Code List Representation (genericode) Version 1.0

Committee Specification 02

06 April 2022

This stage:
https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/genericode-v1.0-cs02.html
https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/genericode-v1.0-cs02.pdf
https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/genericode-v1.0-cs02.xml (Authoritative)

Previous stage:
https://docs.oasis-open.org/codelist/genericode/v1.0/csd04/genericode-v1.0-csd04.html
https://docs.oasis-open.org/codelist/genericode/v1.0/csd04/genericode-v1.0-csd04.pdf
https://docs.oasis-open.org/codelist/genericode/v1.0/csd04/genericode-v1.0-csd04.xml (Authoritative)

Latest stage:
https://docs.oasis-open.org/codelist/genericode/v1.0/genericode-v1.0.html
https://docs.oasis-open.org/codelist/genericode/v1.0/genericode-v1.0.pdf

Technical Committee:
OASIS Code List Representation TC

Chair:
Andrea Caccia (andrea.caccia@studiocaccia.com), Individual

Editor:
G. Ken Holman (gkholman@CraneSoftwrights.com), Crane Softwrights Ltd.

Additional artefacts:
This prose specification is one component of a Work Product that also includes:

- Documentation support files:
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/art/
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/db/
- JSON sample instances:
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/json-example/
- Schematron auxiliary rule constraints:
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/sch/
- XML sample instances:

- https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/xml/

- XML structural constraints:
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/xsd/
- Sample JSON translation in XSLT:
  - https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/xslt/

The ZIP containing the complete files of this release is found in the directory:

- https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/genericode-v1.0-cs02.zip

Declared XML Namespaces:

http://docs.oasis-open.org/codelist/ns/genericode/1.0/
http://docs.oasis-open.org/codelist/ns/rule/1.0/

Abstract:

This specification defines the genericode v1.0 vocabulary, rules, and serialization.

Status:

This document was last revised or approved by the OASIS Code List Representation Technical Committee on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=codelist#technical.

TC members should send comments on this document to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/codelist/.

This specification is provided under the RF on Limited Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (https://www.oasis-open.org/committees/codelist/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:

[**genericode-1.0**] *Code List Representation (genericode) Version 1.0.* Edited by G. Ken Holman. 06 April 2022. OASIS Committee Specification 02. https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/genericode-v1.0-cs02.html. Latest stage: https://docs.oasis-open.org/codelist/genericode/v1.0/genericode-v1.0.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the “OASIS IPR Policy”). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’ procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name “OASIS” is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for guidance.

Table of Contents

Appendixes

1 Introduction

1.1 Overview

1.1.1 Key/value semantics

The key-value store concept is a long-understood data storage paradigm for managing as associative array arrangement of information. The keys of the store are unique values expressed either as monolithic values or the combination of multi-faceted components. The keys are unique to ensure unambiguous association to them with the information values that belong with a given key value.

The associated information values may be simple values or richly-structured values, or may not exist at all for a given key value. Simply the enumeration of a set of unique values can be regarded as a useful key-value association without any values specified. It may be that users of a set of keys implicitly understand the concept associated with each key value without the need to explicitly reify the concept in any associated information values.

When moving information between key-value stores, or publishing the information found in a key-value store, an IT-enabled serialization format needs to accommodate the collection of information as a whole, the individual key values however composed, and the optional though potentially voluminous information item values associated with each key.

Supporting multiple ad hoc or colloquial expressions of key-value associations, whether IT-enabled in a data format or not IT-enabled in arbitrary and opaque publication formats such as PDF and HTML tables, can be a burden for publishers and consumers alike. Adopting a single standardized IT-enabled key-value serialization will promote easier publishing and easier ingestion of the pertinent aspects of a set of keys and their associated information values.

1.1.2 IT-enabled expression of coded domains

ISO/IEC 14662 *Open-edi reference model* provides standards for the inter-working of organizations through interconnected information technology (IT) systems. See Appendix F, *The Open-edi reference model perspective of code lists (Non-Normative)* for the relationship between the semantic and IT-enabling perspectives of business transactions in, respectively, the Business Operational View (BOV) and the Functional Services View (FSV).

ISO/IEC 15944-10 *IT-enabled coded domains as semantic components in business transactions* details important aspects of the BOV of a “set of codes representing X”, where X is a semantic umbrella concept scope and individual codes are distinct and separate semantic concepts within that scope. Simple umbrella semantic examples are days of the week, countries of the world, and financial currencies. Individual distinct semantic concepts are “Tuesday” and “Thursday”, “Australia” and “Tanzania”, and “the US dollar” and “the Euro”.

