



Guidelines for the Use of Code Lists

Document Metadata

Property	Value
Date	2018-05-30
Status	Accepted
Version	1.00
Authors	Makx Dekkers – AMI Consult Daniel Brule – PwC EU Services Alexandru Droscariu – PwC EU Services Ioana Novacean – PwC EU Services
Reviewed by	Nikolaos Loutas – PwC EU Services
Approved by	Susanne Wigard – European Commission, ISA ² Programme

This study was prepared for the ISA Programme by:

PwC EU Services

Disclaimer:

The views expressed in this report are purely those of the authors and may not, in any circumstances, be interpreted as stating an official position of the European Commission.

The European Commission does not guarantee the accuracy of the information included in this study, nor does it accept any responsibility for any use thereof.

Reference herein to any specific products, specifications, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favouring by the European Commission.

All care has been taken by the author to ensure that s/he has obtained, where necessary, permission to use any parts of manuscripts including illustrations, maps, and graphs, on which intellectual property rights already exist from the titular holder(s) of such rights or from her/his or their legal representative.

Table of Contents

1	INTRODUCTION	3
1.1	CONTEXT	3
1.2	SCOPE AND OBJECTIVES.....	4
1.3	TARGET AUDIENCE	4
1.4	STRUCTURE	5
2	IDENTIFYING THE NEED FOR A CODE LIST AND SELECTING THE RIGHT ONE	6
3	MANAGEMENT OF THE CODE LIST LIFECYCLE.....	8
3.1	DESIGN.....	8
3.2	MANAGE CHANGES	11
3.3	RELEASE	12
3.4	RETIRE.....	14
3.5	USE AND EXTEND	15
3.6	CREATE MAPPINGS.....	19
3.7	MANAGE CODE LIST QUALITY	20
3.8	COMMUNICATION	20
4	CODE LIST GOVERNANCE	22
4.1	CONTEXT AND THE NEED FOR GOVERNANCE	22
4.2	GOVERNANCE TASKS AND STRUCTURE	22
4.3	SKILLS AND EXPERTISE REQUIRED FOR THE GOVERNANCE AND MANAGEMENT OF CODE LISTS	24
5	SUMMARY	25

List of Figures

FIGURE 1: EXCERPT OF ACTIVITYTYPECODE CEN BII CODE LIST	4
FIGURE 2: SCENARIOS FOR CODE LIST CONSUMERS	7
FIGURE 3: EXAMPLE OF MULTILINGUAL LABELLING IN XML	9
FIGURE 4: EXAMPLE OF MULTILINGUAL LABELLING IN RDF	9
FIGURE 5: EXAMPLE OF CONCEPT RELATIONSHIPS	10
FIGURE 6: EXAMPLE OF VERSIONING IN XML	12
FIGURE 7: EXAMPLE OF DEPRECATION IN XML	15
FIGURE 8: EXAMPLE OF DEPRECATION IN RDF	15
FIGURE 9: EXAMPLE INITIAL LIST	17
FIGURE 10: EXAMPLE PATTERN FOR NEW VALUES	17
FIGURE 11: EXAMPLE COMBINATION OF INITIAL LIST AND EXTENSION PATTERN	17
FIGURE 12: EXAMPLE OF XML INSTANCES AFTER EXTENSION	17
FIGURE 13: EXAMPLE OF EXTENSION FIELD IN THE SCHEMA FOR ACCOMMODATING NEW VALUES	18
FIGURE 14: EXAMPLE OF ADDITIONAL ATTRIBUTE	18
FIGURE 15: EXAMPLE XML INSTANCES.....	18
FIGURE 16: EXAMPLE OF DOCUMENTATION-BASED EXTENSION	19
FIGURE 17: TASKS RELATED TO CODE LIST GOVERNANCE	23
FIGURE 18: EXAMPLES OF CODE LIST GOVERNANCE TASKS	23
FIGURE 19: 10 TIPS FOR CODE LIST MANAGEMENT AND GOVERNANCE	25

1 INTRODUCTION

This document is part of TASK-06 “Tools and Methodologies” of [ISA²](#) Action 2016.07 “Promoting semantic interoperability among EU Member States”, commonly known as [SEMIC¹](#). This task aims to provide updates to the tools and methodologies developed by the SEMIC action. The current report’s purpose is to provide practical guidance to less experienced organisations on code lists.

1.1 Context

The SEMIC action has previously delivered extensive work in and around the development, maintenance, management, and governance of data specifications. Typically, the aforementioned work has focused on data models. In order to better support public administrations in their interoperability and information governance and management efforts, this document aims to act as a guide to developing or managing and consuming reusable code lists.

This deliverable builds upon the work previously carried out by the SEMIC action and included in [“Methodology and Tools for Metadata Governance and Management for EU Institutions”²](#).

Code lists are lists of values in a predefined set that can be used in metadata and that help metadata creators in selecting from a set of descriptors, thereby enhancing consistency and helping to avoid errors.

Code lists can be implemented using several technologies, in particular XML and RDF. In the case of RDF, code lists are usually called ‘controlled vocabularies’. For brevity, the term ‘*code list*’ is used throughout this document, irrespective of the expression in XML or RDF. Wherever necessary, the specific approaches related to implementation in XML and RDF will be explained.

For business and government messages, the use of code lists is an essential part of document alignment and data harmonisation. Many documents used in information exchanges require information about location, currency, dates, measurements, etc. The way this information is represented commonly differs in more or less subtle ways between countries and languages, despite being the same. IT systems therefore could either misunderstand this information, or not process it at all. The solution to this ambiguity is to have the information coded, creating an unambiguous way of representing it. Code lists minimise errors and reduce ambiguity, providing an essential contribution to interoperability.

Code lists have various uses. For instance, the MDR Data Theme vocabulary³ used in the DCAT Application Profile for data portals in Europe (DCAT-AP)⁴, as a common set of values for data themes, helps datasets published in different places to be classified according to a unified classification scheme. Entities such as the European Data Portal benefit from these interoperability tools, as they aggregate metadata from various

¹ SEMIC: <https://joinup.ec.europa.eu/collection/semantic-interoperability-community-semic>

² https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/methodology_and_tools_for_metadata_governance_and_management_for_eu_institutions.pdf

³ <http://publications.europa.eu/mdr/authority/data-theme/>

⁴ <https://joinup.ec.europa.eu/solution/dcat-application-profile-data-portals-europe>

catalogues⁵. Other examples in the context of the European Commission include: the work of the Publications Office on the [Metadata Registry Name Authority Lists](#)⁶ (MDR NAL) and [EuroVoc](#)⁷, and the work on the [Asset Description Metadata Schema](#) (ADMS) Controlled Vocabularies⁸. The European Committee for Standardisation (CEN) publishes Business Interoperability Interfaces (BII) code lists such as the ActivityTypeCode list, which contains a range of codes and the activities they represent, as illustrated in Figure 1.

Code	Value
A	Housing and community amenities
B	Social protection
C	Recreation, culture and religion
D	Defence
E	Environment
F	Economic and financial affairs
G	Production, transport and distribution of gas and heat
H	Health
I	Airport-related activities
J	Exploration and extraction of gas and oil
K	Port-related activities / Maritime or inland waterway

Figure 1: Excerpt of ActivityTypeCode CEN BII code list

1.2 Scope and objectives

This report aims to analyse all relevant aspects concerning the management and governance of a code list, following its life cycle through development, maintenance, distribution, publication, and reuse, as well as quality and communication.

1.3 Target audience

The report considers the perspectives of the publishers of code lists on one hand, and the consumers of code lists on the other hand. The publishers of code lists can be the consumers of their own work. For the purposes of this document, the audience is assumed to consist, for the most part, of public administration representatives and IT professionals developing information systems for public administration.

