Draft
CEFACT/11SEP/2017

UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE

UNITED NATIONS CENTRE FOR TRADE FACILITATION
AND ELECTRONIC BUSINESS (UN/CEFACT)

Code Management User Guide

Approved: UN/CEFACT Bureau ____________________

Version: 0.2
Release: 3

TABLE OF CONTENTS

Contents

DRAFT

DRAFT

1 About this document

This user guideline describes how to define and apply restrictions and extensions to code lists in UN/EDIFACT messages as well as UN/CEFACT XML messages. In addition, it describes example processes for validating those messages. The process could be done in one-phase, for which message structure and value constraints are validated simultaneously (so-called 'coupled') or in two-phases, for which these constraints are validated separately (so-called 'decoupled').

Parts in this document are excerpts from the XML Naming and Design Rules (UN/CEFACT XML NDR Rules 2.2), UN/EDIFACT Syntax Implementation Guidelines and OASIS Genericode/CVA. They give guidance on how to apply these rules in a real-life environment. The latest version of the UN/CEFACT XML NDR Rules, version 2.2, allows decoupling of selective or all qualified data types from a set of value enumerations.

1.1 Executive summary

Codes are an essential component of any Machine-To-Machine information flow. Codes have been developed over time to facilitate the flow of compressed, standardized values that can be easily validated for correctness to ensure consistent semantics. In a real-life environment, there exist external circumstances (business needs, laws) that require the extending or restricting (sub-setting) of standardized code lists in UN/EDIFACT or UN/CEFACT XML messages. Many international, national and sectoral agencies create and maintain code lists relevant to their area. If required to be used within an information flow, these code lists will be stored in their own environment and referred to as external code lists. Although the standardization procedures define how extensions can be realized by starting a Data Maintenance Request (DMR) there may be time constraints that solutions need to be found for the time until the final update of the standardized code lists are published.

The UN/CEFACT Code Management project defines the procedures, rules and methodologies for the following identified issues.

1. Version compatibility
The ability to use any version of a code list in association with any version of a message, i.e. decoupling the versioning of code lists from the business message versions.

2. Extending code lists
Evaluate if permanent extensions are possible and desirable.

3. Restricting code lists
Provide rules and methodology for restricting code lists for use within specific context. Users of the UN/CEFACT libraries may identify any sub-set they wish from a specific code list for their own community requirements.

4. Code list validation rules
Provide rules and methodology for how to validate instance documents against an XML Schema or UN/EDIFACT message type in respect to code lists.

5. Temporary codes
Provide rules and methodology for the inclusion of temporary codes that will be replaced by a permanent code at the next UN/CEFACT standardized release, in essence a temporary extension.

DRAFT

6. Externally maintained code lists
Define rules and procedures for referencing code lists maintained by organizations external to UN/CEFACT, e.g. ISO, ICC, W3C, UNECE.

7. Publication format for code lists
A standard exchange format for code lists.

1.2 Status of this document

This document has been developed in accordance with the UN/CEFACT/TRADE/22 Open Development Process for Guidelines and approved for publication by the UN/CEFACT Bureau.

1.3 Revision history

Version	Release	Date	Comment
0.1.1	Internal draft from SCRDM Project Team	2016-07-25	
0.2.1	Adjusted by the Code Management Project Team	2017-08-08	
0.2.2	Adjusted by the Code Management Project Team	2017-08-31	
0.2.3	Adjusted by the Code Management Project Team	2017-09-11	

DRAFT

2 Project Team

2.1 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this technical specification.

2.2 Project Team Participants

Project Team Lead:

Rolf Wessel

Lead editor:

Gerhard Heemskerk

Editing Team:

Akio Suzuki
Andreas Pelekies
Eric Cohen
Tayfun Mermer
Mary Kay Blantz
Sue Probert
Niki Dieckmann
Frans van Diepen
Gait Boxman
G. Ken Holman
Lance Thompson
Jörg Walther

DRAFT

3 Introduction

The main audiences for this document are primarily.

- Corporate Chief Technology Officers - Government
- Corporate Chief Technology Officers – Private Sector
- UN/CEFACT Bureau and Vice Chair persons

3.1 Structure of this document

- Chapter: 4 User Requirements
- Chapter: 5 Using code lists in a real-life environment
- Chapter: 6 Validating UN/EDIFACT document instances
- Chapter: 7 Validating UN/CEFACT XML document instances
- Chapter: 8 Publication Format Code Lists
- Chapter: 9 Definition of terms

3.2 Related Documents

- UN/EDIFACT Directory, Part 4 United Nations Rules for Electronic Data Interchange for Administration, Commerce and Transport, Chapter 2.3 - UN/EDIFACT Syntax Implementation Guidelines.
- UN/CEFACT XML Naming and Design Rules for CCTS 2.01 Version 2.2 dated MM/DD/2017.
- ISO 20625 EDIFACT - Rules for generation of XML scheme files (XSD) on the basis of EDI(FACT) implementation guidelines
- Schematron ISO/IEC 19757-3
- OASIS Context/value association using Genericode 1.0
- OASIS Genericode 1.0

3.3 Purpose and scope

The business goals of this document are:

- To summarize the steps for creating and/or using extended, restricted, user-defined (permanent or temporary) code lists and code lists published by other organizations in a real-life environment.
- To give guidance for validating electronic documents (electronic business messages) where the steps above are applied.

DRAFT

4 User requirements

The essence of all user requirements is flexibility to handle external circumstances (urgent business needs, laws) that require the extension, restriction of standardized code lists and/or user-defined code values (permanent or temporary).

The requirements gathering phase of the Code Management Project has provided below list:

- Using own code lists
- Referring to the code list version actually used
- Extending code lists (extension)
- Restricting code lists (restriction)
- Combining code lists (union)
- Choosing code lists (choice)
- Allowing temporary codes
- Validating code constraints of above requirements
- Using internationally harmonized code lists (UN/CEFACT and others)
- Maintaining code lists in an easy manner
- Obtaining code lists from a standardized publication format

4.1 The challenge of Interoperability

Interoperability is looking at how disparate systems understand each other. In this respect, it is about receiving code values and behaving as expected. Code values take an important role in the exchange of transaction data between trading partners. For example, in the case of a Purchase Order, the receiving system understands the message so that it is now able to read the Order and start or continue the process at this stage in the Supply Chain.

The challenge is that most implementations are separate and different and no one major player is able to force alignment globally. Typically, misinterpretations occur both before and after implementations. User-defined code values are often misinterpreted because the use is not documented properly and therefore systems cannot process these values. The other challenge is that not everyone needs to implement all standardized code lists and/or code values specified in the standard as it may not be applicable to them.

4.2 The challenge of Conformance

Conformance is measuring how a document instance makes use of a given standard or specification. Compliant means that some features in the standard specification are not implemented, but all features implemented are covered by the specification, and in accordance with it.

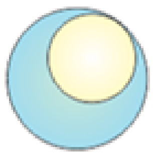


Figure 1: Compliant

4.2.1 Conformance and UN/EDIFACT

In the case of UN/EDIFACT messages there is no technical link between the published message structure and codes used by it. The message structure and codes values used by a community are specified or referenced within the community Message Implementation Guide (MIG). In practice user communities often want to be compliant with a published

DRAFT

United Nations Standard Message (UNSM) whilst referring to any version of code lists, restricted, extended or user-defined code lists (permanent or temporary). To be compliant, the community message standard must be directly derived from an approved UNSM and having the same function. Therefore, a UN/EDIFACT document instance is commonly only conformant with a community MIG.

