mention that the fixed id for an
20 AgencyScheme is AGENCY_SCHEME whereas it should be AGENCIES.

21 **Rectification**

22 The reference to AGENCY_SCHEME is changed to AGENCIES.

23 1 Introduction

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

- 29
- 30 • the availability of an abstract information model capable of supporting time-
31 series and cross-sectional data, structural metadata, and reference metadata
32 (SDMX-IM)
- 33 • standardised XML schemas derived from the model (SDMX-ML)
- 34 • the use of web-services technology (XML, XSD, WSDL, WADL)
- 35

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

40
41 However, it is important to emphasize that the SDMX-RR provides support for the
42 submission and retrieval of all SDMX structural metadata and provisioning metadata.
43 Therefore, the Registry not only supports the data sharing scenario, but this
44 metadata is also vital in order to provide support for data and metadata
45 reporting/collection, and dissemination scenarios.

46
47 Standard formats for the exchange of aggregated statistical data and metadata as
48 prescribed in SDMX v2.1 are envisaged to bring benefits to the statistical community
49 because data reporting and dissemination processes can be made more efficient.

50
51 As organisations migrate to SDMX enabled systems, many XML (and conventional)
52 artefacts will be produced (e.g. Data Structure, Metadata Structure, Code List and
53 Concept definitions (often collectively called structural metadata), XML schemas
54 generated from data and metadata structure definitions, XSLT style-sheets for
55 transformation and display of data and metadata, terminology references, etc.). The
56 SDMX model supports interoperability, and it is important to be able to discover and
57 share these artefacts between parties in a controlled and organized way.

58
59 This is the role of the registry.

60
61 With the fundamental SDMX standards in place, a set of architectural standards are
62 needed to address some of the processes involved in statistical data and metadata
63 exchange, with an emphasis on maintenance, retrieval and sharing of the structural
64 metadata. In addition, the architectural standards support the registration and
65 discovery of data and referential metadata.

66
67 These architectural standards address the ‘how’ rather than the ‘what’, and are
68 aimed at enabling existing SDMX standards to achieve their mission. The
69 architectural standards address registry services which initially comprise:

- 70
- 71 • structural metadata repository
- 72 • data and metadata registration
- 73 • query

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

77 **2 Scope and Normative Status**

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

81
82 In this document, functions and behaviours of the Registry Interfaces are described
83 in four ways:

- 84
- 85 • in text
- 86 • with tables
- 87 • with UML diagrams excerpted from the SDMX Information Model (SDMX-IM)
- 88 • with UML diagrams that are not a part of the SDMX-IM but are included here
89 for clarity and to aid implementations (these diagram are clearly marked as
90 “Logical Class Diagram ...”)

91
92 Whilst the introductory section contains some information on the role of the registry, it
93 is assumed that the reader is familiar with the uses of a registry in providing shared
94 metadata across a community of counterparties.

95
96 Note that sections 5 and 6 contain normative rules regarding the Registry Interface
97 and the identification of registry objects. Further, the minimum standard for access to
98 the registry is via a REST interface (HTTP or HTTPS), as described in the
99 appropriate sections. The notification mechanism must support e-mail and
100 HTTP/HTTPS protocols as described. Normative registry interfaces are specified in
101 the SDMX-ML specification (Part 03 of the SDMX Standard). All other sections of this
102 document are informative.

103
104 Note that although the term “authorised user” is used in this document, the SDMX
105 standards do not define an access control mechanism. Such a mechanism, if
106 required, must be chosen and implemented by the registry software provider.

107 **3 Scope of the SDMX Registry/Repository**

108 **3.1 Objective**

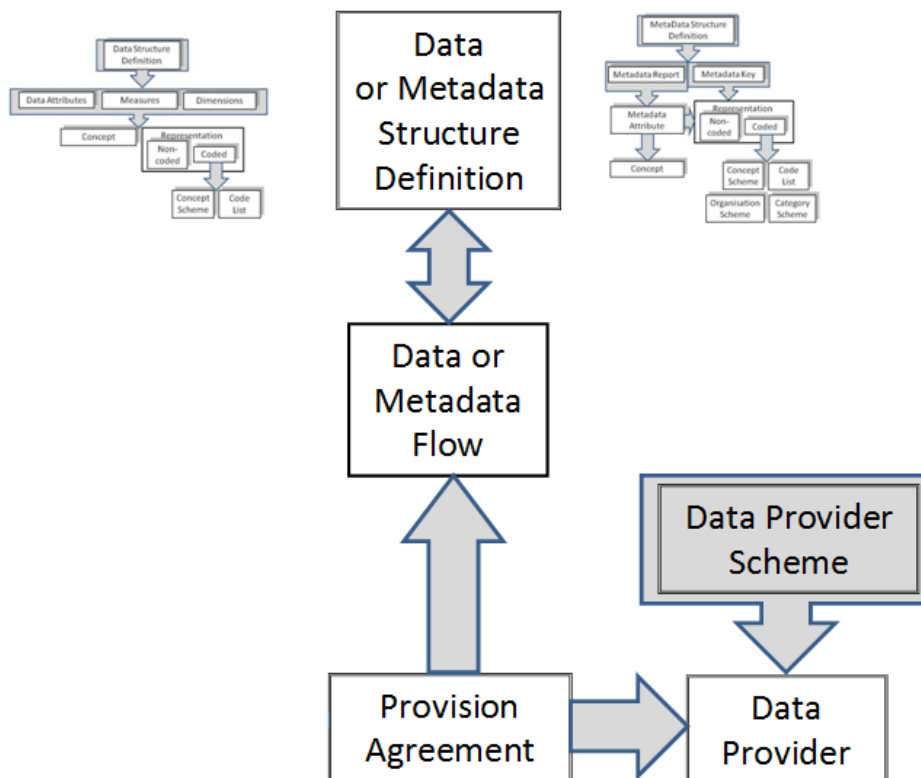
109 The objective of the SDMX registry/repository is, in broad terms, to allow
110 organisations to publish statistical data and reference metadata in known formats
111 such that interested third parties can discover these data and interpret them
112 accurately and correctly. The mechanism for doing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

- 113
- 114 1. To maintain and publish structural metadata that describes the structure and
115 valid content of data and reference metadata sources such as databases,
116 metadata repositories, data sets, metadata sets. This structural metadata
117 enables software applications to understand and to interpret the data and
118 reference metadata in these sources.
- 119 2. To enable applications, organisations, and individuals to share and to
120 discover data and reference metadata. This facilitates data and reference
121 metadata dissemination by implementing the data sharing vision of SDMX.

122 **3.2 Structural Metadata**

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

- 125
- 126 • agreeing and creating a specification of the structure of the data (called a
127 Data Structure Definition or DSD in this document but also known as “key
128 family”) which defines the dimensions, measures and attributes of a dataset
129 and their valid value set
 - 130 • if required, defining a subset or view of a DSD which allows some restriction
131 of content called a “dataflow definition”
 - 132 • agreeing and creating a specification of the structure of reference metadata
133 (Metadata Structure Definition) which defines the attributes and
134 presentational arrangement of a Metadataset and their valid values and
135 content
 - 136 • if required, defining a subset or view of a MSD which allows some restriction
137 of content called a “metadataflow definition”
 - 138 • defining which subject matter domains (specified as a Category Scheme) are
139 related to the Dataflow and Metadataflow Definitions to enable browsing
 - 140 • defining one or more lists of Data Providers (which includes metadata
141 providers)
 - 142 • defining which Data Providers have agreed to publish a given Dataflow and/or
143 Metadataflow Definition - this is called a Provision Agreement
- 144



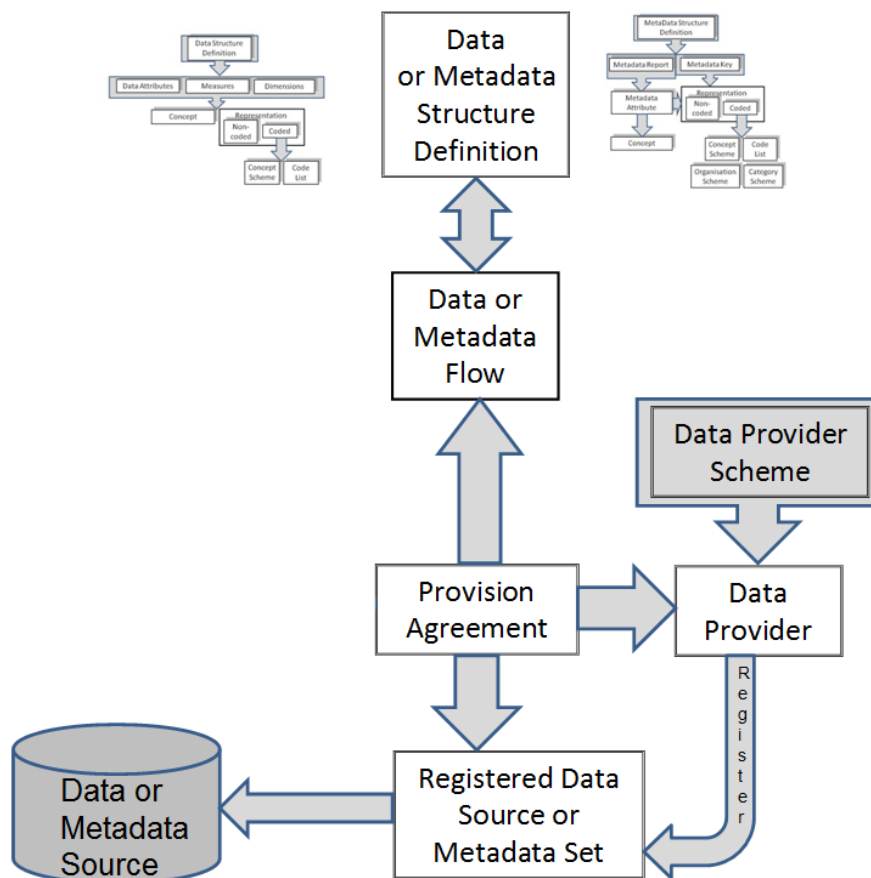
145
146

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

147 **3.3 Registration**

148 Publishing the data and reference metadata involves the following steps for a Data
149 Provider:

- 150
- 151 • making the reference metadata and data available in SDMX-ML conformant
152 data files or databases (which respond to an SDMX-ML query with SDMX-ML
153 data). The data and reference metadata files or databases must be web-
154 accessible, and must conform to an agreed Dataflow or Metadataflow
155 Definition (Data Structure Definition or Metadata Structure Definition subset)
 - 156 • registering the existence of published reference metadata and data files or
157 databases with one or more SDMX registries
158



159
160 **Figure 2: Schematic of Registered Data and Metadata Sources in the SDMX-IM**

161 **3.4 Notification**

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

- 164
- 165 • registry support of a subscription-based notification service which sends an
166 email or notifies an HTTP address announcing all published data that meets
167 the criteria contained in the subscription request

168 **3.5 Discovery**

169 Discovering published data and reference metadata involves interaction with the
170 registry to fulfil the following logical steps that would be carried out by a user