Publishers are the parties that develop, maintain, distribute and publish expressions of code lists. The publisher role is strictly limited to all activities that have to do with how the code lists are managed and does not include the assignment of the vocabulary terms to instance data – which is the role of the consumers.

Consumers of code lists are parties that publish data that needs to be classified using a constrained set of values, and therefore need to find and apply a code list for the description of their data. In this context, the use of a code list requires integration within IT systems that already exist or that are being developed.

⁵ <https://joinup.ec.europa.eu/release/dcat-ap-how-use-mdr-data-themes-vocabulary>

⁶ MDR NAL: <http://publications.europa.eu/mdr/authority/file-type/>

⁷ EuroVoc: <http://eurovoc.europa.eu/drupal/>

⁸ ADMS: https://joinup.ec.europa.eu/svn/adms/ADMS_v1.00/ADMS_SKOS_v1.00.html

1.4 Structure

The structure of this report follows the lifecycle of a code list, and making clear, at each stage, which role is impacted by the activities to be undertaken at that point.

The remainder of this report is structured as follows:

- Section 2 presents the starting point for an organisation's decision to create or reuse a code list, and provides a summary of the selection criteria that could help an organisation select an appropriate code list;
- Section 3 covers various aspects of the management of code lists from a lifecycle perspective, as well as aspects of quality management and communication;
- Section 4 describes practice for the governance of code lists and considers the skills required for the management of code lists;
- Section 5 summarises the main points of the report.

2 IDENTIFYING THE NEED FOR A CODE LIST AND SELECTING THE RIGHT ONE

The decision to create and manage a code list is triggered by an organisation's need for a set of values to be used in documents, IT systems or content management systems in order to constrain input or to classify and organise content. When an organisation or a group of organisations identifies this need, they will start looking for existing code lists that fulfil this need. Searching for a code list could be as basic as using a few keywords in a search engine such as Google, but potential consumers could find it easier to use specialised sources, such as [Joinup](https://joinup.ec.europa.eu/)⁹ or the [Metadata Registry](http://publications.europa.eu/mdr/index.html)¹⁰ or other internal sources of reusable code lists.

Once the need for a code list has been determined, a potential consumer might look for a suitable code list available for reuse, before going through the process of creating a new code list or requesting another organisation to do so. We propose a set of criteria to be used by the potential consumer to select the most appropriate code list in terms of trust, quality and reusability. These criteria are:

- **Fitness-for-purpose:** Is a given code list suitable for the potential consumer in a particular context? For instance, if a country code list contains the name of a country not recognised by the state in which the code list is going to be used or if the national language is not supported, then the code list cannot be used as such.
- **Clarity and consistency:** Is the code list understandable for both machines, and humans, in multiple languages? Are there clear definitions for the codes? Is there overlapping or contradictory information in the code list?
- **Governance and maintenance:** Is the code list being actively maintained? Does the code list come with quality metadata which provide information provenance, quality, reuse conditions etc. Is there a clear governance structure in place to cater for the maintenance, e.g. for change and release management? Are other organisations using it? Is there user support provided?
- The **trustworthiness** of the publisher. Is the publisher a well-established public or private organisation?
- Availability under an appropriate **licence:** Does the licensing structure allow the potential consumer to make use of the code list? Are there any restrictions to use?

If a suitable code list exists, and there are no licensing or other restrictions, the organisation in need can use it and becomes, therefore, a consumer of that code list. If no such code list exists, the organisation can either find a partner willing and able to create and maintain a code list for them, or do this themselves. If another organisation creates and manages this code list, the organisation with the need becomes a consumer of this code list. If the organisation with the need decides to create its own code list, the organisation will then be both the publisher and consumer of the code list. Figure 2 presents these scenarios in a more visual way.

⁹ Joinup: <https://joinup.ec.europa.eu/>

¹⁰ MDR: <http://publications.europa.eu/mdr/index.html>

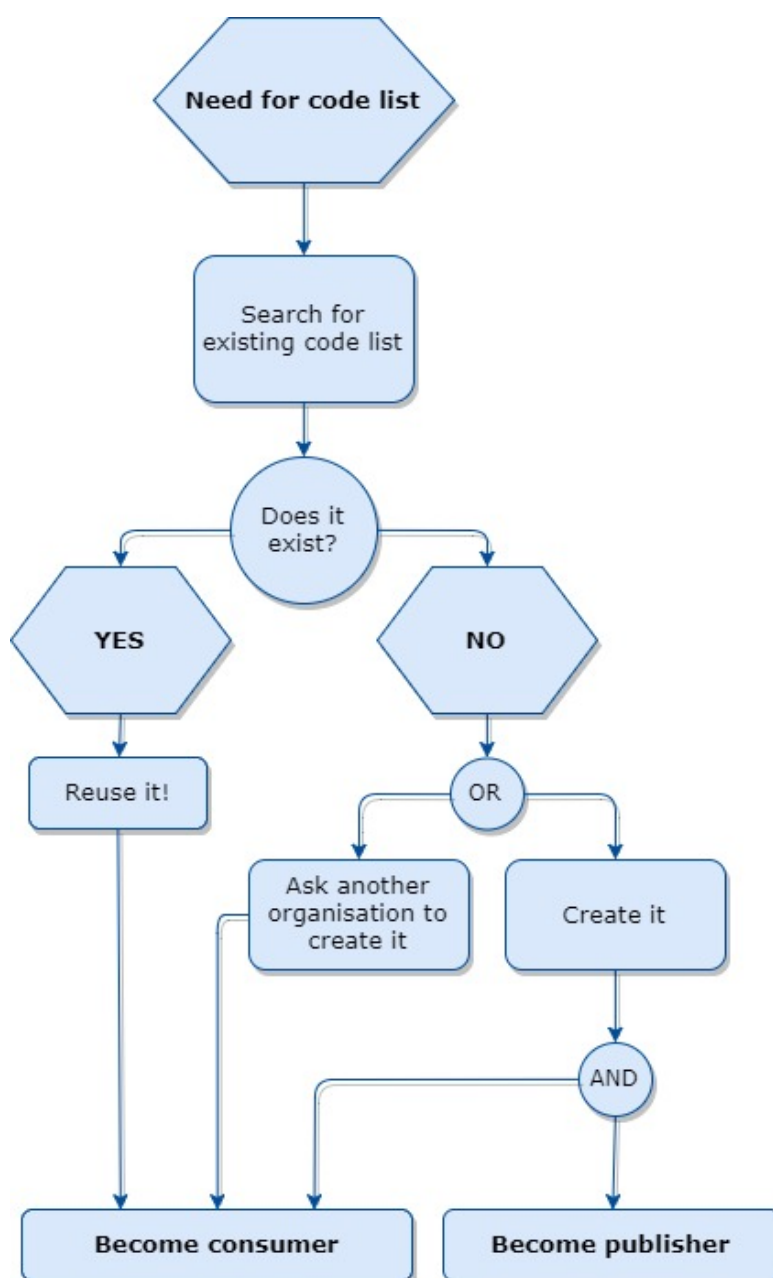


Figure 2: Scenarios for code list consumers

3 MANAGEMENT OF THE CODE LIST LIFECYCLE

3.1 Design

An organisation assumes the role of publisher when it agrees to create and maintain a code list, either for themselves or for another organisation. When the consumer of the code list is another organisation, the consumer and the publisher need to align on what the consumer expects from the code list.

During the design phase, the elaboration of the code list needs to take into account certain aspects, based on design rules such as:

- Having a clearly defined scope, i.e. it should be clear to which characteristic the code list applies. For example, the CEN BII code list [DocumentTypeCode](#)¹¹ contains codes and values strictly related to different types of documents, and nothing else.
- Using terms that are understandable by both the consumers – who will apply the code list in the description of their resources – and the users – who will use the terms in the code list, e.g. as facets in a search application. For instance, the consumers and users of the [Legal Proceeding code list](#)¹² should be able to understand the concepts defined by the codes, as they relate to legal procedures such as “Appeals”.
- Having stable concepts that do not overlap: this is often done by creating separate concepts that differentiate between vaguely related concepts, such as the inclusion of different types of invoice in the aforementioned DocumentTypeCode code list, to avoid overlap.