4.2.2 Conformance and UN/CEFACT XML

In the case of UN/CEFACT XML messages there is a technical link between the published message structure and codes used by it. Using other code values in a XML document instance than published for the data elements of the message will make the document non-conformant, unless ‘decoupling’ has been applied to the message standard (as described within the UN/CEFACT NDR Rules). The term “decoupling” used in this document refers to decoupling selective or all qualified data types from a set of value enumerations (in other words separating codes from the message).

4.2.3 Validation methods

This document provides example validation methods to check whether a document instance conforms or complies to a published UN message standard. The validation of tools is out of scope of this document and so it is assumed some sort of testing will be carried out, which can help trading partners to understand and also verify they are conformant or compliant with the standard or specification.

It is, though, important that users will give a true reflection of the actual level of conformance. Therefore, the conformance statements made by each party should be able to express this in an unambiguous way.

- UN/EDIFACT document instance using code values specified or referred within the MIG is compliant with a published and approved UNSM in case the UN/CEFACT document instance is generate as a UNSM subset, as described in the UN/EDIFACT Message Design Guidelines. The document instance it conformant with a published and approved UNSM in case of pure UNSM, even if non UN code lists or code values are specified within the MIG.
- UN/CEFACT XML document instance using the published code values of the message standard is conformant. It will be non-conformant in case it uses other code values than published for the message standard, unless ‘decoupling’ of code list enumerations (code values) has been applied, as described within the UN/CEFACT XML NDR Rules. Decoupling implies a two-phase validation process as it separates the checking of message structure constraints and code value constraints..

Note:
A two-phase validation process consists of checking the well-formedness of an XML instance document and the message structure constraints. These checks are done at the same time (first phase). In addition, the value constraints, including code lists, will be checked within this process (second phase).

DRAFT

5 Using code lists in a real-life environment

5.1 Introduction

Codes (or enumerated values) are an integral component of any business-to-business information flow. Not only should they be understood by humans but also, they should be fully validated. International standardized codes are harmonized and unambiguous in order to enforce global trade. International standards organizations, but also many international, national and sectoral agencies create code lists. The meaning of a code is essential, and its metadata must be available for the code itself and for the list in which it is adopted. Only then a code could be fully validated for correctness to ensure consistent data. When used within an information flow, these code lists will be explicitly referred to.

5.2 Extended, restricted, user-defined and other organizations code lists

Users of the UN/CEFACT library may identify any sub-set (restriction) or superset (extension) they wish from a specific code list for their own user community requirements by defining code lists. These specific code lists could be based on standardized or user-defined code lists (permanent or temporary). Each type of code list can easily be accommodated with the solutions described in the next chapters.

Note:
The term ‘code lists’, used in this document applies to code lists and identifier lists.

DRAFT

6 Validating UN/EDIFACT document instances

6.1 Introduction

UNSMs are structured in such a way that they can be used by companies, governmental agencies and/or other organizations in many different industries. For most industries, a sub-set of the UNSM has been created because of the restrictive use of the message structure.

Users must bear in mind that to comply with the spirit of sub-sets, any sub-set[4] must always be more restrictive than its parent UNSM. Though validation of restricted, extended, user-defined and other organizations code lists or code values is done against the ones specified within the MIG.

For UN/EDIFACT message implementations five possible scenarios are clearly defined in respect to code lists.

6.2 Restricted code lists

In order to identify the restricted UNSM code list(s), the user community concerned should consider:

- specifying or referring to the restricted code lists or codes values within the MIG.
- referring to above in a Trading Partner Agreement.

6.3 Extended code lists

Since the standards maintenance time-scales may delay the implementation of the required modifications to the UNSM and the code lists repository for some time, users may wish to implement the needed code list(s) and/or code values immediately so that the message can be used in their application.

In order to identify the extended code lists during the interim period, the user community concerned should consider:

- specifying or referring to the extended code lists or code values within the MIG.
- including an appropriate code in element ‘0110 Code list directory version number’ of the UNH (only applicable for message syntax version 4). By this, users can refer to any directory version of a code list, different from the message directory version.
- including an appropriate code in element ‘1131 Code Lists Identification Code’ and/or ‘3055 Code List Responsible Agency’ (if available)[7], in order to identify the code list properly.
- referring to above in a Trading Partner Agreement.

> Note on the use of 1131/3055:
>
> This implies such extension is being expressed per individual code list appearing in such message, combined with the more global indication on the message basis. Whenever data element 3055 is used, data element 1131 is mandatory.

[4] To provide a unique identification for any particular sub-set of a UNSM, users may wish to assign a code for use in the 'Association assigned code' field of the UNH and/or UNG segments.
[7] See ANNEX A (Informative) Usage of data elements 1131/3055 of the UN/EDIFACT Message Design Guidelines

DRAFT

6.4 Choosing or combining code lists

Users may want to choose another code list for an element than published by UN/CEFACT or they even want to combine values from different code lists. Most common is choosing another code list than the published one for the applicable element. The user community concerned should consider:

- specifying or referring to the applicable code list or combined code lists within the MIG. Combined code values from different code lists can be regarded as a user-defined code list (see next paragraph).
- including an appropriate code in element ‘1131 Code Lists Identification Code’ and/or ‘3055 Code List Responsible Agency’ (if available), in order to identify the code list properly.
- referring to above in a Trading Partner Agreement.

Note on the use of 1131/3055:

This implies such choice or combination is being expressed per individual code list appearing in such message, combined with the more global indication on the message basis. Whenever data element 3055 is used, data element 1131 is mandatory.

In practice, a combination of code values from different code lists will be stored as a user-defined code list and referred to within the MIG. As an alternative EDIFACT document instances and code list could be converted to XML where ‘unions’ could be created by the validation process.

6.5 User-defined code lists (permanent or temporary)

User-defined code lists (permanent or temporary) are not uncommon. They often exist in specific industries. If needed, users could create such code lists and specify the code list for the applicable element in the MIG. These code lists should be identified as described in previous paragraph 6.4.

6.6 Code lists published by other organizations

For referencing code lists maintained by organizations external to UN/CEFACT, e.g. ICC, W3C, CODEX, CITES etcetera the same principle as described for user-defined code lists could be applied.

6.7 Validating document instances

During the decades of implementing EDIFACT messages many software tools were created for validating the document instances. Over time systems for processing XML based documents became more popular, and XML allows easy validation in a self-descriptive form, a new way for validating EDIFACT files was introduced.

For users, the below options are available for validating EDIFACT files:

- Traditional in-house validation
- ISO 20625: Converting EDIFACT document instances to XML document instances. By applying this transformation standard validation tools for XML validation can be applied.

DRAFT

When applying ISO 20625 the principles of validating XML-UN/EDIFACT document instances in respect to code lists as described in the next chapters do apply. It is therefore essential that code lists are available in an XML format.

DRAFT

7 Validating UN/CEFACT XML document instances

7.1 Introduction

UN/CEFACT XML messages are structured in such a way that they can be used by companies, governmental agencies and/or other organizations in many different industries. The user requirements regarding code management (see chapter 4), can all be fulfilled when for these UN/CEFACT XML messages ‘decoupling’ has been applied. The present published versions of UN/CEFACT XML message standards validates the messages structure and code values of a document instance simultaneously. Decoupling separates code value validation from message structure validation.

The latest UN/CEFACT XML NDR version allows flexible use of code values, code lists and identifier lists by allowing ‘decoupling’ of code values.

This chapter highlights the example methodologies that should be applied for restricted, extended, user-defined (permanent or temporary) code lists and other organizations code lists or code values.