Some semantic concepts, such as days of the week, are accepted as given without the need for any governing Source Authority or coded domain Source Authority. Most other e-business semantic concepts involved in connecting the IT systems for the inter-working of organizations need such authoritative governance and managed publication of the semantics to be mutually understood by the expression of a code in a coded domain. Such code-level metadata could express linguistic or illustrative explication of the semantics of the given coded value unique key.

Colloquially, such collections of codes of a coded domain are referred to as *code lists*. The genericode specification satisfies the FSV perspective of code lists by providing an IT-enabled expression of coded domains for the direct interchange or open publishing of their content and semantics in a machine-readable syntax.

1.1.3 Genericode

Code lists, often regarded simply as enumerated values, have been with us since long before computers. They should be well understood and easily dealt with by now. Unfortunately, they are not. As is often the case, if you take a fundamentally simple concept, you find that everyone professes to understand it with complete clarity. When you look more closely, you find that everybody has their own unique view of what the problem is and how it should be solved.

If code lists were really so simple and obvious, there would already be a single, well-known and accepted way of handling them in XML. There is no such agreed solution, though. The problem is that while code lists are a well understood concept, people don't actually agree exactly on what code lists are, and how they should be used.

The OASIS Code List Representation format, "*genericode*"[1], is a single semantic model of code lists and accompanying XML serialization (supported by a W3C XML Schema) that can encode a broad range of code list information. The serialization is designed to IT-enable the 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.

1.1.4 What is a Code List? (Non-Normative)

What is a code list, then? Most people would agree that the following is a code list:

{"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"}

Example 1: Days of the week: English, uppercase

This is a perfectly reasonable set of alphabetic codes for representing days of the week. However, so is:

{"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"}

Example 2: Days of the week: English, mixed case

These two code lists are similar, but certainly not identical. That said, they can both be used to represent the days of the week. Of course, you could also use:

{"Dim", "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam"}

Example 3: Days of the week: French, mixed case

which is created from abbreviations for the days of the week in French. Then again, you could use:

{0, 1, 2, 3, 4, 5, 6}

Example 4: Days of the week: numeric

which is suitable as a computer representation, e.g. for a database column. On the other hand:

{"S", "M", "T", "W", "T", "F", "S"}

Example 5: Days of the week: English, single character

is not suitable as a code list for the days of the week, because the values are not unique.

Now suppose that you are using codes to represent days of the week in an application, and you are displaying the days of the week using 3-letter abbreviations in English or French. In that context,

[1]Genericode can be written starting either with an upper-case or lower-case "g". It depends whether genericode is at the start of the sentence or not.

should Example 2 and Example 3 be considered to be code lists, or should they be considered to be display values that would be keyed to either the Example 1 or Example 4 codes? The fact is, they could be either code lists or display values. A value which is a code in one context might only be an associated value for that code in another context. Nothing privileges any of these code lists over the others in terms of ability or suitability to be the code list (except the Example 5 values which are not suitable). There is a choice of code lists that can be used, and the answer to the question “which choice is the best?” depends on the needs of each particular situation.

What the above examples show is that for each *distinct entry* in a code list, there are many possible associated values (we use the term *distinct entry* to express the idea that we are talking a single item that needs to be represented in the code list, rather than about the code value(s) that can be used to identify that item). Some of those associated values are suitable for use in code lists, some are not. This leads to a tabular model, where each row of the table represents a conceptual code, and each column represents an associated value (code list metadata), as follows:

Numeric (key)	**English, upper-case (key)**	**English, mixed case (key)**	**French, mixed case (key)**	**English, single character**
0	SUN	Sun	Dim	S
1	MON	Mon	Lun	M
2	TUE	Tue	Mar	T
3	WED	Wed	Mer	W
4	THU	Thu	Jeu	T
5	FRI	Fri	Ven	F
6	SAT	Sat	Sam	S

Table 1: Days of the week

Notice that the first 4 of the 5 columns have been labeled as “*key*” columns. This means that the values in those columns can be used to uniquely identify the rows, and hence they can be used as code list values. The term *key* is used here similarly to a relational database table.

This is the most common case, where a single column can be used as a key. However, consider the following modification:

Numeric (key)	**English, uppercase (key)**	**English, single char-acter #1**	**English, single char-acter #2**
0	SUN	S	U
1	MON	M	O
2	TUE	T	U
3	WED	W	E
4	THU	T	H
5	FRI	F	R
6	SAT	S	A

Table 2: Days of the week, version 2

Here, the first two columns are each a key column. The last two columns are not individually key columns, but together they form a *compound key*, i.e. while the individual columns do not contain unique values, the pair of values is unique within each row. This is again similar to what happens in some relational databases, that a key for the rows need not be constructed from a single column, but instead may be constructed by combining two or more columns.