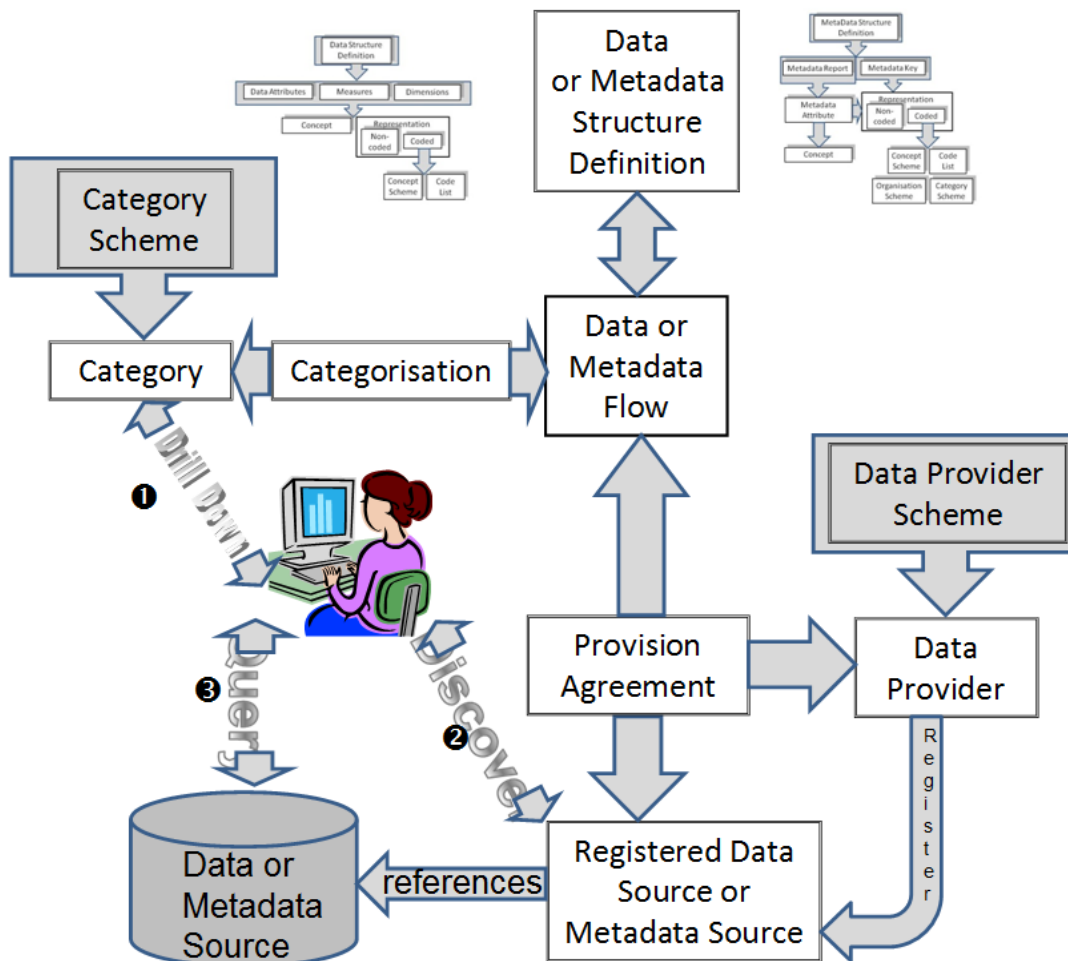
171 interacting with a service that itself interacts with the registry and an SDMX-enabled
 172 data or reference metadata resource:

173

174 • optionally browsing a subject matter domain category scheme to find
 175 Dataflow Definitions (and hence Data Structure Definitions) and
 176 Metadataflows which structure the type of data and/or reference metadata
 177 being sought

178 • build a query, in terms of the selected Data Structure Definition or Metadata
 179 Structure Definition, which specifies what data are required and submitting
 180 this to a service that can query an SDMX registry which will return a list of
 181 (URLs of) data and reference metadata files and databases which satisfy the
 182 query

183 • processing the query result set and retrieving data and/or reference metadata
 184 from the supplied URLs
 185



186

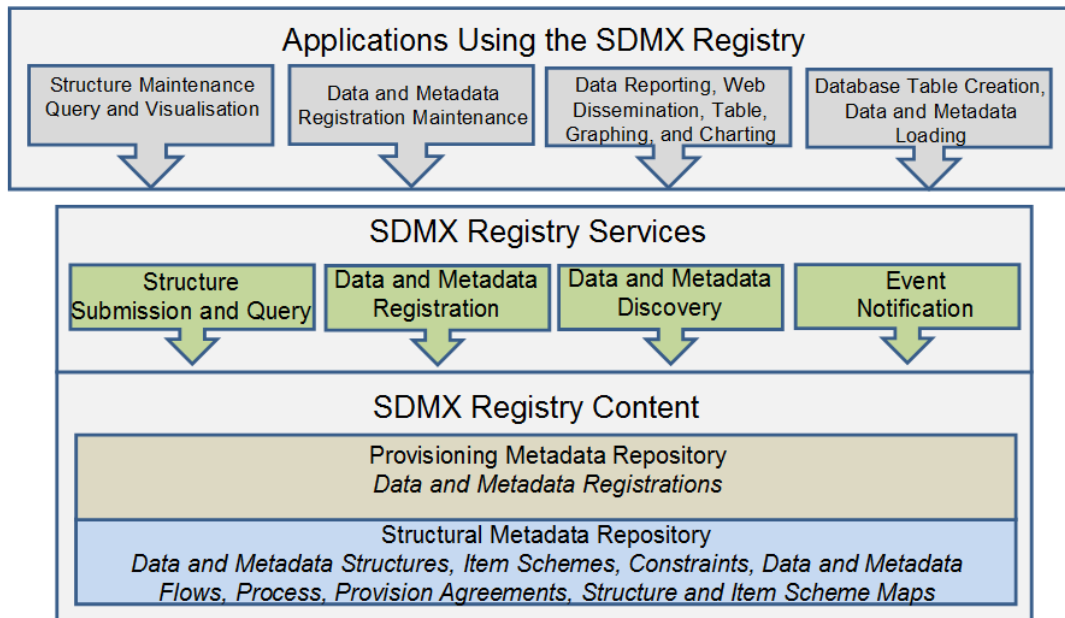
187 **Figure 3: Schematic of Data and Metadata Discovery and Query in the SDMX-IM**

188 4 SDMX Registry/Repository Architecture

189 4.1 Architectural Schematic

190 The architecture of the SDMX registry/repository is derived from the objectives stated
 191 above. It is a layered architecture that is founded by a structural metadata repository

192 which supports a provisioning metadata repository which supports the registry
 193 services. These are all supported by the SDMX-ML schemas. Applications can be
 194 built on top of these services which support the reporting, storage, retrieval, and
 195 dissemination aspects of the statistical lifecycle as well as the maintenance of the
 196 structural metadata required to drive these applications.
 197



198
199

Figure 4: Schematic of the Registry Content and Services

200 **4.2 Structural Metadata Repository**

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

205 Note that the SDMX-ML Submit Structure Request message supports all of the
 206 SDMX structural artefacts. The only structural artefacts that are not supported by the
 207 SDMX-ML Submit Structure Request are::

208
209
210
211

- Registration of data and metadata sources
- Subscription and Notification

212 Separate registry-based messages are defined to support these artefacts.

213 **4.3 Provisioning Metadata Repository**

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

219
220

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

221 **5 Registry Interfaces and Services**

222 **5.1 Registry Interfaces**

223 The Registry Interfaces are:

- 224 • Notify Registry Event
- 225 • Submit Subscription Request
- 226 • Submit Subscription Response
- 227 • Submit Registration Request
- 228 • Submit Registration Response
- 229 • Query Registration Request
- 230 • Query Registration Response
- 231 • Query Subscription Request
- 232 • Query Subscription Response
- 233 • Submit Structure Request
- 234 • Submit Structure Response

235

236 The registry interfaces are invoked in one of two ways:

237

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

241

242 In addition to these interfaces the registry must support a mechanism for querying for
243 structural metadata. This is detailed in 5.2.2.

244

245 All these interactions with the Registry – with the exception of Notify Registry Event –
246 are designed in pairs. The first document – the one which invokes the SDMX-RR
247 interface, is a “Request” document. The message returned by the interface is a
248 “Response” document.

249

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

255 **5.2 Registry Services**

256 **5.2.1 Introduction**

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

259 **5.2.2 Structure Submission and Query Service**

260 This service must implement the following SDMX-ML Interfaces:

261

- 262 • SubmitStructureRequest
- 263 • SubmitStructureResponse

264

265 These interfaces allow structural definitions to be created, modified, and removed in
266 a controlled fashion. It also allows the structural metadata artefacts to be queried and

267 retrieved either in part or as a whole. In order for the architecture to be scalable, the
268 finest-grained piece of structural metadata that can be processed by the SDMX-RR is
269 a MaintainableArtefact (see next section on the SDMX Information Model).
270

271 **5.2.3 Structure Query Service**

272 The registry must support a mechanism for querying for structural metadata. This
273 mechanism can be one or both of the SDMX-ML Query message and the SDMX
274 REST interface for structural metadata (this is defined in Part 7 of the SDMX
275 standards). The registry response to both of these query mechanisms is the SDMX
276 Structure message which has as its root node

277

- 278 • Structure

279

280 The SDMX structural artefacts that may be queried are:

281

- 282 • dataflows and metadataflows
- 283 • data structure definitions and metadata structure definitions
- 284 • codelists
- 285 • concept schemes
- 286 • reporting taxonomies
- 287 • provision agreements
- 288 • structure sets
- 289 • processes
- 290 • hierarchical code lists
- 291 • constraints
- 292 • category schemes
- 293 • categorisations and categorised objects (examples are categorised dataflows
294 and metadataflows, data structure definitions, metadata structure definitions,
295 provision agreements registered data sources and metadata sources)
- 296 • organisation schemes (agency scheme, data provider scheme, data
297 consumer scheme, organisation unit scheme)

298

299 The SDMX query messages that are a part of the SDMX-ML Query message are:

300

- 301 • StructuresQuery
- 302 • DataflowQuery
- 303 • MetadataflowQuery
- 304 • DataStructureQuery
- 305 • MetadataStructureQuery
- 306 • CategorySchemeQuery
- 307 • ConceptSchemeQuery
- 308 • CodelistQuery
- 309 • HierarchicalCodelistQuery
- 310 • OrganisationSchemeQuery
- 311 • ReportingTaxonomyQuery
- 312 • StructureSetQuery
- 313 • ProcessQuery
- 314 • CategorisationQuery
- 315 • ProvisionAgreementQuery
- 316 • ConstraintQuery

317 **5.2.4 Data and Reference Metadata Registration Service**

318 This service must implement the following SDMX-ML Interfaces:

- 319
- 320 • SubmitRegistrationRequest
 - 321 • SubmitRegistrationResponse
 - 322 • QueryRegistrationRequest
 - 323 • QueryRegistrationResponse
- 324

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

333
 334 The Data and Metadata Registration Service MAY validate the following, subject to
 335 the access control mechanism implemented in the Registry:

- 336
- 337 • that the data provider is allowed to register the data-set or metadata-set
 - 338 • that the content of the data set or metadata set meets the validation
 339 constraints. This is dependent upon such constraints being defined in the
 340 structural repository and which reference the relevant Dataflow,
 341 Metadataflow, Data Provider, Data Structure Definition, Metadata Structure
 342 Definition, Provision Agreement
 - 343 • that a queryable data source exists - this would necessitate the registration
 344 service querying the service to determine its existence
 - 345 • that a simple data source exists (i.e. a file accessible at a URL)
 - 346 • that the correct Data Structure Definition or Metadata Structure Definition is
 347 used by the registered data
 - 348 • that the components (Dimensions, Attributes, Measures, Identifier
 349 Components etc.) are consistent with the Data Structure Definition or
 350 Metadata Structure Definition
 - 351 • that the valid representations of the concepts to which these components
 352 correspond conform to the definition in the Data Structure Definition or
 353 Metadata Structure Definition

354
 355 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

356

357 The Registration has three Boolean attributes which may be present to determine
 358 how an SDMX compliant Dataset or Metadata Set indexing application must index

359 the Datasets or Metadata Set upon registration. The indexing application behaviour is
 360 as follows:
 361

Boolean Attribute	Behaviour if Value is “true”
<u>indexTimeSeries</u>	A compliant indexing application must index all the time series keys (for a Dataset registration) or metadata target values (for a Metadata Set registration)
<u>indexDataSet</u>	A compliant indexing application must index the range of actual (present) values for each dimension of the Dataset (for a Dataset registration) or the range of actual (present) values for each Metadata Attribute which takes an enumerated value. 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
<u>indexReportingPeriod</u>	A compliant indexing application must index the time period range(s) for which data are present in the Dataset or Metadata Set