Users of a ‘coupled’ version of the message standard may even want to restrict or extend code values to the code lists schemas or even introduce other code list schemas. By changing the published message standard, the validation process will be non-conformant with the published message standard. In order to be conformant with the published message standard, these users should implement a ‘decoupled’ version of the message standard. The validation process becomes then a two-phase process.

In the below simplified fragment of the qualified data type schema (left column), the qualified data type ‘DocumentCodeType’ is ‘coupled’ by means of the specified code list module (clm61001) which is being imported. The namespace, import declaration and extension base are marked grey.

In the right column, the qualified data type ‘DocumentCodeType’ is ‘decoupled’ by removal of the code list module import and namespace declaration. The extension base ‘DocumentCodeContentType’ is no longer linked to the code list module. Therefore, a simple type ‘DocumentCodeContentType’ has been specified. In addition, the simple type for the list agency ID ‘DocumentCodeListAgencyIDContentType’ does not have any enumeration values.

DRAFT

Qualified Data Type Schema: coupled version	**Qualified Data Type Schema: decoupled version**
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:qdt="urn:un:unece:uncefact:data:Standard:QualifiedDataType:21" xmlns:ccts="urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpecification:2" xmlns:udt="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:21" xmlns:clm61001="urn:un:unece:uncefact:codelist:standard:UNECE:DocumentNameCode:D16B" > <xsd:import namespace="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:21" schemaLocation="UnqualifiedDataType_21p0.xsd"/> <xsd:import namespace="urn:un:unece:uncefact:codelist:standard:UNECE:DocumentNameCode_Invoice:D16B" schemaLocation="../../codelist/standard/UNECE_DocumentNameCode_Invoice_D16B.xsd"/> <xsd:simpleType name=DocumentCodeListAgencyIDContentType">	<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:qdt="urn:un:unece:uncefact:data:Standard:QualifiedDataType:21" xmlns:ccts="urn:un:unece:uncefact:documentation:standard:CoreComponentsTechnicalSpecification:2" xmlns:udt="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:21" targetNamespace="urn:un:unece:uncefact:data:Standard:QualifiedDataType:21" elementFormDefault="qualified" version="21.0"> <xsd:import namespace="urn:un:unece:uncefact:data:standard:UnqualifiedDataType:21" schemaLocation="UnqualifiedDataType_21p0.xsd"/> <xsd:simpleType name=**DocumentCodeContentType**"> <xsd:restriction base="xsd:token"/> </xsd:simpleType>

```
          <xsd:restriction base="xsd:token">
               <xsd:enumeration value="6">
          </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="DocumentCodeType">
  <xsd:simpleContent>
<xsd:extension
base="clm61001:DocumentNameCodeContentType">
       <xsd:attribute name="listID" type="xsd:token"
use="optional" fixed="1001"/>
       <xsd:attribute name="listAgencyID"
type="qdt:DocumentCodeListAgencyIDContentType"
use="optional" fixed="6"/>
       <xsd:attribute name="listVersionID" type="xsd:token"
use="optional" fixed="D16B"/>
       <xsd:attribute name="name" type="xsd:string"
use="optional"/>
       <xsd:attribute name="listURI" type="xsd:anyURI"
use="optional"/>
                    </xsd:extension>
          </xsd:simpleContent>
</xsd:complexType>
```

```
<xsd:simpleType name=
DocumentCodeListAgencyIDContentType">
          <xsd:restriction base="xsd:token"/>
</xsd:simpleType>

<xsd:complexType name="DocumentCodeType">
   <xsd:simpleContent>
 <xsd:extension base="qdt:DocumentCodeContentType">
       <xsd:attribute name="listID" type="xsd:token"
default="1001"/>
       <xsd:attribute name="listAgencyID"
type="qdt:DocumentCodeListAgencyIDContentType"
default="6"/>
        <xsd:attribute name="listVersionID" type="xsd:token"
use="optional" default="D16B"/>
       <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="listURI" type="xsd:anyURI"/>



                    </xsd:extension>
          </xsd:simpleContent>
</xsd:complexType>
```

Figure 2: Coupled and decoupled code lists

In the case of coupled code list modules, the supplementary components of the qualified data type have ‘fixed’ values (marked blue). Using other values in the XML document instance will invoke a validation error during validation.

In the case of decoupled code list modules, the supplementary components of the qualified data type have ‘default’ values (marked yellow), which have to be changed by the user when other codes values are used than those in the referenced code list. This is necessary to avoid misinterpretations.

In the below example, the latest version ID of the code list ‘D17B’ is specified instead of the default ‘D16B’ version. In addition, the code value ‘889’ from code list ‘D17B’ is used for the element ‘TypeCode’.

Qualified data type coupled	***Qualified data type decoupled***
Supplementary components: - listID “**fixed” =1001,** - listAgencyID “**Fixed**” **= 6**, - listVersionID “**Fixed**” = **D16B**	*Supplementary components:* - listID “**Default” =1001,** - listAgencyID “**Default” = 6** - listVersionID “**Default**” = **D16B**
XML document instance fragment	*XML document instance fragment*
<ram:TypeCode listID="1001" listAgencyID="6" listVersionID="D16B">385</ram:TypeCode>	*<ram:TypeCode listID="1001" listAgencyID="6" listVersionID="D17B">889</ram:TypeCode>*

Figure 3: Supplementary Components (coupled and decoupled)

The use of default values in the supplementary components of the qualified data reminds the user of UN/CEFACT available code lists and recommendations.

In below examples, a combination and alternative usage of code lists is specified by XML declarations in the qualified data type. The code list metadata, such as agency ID, is not specified because multiple code lists are declared for a single qualified data type. By this, the metadata of the code lists becomes unambiguous and cannot be validated.

A two-phase validation process, which uses ‘decoupling’ and a rule-based validation language, such as Schematron, solves the problem for these scenarios.

DRAFT

Union: XML declarations qualified data type	Choice: XML declarations qualified data type
<xsd:simpleType name=AccountDutyTypeCode">	<xsd:complexType name="PersonPropertyCodeType">
<xsd:annotation>	<xsd:annotation>
….see annotation….	... see annotation ...
</xsd:annotation>	</xsd:annotation>
<xsd:union memberType=	<xsd:choice>
“clm64437:AccountingTypeCodeContentType”	<xsd:element ref="clm63479:MaritalCode"/>
“clm65153:DutyTaxFeeTyoeCodeContentType”	<xsd:element ref="clm63499:GenderCode"/>
</xsd:simpleType>	</xsd:choice>
	</xsd:complexType>

Figure 4: Union and choice (coupled code list modules)

7.2 One-phase validation process

In order to fulfil all user requirements, as decribed in chapter 4, existing published standardized code lists have to be changed and “saved as” when choosing for a one-phase validation process method[10].

Changing existing message standards is for most users not preferable, because the XML document instance will be non-conformant with the published message standard. For those users, the two-phase validation process methods[11] are available.

For UN/CEFACT XML message implementations five possible scenarios are clearly defined in respect to code lists.

7.2.1 Restricted code lists

In case of allowing users to change existing code list schemas, they could create additional schemas per code list defining those restricted code lists, as described in the NDR specification. The software performing the validation compares the XML message document instance against the restricted code list module schema.

To ensure interoperability the usage of restricted code lists must be agreed on in a Trading Partner Agreement and/or a MIG.

The following steps have to be performed for restriction of a published UN/CEFACT code list:

1. Create a new code list schema file for the restricted code list.
2. Modify the original qualified data type schema so that the corresponding type refers to the newly created code list schema.