Finally, there is no reason why a column should only contain simple values like strings or numbers. A column could also contain a complex compound group of data, such as a fragment of XML:

Numeric (key)	English, uppercase (key)	XHTML
0	SUN	Sunday
1	MON	Monday
2	TUE	Tuesday
3	WED	Wednesday
4	THU	Thursday
5	FRI	Friday
6	SAT	Saturday

Table 3: Days of the week, version 3

Notice that the final XHTML column is not marked as a key column. The values are unique, so it certainly could be used as a key column. However, sometimes you may not wish to mark a column as a key column, even if the values are unique. The values in the column may not make particularly suitable keys. They might be too long to process quickly and conveniently, or they might not be able to be used in a particular context, such as for an XML attribute value. Also, it may be that while the values in a particular column are unique now, there is no guarantee or expectation that they will remain unique as the code list grows or changes in the future.

Once you see the tabular nature that underlies the information that can be associated with code lists, it becomes clear why they can be a source of so much debate. Different users need different subsets of the code list information, and people often assume that the information they need is all the information that anyone needs.

That kind of thinking doesn't work well with code lists, because code lists are sufficiently generic a concept that they are used across messages/documents, applications, and databases. The code list details that you need for the XML schemas often will not be exactly the same as the details that you need for your database or your application. If the code list information cannot be shared easily across these different areas, the result is duplication of effort and potential loss of synchronization between different implementations of the same code list.

The XML schema may only require a set of 3-letter codes to represent the code list. The database may require a set of numeric codes, plus display labels (possibly in different languages). The application may need to know which 3-letter code corresponds to which numeric code, so that it can process the XML and update the database. Also, some information related to a code list might not be appropriate for the XML format. For example, if you have a different image file for each code, it isn't ideal to include this image inline in the code list XML, since it vastly increases the size of the XML, and makes it more difficult to read. So in an XML representation, you are more likely to include some reference (e.g. a URL) to the image. For a database, however, it may be feasible to store the image in a BLOB[2] column in a database.

One last piece of experience from databases is that support for undefined values will be required. Sometimes users will have values that need to be associated with some of the codes in a code list, but won't have values to associate with every code. In that case, the concept of a undefined (nil or null) value is needed.

[2]Binary Large OBject.

1.2 Terminology

1.2.1 Terms and Definitions

coded domain Source Authority (cdSA) , noun

Person, usually an organization, as a Source Authority which sets the rules governing a coded domain

[ISO/IEC 15944-2:2006, 3.14]

Source Authority (SA) , noun

Person recognized by other Persons as the authoritative source for a set of constraints

[ISO/IEC 15944-2:2006, 3.109]

1.3 Normative References

There are no normative references.

1.4 Non-normative References

[**Open-edi**] *ISO/IEC 14662:2010 Information technology - Open-edi reference model*

2 Genericode Model

2.1 Tabular Structure

Genericode has 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*. A *required column* does not allow any row to have an undefined (nil or null) value. An *optional column* allows undefined values.

A genericode *key* is a set of one or more required columns that together uniquely identify each distinct entry in the code list. Optional columns cannot be used for keys. Each code list must have at least one key. Genericode keys are equivalent to what people usually mean when they talk about the "codes" in a code list. However, genericode allows multiple keys for each code list, and there is no single *preferred* key. For code lists that have multiple keys, it is assumed that the choice of which key to use is a *late binding* choice that is specific to the application, technology and/or context in which the code list is used.

2.2 Genericode Document Types

2.2.1 Summary of Document Types

There are 3 kinds of genericode documents, all supported by the one W3C XML Schema:

- · Column Set documents;
- · Code List documents;
- · Code List Set documents.

2.2.2 Column Set Documents

A column set document has the root element `<gc:ColumnSet>`. It contain definitions of genericode columns or keys that can be imported into code list documents or into other column set documents.

2.2.3 Code List Documents

A code list document has the root element `<gc:CodeList>`. It contains metadata describing the code list as a whole, as well as explicit code list data – codes and associated values.