A more complete list of design principles is provided in the [ANSI/NISO standard Z39.19](#)¹³.

Multilingualism

As part of the design process, the publisher has to determine whether the code list will be used in a multilingual environment. If that is the case, several options may be used: multilingual documentation, multilingual labels in the metadata of the terms, in particular in RDF expressions, or language-independent terms, such as numbers or mnemonics. In XML this is mainly achieved by using [xsd:token](#)¹⁴, for instance as in the [UN/CEFACT Currency code list](#)¹⁵, or [xsd:id](#)¹⁶ restricting values to numbers or mnemonics while keeping labels as attributes.

¹¹ CEN BII Code Lists:

https://overheid.vlaanderen.be/sites/default/files/documenten/overheidsopdrachten/e-procurement/CWA16558-Annex-G-BII-CodeLists-V2_0_4.pdf

¹² Legal Proceeding code list:

<http://publications.europa.eu/mdr/resource/authority/procjur/html/procjur-eng.html>

¹³ ANSI/NISO Z39.19-2005 (R2010) Guidelines for the Construction, Format, and Management of Monolingual Controlled Vocabularies. <http://www.niso.org/publications/ansiniso-z3919-2005-r2010-guidelines-construction-format-and-management-monolingual>

¹⁴ W3C. XML Schema Part 2: Datatypes Second Edition. Section 3.3.2 token. <https://www.w3.org/TR/xmlschema-2/#token>

¹⁵ UN/CEFACT Currency Code List: https://docs.oasis-open.org/ubl/os-UBL-2.0/xsd/common/CodeList_CurrencyCode_ISO_7_04.xsd

¹⁶ W3C. XML Schema Part 2: Datatypes Second Edition. Section 3.3.8 ID <https://www.w3.org/TR/xmlschema-2/#ID>

Figure 3 presents a snippet of an XML file illustrating how the multilingualism issue is addressed in the [MDR NAL code list](#)¹⁷, which involves creating local XML elements for each required language. Using the same example, but in its RDF expression, one can observe the use of `skos:prefLabel`¹⁸ properties being used to add language labels in Figure 4.

```
<data-theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" original.entries.date="2015-10-07" current.entries.date="2016-03-02" table.structure.version="01-01.47-20160204" table.id="data-theme" ref.administration="cat.admin.xml" xsi:noNamespaceSchemaLocation="http://publications.europa.eu/mdr/resource/authority-schema/cat-01-01.47-20160204.xsd" version="20160921-0">
  <record deprecated="false" id="DTH0001" IMMC.proposal.date="2015-10-09" IMMC.approval.date="2015-10-09" date.creation="2015-10-08" adm.status="current">
    <authority-code>AGRI</authority-code>
    <label>
      <lg.version lg="bul" script="Cyrillic">
        Селско стопанство, рибарство, горско стопанство и храни
      </lg.version>
      <lg.version lg="ces">Zemědělství, rybolov, lesnictví a výživa</lg.version>
      <lg.version lg="dan">Landbrug, fiskeri, skovbrug og fødevarer</lg.version>
      <lg.version lg="deu">
        Landwirtschaft, Fischerei, Forstwirtschaft und Nahrungsmittel
      </lg.version>
      <lg.version lg="ell" script="Greek">Γεωργία, αλιεία, δασοκομία και τρόφιμα</lg.version>
      <lg.version lg="eng">Agriculture, fisheries, forestry and food</lg.version>
      <lg.version lg="est">Põllumajandus, kalandus, metsandus ja toiduained</lg.version>
      <lg.version lg="fin">Maatalous, kalastus, metsätalous ja elintarvikkeet</lg.version>
      <lg.version lg="fra">Agriculture, pêche, sylviculture et alimentation</lg.version>
      <lg.version lg="gle">Talmhaíocht, iascach, foraoiseacht agus bia</lg.version>
      <lg.version lg="hrv">Poljoprivreda, ribarstvo, šumarstvo i hrana</lg.version>
      <lg.version lg="hun">Mezőgazdaság, halászat, erdőszet és élelmiszer</lg.version>
    </label>
  </record>
</data-theme>
```

Figure 3: Example of multilingual labelling in XML

```
<skos:Concept rdf:about="http://publications.europa.eu/resource/authority/data-theme/AGRI"
  at:deprecated="false">
  <rdf:type rdf:resource="http://publications.europa.eu/ontology/euvoc#DataTheme"/>
  <dc:identifier>AGRI</dc:identifier>
  <at:authority-code>AGRI</at:authority-code>
  <at:op-code>AGRI</at:op-code>
  <atold:op-code>AGRI</atold:op-code>
  <at:start.use>2015-10-01</at:start.use>
  <skos:topConceptOf rdf:resource="http://publications.europa.eu/resource/authority/data-theme"/>
  <skos:inScheme rdf:resource="http://publications.europa.eu/resource/authority/data-theme"/>
  <owl:versionInfo>20161225-1</owl:versionInfo>
  <dc:dateAccepted rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2015-10-09</dc:dateAccepted>
  <dc:created rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2015-10-01</dc:created>
  <dc:dateSubmitted rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2015-10-09</dc:dateSubmitted>
  <euvoc:startDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">2015-10-01</euvoc:startDate>
  <euvoc:status rdf:resource="http://publications.europa.eu/resource/authority/concept-status/CURRENT"/>
  <!--in template name="specific-part-temporary"-->
  <skos:prefLabel xml:lang="bg">Селско стопанство, рибарство, горско стопанство и храни</skos:prefLabel>
  <skos:xl:prefLabel>
    <skos:prefLabel xml:lang="cs">Zemědělství, rybolov, lesnictví a výživa</skos:prefLabel>
    <skos:xl:prefLabel>
      <skos:prefLabel xml:lang="da">Landbrug, fiskeri, skovbrug og fødevarer</skos:prefLabel>
      <skos:xl:prefLabel>
        <skos:prefLabel xml:lang="de">Landwirtschaft, Fischerei, Forstwirtschaft und Nahrungsmittel</skos:prefLabel>
        <skos:xl:prefLabel>
          <skos:prefLabel xml:lang="el">Γεωργία, αλιεία, δασοκομία και τρόφιμα</skos:prefLabel>
          <skos:xl:prefLabel>
            <skos:prefLabel xml:lang="en">Agriculture, fisheries, forestry and food</skos:prefLabel>
```

Figure 4: Example of multilingual labelling in RDF

Licensing

The publisher will also have to decide under which licence the code list is made available for use. Generally, it is important to specify the licence regime of any code list, because otherwise interested parties may assume that the code list is not available for re-use. The default legal view on code lists that are not clearly licensed is that the potential consumer needs to contact the publisher on a case-by-case

¹⁷ MDR NAL Code List : <http://publications.europa.eu/mdr/resource/authority/data-theme/xml/data-theme.xml>

¹⁸ W3C. Simple Knowledge Organization System (SKOS). <https://www.w3.org/2004/02/skos/>

basis¹⁹. It is a good practice to publish code lists under as open a licence as possible, as interested parties can then build upon it to create added value. A suitable open licence could be the ISA Open Metadata Licence v1.1²⁰.