7.2.2 Extended code lists

The same procedure as described in previous paragraph can be applied for extending existing code list module schemas. The software performing the validation compares the XML message document instance against the modified code list module schema and qualified data type schema.

7.2.3 Choosing or combining code lists

The UN/CEFACT NDR specification also describes choosing or combining values from different code lists by using either the xsd:choice or xsd:union elements. There are examples

DRAFT

[10] Both message structure and code values constraints are validated simultaneously.
[11] Message structure and code values constraints are validated separately.

provided in this document for these options (see §7.2). For further details we refer to the UN/CEFACT NDR specification. As mentioned in paragraph 7.2, the xsd:choice and xsd:union implementation within the qualified data type, do not address supplementary component differences, as they can only be declared for a single qualified data type.

7.2.4 User-defined code lists (permanent or temporary)

User-defined code lists, either permanent or temporary, are not uncommon. They often exist in specific industries. If needed, users could create such code lists modules for the applicable qualified data types specified within the qualified data type schema. A user-defined code list can often be regarded as an extended code list (see example §7.2.7). The user creates a new code list schema module and modifies the original qualified data type schema so that the corresponding type refers to the user-defined code list schema.

7.2.5 Code lists published by other organizations

For referencing code lists maintained by organizations external to UN/CEFACT, e.g. ICC, W3C, CODEX, CITES etcetera the same principle as described in the preceding paragraph would be applied. The user modifies the original qualified data type schema so that the corresponding type refers to the user-defined code list schema.

7.2.6 Example for a restricted code list

To demonstrate the methodology the use case of restricting the valid currencies in an XML document instance could be looked at. In this example only the use of the Euro currency should be valid in the corresponding user community. The corresponding schema then could look like shown in Figure 5. In this example, the code list schema is saved as **Invoice_** ISO_ISO3AlphaCurrencyCode_2012-08-31.xsd.

The schema for the qualified data types now needs to be adjusted to the new code list file. Only the relevant parts are shown in the following figure. It is allowed to alter the namespace prefix accordingly. For simplification, the original namespace prefix is kept.

Qualified data type schema	**Code list schema**
<xs:schema ... xmlns:clm5ISO42173A= "urn:un:unece:uncefact:codelist:standard: ISO:ISO3AlphaCurrencyCode:INVOICE" ... elementFormDefault="qualified" version="1.0"> <xs:import namespace="urn:un:unece:uncefact:codelist:standard: ISO:ISO3AlphaCurrencyCode:INVOICE" schemaLocation="Invoice_ ISO_ISO3AlphaCurrencyCode_2012-08-31.xsd"/> ... </xs:schema>	<xs:schema xmlns:clmISO42173AINVOICE= "urn:un:unece:uncefact:codelist:standard: ISO:ISO3AlphaCurrencyCode:INVOICE" xmlns:xs="http://www.w3.org/2001 /XMLSchema" targetNamespace= "urn:un:unece:uncefact:codelist:standard:ISO: ISO3AlphaCurrencyCode:INVOICE" elementFormDefault="qualified" version="9.5"> <xs:simpleType name="ISO3AlphaCurrencyCodeContentType"> <xs:restriction base="xs:token"> <xs:enumeration value="EUR"/> <xs:enumeration value="USD"/> </xs:restriction> </xs:simpleType> </xs:schema>

Figure 5: Restricted code list (code list schema and qualified data type)

DRAFT

7.2.7 Example for an extended code list

To demonstrate the methodology the use case of extending the valid VAT category codes in an XML document instance should be looked at. In this example, the existing code list should be valid and a new code value ‘BB’ should be added. The corresponding code list schema then could look like shown in Figure 6. In this example, the code list schema is saved as VATExtended_UNECE_DutyorTaxorFeeCategoryCode_D17B.xsd.

The schema for the qualified data types now needs to be adjusted to the new code list file. Only the relevant parts are shown in the following figure. It is allowed to alter the namespace prefix accordingly. For simplification, the original namespace prefix is kept.

Qualified data type schema	Code list schema
<xs:schema ...	<xs:schema xmlns:clm65305=
xmlns:	"urn:un:unece:uncefact:codelist:
clm65305="urn:un:unece:uncefact:codelist	standard:UNECE:DutyorTaxorFeeCategoryCode
:standard:UNECE:	:D17B:VATEXTENDED"
DutyorTaxorFeeCategoryCode	xmlns:xs="http://www.w3.org/2001/XMLSchema"
:D17B:VATEXTENDED ...	targetNamespace= "urn:un:unece:uncefact
elementFormDefault="qualified" version="1.0">	:codelist:standard:UNECE
<xs:import namespace="	:DutyorTaxorFeeCategoryCode:D17B
urn:un:unece:uncefact:codelist:standard:UNECE	:VATEXTENDED" elementFormDefault="qualified"
:DutyorTaxorFeeCategoryCode:D17B	version="1.5">
:VATEXTENDED "	<xs:simpleType
schemaLocation="	name="DutyorTaxorFeeCategoryCodeContentType">
VATExtended_UNECE_DutyorTaxorFee	<xs:restriction base="xs:token">
CategoryCode_D17B.xsd"/>	<xs:enumeration value="A"/>
...	<xs:enumeration value="AA"/>
</xs:schema>	<xs:enumeration value="AB"/>
	<xs:enumeration value="AC"/>
	<xs:enumeration value="AD"/>
	<xs:enumeration value="AE"/>
	<xs:enumeration value="B"/>
	<xs:enumeration value="BB"/>
	<xs:enumeration value="C"/>
	<xs:enumeration value="D"/>
	<xs:enumeration value="E"/>
	<xs:enumeration value="F"/>
	<xs:enumeration value="G"/>
	<xs:enumeration value="H"/>
	<xs:enumeration value="I"/>
	<xs:enumeration value="J"/>
	<xs:enumeration value="O"/>
	<xs:enumeration value="S"/>
	<xs:enumeration value="Z"/>
	</xs:restriction>
	</xs:simpleType>
	</xs:schema>

Figure 6: Extended code list (code list schema and qualified data type)

DRAFT

7.2.8 Impacts for a real-life environment

The advantage is that still a one-phase validation can be performed. But the modified code list schema needs to be published and maintained within the user community in order to simplify implementation and keep consistency. All users need to agree on using the modified code list schema and to be non-conformant to the published message standard.

The non-conformance issue can be avoided by applying a two-phase validation process (see next paragraph) in which code list are decoupled from the message standard.

7.3 Two-phase validation process

In a two-phase validation process method[12] structural validation is executed independent of value validation, and done in the first phase of the process. The validation of code values is performed in a second phase following a successful first phase validation. This two-phase validation process method is ideal for users who prefer maximum flexibility regarding code lists and/or code values.

The two-phase validation methods, described in this document, are rule based. Schematron is used as the rule based validation language. Schematron is capable of expressing constraints in ways that other XML schema languages like XML Schema and DTD cannot. For example, it can require that the content of an element be controlled by one of its siblings. Or it can request or require that an element must have specific attributes (e.g. code list metadata and/or specific code values).

Figure 7 illustrates the essence of the two-phase validation process. It shows the distinction between structural constraints validation (phase 1) and value validation (phase 2). Structural validation is typically performed by using XSD schema (marked '1') and value constraint validation is typically performed by using XSLT (marked '2'). As constraints are specified as rules using Schematron, they will be deployed as XSLT code, making it practical for applications.

Trading partners can execute value validation using whatever tools are appropriate to their environment.