362

363 **5.2.5 Data and Reference Metadata Discovery**

364 The Data and Metadata Discovery Service implements the following Registry
 365 Interfaces:

- 366
- 367 • QueryRegistrationRequest
 - 368 • QueryRegistrationResponse
- 369

370 **5.2.6 Subscription and Notification**

371 The Subscription and Notification Service implements the following Registry
 372 Interfaces:

- 373
- 374 • SubmitSubscriptionRequest
 - 375 • SubmitSubscriptionResponse
 - 376 • NotifyRegistryEvent
- 377

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

386
 387 Subscription management involves a user submitting a subscription request which
 388 contains:
 389

- 390
- 391
- 392
- 393
- 394
- 395
- 396
- 397
- 398
- 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 end-points to which an XML notification message can be sent. Supported end-point types will be email (mailto:) and HTTP POST (a normal http:// address)
 - request for a list of submitted subscriptions
 - deletion of a subscription

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

403 **5.2.7 Registry Behaviour**

404 The following table defines the behaviour of the SDMX Registry for the various
 405 Registry Interface messages.

Interface	Behaviour
All	<p>1) If the action is set to “replace” then the entire contents of the existing maintainable object in the Registry MUST be replaced by the object submitted, unless the final attribute is set to “true” in which case the only changes that are allowed are to the following constructs:</p> <ul style="list-style-type: none"> • Name – this applies to the Maintainable object and its contained elements, such a Code in a Code list. • Description - this applies to the Maintainable object and its contained elements, such a Code in a Code list. • Annotation - this applies to the Maintainable object and its contained elements, such a Code in a Code list. • validTo • validFrom • structureURL • serviceURL • uri • isExternalReference

Interface	Behaviour
	<p>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.</p> <p>3) If the action is set to “delete” then the Registry MUST verify that the object can be deleted. In order to qualify for deletion the object must:</p> <p>a) Not have the final attribute set to “true”</p> <p>b) Not be referenced from any other object in the Registry.</p> <p>4) The version rules in the SDMX Schema documentation MUST be obeyed.</p> <p>5) The specific rules for the elements and attributes documented in the SDMX Schema MUST be obeyed.</p>
SubmitStructureRequest	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.
SubmitProvisioningRequest	No additional behaviour.
Submit Registration Request	<p>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.</p> <p>For a queryable datasource the Registry MAY validate that the source exists and can accept an SDMX-ML data query.</p>

406

407 **6 Identification of SDMX Objects**

408 **6.1 Identification, Versioning, and Maintenance**

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

410

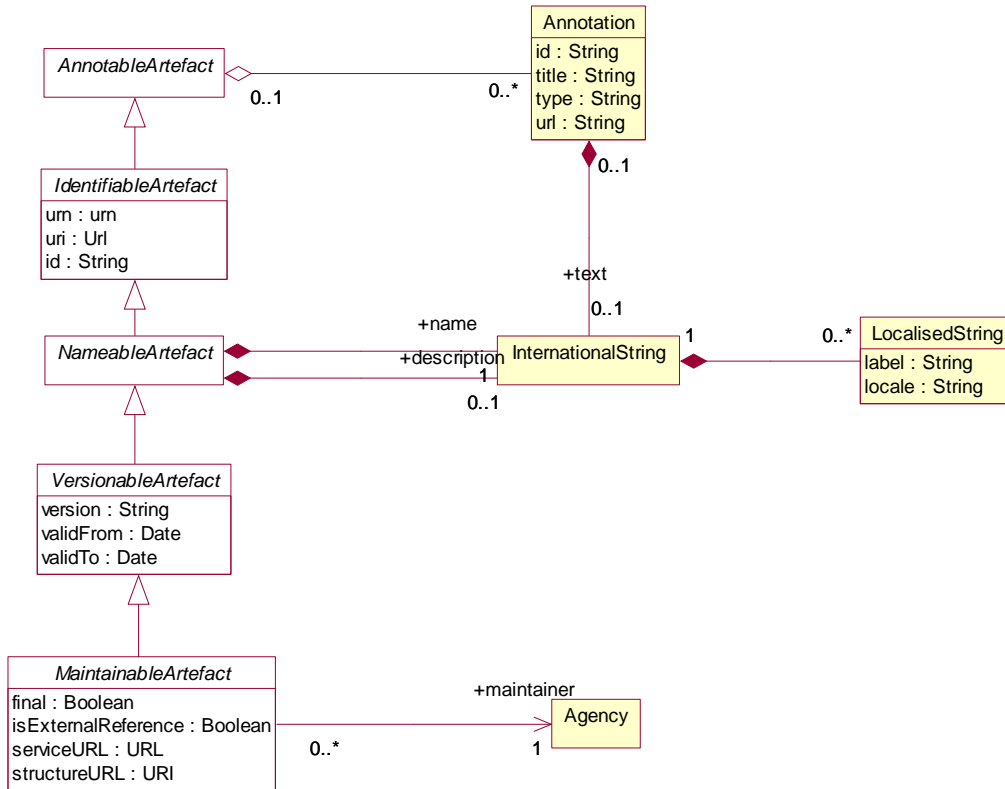
411 • **IdentifiableArtefact** - this gives an object the ability to be uniquely identified
 412 (see following section on identification), to have a user-defined URI, and to
 413 have multi-lingual annotations.

414 • **NamableArtefact** - this has all of the features of IdentifiableArtefact plus the
 415 ability to have a multi-lingual name and description,

416 • **VersionableArtefact** – this has all of the above features plus a version
 417 number and a validity period.

- 418 • **MaintainableArtefact** – this has all of the above features, and indication as
 419 to whether the object is “final” and cannot be changed or deleted, registry and
 420 structure URIs, plus an association to the maintenance agency of the object.

421 **6.1.1 Identification, Naming, Versioning, and Maintenance Model**
 422



423 **Figure 5: Class diagram of fundamental artefacts in the SDMX-IM**
 424

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

- 427
- 428 • Annotable
 - 429 • Identifiable
 - 430 • Nameable
 - 431 • Versionable
 - 432 • Maintainable

Object Type	Data Attributes	Status	Data type	Notes
Annotable	AnnotationTitle	C	string	
	AnnotationType	C	string	
	AnnotationURN	C	string	
	AnnotationText in the form of	C		This can have language-specific variants.

Object Type	Data Attributes	Status	Data type	Notes
	International String			
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 International String	M	string	This can have language-specific variants.
	Description in the form of International String	C	string	This can have language-specific variants.
Versionable	All content as for Identifiable plus			
	version	C	string	This is the version number. If not present the default is 1.0
	validFrom	C	Date/time	
	validTo	C	Date/time	
Maintainable	All content as for Versionable plus			
	final		boolean	Value of "true" indicates that this is a final specification and it cannot be changed except as a new version. Note that providing a "final" object is not referenced from another object then it may be deleted.
	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 structureURI, each of which must return a valid SDMX-ML file.
	serviceURL	C	string	The url of the service that

Object Type	Data Attributes	Status	Data type	Notes
				can be queried for this resource
	structureURL	C	string	The url of the resource.
	(Maintenance) agencyId	M	string	The object must be linked to a maintenance agency.

433

Table 1: Common Attributes of Object Types

434

6.2 Unique identification of SDMX objects

435

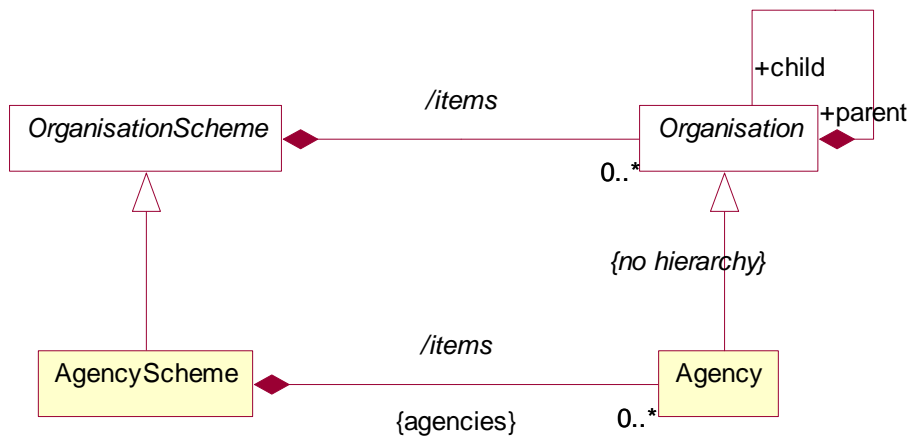
6.2.1 Agencies

436

The Maintenance Agency in SDMX is maintained in an Agency Scheme which itself

437

is a sub class of Organisation Scheme – this is shown in the class diagram below.



438

439

Figure 6: Agency Scheme Model

440

The Agency in SDMX is extremely important. The Agency Id system used in SDMX

441

is an n-level structure. The top level of this structure is maintained by SDMX. Any

442

Agency in this top level can declare sub agencies and any sub agency can also

443

declare sub agencies. The Agency Scheme has a fixed id and version and is never

444

declared explicitly in the SDMX object identification mechanism.

445

In order to achieve this SDMX adopts the following rules:

446

447

1. Agencies are maintained in an Agency Scheme (which is a sub class of Organisation Scheme)

448

2. The agency of the Agency Scheme must also be declared in a (different) Agency Scheme.

449

3. The “top-level” agency is SDMX and maintains the “top-level” Agency Scheme.

450

4. 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.

451

5. The AgencyScheme cannot be versioned and so take a default version number of 1.0 and cannot be made “final”.

452

6. There can be only one AgencyScheme maintained by any one Agency. It has a fixed Id of AGENCIES.

453

454

455

456

457

458

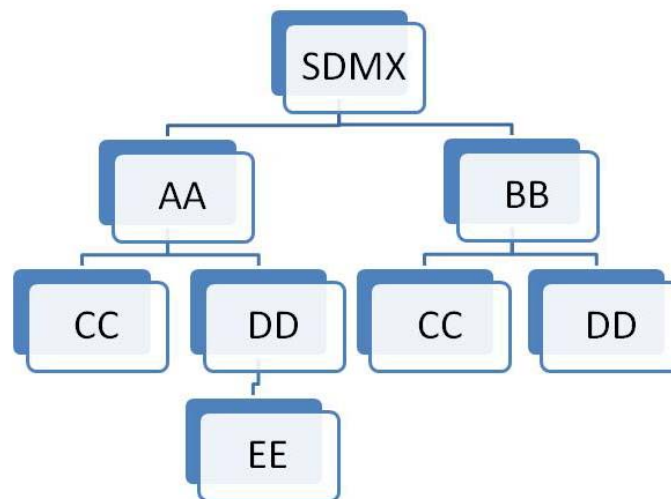
459

- 460 7. The /hierarchy of Organisation is not inherited by Maintenance Agency –
 461 thus each Agency Scheme is a flat list of Maintenance Agencies.
 462 8. The format of the agency identifier is agencyID.agencyID etc. The top-
 463 level agency in this identification mechanism is the agency registered in the
 464 SDMX agency scheme. In other words, SDMX is not a part of the hierarchical
 465 ID structure for agencies. However SDMX is, itself, a maintenance agency
 466 and is contained in the top-level Agency Scheme.
 467

468 This supports a hierarchical structure of agencyID.

469

470 An example is shown below.



471

472

Figure 7: Example of Hierarchic Structure of Agencies

473 The following organizations maintain an Agency Scheme.

474

- 475 • SDMX – contains Agencies AA, BB
- 476 • AA – contains Agencies CC, DD
- 477 • BB – contains Agencies CC, DD
- 478 • DD – Contains Agency EE

479 Each agency is identified by its full hierarchy excluding SDMX.

480

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

482

483 An example of this is shown in the XML snippet below.

484

```

]<structure:Codelists>
]<structure:Codelist id="CL_BOP" agencyID="SDMX" version="1.0"
urn="urn:sdmx.org.sdmx.infomodel.codelist.Codelist=SDMX:CL_BOP[1.0]">
  <common:Name>name</common:Name>
</structure:Codelist>
]<structure:Codelist id="CL_BOP" agencyID="AA" version="1.0"
urn="urn:sdmx.org.sdmx.infomodel.codelist.Codelist=AA:CL_BOP[1.0]" >
  <common:Name>name</common:Name>
</structure:Codelist>
]<structure:Codelist id="CL_BOP" agencyID="AA.CC" version="1.0"
urn="urn:sdmx.org.sdmx.infomodel.codelist.Codelist=AA.CC:CL_BOP[1.0]" >
  <common:Name>name</common:Name>
</structure:Codelist>
]<structure:Codelist id="CL_BOP" agencyID="BB.CC" version="1.0"
urn="urn:sdmx.org.sdmx.infomodel.codelist.Codelist=BB.CC:CL_BOP[1.0]">
  <common:Name>name</common:Name>
</structure:Codelist>
</structure:Codelists>

```

485
486

Figure 8: Example Showing Use of Agency Identifiers

487

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

490 **6.2.2 Universal Resource Name (URN)**

491 **6.2.2.1 Introduction**

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

501 **6.2.2.2 URN Structure**

502 **Case Rules for URN**

503

504 For the URN, all parts of the string are case sensitive. The Id of any object must be
505 UPPER CASE. Therefore, CRED_ext_Debt is invalid and it should be
506 CRED_EXT_DEBT.