Relationships

Another aspect to be taken into account is that of relationships between the terms of a code list. While some concepts do not lend themselves easily to the expression of relationships (such as country names in a list of countries), others are perfectly suited for such expressions. For instance, in [EuroVoc](#)²¹, the concept “single-family housing” is the preferred term to “[house](#)”. From the “single-family housing” entry, one can learn about related concepts, either hierarchically or by equivalence. In the example in Figure 5, we see hierarchical relationships (the term is part of microthesaurus 2846 construction and town planning), while “housing” is a broader term for it.

single-family housing

UF *house*
villa

28 SOCIAL QUESTIONS

MT 2846 construction and town planning

BT1 housing

URI <http://eurovoc.europa.eu/1693>

Has Exact Match
single family dwelling (GEMET)

Figure 5: Example of concept relationships

Technologies

As part of the design and development effort, the publisher needs to express the conceptual design in particular syntaxes: in a Linked Data paradigm in RDF, e.g. as SKOS concepts in a SKOS concept scheme, and as [XSD enumerations](#)²², or [OASIS Genericcode](#)²³ (a technique used recently by UBL²⁴, for example) for XML environments. Additionally, a code list can be published as a dataset or as a file, but could also work as a service, in a similar way as linked data. The release should always be accompanied by thorough documentation.

The two expression approaches also have consequences for the access that consumers have to the vocabulary: in a Linked Data environment, vocabulary terms can be used by reference, i.e. as URIs to individual concepts, while in an XML

¹⁹ Data and metadata licensing: https://joinup.ec.europa.eu/sites/default/files/document/2015-05/d2.1.2_training_module_2.5_data_and_metadata_licensing_v1.00_en.pdf

²⁰ ISA Open Metadata Licence v1.1. <https://joinup.ec.europa.eu/licence/isa-open-metadata-licence-v11>

²¹ EuroVoc: <http://eurovoc.europa.eu/drupal/>

²² W3C. XML Schema Part 2: Datatypes Second Edition. Section 4.3.5 enumeration. <https://www.w3.org/TR/xmlschema-2/#rf-enumeration>

²³ OASIS Genericcode: <http://docs.oasis-open.org/codelist/cs-genericcode-1.0/doc/oasis-code-list-representation-genericcode.html>

²⁴ Code lists in XML business documents: <https://www.ibm.com/developerworks/library/x-ind-ublcodel/x-ind-ublcodel-pdf.pdf>

environment, it is not possible to simply use the URI. However, in XML, one can use the codes without importing the full code list. Furthermore, codes can be used in XML independently of the format they are published in. However, if there is an intention to perform schema validation, the code list does need to be imported into the schema (if the code list is expressed as XML schema).

Furthermore, while URIs are a key priority in a Linked Data environment, they carry less weight on the XML side. In XML, identifiers end up not as part of the content itself, but part of the metadata²⁵.

3.2 Manage changes

It is a good practice to provide a strong commitment to sustain a code list by using a transparent change management process. One example is the Publications' Office strong commitment to maintain the Named Authority Lists. Commitment to maintenance also means commitment to the continuous improvement of the code list. In order to measurably improve a code list, certain aspects can serve as performance indicators: number of change requests, number of releases, the time elapsed between the receipt of a change request and the closing of the change management process for said request, the effort required needed to execute the change management operation, etc.²⁶

An explicit change management policy²⁷ needs to state how change requests are captured, how these requests are assessed, who decides which action to take and who implements the changes. This is exemplified in the change management policy of some data specifications, for instance that of [DCAT-AP](#)²⁸. The person or organisation requesting the change should also be informed on any decision following the change request.

Both the publisher and the consumer are responsible for backwards compatibility. The consumer must use the code list in such a way that adaptations to codes do not violate the conformance to definitions, relationships in the code list or anything else. Changing the meaning of a term in a list makes any subsequent version incompatible with previous versions²⁹.

Additionally, there should be some form of agreement between publishers and consumers on whether the publisher should proactively inform the consumers of any new version of the code list, or whether consumers should check for updates following some pre-defined release schedule. In that spirit, pre-defined and fixed release cycles contribute to making new versions predictable and increasing the chances that consumers are aware of the updates in a timely manner. This issue is particularly important, as for some code lists (e.g. country codes) there could be legal implications or liabilities of different sorts in the case of supplying the wrong version or not recognising codes in the most current version.

²⁵ XML attributes: https://www.w3schools.com/xml/xml_attributes.asp

²⁶ Report on implementation of a Metadata Management pilot for DG COMP
https://joinup.ec.europa.eu/sites/default/files/document/2015-09/report_on_implementation_of_a_metadata_management_pilot_for_dg_comp.pdf

²⁷ See "Process and Methodology for Developing Core Vocabularies" for an example of a change management process
<https://joinup.ec.europa.eu/document/process-and-methodology-developing-core-vocabularies>

²⁸ Change management policy of DCAT-AP: <https://joinup.ec.europa.eu/document/change-and-release-management-policy-dcat-ap>

²⁹ Vocabulary Management Note: <https://www.w3.org/wiki/VocabManagementNote>

3.3 Release

The publisher has to choose which distributions will be made available, a decision that depends on the potential consumer-base of the code list. As mentioned in section 3.1, code lists can be distributed in different forms, as a file, as a dataset, or as a service. In order to facilitate dissemination, the code list should be hosted on a stable platform, such as an official website that establishes the publisher as the authoritative source, and have a persistent identifier.

Publication of a new release should come with some form of statement of authenticity. This could be as easy as having an official website with the distribution stating that this is the only valid source, as mentioned above. This practice enables the establishment of the authority of the owner of the code list.

When a new version of a code list is released, stakeholders should also receive associated documentation. For each release, a versioning system with numbers is strongly recommended to allow users to easily determine whether they use the latest version of the code list. It also contributes to a well-designed and well-maintained code list.

Versioning in XML

When implementing changes to XML schemas, the version number of the schema should increase, and any updates should be reflected in a change log. The version number can be implemented at several levels in an XML code list:

- In the file name;
- In the namespace;
- Through the schema version attribute (<xsd:schema... version="1.0">);
- Changing the name or location of the schema;
- Adding a schema-version attribute to the root element³⁰.

```
<?xml version="1.0" encoding="UTF-8"?>
<data-theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  original.entries.date="2015-10-07"
  current.entries.date="2016-03-02"
  table.structure.version="01-01.47-20160204"
  table.id="data-theme"
  ref.administration="cat.admin.xml"
  xsi:noNamespaceSchemaLocation="http://publications.europa.eu/mdr/resource/authority-schema/cat-01-01.47-20160204.xsd"
  version="20160921-0">
```

Figure 6: Example of versioning in XML

Figure 6 shows an example of versioning being done in the schema version attribute, using both a number and a date for the versioning.

Versioning in RDF

In contrast to XML code lists, where there are different versions of the schemas themselves, for RDF, we simply see different versions of the codes. However, this only occurs as a way to keep track of minor changes, which do not change a code's definition and semantics. For significant changes in the semantics of a code, a new one needs to be created. For example, in Dublin Core, we see that when the concept "license" suffered a relatively minor change (deleting an additional comment), this is reflected in the creation of a new code:

³⁰ Versioning in XML: <http://www.xfront.com/Versioning.pdf>
30/05/2018

<http://dublincore.org/usage/terms/history/#license-002>, which replaces <http://dublincore.org/usage/terms/history/#license-001>.

Formats

Code lists can be published in many different formats, including spreadsheets or text documents. Commonly used formats are of course XML and RDF, but also CSV and HTML. CSV and HTML are particularly user-friendly formats, as they allow the consumer to open the code list in a simple spreadsheet software (for CSV) or any Web browser (for HTML).

A commonly encountered format in this domain is also SKOS (Simple Knowledge Organisation System), a W3C standard specifically designed for representation of structured controlled vocabularies such as code lists³¹. SKOS provides a standard way to represent knowledge organisation systems using RDF. As such, the RDF distribution of a code list would be modelled in SKOS. Using unique URIs as identifiers for concepts, SKOS enables the expression of a relationship between a concept and another SKOS-based concept using the URI of this second concept³². One of the main advantages of SKOS is that it allows publishers to store additional metadata. For even more granular metadata, SKOS-XL is an extension to SKOS designed for assigning metadata to a label³³. SKOS-XL therefore facilitates versioning and multilingualism, for which it is used by organisations such as the [Publications Office of the EU](#)³⁴.