In addition to the validation performed by the inhouse application, trading partners may use one of the following commonly used standards for value constraints validation.

1.	Schematron/XSLT	(ISO/IEC 19757-3 / W3C)
2.	Schematron/XSLT using Genericode/CVA	(OASIS)

DRAFT

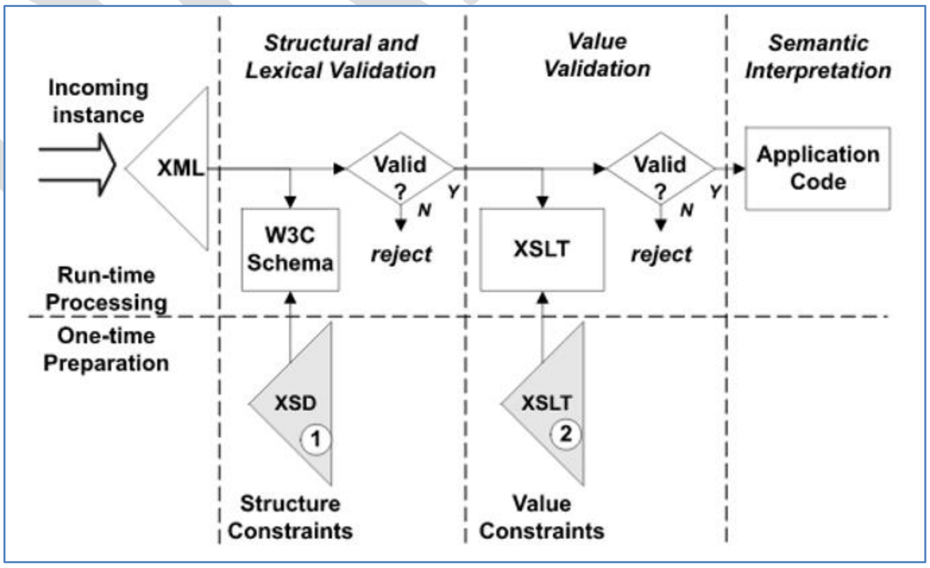


Figure 7: Two-phase validation process

[12] Message structure and code values constraints are validated separately.

7.3.1 ISO Schematron/XSLT

Schematron is a rule-based validation language that uses context expressions. A Schematron schema makes assertions applied to a specific context within the XML document. If an assertion fails a diagnostic message can be displayed. In order to implement the context expressions used in the Schematron rules, XPath is used with various extensions provided by XSLT. As path expressions are built on top of XPath and XSLT, one should implement Schematron using XSLT (an assert element has a test attribute, which is an XSLT pattern). XML documents have data elements to be validated. The context location of those data elements is represented using XPath.

For UN/CEFACT XML message implementations five possible scenarios are clearly defined in respect to code lists. In below paragraphs, these are specified for both two-phase validation process methods.

7.3.1.1 Restricted code lists

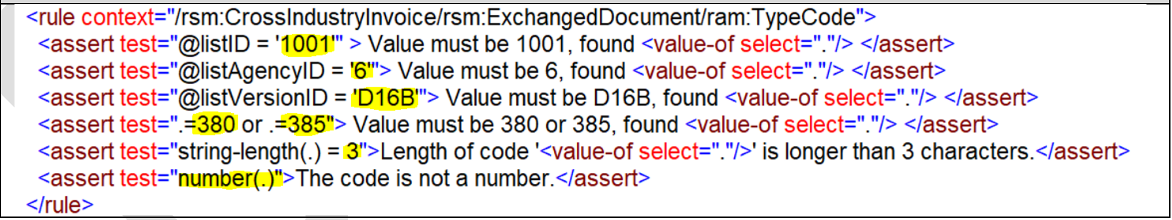
The restricted (code) values for a specific context within the XML document, such as ExchangedDocument/TypeCode, can be expressed as an assertion in a Schematron rule. In addition, assertions for the supplementary components can be included. From the Schematron file an XSLT file can be generated automatically using a tool.

In below example, the allowed code values and supplementary codes have been specified as a Schematron rule.

This simplified example allows only the exchanged document type codes (in an invoice):

- code values ‘380’ and ‘385’
- *code list ID ‘1001’*
- *list agency ID ‘6’*
- *code list version ‘D16B’.*
- *code value length is restricted to 3 characters and must be numeric.*

Schematron rule



```
<rule context="/rsm:CrossIndustryInvoice/rsm:ExchangedDocument/ram:TypeCode">
  <assert test="@listID = '1001'" > Value must be 1001, found <value-of select="."/> </assert>
  <assert test="@listAgencyID = '6'"> Value must be 6, found <value-of select="."/> </assert>
  <assert test="@listVersionID = 'D16B'"> Value must be D16B, found <value-of select="."/> </assert>
  <assert test=".=380 or .=385"> Value must be 380 or 385, found <value-of select="."/> </assert>
  <assert test="string-length(.) = 3">Length of code '<value-of select="."/>' is longer than 3 characters.</assert>
  <assert test="number(.)">The code is not a number.</assert>
</rule>
```

Figure 8: Schematron rule (restricted code list)

A user most likely wants to link code values from his ‘restricted’ UN/CEFACT code list schema module instead of specifying each allowed code value within an assertion manually. An important feature to note is that, because of XSLT's *Document()* function, a Schematron assertion test can refer to data in a different document from the context document. This allows Schematron to be used to validate against a code list located externally to the schema (this can be in any XML document type). Although the function *Document()* includes external codes values for this purpose, it would still be quite some time consumed to write the needed code.

7.3.1.2 Extended code lists

The extended code values for a specific context within the XML document instance, such as ExchangedDocument/TypeCode, can be expressed as assertions in a Schematron rule. The

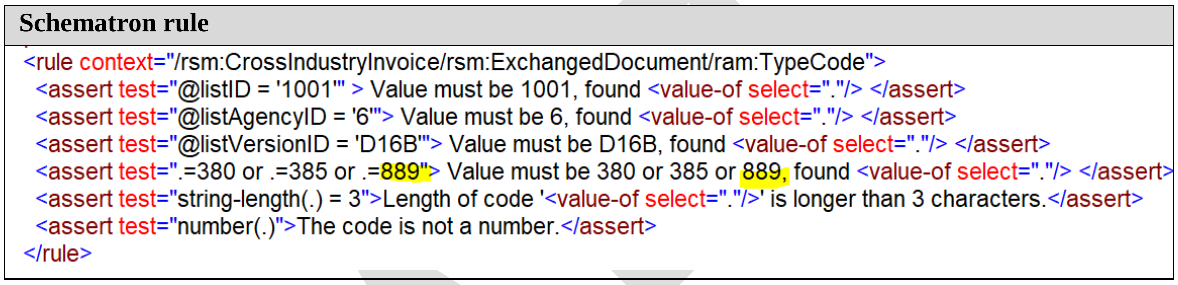
DRAFT

extended code values could be added to an existing assertion or by adding an assertion next to the one holding the base set of code values. In addition, assertions for the supplementary components can be included as well. From the Schematron file an XSLT file can be generated automatically by using a tool.

In below example, the allowed code values and supplementary codes have been specified as a Schematron rule.