2.2.4 Code List Set Documents

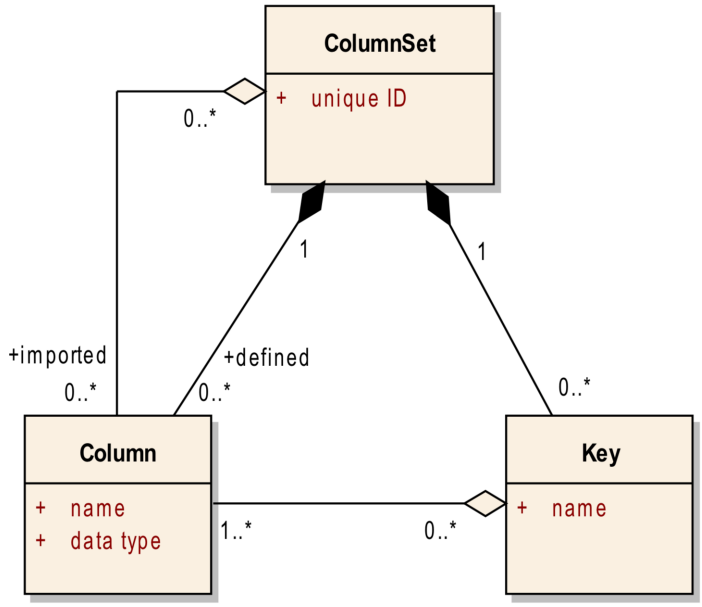
A code list set document has the root element `<gc:CodeListSet>`. It contains references to particular versions of code lists, and can also contain version-independent references to code lists. A code list set document can be used to define a particular *configuration* of versions of code lists that are used by a project, application, standard, etc.

2.3 Column Sets – Columns and Keys

A column set is a set of definitions of genericode columns and/or keys. A column defines a particular metadata value that can be defined for each distinct entry in a code list. A key defines a set of one or more columns.

It is not necessary to use separate column set documents. A genericode code list document can contain all of the required column and key definitions. Column set documents are provided as a convenience mechanism for sharing column and/or key definitions between multiple code lists.

Figure 1. UML relationships of keys and columns



This figure is in UML notation. Each column set must have a unique ID. For a column set defined within a code list document, the code list document's unique identifier is used. A column set can define any number of columns. It can also reference any number of columns from other column sets (in column set documents or code list documents). A column set can also define any number of keys. Each key is defined by one or more of the columns in the column set (either defined or imported). Keys are used to uniquely identify the rows (distinct entries) of code lists. Columns and keys are uniquely named within the column set that defines them, and each can also be uniquely identified using a specific URI if required additionally.

The matching genericode W3C XML Schema (WXS) representation of column set content is:

This figure is in XML Spy® notation. A default datatype library URI can be provided to identify which datatype library should be used for columns which do not explicitly specify a datatype library. If this URI is not provided, the datatype library defaults to the W3C XML Schema (WXS) datatype library.

A column set definition contains optional user annotation information (Annotation), and then identification and location information (Identification). A column set has a short name, any number of long names and a version.

A column set is uniquely identified by a canonical URI. Particular versions of the column set are uniquely identified by a canonical version URI. Location URIs can also be provided to suggest URLs from which an XML genericode column set instance may be retrieved (at the discretion of an application). *Alternative location URIs* can be provided to suggest URLs from which non-genericode representations of the column set can be retrieved. Canonical URIs and canonical version URIs must not be used as *de facto* location URIs for retrieving column set instances (nor anything else). The column set definition can also list the details of the agency which is responsible for publishing and/or maintaining the column set information.

A column definition (Column) contains a unique ID for the column and its use (required or optional). It also contains a short name (token) for the column, any number of long names, and optional extra canonical identification URIs. The datatype information for the column is contained in its Data element.

The Data structure is based on the data element in RELAX NG. The datatype is specified as a Type from a DatatypeLibrary. If the datatype library is not specified, it is inherited from the DatatypeLibrary

attribute of the enclosing column set definition. It otherwise defaults to the W3C XML Schema (WXS) datatype library.

If the data is XML (complex valued), the DataTypeLibrary is set to the namespace URI for the XML (or to "*" if any namespace[3] is allowed), and the Type is set to the root element name for the XML data (or to "*" if any root element is allowed).

Data definitions can contain Parameter elements which define facets that refine the datatype. When using the WXS datatype library, these are just the usual WXS datatype facets.

If a column is defined in an external column set or code list document, it is referenced using a ColumnRef. The column reference must have an ID just as a column definition would, but it also has an ExternalRef which contains the column's ID in the external document. The external column set or code list is identified by a CanonicalVersionUri and/or by any LocationUri information that is provided.