Publishing tools and platforms

Code list publishers have a variety of tools at their disposal, both Open Source and commercial products. There are tools on the market that cover the full lifecycle of a code list, from design and release to change management and retirement. For instance, [Unilexicon](#)³⁵ is an Open Source solution that can be used to create code lists. It has a very straightforward registration process and features an easy-to-use visual editor. A solution that might be used simply to publish a code list is [Skosmos](#)³⁶. For publishers that are looking for collaborative tools, [VocBench](#)³⁷ might be the most appropriate option.

Licensing

Section 3.1 contains the basic information regarding the licensing approach a code list publisher might take. It is also good to know that some code list publishing tools include the possibility to provide licensing information about the data in a code list. This can encourage the broad dissemination and reuse of a given code list.

³¹ SKOS Reference: <https://www.w3.org/TR/2009/REC-skos-reference-20090818/>

³² Improve your taxonomy management using the W3C SKOS standard: <https://www.ibm.com/developerworks/library/x-skostaxonomy/index.html>

³³ SKOS-XL Reference: <https://www.w3.org/TR/skos-reference/skos-xl.html>

³⁴ EU Publications: <https://publications.europa.eu/en/home>

³⁵ Unilexicon. Taxonomy editor and tagging suite. <https://unilexicon.com/>

³⁶ Skosmos. Open source web-based SKOS browser and publishing tool. <http://skosmos.org/>

³⁷ VocBench, a web-based, multilingual, collaborative development platform for managing OWL ontologies, SKOS (XL) thesauri and generic RDF datasets. <http://vocbench.uniroma2.it/>

Getting new versions to consumers

Code list publishers need some way to communicate to consumers about new versions of a code list. When the publisher and the consumer are not in the same organisation or group of organisations, a generic solution is necessary to facilitate the timely transmission of updates of the code list to consumers. New versions should generally come with updates in documentation and in the change log of the code list. Maintaining communication to consumers may include pushing the new versions to them and allowing them to access both the new version and the updates in documentation. This can be done either via a mailing list or online community, such as Joinup or open source software development platforms like [GitHub](#)³⁸ or [Sourceforge](#)³⁹.

3.4 Retire

There are situations where elements of a code list or even complete code lists need to be retired. This may be the result of changes in the real world, such as reorganisation of regions in a country in which a region ceases to exist, for example when it is merged into a large region. Another reason may be that a particular code or code list is no longer used for descriptions of the resources.

Retirement should be handled with extreme care in order not to invalidate existing data, and should only be done if it can be established that the codes or the code list are not used anywhere, or that all consumers of the codes or list can reclassify the existing data before the retirement date of the code or list. This means a code will almost never be removed, just marked as retired, in order to ensure that data does not become invalid. Another possibility, used in more complex cases, would be to assign temporal attributes to a term that would indicate from when and until when a term is valid. Depending on the features of the tool being used to manage a code list, the publisher could even indicate what becomes of a retired code, where it is reintegrated or which new codes it is split into.

For instance, the [MDR NAL](#) country code list includes “Czechoslovakia” as a term, although this country does not currently exist. The term’s metadata includes the fact that it was only in use for a given period of time, and it is listed as a predecessor of “Slovakia”. However, as the term may still need to be used (for instance, as a person’s country of birth), it cannot be retired.

Retired codes should never be reused with a different meaning. An example where this rule was not respected was the re-assignment of country code “cs” from Czechoslovakia to Serbia and Montenegro in ISO 3166⁴⁰.

³⁸ GitHub. <https://github.com/>

³⁹ Sourceforge. <https://sourceforge.net/>

⁴⁰ International Organization for Standardization (ISO). Country Codes - ISO 3166. <https://www.iso.org/iso-3166-country-codes.html>. See also: https://en.wikipedia.org/wiki/ISO_3166-2:CS

Retiring elements in XML

In XML, there is no formal way to retire elements. One possibility is to add it as [xsd:documentation](#)⁴¹, in order to inform consumers. Another way would be to use a status attribute. The second way is illustrated by Figure 7.

```
<record deprecated="true">
  id = "DTH0006"
  IMMC.proposal.date = "2015-10-09"
  IMMC.approval.date = "2015-10-09"
  date.creation = "2015-10-01"
  adm.status = "current">
```

Figure 7: Example of deprecation in XML

Retiring elements in RDF

In RDF, elements can be retired by adding a status property on a term, in a manner equivalent to how this is done in XML. Figure 8 shows an example of a deprecated element.

```
<skos:Concept rdf:about
=http://publications.europa.eu/resource/authority/data-theme/TRANS
at:deprecated= "true">
```

Figure 8: Example of deprecation in RDF

3.5 Use and extend

Integration is the process of combining data from different sources and providing a reconciled view on it. The consumer team in charge of IT systems maintenance captures and manages the requirements for integration of code lists in existing IT systems. This team can use an existing code list as it is or opt to extend it if necessary.

- In XML, the integration is done using the [xsd:import](#)⁴² tag and relying on a copy of the code list. It is also theoretically possible to use a remote schema location with the `<import>` tag. However, this can only work when the consumer can reach this location, which assumes a network connection and may not work even then if the connection is through a proxy). Using remote schema locations may be done using XML catalogues⁴³.
- Using a copy of the code list file is also possible in RDF, but this is not recommended practice unless it is done through caching. In any case, reference to an individual code should use the URI of that code.

If consumers identify that some concepts are missing from a code list, they may want to create a change request asking for the addition of those concepts. They can also create change requests to report errors, to provide translations, and more. This is easier than starting from scratch, but extensions should be documented as separate

⁴¹ W3Schools. XML Schema documentation Element.

https://www.w3schools.com/xml/el_documentation.asp

⁴² W3Schools. XML Schema import Element. https://www.w3schools.com/xml/el_import.asp

⁴³ XML catalogues are documents describing a mapping between external entity references and locally cached equivalents.

vocabularies to ensure that those not interested by the extension can determine which version would be most appropriate for their use.

When a consumer uses a remote code list, control is entirely with the code list publisher. Code list consumers might prefer exerting some degree of control over the version they use and could therefore find it preferable to use a local copy and replace it (or not) with a newer version at a convenient time.

For XML, it is not uncommon for XML schemas not to import XSD code lists into the schema, but to leave either the entire code list or its version unspecified. The instance document can then mention a specific code list and/or code list version. In this case, validation in some other manner than schema validation would be required.

Extending an XML code list

In the XML realm, the options for extending a code lists are limited, since extensibility is not part of the specification as such. While some believe lists should not be extended, this is not realistic in practice. Backwards compatibility is also an issue, along with the fact that after publishing, the publisher cannot control what consumers do with the code list, including for instance using an older version. However, there are some possible approaches to extending XML code lists:

- Editing the original schema to add new values: this is an easy solution, but it requires editing the original schemas, which presumes control over the schemas. This is in effect creating a new version of the code list, which might be dedicated to exclusively internal use.
- Creating a new list and joining it to the original using the [xsd:union](#)⁴⁴ tag allows the original list to remain unchanged, but all values must be known at design time. Additionally, the xsd:union tag is not always supported by tools.
- Creating a pattern and combining it with the original enumerated type: this enables the use of the same element for all data, and validation is done by the parser, but the content of the element must be parsed to determine whether it's extended, and the xsd:union tag must be supported. The example below illustrates how this method can be used to extend a list involving shoe sizes available for a given model. Figure 9 shows the initial list, while Figure 10 presents a pattern for the new values. The pattern essentially calls for any new string. The "x:" is a delineator between the initial elements and the extensions. The last step is using xsd:union to combine the initial list and the newly created pattern, as illustrated in Figure 11. The impact of the extension can be observed in Figure 12, which shows the new values can be validated by the parser.