This simplified example allows only the exchanged document type codes (in an invoice):

- code values ‘380’, ‘385’ and ‘889’ (added),
- *code list ID ‘1001’*
- *list agency ID ‘6’*
- *code list version ‘D16B’.*
- *The length of the code value is restricted to 3 characters and must be numeric.*



Schematron rule

```
<rule context="/rsm:CrossIndustryInvoice/rsm:ExchangedDocument/ram:TypeCode">
  <assert test="@listID = '1001'" > Value must be 1001, found <value-of select="."/> </assert>
  <assert test="@listAgencyID = '6'"> Value must be 6, found <value-of select="."/> </assert>
  <assert test="@listVersionID = 'D16B'"> Value must be D16B, found <value-of select="."/> </assert>
  <assert test=".=380 or .=385 or .=889"> Value must be 380 or 385 or 889, found <value-of select="."/> </assert>
  <assert test="string-length(.) = 3">Length of code '<value-of select="."/>' is longer than 3 characters.</assert>
  <assert test="number(.)">The code is not a number.</assert>
</rule>
```

Figure 9: Schematron rule (extemded code list)

A user most likely wants to link code values from his ‘extended’ UN/CEFACT code list schema module instead of specifying each allowed code value within an assertion manually. The function *Document()* could be used to link external located code values for this purpose.

7.3.1.3 Choosing or combining code lists

Combined code lists can be achieved by adding multiple assertions using function *Document()*) in order to refer to multiple code lists or by specifying the combined code values as one or multiple assertion. Alternative code lists to choose from, can be specified as different Schematron rules referring to externally located code lists using the Schematron function *Document()* or by specifying the code values as an assertion.

7.3.1.4 User-defined code lists (permanent or temporary)

User-defined code lists, either permanent or temporary, are not uncommon. They often exist in specific industries. These code lists could be regarded as additional or extended code lists. For both assertions within Schematron rules can be used to specify the code values or refer to externally located code lists using the Schematron function *Document()*.

7.3.1.5 Code lists published by other organizations

An external maintained code list could be treated as a user defined code list using assertions to specify the needed code values or refer to externally located code lists using the Schematron function *Document()*.

7.3.1.6 Impacts for a real-life environment

From a user-perspective, the Schematron/XSLT validation method requires users to take the following steps:

DRAFT

- Create code lists (including metadata) in such a way that Schematron rules can validate these data.
- Write Schematron rules for checking the allowed code value(s), supplementary components, appropriate document context(s), all including error messages.
- Use a tool which generates the XSLT file from the Schematron file.
- Create an environment managing the Schematron rules in order to easy maintenance on code lists and code values.

7.3.2 ISO Schematron/XSLT – using Genericode/CVA

This method uses, in addition to ISO Schematron/XSLT, a standard representation format of code lists named ‘genericode’ and associations that link context and values named ‘ContextValueAssociation’. It is a more user-friendly and code-management-orientated method and eases implementation through the use of a freely available tool for the creation of the Schematron/XSLT files.

In this method, the base code lists remain untouched. The extended, restricted, user-defined codes (permanent or temporary) are specified in separate genericode files, each with their own identifying list-level metadata.

The Context/Value Association (CVA) file specifies the XPath contexts of an XML document instance and the genericode file(s) applicable to each context. Unlike XSD enumerations binding the same enumeration to all contexts of a globally-declared and re-used business artefact (BBIE) in a message standard, the use of XPath in CVA provides for specifying different code lists at different contexts of one BBIE. Perhaps the user needs to validate against different lists of currency codes at different ‘currency code locations’ of a single XML document.

In other words, validation can be done on different context levels:

Context levels	Context address as specified in CVA file (examples)
System-wide	address="ram:SpecifiedTradeProduct/ram:TypeCode"/>
Document-wide	address="rsm:CrossIndustryInvoice//ram:InvoiceCurrencyCode"/>
Element specific	address="rsm:CrossIndustryInvoice /rsm:SupplyChainTradeTransaction /ram:IncludedSupplyChainTradeLineItem /ram:SpecifiedTradeProduct /ram:ColourCode"/>

Figure 10: Context levels and contest address

The Schematron expressions[15] leverage any code list metadata found in the BBIE’s supplementary components to ensure the appropriate genericode expression of codes is used in the given XML document instance. Finally, these XML expressions can be processed by applications creating visual interfaces in order to tailor drop-down lists of coded value domains presented to users.

A genericode file contains the following data which can be used during validation:

- code list values
- code list metadata

[15] Schematron rules are generated automatically by a free of charge tool, but users could write the rules themselves.

DRAFT

The code value found in the XML document instance will be checked against the genericode files linked by association. The location of a genericode file is declared with URI address and the identity of each code list is unique. An association links the document's context with a set of genericode files.

Any supplementary component (metadata) present in the XML document instance is also checked against the code list value metadata specified in the genericode file. All community members use the same message schemas for the initial structural constraints, while the many and varied and contextual requirements for value validation agreed upon between trading partners, perhaps even in real time, are realized as needed.

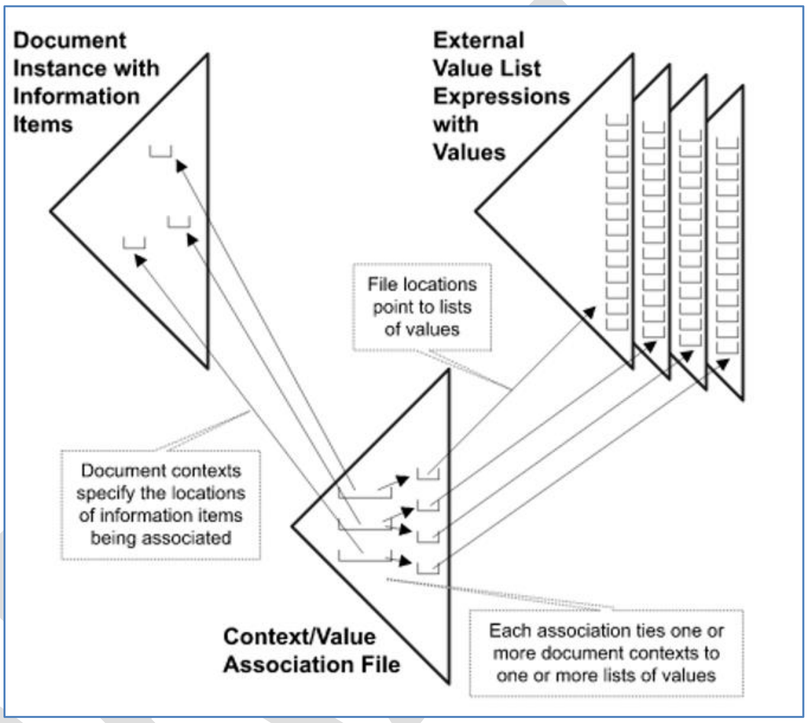

Document Instance with Information Items
External Value List Expressions with Values
File locations point to lists of values
Document contexts specify the locations of information items being associated
Context/Value Association File
Each association ties one or more document contexts to one or more lists of values

Figure 11: Concept of ISO Schematron/XSLT – using Genericode/CVA .

The Context/Value Association

The Context/Value Association file format is an XML vocabulary using address expressions to specify hierarchical document contexts and their associated constraints. A document context specifies one or more locations found in an XML document or other similarly structured hierarchy of information. A constraint is expressed as either an explicit expression evaluation or as a value inclusion in one or more controlled vocabularies of values.

This file format specification assumes a controlled vocabulary of values is expressed in an external resource described by the OASIS genericode standard.

For each code list scenario, the applicable CVA 'Value lists' (code lists) and 'Contexts' (associations) will be described in the following paragraphs.

The concept of a masquerade

The CVA file employs the concept of a masquerade. The masquerade overlays the complete list's metadata in place of the customized code list's metadata during the validation process in real time. This prevents confusion and ambiguity regarding the identity of the customized code list which is not and should not be identified as a complete list in its metadata.