507

508 The generic structure of the URN is as follows:

509

```

510 SDMXprefix.SDMX-IM-package-name.class-name=agency-
511 id:maintainedobject-id(maintainedobject-version).*container-
512 object-id.object-id

```

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

514

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

518
 519 The Maintenance agency identifier is separated from the maintainable artefact
 520 identifier by a colon ':'. All other identifiers in the SDMX URN syntax are separated by
 521 a period(.

522 **6.2.2.3 Explanation of the generic structure**

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

525
 526 **SDMXPrefix:** urn:sdmx:org.

527
 528 **SDMX-IM package name:** sdmx.infomodel.package=
 529

530 The packages are:

- 531 base
- 532 codelist
- 533 conceptscheme
- 534 datastructure
- 535 categoryscheme
- 536 registry
- 537 metadatastructure
- 538 process
- 539 mapping

540
 541 **maintainable-object-id** is the identifier of the maintainable object. This will always
 542 be present as all identifiable objects are either a maintainable object or contained in a
 543 maintainable object.

544 **(maintainable-object-version)** is the version of the maintainable object and is
 545 enclosed in round brackets (). It will always be present.

546 **container-object-id** is the identifier of an intermediary object that contains the actual
 547 object which the URN is identifying. It is not mandatory as many actual objects do not
 548 have an intermediary container object. For instance, a Code is in a maintained object
 549 (Code List) and has no intermediary container object, whereas a Metadata Attribute
 550 has an intermediary container object (Report Structure) and may have an
 551 intermediary container object which is its parent Metadata Attribute. For this reason
 552 the container object id may repeat, with each repetition identifying the object at the
 553 next-lower level in its hierarchy. Note that if there is only a single containing object in
 554 the model then it is NOT included in the URN structure. This applies to Attribute
 555 Descriptor, Dimension Descriptor, and Measure Descriptor where there can be only
 556 one such object and this object has a fixed id. Therefore, whilst each of these has a
 557 URN, the id of the Attribute Descriptor, Dimension Descriptor, and Measure
 558 Descriptor is not included when the actual object is a Data Attribute or a
 559 Dimension/Measure Dimension/ Time Dimension, or a Measure.

560
 561 Note that although a Code can have a parent Code and a Concept can have a parent
 562 Concept these are maintained in a flat structure and therefore do not have a
 563 container-object-id.

564
 565 For example the sequence is agency:DSDid(version).DimensionId and not
 566 agency:DSDid(version).DimensionDescriptorId.DimensionId.

567

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

571

572 **Generic Examples of the URN Structure**

573

574 Actual object is a maintainable

575 SDMXPrefix.SDMX-IM package name.classname=agency
576 id:maintained-object-id(version)

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

579

580 SDMXPrefix.SDMX-IM package name.classname=agency
581 id:maintained-object-id(version).object-id

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

584

585 SDMXPrefix.SDMX-IM package name.classname=agency
586 id:maintained-object-id(version).contained-object-id.object-id

587

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

590

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

597

598 SDMXPrefix.SDMX-IM package name.classname=agency
599 id:maintained-object-id(version).contained-object-id.object-id

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

602

603 In this case the generic syntax is the same as for the example above as the parent
604 object is regarded as a containing object, even if it is of the same type. An example
605 here is a Metadata Attribute where the contained objects are Report Structure (first
606 contained object id) and Metadata Attribute (subsequent contained object ids). The
607 example below shows the generic structure for MSD/Report Structure/Metadata
608 Attribute/Metadata Attribute

609

610 SDMXPrefix.SDMX-IM package name.classname=agency
611 id:maintained-object-id(version).contained-object-id.
612 contained-object-id contained-object-id.object-id

613 **Concrete Examples of the URN Structure**

614

615 The Data Structure Definition CRED_EXT_DEBT version 1.0 maintained by the top
616 level Agency TFFS would have the URN:

617

618 urn:sdmx:org.sdmx.infomodel.datastructure.DataStructure=TFFS:CRED_EXT_
619 DEBT(1.0)

620 The URN for a code for Argentina maintained by ISO in the code list CL_3166A2
621 version 1.0 would be:

622

623 urn:sdmx:org.sdmx.infomodel.codelist.Code=ISO:CL_3166A2(1.0).AR

624 The URN for a category (id of 1) which has parent category (id of 2) maintained by
625 SDMX in the category scheme SUBJECT_MATTER_DOMAINS version 1.0 would
626 be:

627

628 urn:sdmx:org.sdmx.infomodel.categoryscheme.Category=SDMX:SUBJE
629 CT_MATTER_DOMAINS(1.0).1.2

630 The URN for a Metadata Attribute maintained by SDMX in the MSD
631 CONTACT_METADATA version 1.0 in the Report Structure CONTACT_REPORT
632 where the hierarchy of the Metadata Attribute is
633 CONTACT_DETAILS/CONTACT_NAME would be:

634

635 urn:sdmx:org.sdmx.infomodel.metadatastructure.MetadataAttribut
636 e=SDMX:CONTACT_METADATA(1.0).CONTACT_REPORT.CONTACT_DETAILS.CO
637 NTACT_NAME

638 The TFFS defines ABC as a sub Agency of TFFS then the URN of a Dataflow
639 maintained by ABC and identified as EXTERNAL_DEBT version 1.0 would be:

640

641 urn:sdmx:org.sdmx.infomodel.datastructure.Dataflow=TFFS.ABC:EX
642 TERNAL_DEBT(1.0)

643

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

648

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

651 • **Maintainable Artefacts:** id, version, agencyId,

652

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

655 **6.2.3 Table of SDMX-IM Packages and Classes**

656 The table below lists all of the packages in the SDMX-IM together with the concrete
657 classes that are in these packages and whose objects have a URN.
658

Package	URN Classname (model classname where this is different)
base	Agency
	OrganisationUnitScheme
	AgencyScheme
	DataProviderScheme
	DataConsumerScheme
	OrganisationUnit
	DataProvider
	DataConsumer
datastructure	DataStructure (DataStructureDefinition)
	AttributeDescriptor
	DataAttribute
	GroupDimensionDescriptor
	DimensionDescriptor
	Dimension
	MeasureDimension
	TimeDimension
	MeasureDescriptor
	PrimaryMeasure
	Dataflow (DataflowDefinition)
metadatastructure	MetadataTarget
	DimensionDescriptorValueTarget
	IdentifiableObjectTarget
	ReportPeriodTarget
	DataSetTarget
	ReportStructure
	MetadataAttribute
	MetadataStructure (MetadataStructureDefinition)
	Metadataflow (MetadataflowDefinition)
process	Process
	ProcessStep
	Transition
registry	ProvisionAgreement
	AttachmentConstraint
	ContentConstraint
	Subscription
mapping	StructureMap
	StructureSet
	ComponentMap

Package	URN Classname (model classname where this is different)
	ConceptSchemeMap
	OrganisationSchemeMap
	CodelistMap
	CategorySchemeMap
	ReportingTaxonomyMap
	ConceptMap
	OrganisationMap
	CodeMap
	HybridCodelistMap
	CategoryMap
	HybridCodeMap
	ReportingCategoryMap
codelist	Codelist
	HierarchicalCodelist
	Hierarchy
	Hierarchy
	Code
	HierarchicalCode
	Level
categoryscheme	CategoryScheme
	Category
	Categorisation
	ReportingTaxonomy
	ReportingCategory
conceptscheme	ConceptScheme
	Concept

659

Table 2: SDMX-IM Packages and Contained Classes

660 **6.2.4 URN Identification components of SDMX objects**

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

664

665 * indicates that the object is maintainable.

666

667 Note that for brevity the URN examples omit the prefix. All URNs have the prefix

668

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

670

SDMX Class	Key attribute(s)	Example of URN
Agency	The URN for an Agency is shown later in this table. 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.	IMF Sub agency in the IMF AGENCIES IMF.SubAgency1
*ConceptScheme	conceptSchemeAgencyId:conceptSchemeId(version)	SDMX:CROSS_DOMAIN_CONCEPTS(1.0)
Concept	conceptSchemeAgencyId: conceptSchemeId(version).conceptId	SDMX:CROSS_DOMAIN_CONCEPTS(1.0).FREQ
*Codelist	codeListAgencyId:codeListId(version)	SDMX:CL_FREQ(1.0)
Code	codeListAgencyId:codelistId(version).codeId	SDMX:CL_FREQ(1.0).Q

*Hierarchical Codelist	hierachicalCodelistAgencyId: hierachicalCodelistId(version)	UNESCO:CL_EXP_SOURCE(1.0)
Hierarchy	hierachicalcodeListAgencyId: hierachicalcodelistId(version).Hierarchy	UNESCO:CL_EXP_SOURCE(1.0). H-C-GOV
Level	hierachicalcodeListAgencyId: hierachicalcodelistId(version).Hierarchy.Level	ESTAT:HCL_REGION(1.0).H_1.COUNTRY
HierarchicalCode	hierachicalCodeListAgencyId: hierachicalcodelistId(version).hierarchy.hierarc hicalCode	UNESCO:CL_EXP_SOURCE(1.0). H-C-GOV.GOV_CODE1
*DataStructure	dataStructureDefintitionAgencyId: dataStructureDefintitionId(version)	TFFS:EXT_DEBT(1.0)
Dimension Descriptor Measure Descriptor Attribute Descriptor	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).DimensionDescriptor TFFS:EXT_DEBT(1.0).MeasureDescriptor TFFS:EXT_DEBT(1.0).AttributeDescriptor
GroupDimension Descriptor	dataStructureDefinitionAgencyId: dataStructureDefinitionId(version). groupDimensionDescriptorId	TFFS:EXT_DEBT(1.0).SIBLING
Dimension	dataStructureDefinitionAgencyId: dataStructureDefinition (version). dimensionId	TFFS:EXT_DEBT(1.0).FREQ
TimeDimension	dataStructureDefinitionAgencyId: dataStructureDefinition (version). timeDimensionId	TFFS:EXT_DEBT(1.0).TIME_PERIOD
Measure Dimension	dataStructureDefinitionAgencyId: dataStructureDefinition (version).	TFFS:EXT_DEBT(1.0).STOCK_FLOW

	measureDimensionId	
DataAttribute	dataStructureDefinitionAgencyId: dataStructureDefinition (version). dataAttributeId	TFFS:EXT_DEBT(1.0).OBS_STATUS
PrimaryMeasure	dataStructureDefinitionAgencyId: dataStructureDefinition (version). primaryMeasureId	TFFS:EXT_DEBT(1.0).OBS_VALUE
*Category Scheme	categorySchemeAgencyId: categorySchemeId(version)	IMF:SDDS(1.0)
Category	categorySchemeAgencyId: categorySchemeId(version). categoryId.categoryId categoryId.categoryId etc.	IMF:SDDS(1.0): level_1_category.level_2_category ...
*Reporting Taxonomy	reportingTaxonomyAgencyId: reportingTaxonomyId(version)	IMF:REP_1(1.0)
ReportingCategory	reportingTaxonomyAgencyId: reportingTaxonomyId(version) reportingcategoryId.reportingcategoryId	IMF:REP_1(1.0): level_1_repcategory.level_2_repcategory ...
*Categorisation	categorisationAgencyId: categorisationId(version)	IMF:cat001(1.0)
*Organisation Unit Scheme	organisationUnitSchemeAgencyId: organisationUnitSchemeId(version)	ECB:ORGANISATIONS(1.0)
Organisation Unit	organisationUnitSchemeAgencyId: organisationUnitSchemeId(version). organisationUnitId	ECB:ORGANISATIONS(1.0).1F
*AgencyScheme	agencySchemeAgencyId: agencySchemeId(version)	ECB:AGENCIES(1.0)