A key definition (Key) contains an ID for the key. It also contains a short name (token) for the key, any number of long names, and optional extra canonical identification URIs. The columns which together form the key are referenced using one or more ColumnRef elements. The Ref attribute of each contains the ID of either a Column or ColumnRef in the column set. Only required (not optional) columns may be used within a key (note that this rule is not able to be enforced using the genericode WXS Schema alone).

If a key is defined in an external column set or code list document, it can be referred to using a KeyRef. The key reference must have an ID, and also has an ExternalRef which contains the key's ID in the external document. The external column set or code list is identified by a CanonicalVersionUri and/or by any LocationUri information that is provided.

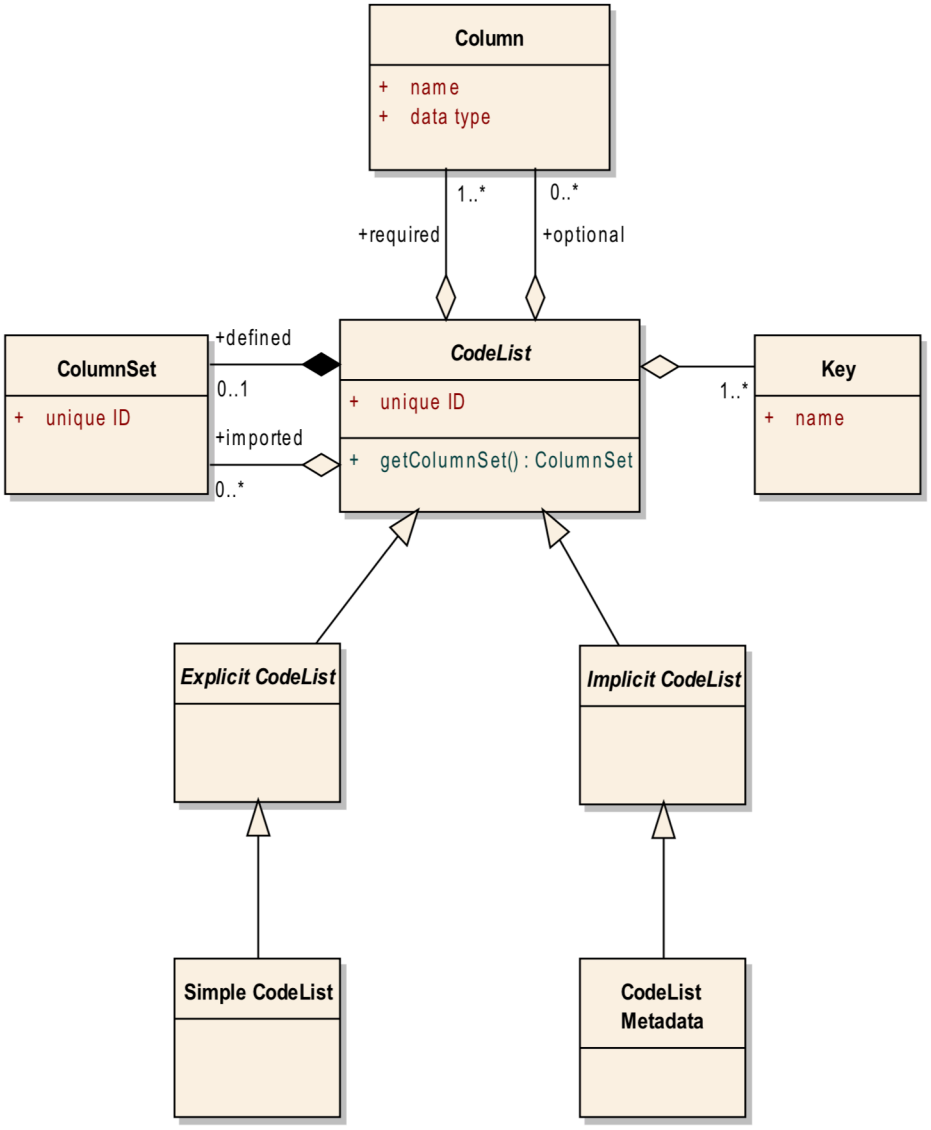
2.4 Code lists

A code list can contain its own embedded column set definition. It can also import columns and keys from any number of external column sets (in column set documents and/or code list documents). In the simplest case, what a code list provides is information (metadata) about the code list and (optionally) a set of rows, where each row defines a *distinct entry* in the code list.

A code list document that contains only information (metadata) about the code list as a whole is known as a *CodeList Metadata* document. If the code list document defines (zero or more) row, it is a *Simple CodeList*. These are the only kinds of code list that are supported in this version of the specification.

[3]Any namespace except the genericode namespace.