A data element citing the full list will successfully validate against the extended or restricted list using the masquerade of the full list. This ensures multiple extended or restricted lists of the same full list can be uniquely identified and managed by their respective distinguished metadata.

The concept of masquerade is also used in case of combining code lists, in which of them is taken as the masquerade overlay. Different trading partners can mutually use different sets of code lists.

Masquerade
In this example, the masquerade overlays the “ISO3AlphaCurrencyCode” list’s metadata in place of the “InvoiceCurrencyTypeCodes” code list’s metadata.
Eaxmple: Value lists
<ValueLists> <ValueList xml:id="InvoiceCurrencyCodesD17B" masqueradeUri= "../gc/ISO_ISO3AlphaCurrencyCode_2012-08-31.gc"/> uri="../gc/InvoiceCurrencyTypeCodes.gc"/> </ValueLists>

Figure 12: Concept of masquerade

7.3.2.1 Restricted code lists

A restricted code list is a shorter version of the applicable full-list genericode file. The masquerade ensures re-use of the metadata specified in the UNECE full code list.

Restricting code values	
In this example, the invoice currency code list (restricting of ISO code list) is used only for the TaxCurrencyCode element specified with the Header Trade Settlement component.	
Example: Contexts	*Example: Value lists*
<Contexts> <Context values=" InvoiceTaxCurrencyCodesD17B" metadata="cctsV2.01-code" address=" rsm:CrossIndustryInvoice/ rsm:SupplyChainTradeTransaction/ram :ApplicableHeaderTradeSettlement/ram:TaxCurrencyCode"/> </Contexts>	<ValueLists> <ValueList xml:id="InvoiceTaxCurrencyCodesD17B" masqueradeUri="../gc/ ISO_ISO3AlphaCurrencyCode_2012-08- 31.gc"/> uri="../gc/InvoiceTaxCurrencyTypeCodes.gc"/> </ValueLists>

Figure 13: Restricted code list

7.3.2.2 Extended code lists

The extended code list is a genericode containing only the extended code values compared to the version of the applicable full-list genericode file. The masquerade ensures re-use of the metadata specified in the full-list genericode file.

The CVA file would express the union of the full-list genericode file and the extended genericode file. The masquerade would make the entire list appear to have the full-list genericode file list’s metadata. In this way at no time is there an ambiguous publication of a mixed list with metadata that could be confused with the metadata of the published list. When the published list is revised, the extended code values are incorporated as in extended genericode file.

DRAFT

Extending code values	
In this example, the ISO 3 alpha currency code list (base list) has been extended by the new ISO 3 alpha currency code list (containing only new currency code values). The code list is used only for the TaxCurrencyCode element specified with the Header Trade Settlement component.	
Example: Contexts	*Example: Value lists*
</Contexts> <Context values=" ISO_ISO3AlphaCurrencyCode_2012-08-31 NEW_ISO3AlphaCurrencyCode_2017-09-08" metadata="cctsV2.01-code" address=" rsm:CrossIndustryInvoice"/> </Contexts>	<ValueLists> <ValueList xml:id=" ISO_ISO3AlphaCurrencyCode_2012-08-31" uri="../gc/ ISO_ISO3AlphaCurrencyCode_2012-08- 31.gc"/> <ValueList xml:id="NEW_ISO3AlphaCurrencyCode" masqueradeUri= "../gc/ISO_ISO3AlphaCurrencyCode_2012- 08-31.gc "/> uri="../gc/ NEW_ISO3AlphaCurrencyCode.gc "/> </ValueLists>

Figure 14: Extended code list

7.3.2.3 Choosing or combining code lists

Combining code values of different code lists is the essence of genericode/CVA. Users can create as many code lists as needed. A union of code lists means specifying multiple ‘Value lists’ and specifying these within the ‘Context value’ in the CVA file.

Combining code values	
In this example, the transport means type code list is combined with the transport means type code list of recommendation 28.	
Example: Contexts	*Example: Value lists*
<Contexts> <Context values=" UNECE_TransportMeansTypeCode_2007 UNECE_Rec28_Codes_for_Types_of_ Means_of_Transport_2007" metadata="cctsV2.01-code" address=" rsm:CrossIndustryInvoice/rsm: SupplyChainTradeTransaction/ ram:ApplicableHeaderTradeDelivery/ ram:RelatedSupplyChainConsignment/ ram:SpecifiedLogisticsTransportMovement/ ram:UsedLogisticsTransportMeans/ram:TypeCod e "/> </Contexts>	</ValueLists> <ValueList xml:id="UNECE_TransportMeansTypeCode_2007" uri="../gc/UNECE_TransportMeansTypeCode_2007. gc"/> <ValueList xml:id="UNECE_Rec28_Codes_for_Types_of_ Means_of_Transport_2007" masqueradeUri= "../gc/UNECE_TransportMeansTypeCode_2007.gc"/ > </ValueLists>

Figure 15: Comnined code list

7.3.2.4 User-defined code lists (permanent or temporary)

A user-defined code list is a genericode file containing only the user-defined code values. This genericode file would have its own identity. The user-defined permanent and/or temporary code values may be adopted in a new version of a standardized code list.

User-defined code values
In this example, the user-defined end item type code list is only applicable for the element EnditemTypeCode used within the below specified XPATH.

DRAFT

Example: Contexts	*Example: Value lists*
<Contexts> <Context values=" User_Defined_Enditem_TypeCode_2017" metadata="cctsV2.01-code" address=" rsm:CrossIndustryInvoice/rsm:SupplyChainTradeTransaction/ram:IncludedSupplyChainTradeLineItem/ram:SpecifiedTradeProduct/ram:EndItemTypeCode"/> </Contexts>	</ValueLists> <ValueList xml:id="User_Defined_Enditem_TypeCode_2017" uri="../gc/ User_Defined_Enditem_TypeCode_2017.gc"/> </ValueLists>

Figure 16: User-defined code list

7.3.2.5 Code lists published by other organizations

An external maintained code list could be treated as a user defined code list (see previous paragraph).

7.3.2.6 Impacts for a real-life environment

From a user-perspective, the Schematron/XSLT using genericode/CVA method offers users the following advantages:

- A user-friendly code management solution solving all issues identified by the code management project team.
- Easy implementation through the use of a freely available tool for the creation of the Schematron/XSLT files.
- Users can focus on the maintenance of genericode files and context associations, without having to write extensive files expressing their needs.
- Code list values and metadata are stored in a standardized file format (genericode).
- Associations between document context and applicable code lists are stored in a standardized file format (CVA).
- By using the ‘masquerade’ function, unions of code lists are recognized as one single code list during validation and can be presented in user dropdown lists.
- Through the existence of genericode files and the ‘masquerade’ function, the supplementary components can be checked to avoid any ambiguity.

DRAFT

9 Publication format code lists

9.1 Genericode

Genericode is a standard format for defining code lists.

The genericode standard offers:

- a XML format designed to support interchange or distribution of machine-readable code list information between systems.
- a XML format that can be transformed into formats suitable for run-time usage, or loaded into systems that perform run-time processing using code list information.
- a tabular structure for code list information:
    - each row in the table represents a single distinct entry in the code list, i.e. each row represents a single uniquely identifiable item in the code list.
    - each column in the table represents a metadata value that can be defined for each distinct entry in the code list. Each column is either required or optional.