⁴⁴ W3Schools. XML Schema union Element. https://www.w3schools.com/xml/el_union.asp
30/05/2018

```
<xsd:simpleType name="ShoeSizeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="36"/>
    <xsd:enumeration value="37"/>
    <xsd:enumeration value="38"/>
    <xsd:enumeration value="39"/>
    <xsd:enumeration value="40"/>
    <xsd:enumeration value="41"/>
    <xsd:enumeration value="42"/>
    <xsd:enumeration value="43"/>
    <xsd:enumeration value="44"/>
    <xsd:enumeration value="45"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 9: Example initial list

```
<xsd:simpleType name="StringPatternType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="x:\S.*"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 10: Example pattern for new values

```
<xsd:simpleType name="NewShoeSizeType">
  <xsd:union memberTypes="ShoeSizeType StringPatternType"/>
</xsd:simpleType>
<xsd:element name="ShoeSize" type="NewShoeSizeType"/>
```

Figure 11: Example combination of initial list and extension pattern

```
<ShoeSize>Black</ShoeSize>
<ShoeSize>x:35</ShoeSize>
```

Figure 12: Example of XML instances after extension

- Using a separate field for extensions refers to including an extension field in the schema in order to accommodate additional values. This forgoes changing the original schema, and allows changes after design time, as well as being in line with the XML schema specifications. However, the values do need to occur in elements rather than attributes. The example below illustrates how this method can be used to extend a list involving shoe sizes available for a given model. Figure 13 shows the initial list, where an extension is already being prepared as an extension field. Figure 14 presents the creation of an additional attribute named "extension" to accommodate the new values. Figure 15 presents some XML instances after the extension.

```
<xsd:simpleType name="ShoeSizeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="36"/>
    <xsd:enumeration value="37"/>
    <xsd:enumeration value="38"/>
    <xsd:enumeration value="39"/>
    <xsd:enumeration value="40"/>
    <xsd:enumeration value="41"/>
    <xsd:enumeration value="42"/>
    <xsd:enumeration value="43"/>
    <xsd:enumeration value="44"/>
    <xsd:enumeration value="45"/>
    <xsd:enumeration value="Extension"/>
  </xsd:restriction>
</xsd:simpleType>
```

Figure 13: Example of extension field in the schema for accommodating new values

```
<xsd:complexType name="NewShoeSizeType">
  <xsd:simpleContent>
    <xsd:extension base="ShoeSizeType">
      <xsd:attribute name="extension" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="ShoeSize" type="ShoeSizeType"/>
```

Figure 14: Example of additional attribute

```
<ShoeSize>40</ShoeSize>
<ShoeSize extension="35">Extension</ShoeSize>
```

Figure 15: Example XML instances

- When there is no requirement to validate the schema in one pass, a code list could also be extended using a documentation-based approach. This can involve either the use of the `xsd:union` tag in combination with a string or the use of [xsd:annotation](https://www.w3schools.com/xml/el_annotation.asp)⁴⁵, wherein additional values are added in the `xsd:documentation` tag⁴⁶. Figure 16 presents an example of this method being used to extend a list involving shoe sizes available for a given model.

⁴⁵ W3Schools. XML Schema annotation Element. https://www.w3schools.com/xml/el_annotation.asp

⁴⁶ Extend enumerated lists in XML schema: <https://www.ibm.com/developerworks/library/x-extenum/>

```

<xsd:simpleType name="ShoeSizeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="36"/>
    <xsd:enumeration value="37"/>
    <xsd:enumeration value="38"/>
    <xsd:enumeration value="39"/>
    <xsd:enumeration value="40"/>
    <xsd:enumeration value="41"/>
    <xsd:enumeration value="42"/>
    <xsd:enumeration value="43"/>
    <xsd:enumeration value="44"/>
    <xsd:enumeration value="45"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:element name="ShoeSize" type="ShoeSizeType"/>

<xsd:simpleType name="ExtShoeSizeType">
  <xsd:union memberTypes="ShoeSizeType xsd:string"/>
</xsd:simpleType>
<xsd:element name="ShoeSize_docbased" type="ExtShoeSizeType"/>

```

Figure 16: Example of documentation-based extension

Extending an RDF code list

A consumer extending an RDF code list is technically not extending the list but creating a new one based on the original. If a concept is considered important enough, one may ask the manager of the original code list to consider adding the term to the code list using the normal change request channels.

3.6 Create mappings

As a consumer, an organisation may be interested in reclassifying some concepts when reusing an existing code list, meaning they would effectively become publishers in their own right by making changes to a code list and thus creating a new version. In other cases, mappings between code lists may need to be created when data from different sources are being integrated or when systems are being connected. Declaring relationships between codes in one code list to codes in another code list may come up in the design phase, when first creating the code list or at a later stage. If data resources managed by the user already have been classified using an existing code list and there is a need to replace this classification by a new one, the user can re-classify the resources by mapping the existing list to the new one. For example, national data portals may map a local dataset theme vocabulary to the MDR data themes NAL. This scenario also applies to data exchange, e.g. when a system collects new data by showing the user a drop-down list based on a code list to select the value for a certain field.

Mapping itself can be done by defining relations between entities of two code lists. Relations and links can be based on the Simple Knowledge Organization System (SKOS)⁴⁷, which is a common data model for sharing a linking knowledge organisation system. The following mappings exist in SKOS: exact match, close match, broad match, narrow match, related match. SKOS can be used both in RDF and XML. For instance, it is used by MDR NALs in both forms.

⁴⁷ W3C. Simple Knowledge Organization System (SKOS). <https://www.w3.org/2004/02/skos/>
30/05/2018

The code list may need to be mapped with other vocabularies to facilitate cross-referencing of terms in different vocabularies, for example creating links from the MDR Language NAL⁴⁸ to the Library of Congress Codes for the Representation of Names of Languages Part 2 (ISO639-2)⁴⁹ vocabulary. The creation of mappings between controlled vocabularies can be a prerequisite to implementation of data transformation routines. Additionally, mapping is an additive process, which can add semantics that are otherwise missing from a vocabulary⁵⁰. To make use of the mappings, they should be published⁵¹ in human and machine-readable formats. SKOS mapping specifications⁵² contain further details on how to map concepts from different schemes.

In XML, the <xsd:id> and <xsd:idref> tags allow cross-referencing. For example, in the MDR NAL [currency code list](#) references [the XML schema](#).

3.7 Manage code list quality

Code list quality management subscribes to certain quality requirements common to reference data:

- The codes should suit the use case of the code list;
- The code list should be maintained and reused;
- The definitions of the terms should be clear to both humans and machines.

Before starting, the quality requirements have to be explicitly stated. In addition, a way to measure the quality level should be determined. The quality of a code list is for a large part determined by the need for such a code list. Certain quality aspects should already be considered in the design phase, as mentioned in section 3.1. Once development has started, the quality should be continuously evaluated against the pre-defined requirements. For instance, publishers could ensure that the most important design principles are implemented by allowing review by code list consumers or individuals not involved in the development of the code list. This would cover aspects such as having a clearly defined scope and having understandable terms as well stable concepts.

For more low-level aspects of code list quality management, publishers can rely on the features of certain code list management tools. Some software provides features like data traceability, configurable quality policies, allowing (or disallowing) duplicate terms, auditing terms, and consistency control.