Agency	agencySchemeAgencyId: agencySchemeId(version). agencyId	ECB:AGENCY(1.0).AA
*DataProvider Scheme	dataProviderSchemeAgencyId: dataProviderSchemeId(version)	SDMX:DATA_PROVIDERS(1.0)
DataProvider	dataProviderSchemeAgencyId: dataProviderSchemeId(version) dataProviderId	SDMX:DATA_PROVIDERS(1.0).PROVIDER_1
*DataConsumer Scheme	dataConsumerSchemeAgencyId: dataConsumerSchemeId(version)	SDMX:DATA_CONSUMERS(1.0)
Data Consumer	dataConsumerSchemeAgencyId: dataConsumerSchemeId(version) dataConsumerId	SDMX:DATA_CONSUMERS(1.0).CONSUMER_1
*Metadata Structure	MSDAgencyId:MSDId(version)	IMF:SDDS_MSD(1.0)
MetadataTarget	MSDAgencyId: MSDId(version).metadataTargetId	IMF:SDDS_MSD(1.0).AGENCY
Dimension DescriptorValues Target	MSDAgencyId: MSDId(version). metadataTargetId.keyDescriptorValueTargetId	IMF:SDDS_MSD(1.0).AGENCY.KEY
Identifiable ObjectTarget	MSDAgencyId: MSDId(version).metadataTargetId.identifiable ObjectTargetId	IMF:SDDS_MSD(1.0).AGENCY.STR-OBJECT
DataSetTarget	MSDAgencyId: MSDId(version).metadataTargetId.dataSet TargetId	IMF:SDDS_MSD(1.0).AGENCY.D1101
ReportPeriod Target	MSDAgencyId: MSDId(version).metadataTargetId.reportPeriod TargetId	IMF:SDDS_MSD(1.0).AGENCY.REP_PER

ReportStructure	MSDAgencyId: MSDId(version).reportStructureId	IMF:SDDS_MSD(1.0).AGENCY_REPORT
Metadata Attribute	MSDAgencyId: MSDId(version).reportStructureId.metadataAttributeId	IMF:SDDS_MSD(1.0).AGENCY_REPORT.COMPILED
*Dataflow	dataflowAgencyId: dataflowId(version)	TFFS:CRED_EXT_DEBT(1.0)
*Provision Agreement	provisionAgreementAgencyId:provisionAgreementId(version)	TFFS:CRED_EXT_DEBT_AB(1.0)
*Content Constraint	constraintAgencyId:ContentConstraintId(version)	TFFS:CREDITOR_DATA_CONTENT(1.0)
*Attachment Constraint	constraintAgencyId: attachmentConstraintId(version)	TFFS:CREDITOR_DATA_ATTACHMENT_CONSTRAINT_ONE(1.0)
*Metadataflow	metadataflowAgencyId: metadataflowId(version)	IMF:SDDS_FLOW(1.0)
*StructureSet	structureSetAgencyId: structureSetId(version)	SDMX:BOP_STRUCTURES(1.0)
StructureMap	structureSetAgencyId: structureSetId(version). structureMapId	SDMX:BOP_STRUCTURES(1.0).TABLE1_TABLE2
Component Map	structureSetAgencyId: structureSetId(version). structureMapId. componentMapId	SDMX:BOP_STRUCTURES(1.0).TABLE1_TABLE2. REFAREA_REPCOUNTRY
CodelistMap	structureSetAgencyId: structureSetId(version). codelistMapId	SDMX:BOP_STRUCTURES(1.0).CLREFAREA_CLREPCOUNTRY
CodeMap	structureSetAgencyId: structureSetId(version).	SDMX:BOP_STRUCTURES(1.0).CLREFAREA_CLREPCOUNTRY. DE_GER

	codeListMapId. codeMapId	
Category SchemeMap	structureSetAgencyId: structureSetId(version). categorySchemeMapId	SDMX:BOP_STRUCTURES(1.0).SDMX_EUROSTAT
CategoryMap	structureSetAgencyId: structureSetId(version). categorySchemeMapId. categoryMapId	SDMX:BOP_STRUCTURES(1.0).SDMX_EUROSTAT.TOURISM_M AP
Organisation SchemeMap	structureSetAgencyId: structureSetId(version). organisationSchemeMapId	SDMX:BOP_STRUCTURES(1.0).DATA_PROVIDER_MAP
Organisation Map	structureSetAgencyId: structureSetId(version). organisationSchemeMapId. organisationMapId	SDMX:BOP_STRUCTURES(1.0).DATA_PROVIDER_MAP.IMF_1C0
Concept SchemeMap	structureSetAgencyId: structureSetId(version). conceptSchemeMapId	SDMX:BOP_STRUCTURES(1.0).SDMX_OECD
ConceptMap	structureSetAgencyId: structureSetId(version). conceptSchemeMapId. conceptMapId	SDMX:BOP_STRUCTURES(1.0).SDMX_OECD.COVERAGE_AVAI LABILITY
Reporting TaxonomyMap	structureSetAgencyId: structureSetId(version). reportingTaxonomyMapId	SDMX:BOP_STRUCTURES(1.0).TAXMAP
Reporting CategoryMap	structureSetAgencyId: structureSetId(version). reportingCategoryId	SDMX:BOP_STRUCTURES(1.0).TAXMAP.TOPCAT

HybridCodelist Map	structureSetAgencyId: structureSetId(version). hybridCodelistMapId.	SDMX:BOP_STRUCTURES(1.0).COUNTRY_HIERARCHYMAP
HybridCodeMap	structureSetAgencyId: structureSetId(version). hybridCodelistMapId. hybridCodeMapId	SDMX:BOP_STRUCTURES(1.0).COUNTRY_HIERARCHYMAP.CO DEMAP1
*Process	processAgencyId: processId{version]	BIS:PROCESS1(1.0)
ProcessStep	processAgencyId: processId(version). processStepId	BIS:PROCESS1(1.0).STEP1
Transition	processAgencyId: processId(version). processStepId transitionId	BIS:PROCESS1(1.0).STEP1.TRANSITION1
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 registryURN that is returned from the registry in the subscription response message.

Table 3: Table of identification components for SDMX Identifiable Artefacts

672 7 Implementation Notes

673 7.1 Structural Definition Metadata

674 7.1.1 Introduction

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

682
683 At a minimum, structural metadata definitions may be submitted to and queried from
684 the registry via an HTTP/HTTPS POST in the form of one of the SDMX-ML registry
685 messages for structural metadata and the SDMX Query message for structure
686 queries. The use of SOAP is also recommended, as described in the SDMX Web
687 Services Guidelines. The message may contain all structural metadata items for the
688 whole registry, structural metadata items for one maintenance agency, or individual
689 structural metadata items.

690
691 Structural metadata items

- 692 • may only be modified by the maintenance agency which created them
- 693 • may only be deleted by the agency which created them
- 694 • may not be deleted if they are referenced from other constructs in the
695 Registry

696
697 The level of granularity for the maintenance of SDMX Structural Metadata objects in
698 the registry is the Maintainable Artefact. In other words, any function such as add,
699 modify, delete is at the level of the Maintainable Artefact. For instance, if a Code is
700 added to a Code List, or the Name of a Code is changed, the Registry must replace
701 the existing Code List with the submitted Code List of the same Maintenance
702 Agency, Code List, Id and Version.

703
704 The following table lists the Maintainable Artefacts.
705

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
	Data Consumer Scheme	Data Consumer
	Reporting Taxonomy	Reporting Category
Structure	Data Structure Definition	Dimension Descriptor Group Dimension

Maintainable Artefacts		Content
Abstract Class	Concrete Class	
		Descriptor Dimension Measure Dimension Time Dimension Attribute Descriptor Data Attribute Measure Descriptor Primary Measure
	Metadata Structure Definition	Metadata Target, Dimension Descriptor Values Target Identifiable Object Target Report Period Target Data SetTarget Report Structure Metadata Attribute
Structure Usage	Dataflow Definition	
	Metadataflow Definition	
None	Process	Process Step
None	Structure Set	Component Map Concept Scheme Map Codelist Map Category Scheme Map Reporting Taxonomy Map Organisation Scheme Map Concept Map Code Map Category Map Organisation Map Reporting Category Map Hybrid Codelist Map Hybrid Code Map
None	Provision Agreement	
None	Hierarchical Codelist	Hierarchy Hierarchical Code

706 **Table 4: Table of Maintainable Artefacts for Structural Definition Metadata**

707 **7.1.2 Item Scheme, Structure**

708 The artefacts included in the structural definitions are:

709

- 710 • All types of Item Scheme (Codelist, Concept Scheme, Category Scheme,
711 Organisation Scheme - Agency Scheme, Data Provider Scheme, Data
712 Consumer Scheme, Organisation Unit Scheme)

- 713 • All types of Structure (Data Structure Definition, Metadata Structure
714 Definition)

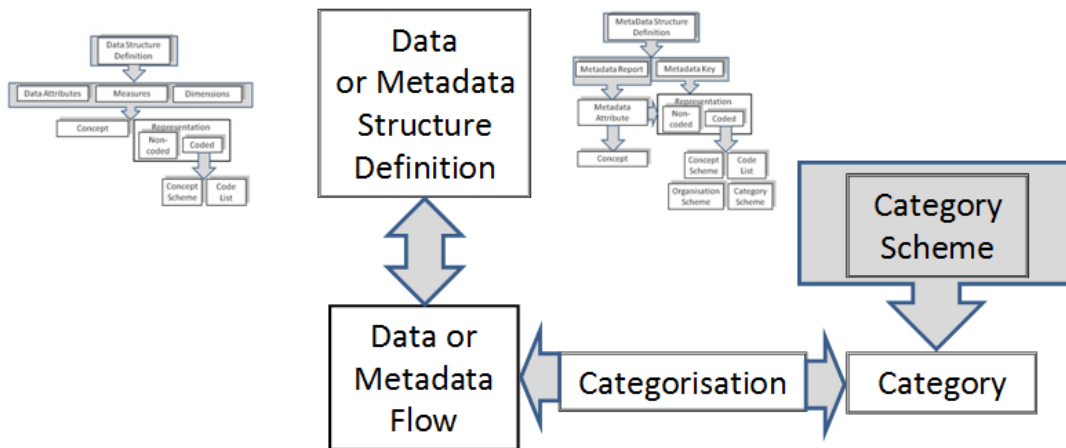
- 715 • All types of Structure Usage (Dataflow Definition, Metadataflow Definition)