Genericode is used, among other things, by UBL (Universal Business Language) and FpML (Financial Products Markup Language). Genericode is advised by the Dutch government, the education world (New Zealand) and electronic commerce (e-Commerce, EDI) as complementary to UBL. The standard is used worldwide.

Besides, genericode files are an essential component within the code value validation method 'Schematron/XSLT using Genericode/CVA'. In fact, they could be used as a component within every code validation environment. In addition, the genericode standard format for defining code lists is translation syntax independent. From a genericode file, XSD code list schema modules or any other format could be created. This will ease the maintenance of code lists in environments, such as where UN/EDIFACT and UN/CEFACT XML use the same code list repository.

The two-phase validation methods, described in this document and beyond, will benefit from a publication of code lists in one single representation format. Both UN/EDIFACT and UN/CEFACT XML messages may use one or more code lists during a two-phase validation process.

9.2 Code list Document

The OASIS Code List Representation format[16], "genericode", is a single model and XML format (with a W3C XML Schema) that can encode a broad range of code list information. The XML format is designed to support interchange or distribution of machine-readable code list information between systems. Note that genericode is not designed as a run-time format for accessing code list information, and is not optimized for such usage. Rather, it is designed as an interchange format that can be transformed into formats suitable for run-time usage, or loaded into systems that perform run-time processing using code list information.

[16] http://docs.oasis-open.org/codelist/cs-genericode-1.0/doc/oasis-code-list-representation-genericode.pdf

DRAFT

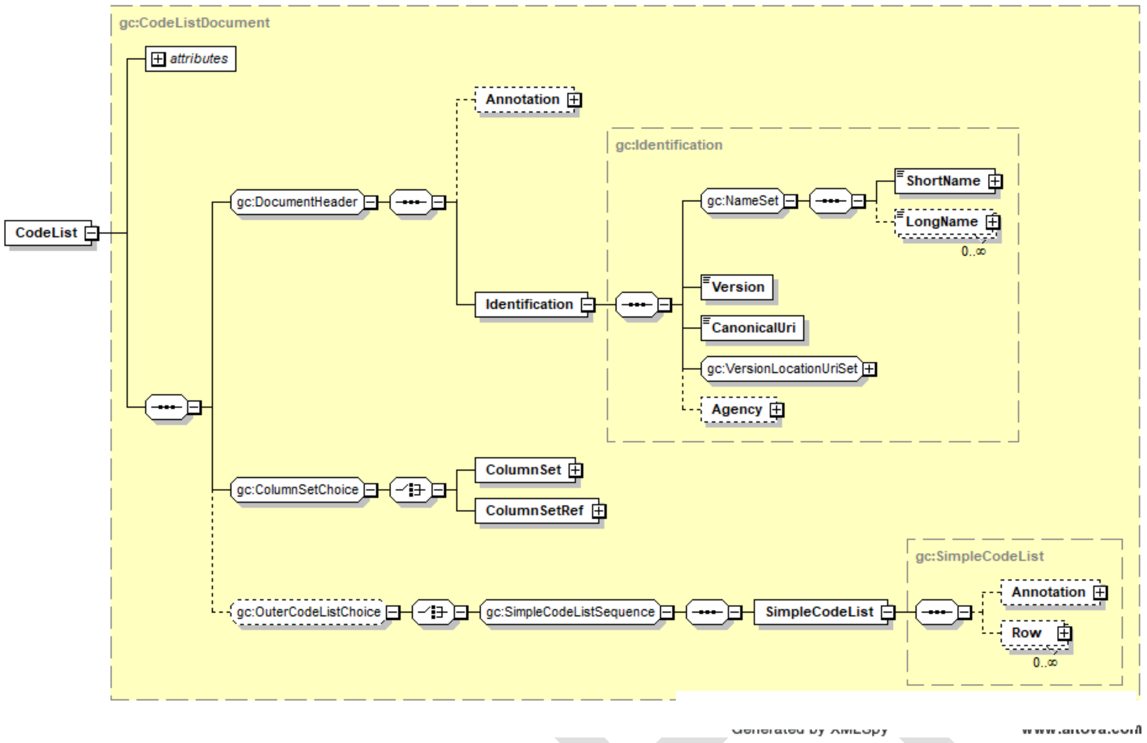


Figure 17: Code List Document Schema

DRAFT

9.4 Example UNECE_DocumentNameCode_Invoice_D16B.gc

```
?xml version="1.0" encoding="UTF-8"?>
<gc:CodeList xmlns:gc="http://docs.oasis-open.org/codelist/ns/genericode/1.0/">
  <Identification>
    <ShortName>DocumentNameCode_Invoice</ShortName>
    <LongName xml:lang="en">Document Name Code_Invoice</LongName>
    <Version>D16B</Version>
    <CanonicalUri>urn:un:unece:uncefact:codelist:standard:UNECE:DocumentNameCode_Invoice</CanonicalUri>
<CanonicalVersionUri>urn:un:unece:uncefact:codelist:standard:UNECE:DocumentNameCode_Invoice:D16B</CanonicalVe
rsionUri>
    <Agency>
      <LongName xml:lang="en">United Nations Economic Commission for Europe</LongName>
      <Identifier>6</Identifier>
    </Agency>
  </Identification>
  <ColumnSet>
    <Column Id="code" Use="required">
      <ShortName>Code</ShortName>
      <Data Type="normalizedString"/>
    </Column>
    <Column Id="name" Use="required">
      <ShortName>Name</ShortName>
      <Data Type="string"/>
    </Column>
    <Column Id="description" Use="required">
      <ShortName>Description</ShortName>
      <Data Type="string"/>
    </Column>
    <Key Id="codeKey">
      <ShortName>CodeKey</ShortName>
      <ColumnRef Ref="code"/>
    </Key>
  </ColumnSet>
  <SimpleCodeList>
    <Row>
      <Value ColumnRef="code">
        <SimpleValue>80</SimpleValue>
      </Value>
      <Value ColumnRef="name">
        <SimpleValue>Debit note related to goods or services</SimpleValue>
      </Value>
      <Value ColumnRef="description">
        <SimpleValue>Debit information related to a transaction for goods or services to the relevant party.</SimpleValue>
      </Value>
    </Row>
    ……………………………..
    ……………………………….
  </SimpleCodeList>
</gc:CodeList>
```

Figure 18: Example Genericode file

DRAFT

10 Definition of Terms

Term	Definition
Restriction	Removing code values from an existing code lists or by saving the changed one as a new code list.
Choice (of code lists)	XML Schema choice element allows only one of the elements contained in the <choice> declaration to be present within the containing element. In other words one one of the code lists is applicable for the element involved.
Coupled	During the validation of the document instance, code values are validated simultaneously with the message structure constraints (one-phase validation process).
EDIFACT	The EDIFACT standard provides a set of syntax rules to structure data for interactive exchange of standard messages between multi-country and multi-industry.
Extension	Adding new code values to an existing code list or by saving the changed one as a new code list.
ISO	International Standards Organization
One-phase validation process	Both message structure and code values constraints are validated simultaneously.
Schematron	Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees.
Sub-set	See restriction
Superset	See extension
Two-phase validation process	Only message structure constraints are validated during this process phase.
Union (of code lists)	The union element defines a simple type as a collection (union) of values from specified simple data types. In other words it combines one or more code lists.
Uncoupled	During the validation of the document instance, code values are not validated simultaneously with the message structure constraints, but validated in a next phase (two-phase validation process).
Validating	Checking that a document instance meets specifications and fulfills its intended purpose.
XML	Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable through use of tags that can be created and defined.
XSD	XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document.