Figure 2. UML for code lists

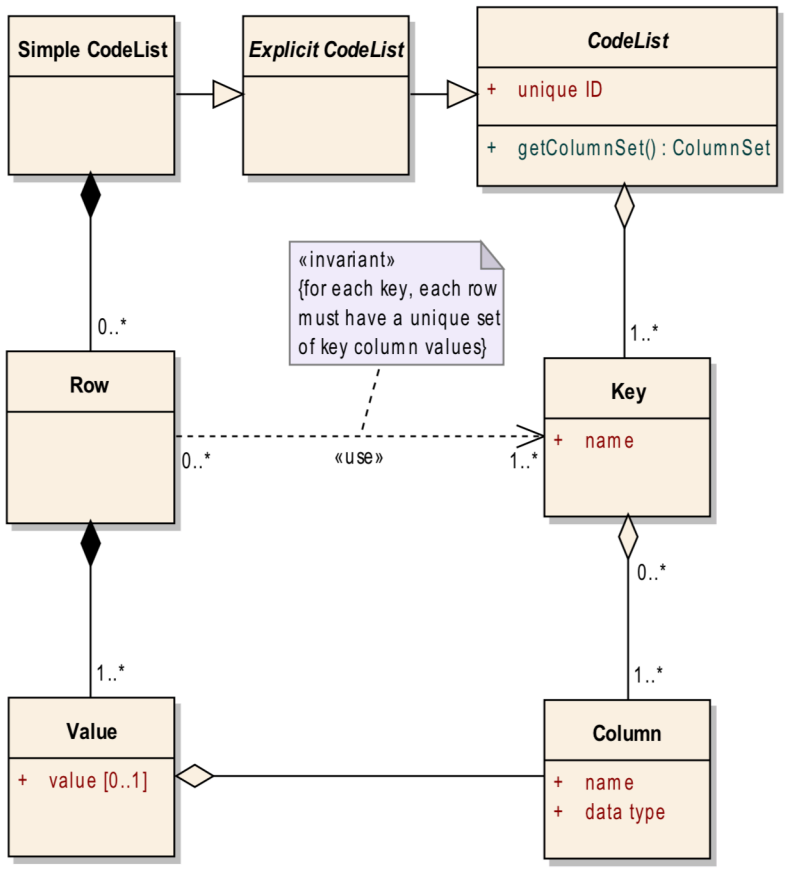


There is an important difference between a *CodeList Metadata* document and *Simple CodeList* that contains zero rows (zero distinct entries). The former does not provide information on how many *distinct entries* are contained in the code list. The latter explicitly indicates that a particular version of the code list contains zero distinct entries, i.e. the particular version of the code list is empty. A CodeList Metadata document does not provide any indication about whether a code list is empty or not.

A *CodeList Metadata* document is a special case of a *Simple CodeList* document. The differences will be discussed explicitly where appropriate.

A *Simple CodeList* is modeled as follows:

Figure 3. UML for simple code lists



A *Simple CodeList* contains zero or more rows (it is necessary to support empty code lists to allow for code lists that are empty now, but will be populated in future versions). Each *Row* defines a single *distinct entry* in the code list.

A *Row* contains one or more *Values*, where each of those values corresponds to a distinct column in the code list. At least one value is required, because a code list has to have at least one key, and each key requires at least one column. As a consequence, a *Row* must have at least one *Value*. Additionally, a *Row* must contain a defined *Value* for each of the *required columns* in the code list, i.e. for those columns for which a *Value* must be defined (non-null) for each *Row* (distinct entry) in the code list.

Each *Value* is associated with a single distinct column of the code list. For each *Key* in the code list, the values associated with the columns for that key must form a unique set, i.e. no two rows are allowed to have the same set of values for the same key columns. Note that this uniqueness requirement cannot be enforced using the genericode WXS Schema for code list documents, which is structured as follows:

Many of these elements and types have appeared already in section Section 2.3, “Column Sets – Columns and Keys”, so the explanations will not be repeated here. A code list document can either define its own embedded ColumnSet, or refer to an externally defined column set using a ColumnSetRef.

A ColumnSetRef contains the canonical version URI which uniquely identifies a referenced column set or code list document which contains the column set. It can also contain suggested URLs from which to retrieve the column set or code list. Canonical version URIs must not be used as *de facto* location URIs for retrieving column set instances (nor anything else).

A code list document that contains a SimpleCodeList element is a Simple CodeList. If the code list document does not contain a SimpleCodeList element, then it is a CodeList Metadata document.

The genericode WXS Schema representation of a SimpleCodeList is

A SimpleCodeList contains zero or more Row elements. Each Row contains one or more Value elements.