716 **7.1.3 Structure Usage**

717 **7.1.3.1 Structure Usage: Basic Concepts**

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

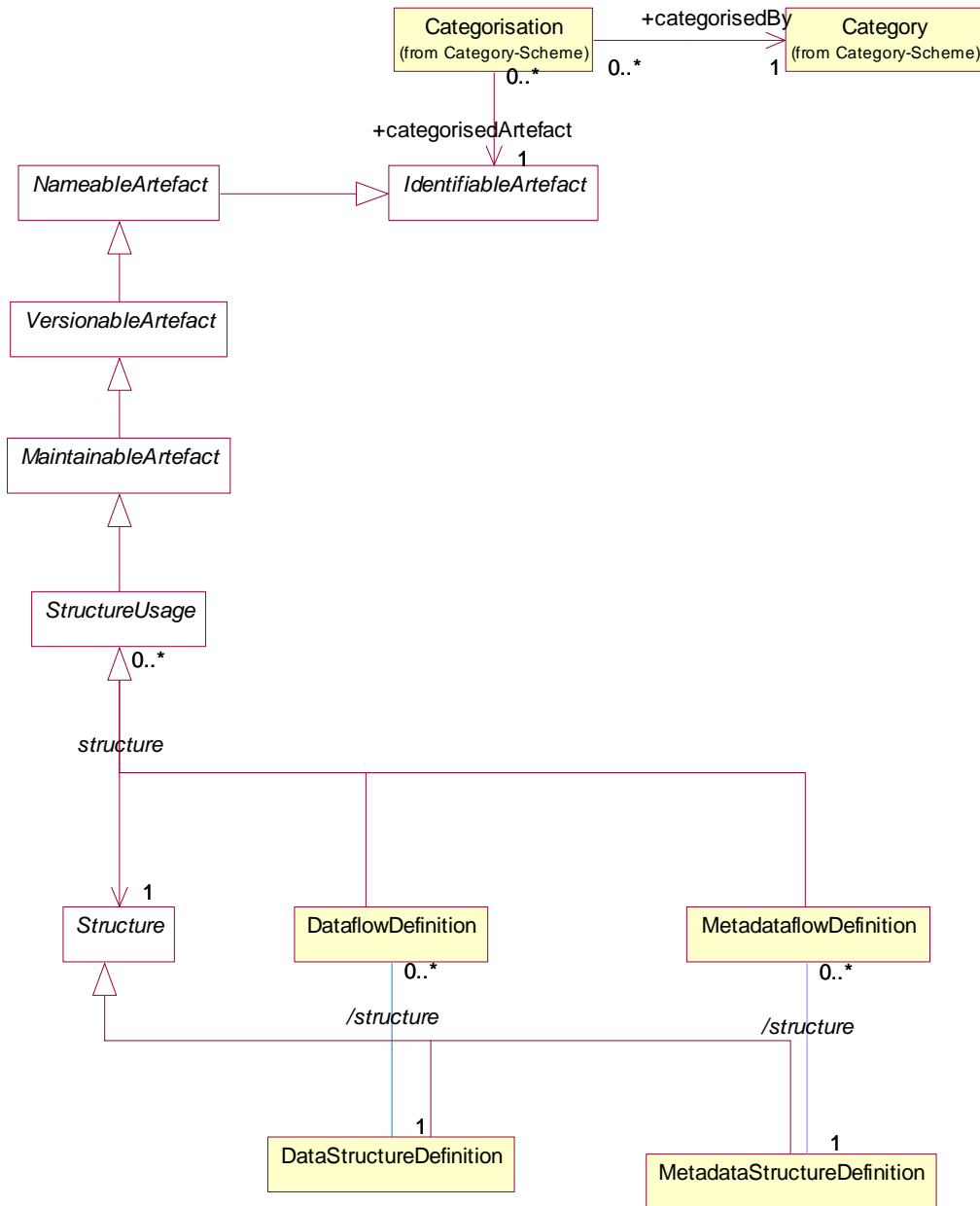
724 **7.1.3.2 Structure Usage Schematic**



725
 726
 727

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

728 **7.1.3.3 Structure Usage Model**



729
730

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

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

733

734

- Dataflow Definition to Data Structure Definition

735

- Metadataflow Definition to Metadata Structure Definition

736

The following links may be created by means of a Categorisation

737 • Categorisation to Dataflow Definition and Category

738 • Categorisation to Metadataflow Definition and Category

739 **7.2 Data and Metadata Provisioning**

740 **7.2.1 Provisioning Agreement: Basic concepts**

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

751

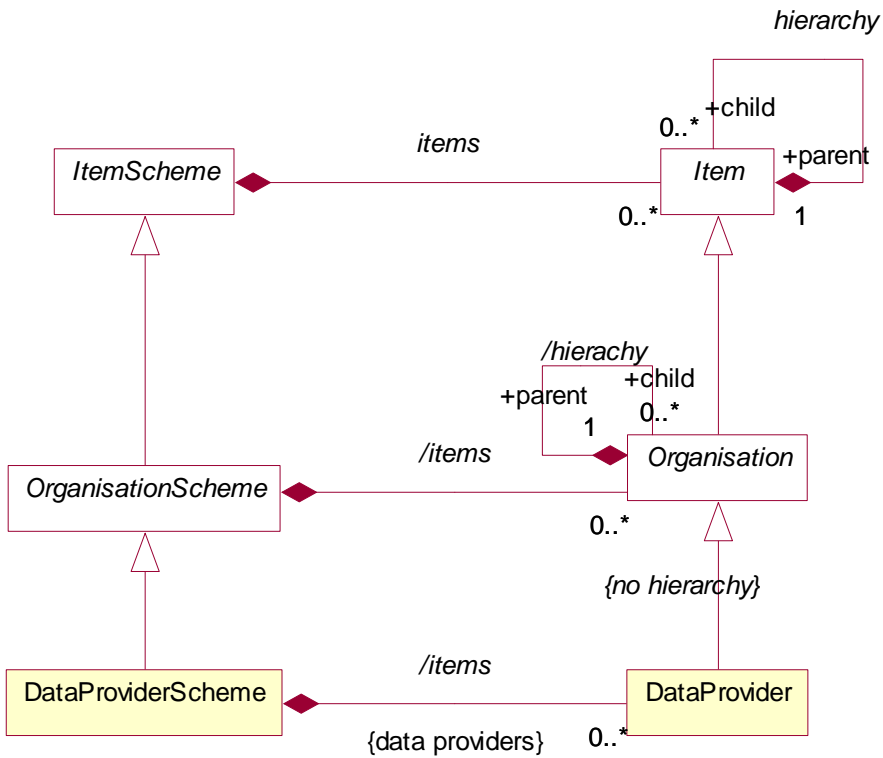
752 Note that in the SDMX-IM the class “Data Provider” encompasses both data and
753 metadata and the term “data provisioning” here includes both the provisioning of data
754 and metadata.

755

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

760 **7.2.2 Provisioning Agreement Model – pull use case**

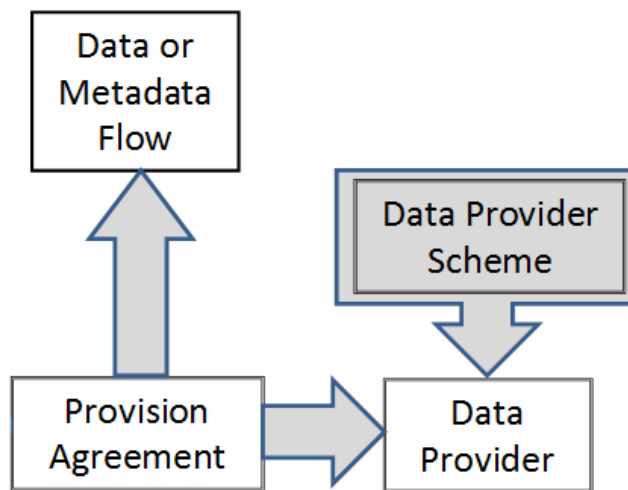
761 An organisation which publishes statistical data or reference metadata and wishes to
762 make it available to an SDMX enabled community is called a Data Provider. In terms
763 of the SDMX Information Model, the Data Provider is maintained in a Data Provider
764 Scheme.



765
766
767
768
769
770
771

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



772
773
774
775
776

Figure 12: Schematic of the Provision Agreement

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

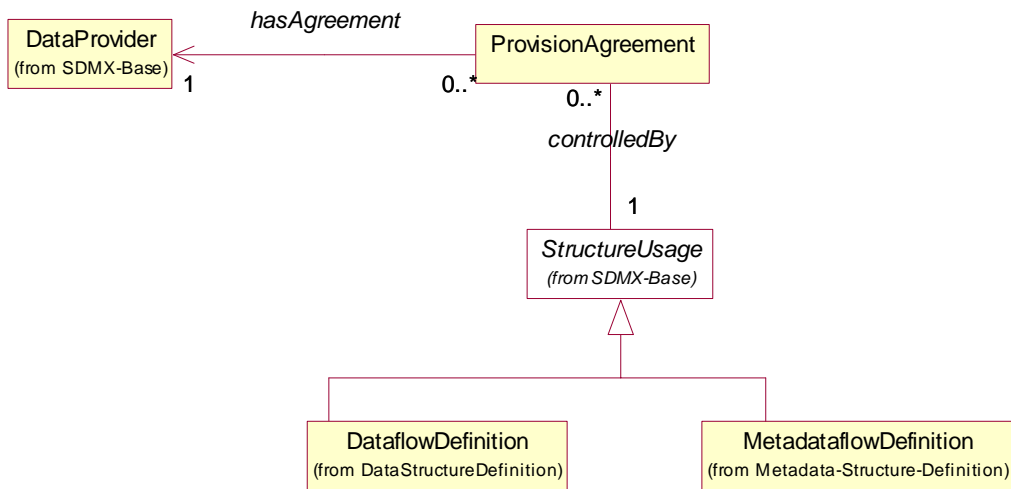


Figure 13: Logical class diagram of the information contained in the Provision Agreement

777
778
779

780 A Provision Agreement is structural metadata. Each Provision Agreement must
781 reference a Data Provider and a Dataflow or Metadataflow Definition. The Data
782 Provider and the Dataflow/Metadataflow Definition must exist already in order to set
783 up a Provision Agreement.

784 **7.3 Data and Metadata Constraints**

785 **7.3.1 Data and Metadata Constraints: Basic Concepts**

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

791
792 Constraints comprise the specification of subsets of key or target values or attribute
793 values that are contained in a Datasource, or is to be provided for a Dataflow or
794 Metadataflow Definition, or directly attached to a Data Structure Definition or
795 Metadata Structure Definition. This is important metadata because, for example, the
796 full range of possibilities which is implied by the Data Structure Definition (e.g. the
797 complete set of valid keys is the Cartesian product of all the values in the code lists
798 for each of the Dimensions) is often more than is actually present in any specific
799 Datasource, or more than is intended to be supplied according to a specific Dataflow
800 Definition.

801

802 Often a Data Provider will not be able to provide data for all key combinations, either
803 because the combination itself is not meaningful, or simply because the provider
804 does not have the data for that combination. In this case the Data Provider could
805 constrain the Datasource (at the level of the Provision Agreement or the Data
806 Provider) by supplying metadata that defines the key combinations or cube regions
807 that are available. This is done by means of a Constraint. The Content Constraint is
808 also used to define a code list sub set which is used to populate a Partial Code List.