3.8 Communication

Communication is a cross-cutting horizontal process that covers the code list lifecycle throughout its entirety. Therefore, a communication plan needs to cover the way all parties communicate. Different consumer groups will have different communication needs. It is up to the publisher to find suitable ways to communicate with all stakeholders. This involves determining how to engage stakeholders, how to

⁴⁸ MDR NAL: <http://publications.europa.eu/mdr/authority/language/>

⁴⁹ ISO 639.2: <http://www.loc.gov/standards/iso639-2/langhome.html>

⁵⁰ Strategies for Vocabulary Design and Development?
<https://ecommons.cornell.edu/bitstream/handle/1813/42443/82-89-Paper.pdf?sequence=3&isAllowed=y>

⁵¹ Core Data Model Mapping Directory: <http://mapping.semic.eu>

⁵² SKOS mapping specifications: <https://www.w3.org/2004/02/skos/mapping/spec/2004-11-11.html>

announce updates and changes to the code list, choosing the communication channels that are suited for each group of consumers and organising events.

Depending on the main purpose of the code list, the publishers might take different approaches to communication entirely, depending on whether the code list has been created on request, or it exists as a general-purpose code list where the aim is to raise awareness among potential consumers.

- For a code list that has been created on the request of some organisation or group of organisations, changes to the code list might also be initiated by the same organisation. As such, communication will involve contributions from both sides, including how the requesting organisation chooses to govern its requirements. In this case, for the publisher, communication will be almost entirely concerning updates and changes to the code list.
- For a general-purpose code list, the publisher aims to ensure that as many potential users as possible are aware of the code list and able to access its updates. Here, communication might take more externally-focused forms, such as blog posts, news items, social media activity, etc. Most users of a general-purpose code list will not request changes, and if they do, it is up to the publisher to choose when/if to implement them.

Documentation is one of the main means of communication between publishers and other stakeholders. Aspects including licensing, change management, versioning strategy, policy elements, etc. can be made clear through the use of thorough documentation.

- A clear licensing policy informs decisions by potential consumers regarding the possibility of re-using a code list.
- Communications regarding the change management approach of a given code list help consumers interested in maintaining their content over time automatically.
- For publishers, having a versioning strategy provides several benefits, particularly in terms of facilitating updates to the code list. Appropriately-implemented versioning also makes vocabularies more attractive for potential consumers.
- Another important aspect of communication has to do with the policy environment. Publishers can use documentation to justify investing in the creation and maintenance of code lists. Additionally, potential consumers need background information on the publisher to be able to make a sound decision regarding re-using an existing code list.

Good communication also involves promoting code list releases, managing the culture around the code list and providing training to potential users. Publishers should take advantage of the opportunity to communicate to consumers by informing them that there is a management team in place. Such a management team can provide support for not just using a code list, but also in terms of change management and other aspects.

4 CODE LIST GOVERNANCE

4.1 Context and the need for governance

Governance is permanent and active as long as a code list is still used or needed. In the context of data specifications, governance represents “the set of roles and responsibilities, cohesive policies and principles, and decision-making processes that define, govern, and regulate the lifecycle of data specifications”⁵³.

The governance of a code list consists of determining the different actors and their roles in its development and management. This implies setting up a decision structure, delegation and escalation paths and clearly defining all roles in the team that creates and maintains the code list. Some good practices of code list governance to take into account are:

- Involving direct stakeholders (e.g. key consumers) in the governance process;
- Using interoperable tools based on open standards for supporting both the governance and the management of code lists.

In special cases, the publisher is the same as the consumer of a code list. It is recommended to separate the governance of code lists from the governance of the IT systems in which the code lists will be integrated, but to coordinate the two in case of reuse. There are at least two main reasons for this:

- Code lists have their own life cycle as standalone, reusable components.
- The requirements for code lists should be as generic as possible (decoupled from any specific needs of IT systems) to ensure a higher degree of reusability⁵⁴.

There may be significant differences in the governance of code lists depending on how often they are likely to change. For instance, a list of public services is likely to change with relative frequency, while country codes only change exceptionally. This could impose specific decisions regarding the governance structure, the decision-making mechanisms, and frequency of planned updates.

Another point to be considered regarding code list governance concerns the provenance of the codes. Some lists are simply facts from the real world and enshrined in law, such as lists of hazardous substances, country names, or marital statuses recognised in a jurisdiction. In such cases, the publisher may not be the authoritative source but will follow the lead and act on behalf of the authoritative source being the legislation.

4.2 Governance tasks and structure

In terms of code list governance, as for any other data specification, certain tasks need to be fulfilled, as shown in Figure 17.

⁵³ Metadata Governance and Management: https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/methodology_and_tools_for_metadata_governance_and_management_for_eu_institutions.pdf

⁵⁴ In practice, this is not always possible. Some code lists are destined for use in a specific community, for very specific needs, which would essentially force the consumers in that community to adapt the release cycles of their applications to the one of the code list.

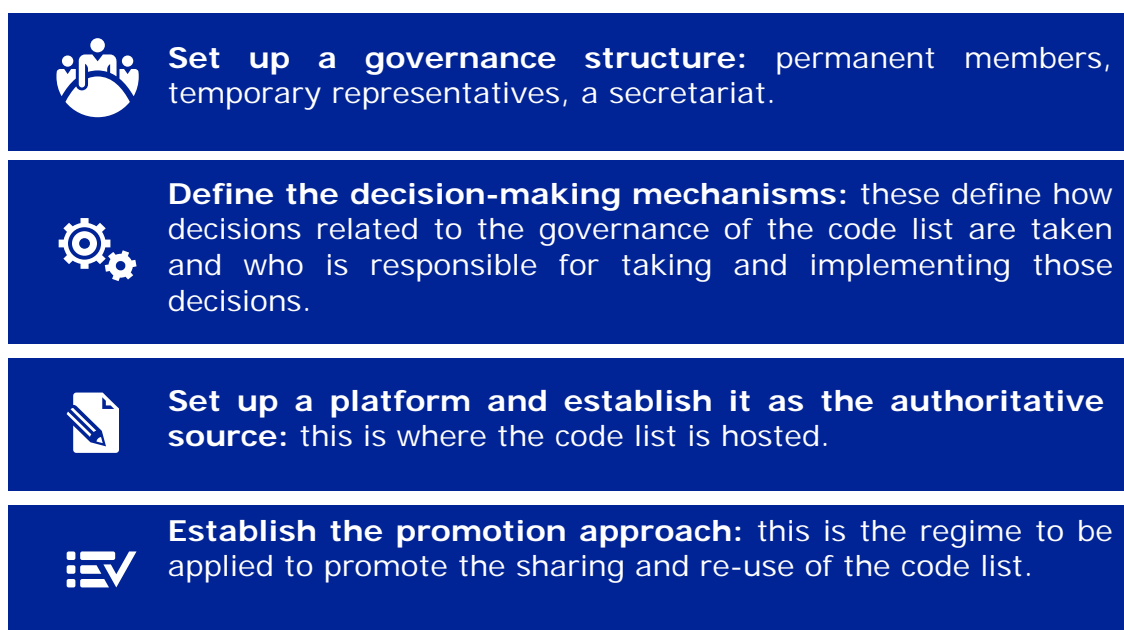


Figure 17: Tasks related to code list governance

Figure 18 takes the tasks related to code list governance described in Figure 17 and provides examples of how these tasks may be carried out.