The Value container element is needed to allow optional user annotations of individual values in the code list. It has a ColumnRef attribute which contains the unique document ID of the associated column. A Value element can contains either a SimpleValue containing a textual value, or a ComplexValue containing a balanced (well-formed) XML fragment from a namespace other than the genericode namespace.

If a `Value` element does not contain either a `SimpleValue` element or a `ComplexValue` element, then the value is undefined. Only *optional columns* are allowed to have undefined values. Also, if a `Row` element does not contain a `Value` element corresponding to a particular column, then the row's value for that column is undefined.

Note that the ColumnRef attribute of a Value is optional. If it is not provided, it is assumed that the column is the one which follows the column associated with the previous value in the row. If the first Value in a Row does not have a ColumnRef, it is assumed to be associated with the first column in the column set. It is an error if a row contains more than one value for the same column, or if it does not contain a value for a required column.

The genericode WXS Schema is not able to validate that the contents of a Value match the datatype of the associated column. Other validation mechanisms should be used to perform datatype validation.

2.5 Code list sets

A CodeList Set lists a configuration of code lists and/or codelist versions. CodeList Sets can be used to provide lists of the code lists or code list versions that are associated with a particular version of an application or specification. The genericode WXS Schema structure for CodeList Set documents is:

Many of these elements and types have appeared already in Section 2.3, "Column Sets – Columns and Keys", so the explanations will not be repeated here. A code list set document contains a series of zero or more CodeListRef, CodeListSet, or CodeListSetRef elements.

A CodeListRef is a reference to a code list or to a version of a code list. If the CanonicalVersionUri is defined, then the LocationUri elements (if any) contain retrieval URIs for genericode CodeList documents. If the CanonicalVersionUri is not defined, then the LocationUri elements (if any) contain retrieval URIs for genericode CodeList Metadata documents. Note that canonical URIs and canonical version URIs must not be used as *de facto* location URIs for retrieving code list instances (nor anything else).

A CodeListSet element is used to define an embedded code list set within a larger code list set document. It allows a single *CodeList Set* document to carry information on multiple code list sets. Each embedded `CodeListSet` element has the same structure as a `CodeListSet` document.

A CodeListSetRef is a reference to a code list set or to a version of a code list set. If the CanonicalVersionUri is defined, then the LocationUri elements (if any) contain retrieval URIs for genericode CodeList Set documents. If the CanonicalVersionUri is not defined, then the LocationUri elements (if any) contain retrieval URIs for genericode CodeList Set Metadata documents. Just as for code list references, canonical URIs and canonical version URIs must not be used as *de facto* location URIs for retrieving code list instances (nor anything else).

A *CodeList Set* does not contain definitions of code lists, it only refers to the code list and code list versions which are a part of the particular version of the *CodeList Set*. It should also be noted that a code list set may contain a reference to a code list or code list set without specifying a particular version of the code list or code list set, and it may contain a reference to a code list or code list version or code list set or code list set version without specifying a location for retrieving a genericode

definition of that code list (metadata) or code list version or code list set (metadata) or code list set version. This is to support situations where

- · the code list definition or code list set definition is known to the users, and no location needs to be published. This may be because users have an application which maps the canonical URI or canonical version URI to a local definition;
- · the code list or code list set is sufficiently well-known (e.g. ISO 3-letter country codes) that users only need to have it uniquely identified, and do not need to have it enumerated or defined for them.

2.6 Namespaces

The genericode Schema makes use of two namespace URIs. The "`gc`" XML prefix refers to the namespace URI

http://docs.oasis-open.org/codelist/ns/genericode/1.0/

which is the main genericode namespace URI. The "rule" XML prefix refers to the namespace URI

http://docs.oasis-open.org/codelist/ns/rule/1.0/

which is used in the identification of auxiliary rules in the Schema. These are rules that cannot be enforced using the XML Schema itself; they appear in the Schema in `<rule:text>` elements within `<xsd:documentation>` elements.

3 Genericode XML Serialization

3.1 Schema Version and Namespace

Schema version: 1.0

Target namespace: http://docs.oasis-open.org/codelist/ns/genericode/1.0/

3.2 Notation

Multiplicity	Minimum Occurrence	Maximum Occurrence
1	1	1
?	0	1
+	1	unbounded
*	0	unbounded
{m,n}	m	n
{m,}	m	unbounded

ANY – any element in any namespace.

ANY[##other] – any element in any namespace other than the target namespace of the genericode XML Schema.

(A , B , C , ...) – sequence of items (A, B, C, etc.).