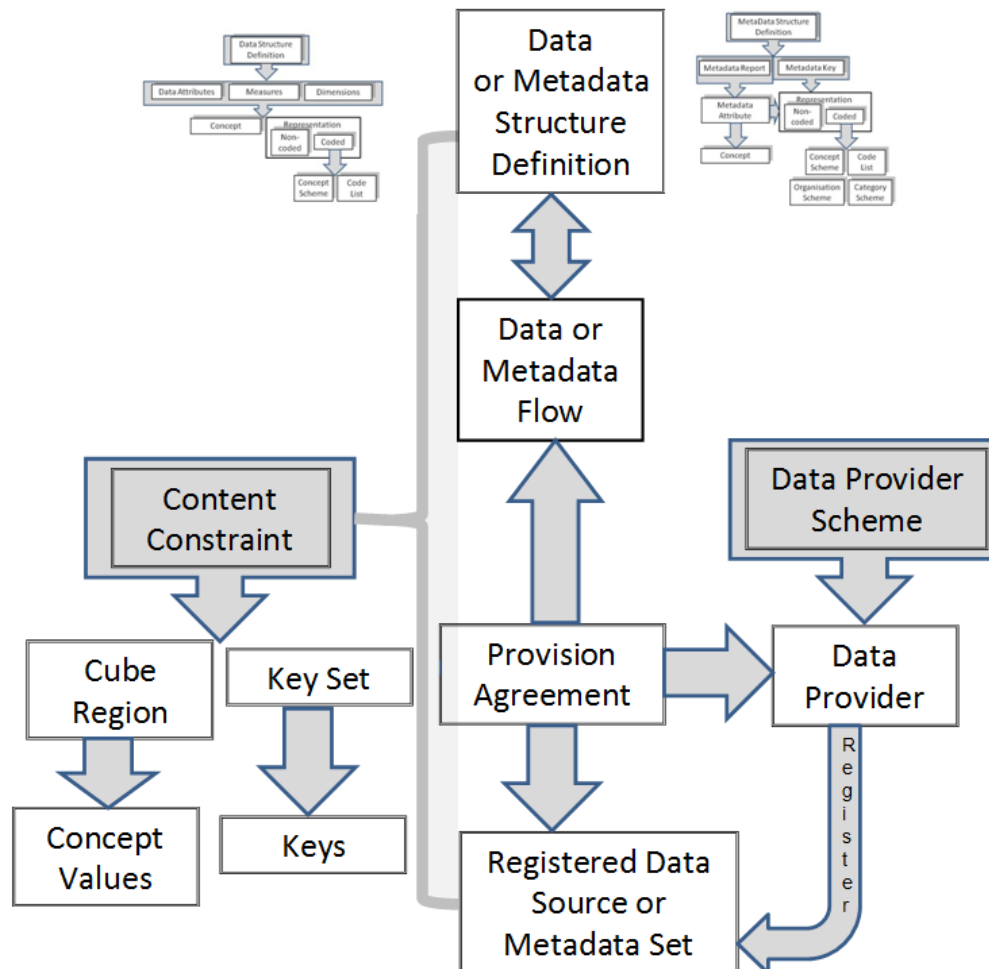
809

810 Furthermore, it is often useful to define subsets or views of the Data Structure
 811 Definition which restrict values in some code lists, especially where many such
 812 subsets restrict the same Data Structure Definition. Such a view is called a Dataflow
 813 Definition, and there can be one or more defined for any Data Structure Definition.
 814

815 Whenever data is published or made available by a Data Provider, it must conform to
 816 a Dataflow Definition (and hence to a Data Structure Definition). The Dataflow
 817 Definition is thus a means of enabling content based processing.
 818

819 In addition, Constraints can be extremely useful in a data visualisation system, such
 820 as dissemination of statistics on a website. In such a system a Cube Region can be
 821 used to specify the Dimension codes that actually exist in a datasource (these can be
 822 used to build relevant selection tables), and the Key Set can be used to specify the
 823 keys that exist in a datasource (these can be used to guide the user to select only
 824 those Dimension code values that will return data based on the Dimension values
 825 already selected).

826 **7.3.2 Data and Metadata Constraints: Schematic**



827
 828 **Figure 14: Schematic of the Constraint and the Artefacts that can be Constrained**
 829

845 **7.4 Data and Metadata Registration**

846 **7.4.1 Basic Concepts**

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

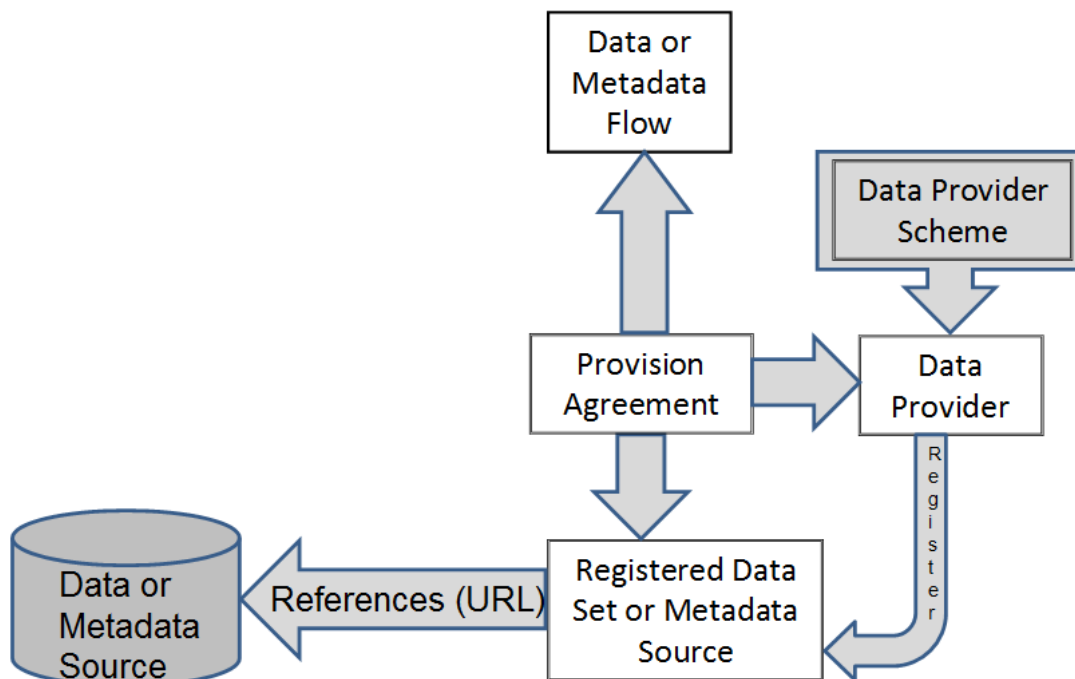
852
 853 The Data Provider wishes to make this new data available to one or more data
 854 collectors in a “pull” scenario, or to make the data available to data consumers. To do
 855 this, the Data Provider registers the new dataset with one or more SDMX conformant
 856 registries that have been configured with structural and provisioning metadata. In
 857 other words, the registry “knows” the Data Provider and “knows” what data flows the
 858 data provider has agreed to make available.

859
 860 The same mechanism can be used to report or make available a metadata set.

861
 862 SDMX-RR supports dataset and metadata set registration via the Registration
 863 Request, which can be created by the Data Provider (giving the Data Provider
 864 maximum control). The registry responds to the registration request with a
 865 registration response which indicates if the registration was successful. In the event
 866 of an error, the error messages are returned as a registry exception within the
 867 response.

868 **7.4.2 The Registration Request**

869 **7.4.2.1 Registration Request Schematic**



870
 871 **Figure 16: Schematic of the Objects Concerned with Registration**
 872

873 **7.4.2.2 Registration Request Model**

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

- 878
- 879 • validFrom
 - 880 • validTo
 - 881 • lastUpdated

882

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

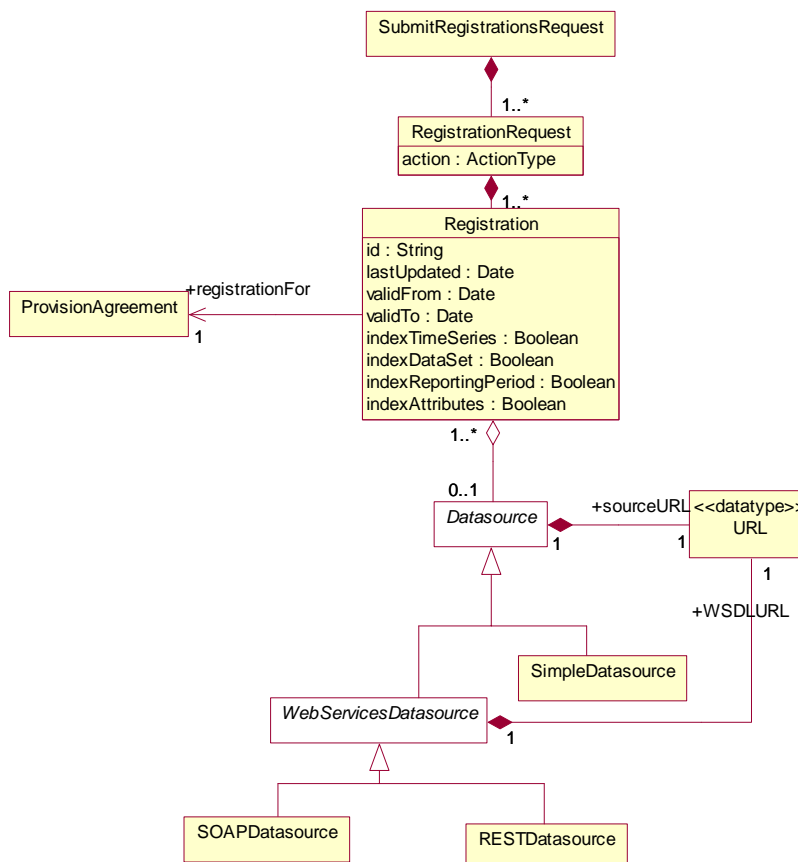
885

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

887

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 Submit Registration Request
Delete	Delete the existing Registration identified by the id in the Registration of the Submit Registration Request

888



889

890

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

891 The Query Datasource is an abstract class that represents a data source which can
 892 understand an SDMX-ML query (SOAPDatasource) or RESTful query
 893 (RESTDatasource) and respond appropriately. Each of these different Datasources
 894 inherit the dataURL from Datasource, and the QueryDatasource has an additional
 895 URL to locate a WSDL or WADL document to describe how to access it. All other
 896 supported protocols are assumed to use the Simple Datasource URL.

897
 898 A Simple Datasource is used to reference a physical SDMX-ML file that is available
 899 at a URL.

900
 901 The Registration Request has an action attribute which defines whether this is a new
 902 (append) or updated (replace) Registration, or that the Registration is to be deleted
 903 (delete). The id is only provided for the replace and delete actions, as the Registry
 904 will allocate the unique id of the (new) Registration.

905
 906 The Registration includes attributes that state how a Simple Datasource is to be
 907 indexed when registered. The Registry registration process must act as follows.

908
 909 Information in the data or metadata set is extracted and placed in one or more
 910 Content Constraints (see the Constraints model in the SDMX Information Model –
 911 Section 2 of the SDMX Standards). The information to be extracted is indicated by
 912 the Boolean values set on the Provision Agreement as shown in the table below.

913

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 <u>dataset</u> . Extract the Reporting Begin and Reporting End from the Header of the Message containing the data set, and create a Reference Period constraint.
indexAttributes	<p>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 SDMX-IM, and the associated Selection Value</p> <p>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 SDMX-IM. Note that the content is not stored in the Selection Value.</p>

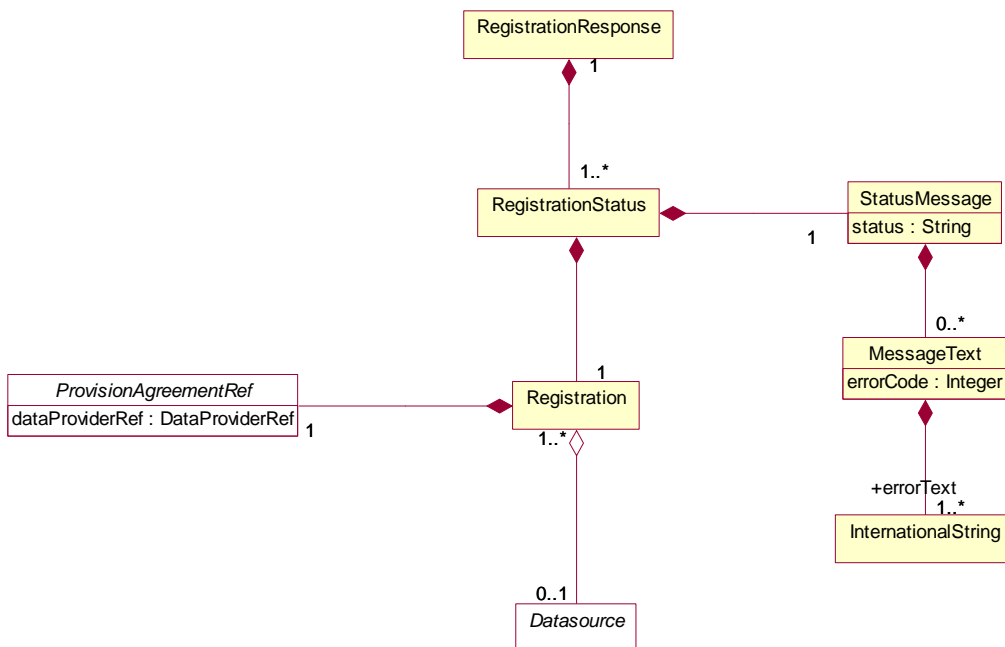
914
 915 Constraints that specify the contents of a Query Datasource are submitted to the
 916 Registry in a Submit Structure Request.
 917
 918 The Registration must reference the Provision Agreement to which it relates.