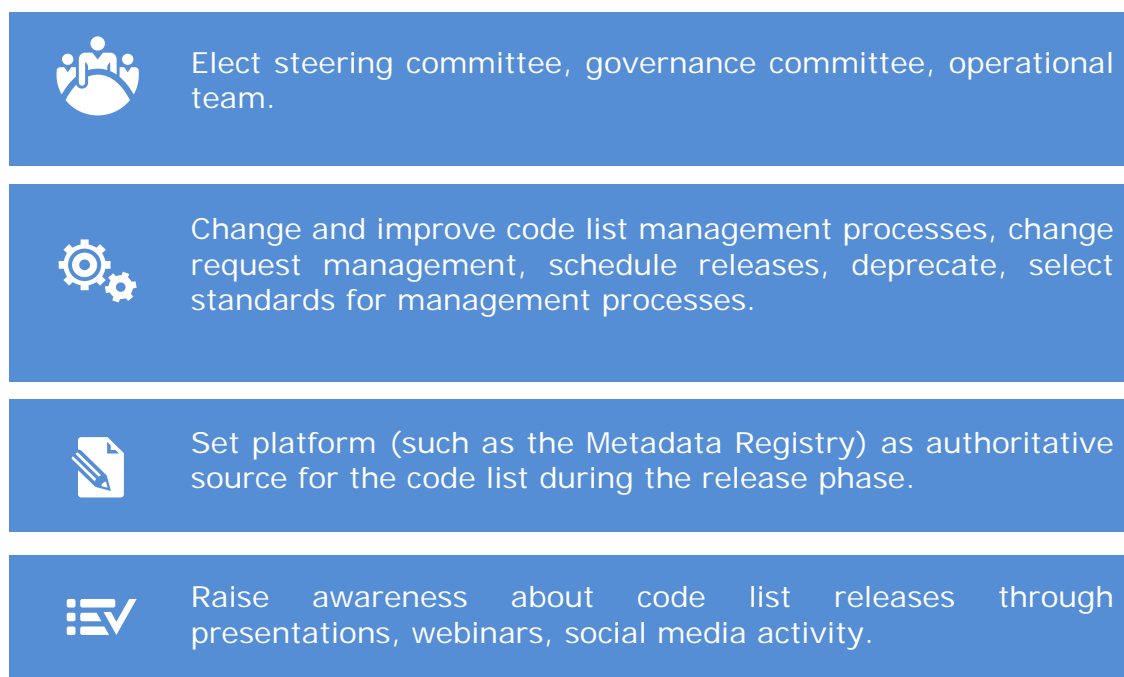


Figure 18: Examples of code list governance tasks

It must be clear to the consumers of a code list who the owner or responsible organisation or individual is. Contact data should be available⁵⁵, ideally as part of the

⁵⁵ Issues in Vocabulary Management:
http://groups.niso.org/apps/group_public/download.php/18054/TR-06-201x_Issues_in_Vocabulary_Management.pdf

metadata of the code list. Metadata, along with documentation, support the usability of a code list.

4.3 Skills and expertise required for the governance and management of code lists

The governance and management of a code list require people appointed in different roles across the structures, from the steering committee, through the governance committee and the operational team.

The steering committee generally consists of representatives of the publishing organisation and has the purpose of setting the strategic direction for the code list. The members of this structure should come from both the business side and the technical side, and they should be able to take decisions on scope and goals. They review the progress and solve conflicts, as well as appointing members of the governance committee and the operational team.

The governance committee counts among its members the main stakeholders of a code list or a group of code lists used in the same context or by the same group of consumers. This committee will usually also be responsible for the governance of other, related data models. They take decisions regarding operational support that the operational team may need, and oversee the latter's compliance, as well as developing, disseminating and enforcing the required procedures.

The operational team handles the day-to-day aspects of the work, managing the design of the code list, its development and support around it. The operational team performs tasks related to the management of the code list rather than governance. Table 1 contains an overview of the correspondence between activity area and required skills.

Table 1: Skills required and corresponding area of activity

Skill	Activity
Domain expertise	Governance; management
Information management	Management
Technical expertise (knowledge about technical approaches, formats, etc.), documentation and publication	Management
Release management	Governance; management
Standardisation expertise	Governance; management

5 SUMMARY

This section summarises the main takeaways and good practices related to code list management and governance. These takeaways are meant to structure the way stakeholders approach code lists throughout their lifecycle. Figure 19 presents the report's conclusions as a list of tips for code list management and governance.

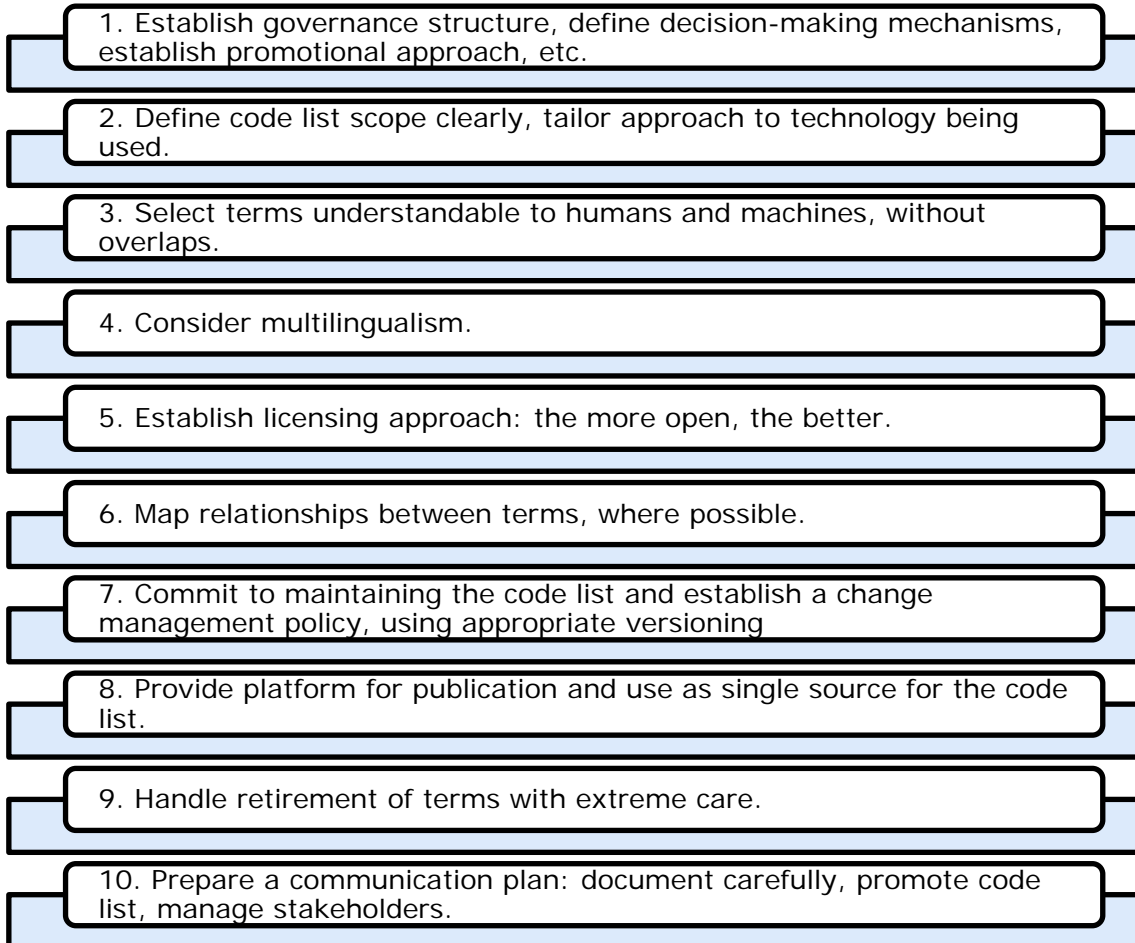
- 
1. Establish governance structure, define decision-making mechanisms, establish promotional approach, etc.
 2. Define code list scope clearly, tailor approach to technology being used.
 3. Select terms understandable to humans and machines, without overlaps.
 4. Consider multilingualism.
 5. Establish licensing approach: the more open, the better.
 6. Map relationships between terms, where possible.
 7. Commit to maintaining the code list and establish a change management policy, using appropriate versioning
 8. Provide platform for publication and use as single source for the code list.
 9. Handle retirement of terms with extreme care.
 10. Prepare a communication plan: document carefully, promote code list, manage stakeholders.

Figure 19: 10 tips for code list management and governance