919 **7.4.3 Registration Response**

920 After a registration request has been submitted to the registry, a response is returned
 921 to the submitter indicating success or failure. Given that a registration request can
 922 hold many Registrations, then there must be a registration status for each
 923 Registration. The Submit Registration class has a status field which is either set to
 924 “Success”, “Warning” or “Failure”.

925
 926 If the registration has succeeded, a Registration will be returned - this holds the
 927 Registry-allocated Id of the newly registered Datasource plus a Datasource holding
 928 the URL to access the dataset, metadataset, or query service.

929
 930 The Registration Response returns set of registration status (one for each
 931 registration submitted) in terms of a Status Message (this is common to all Registry
 932 Responses) that indicates success or failure. In the event of registration failure, a set
 933 of Message Text are returned, giving the error messages that occurred during
 934 registration. It is entirely possible when registering a batch of datasets, that the
 935 response will contain some successful and some failed statuses. The logical model
 936 for the Registration Response is shown below:
 937



938
 939 **Figure 18: Logical class diagram showing the registration response**

940 **7.5 Subscription and Notification Service**

941 The contents of the SDMX Registry/Repository will change regularly: new code lists
 942 and key families will be published, new datasets and metadata-sets will be
 943 registered. To obviate the need for users to repeatedly query the registry to see when

944 new information is available, a mechanism is provided to allow users to be notified
945 when these events happen.

946

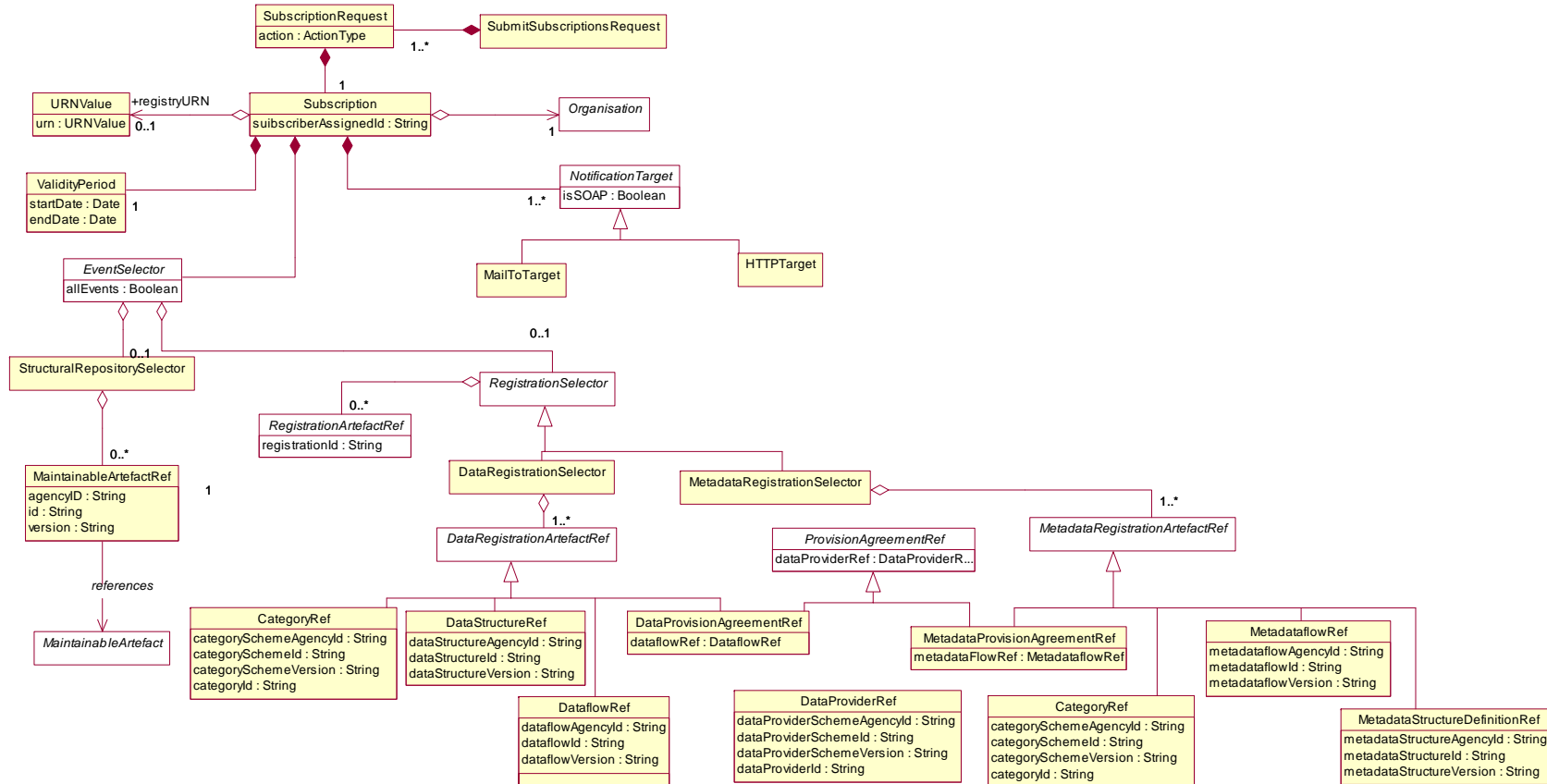
947 A user can submit a subscription in the registry that defines which events are of
948 interest, and either an email and/or an HTTP address to which a notification of
949 qualifying events will be delivered. The subscription will be identified in the registry by
950 a URN which is returned to the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

955

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

960

961 **7.5.1 Subscription Logical Class Diagram**
962



963
964

Figure 19: Logical Class Diagram of the Subscription

965 **7.5.2 Subscription Information**

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

- 968
- 969 1. A set of URIs describing the end-points to which notifications must be sent if
970 the subscription is activated. The URIs can be either mailto: or http: protocol.
971 In the former case an email notification is sent; in the latter an HTTP POST
972 notification is sent.
 - 973 2. A user-defined identifier which is returned in the response to the subscription
974 request. This helps with asynchronous processing and is NOT stored in the
975 Registry.
 - 976 3. A validity period which defines both when the subscription becomes active
977 and expires. The subscriber may be sent a notification on expiration of the
978 subscription.
 - 979 4. A selector which specifies which type of events are of interest. The set of
980 event types is:
- 981

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 SDMX-ML data file or an SDMX conformant database.
METADATA_REGISTRATION_EVENTS	Whenever a published metadataset is registered. This can be either a SDMX-ML reference metadata file or an SDMX conformant database.
ALL_EVENTS	All events of the specified EventType

982 **7.5.3 Wildcard Facility**

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

- 986
- 987 • agencyID
 - 988 • id
 - 989 • version

990

991 Examples of wildcarded identifier components for an identified object type of Codelist
992 are shown below.

993

994 AgencyID = %
995 Id = %
996 Version = %

997

998 This subscribes to all Codelists of all versions for all agencies.

999

1000 AgencyID = AGENCY1
1001 Id = CODELIST1
1002 Version = %

- 1003
- 1004 This subscribes to all versions of Codelist CODELIST1 maintained by the agency
- 1005 AGENCY1
- 1006
- 1007 AgencyID = AGENCY1
- 1008 Id = %
- 1009 Version = %
- 1010
- 1011 This subscribes to all versions of all Codelist objects maintained by the agency
- 1012 AGENCY1
- 1013
- 1014 AgencyID = %
- 1015 Id = CODELIST1
- 1016 Version = %
- 1017
- 1018 This subscribes to all versions of Codelist CODELIST1 maintained by the agency
- 1019 AGENCY1
- 1020
- 1021 Note that if the subscription is to the latest version then this can be achieved by the *
- 1022 character
- 1023
- 1024 i.e. Version = *
- 1025
- 1026 Note that a subscription using the URN mechanism cannot use wildcard characters.

1027 **7.5.4 Structural Repository Events**

1028 Whenever a maintainable artefact (data structure definition, concept scheme,
 1029 codelist, metadata structure definition, category scheme, etc.) is added to, deleted
 1030 from, or modified in the structural metadata repository, a structural metadata event is
 1031 triggered. Subscriptions may be set up to monitor all such events, or focus on
 1032 specific artefacts such as a Data Structure Definition.

1033 **7.5.5 Registration Events**

1034 Whenever a dataset or metadata-set is registered a registration event is created. A
 1035 subscription may be observing all data or metadata registrations, or it may focus on
 1036 specific registrations as shown in the table below:

1037

Selector	Comment
DataProvider	Any datasets or metadata sets registered by the specified data provider will activate the notification.
ProvisionAgreement	Any datasets or metadata sets registered for the provision agreement will activate the notification.
Dataflow (&Metadataflow)	Any datasets or metadata sets registered for the specified dataflow (or metadataflow) will activate the notification.
DataStructureDefinition & MetadataStructureDefinition	Any datasets or metadata sets registered for those dataflows (or metadataflows) that are based on the specified Data Structure Definition will

Selector	Comment
	activate the notification.
Category	Any datasets or metadata sets registered for those dataflows, metadataflows, provision agreements that are categorised by the category.

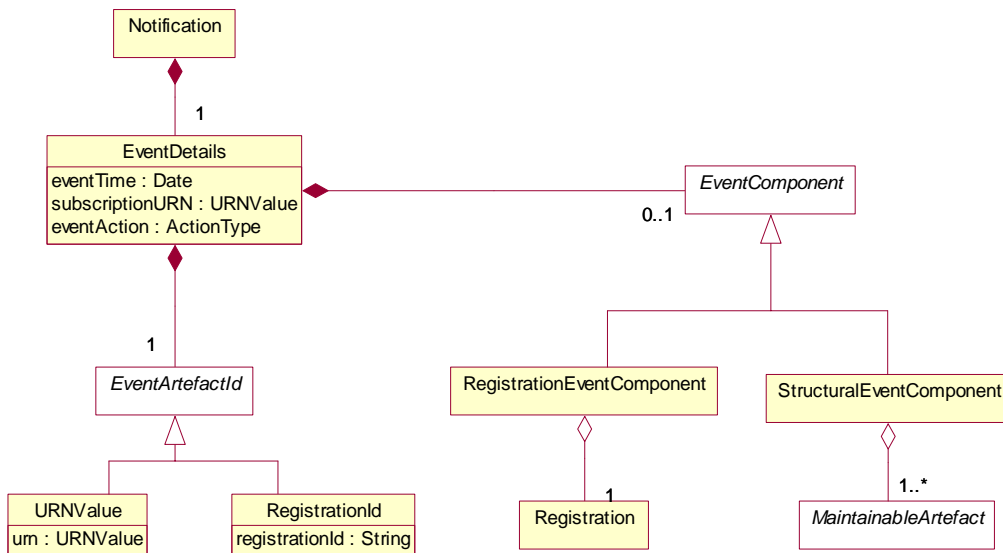
1038

1039 The event will also capture the semantic of the registration: deletion or replacement
 1040 of an existing registration or a new registration.

1041 **7.6 Notification**

1042 **7.6.1 Logical Class Diagram**

1043



1044

1045

Figure 20: Logical Class Diagram of the Notification

1046

1047 A notification is an XML document that is sent to a user via email or http POST
 1048 whenever a subscription is activated. It is an asynchronous one-way message.

1049

1050 Regardless of the registry component that caused the event to be triggered, the
 1051 following common information is in the message:

1052

1053

1054

1055

1056

1057

- 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.

1058

1059

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

1060 **7.6.2 Structural Event Component**

1061

1062

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).

- 1063 **7.6.3 Registration Event Component**
- 1064 The notification will contain the Registration.
- 1065