(A | B | C | ...) – choice between items (A, B, C, etc.).

3.3 Table of Schema Definitions

3.3.1 Global Elements

· Section 3.4.5, “CodeList (Element)”

· Section 3.4.8, “CodeListSet (Element)”

· Section 3.4.16, “ColumnSet (Element)”

3.3.2 Global Complex Types

· Section 3.4.1, “Agency (Complex Type)”

· Section 3.4.2, “Annotation (Complex Type)”

· Section 3.4.3, “AnyOtherContent (Complex Type)”

· Section 3.4.4, “AnyOtherLanguageContent (Complex Type)”

· Section 3.4.6, “CodeListDocument (Complex Type)”

· Section 3.4.7, “CodeListRef (Complex Type)”

· Section 3.4.10, “CodeListSetDocument (Complex Type)”

· Section 3.4.11, “CodeListSetRef (Complex Type)”

· Section 3.4.12, “Column (Complex Type)”

· Section 3.4.14, “ColumnRef (Complex Type)”

· Section 3.4.17, “ColumnSet (Complex Type)”

· Section 3.4.20, “ColumnSetDocument (Complex Type)”

· Section 3.4.21, “ColumnSetRef (Complex Type)”

· Section 3.4.22, “Data (Complex Type)”

· Section 3.4.23, “DataRestrictions (Complex Type)”

· Section 3.4.24, “DatatypeFacet (Complex Type)”

· Section 3.4.28, “GeneralIdentifier (Complex Type)”

· Section 3.4.30, “Identification (Complex Type)”

· Section 3.4.33, “Key (Complex Type)”

· Section 3.4.35, “KeyColumnRef (Complex Type)”

· Section 3.4.36, “KeyRef (Complex Type)”

· Section 3.4.38, “LongName (Complex Type)”

· Section 3.4.39, “MimeTypedUri (Complex Type)”

· Section 3.4.44, “Row (Complex Type)”

· Section 3.4.45, “ShortName (Complex Type)”

· Section 3.4.46, “SimpleCodeList (Complex Type)”

· Section 3.4.48, “SimpleValue (Complex Type)”

· Section 3.4.50, “Value (Complex Type)”

3.3.3 Global Simple Types

· Section 3.4.49, “UseType (Simple Type)”

3.3.4 Global Model Groups

· Section 3.4.9, “CodeListSetChoice (Model Group)”

· Section 3.4.13, “ColumnChoice (Model Group)”

· Section 3.4.18, “ColumnSetChoice (Model Group)”

· Section 3.4.19, “ColumnSetContent (Model Group)”

· Section 3.4.26, “DocumentHeader (Model Group)”

· Section 3.4.31, “IdentificationRefUriSet (Model Group)”

· Section 3.4.32, “IdentificationVersionUriSet (Model Group)”

· Section 3.4.34, “KeyChoice (Model Group)”

· Section 3.4.40, “NameSet (Model Group)”

· Section 3.4.42, “OuterCodeListChoice (Model Group)”

· Section 3.4.47, “SimpleCodeListSequence (Model Group)”

· Section 3.4.51, “ValueChoice (Model Group)”

· Section 3.4.53, “VersionLocationUriSet (Model Group)”

3.3.5 Global Attribute Groups

· Section 3.4.15, “ColumnReference (Attribute Group)”

· Section 3.4.25, “DefaultDatatypeLibrary (Attribute Group)”

· Section 3.4.27, “ExternalReference (Attribute Group)”

· Section 3.4.29, “IdDefinition (Attribute Group)”

· Section 3.4.37, “Language (Attribute Group)”

· Section 3.4.41, “OptionalUseDefinition (Attribute Group)”

· Section 3.4.43, “RequiredUseDefinition (Attribute Group)”

· Section 3.4.52, “ValueIdentification (Attribute Group)”

3.4 Global Schema Definitions in Alphabetic Order

3.4.1 Agency (Complex Type)

Details of an agency which produces code lists or related artifacts.

Content Model: (ShortName? , LongName* , Identifier*)

Mixed Content: No

Elements:

Element	Type	Description
ShortName	Section 3.4.45, “ShortName (Complex Type)”	Short name (without whitespace) for the agency.
LongName	Section 3.4.38, “LongName (Complex Type)”	Human-readable name for the agency.
Identifier	Section 3.4.28, “GeneralIdentifier (Complex Type)”	Identifier for the agency.

3.4.2 Annotation (Complex Type)

User annotation information.

Content Model: (Description* , AppInfo?)

Mixed Content: No

Elements:

Element	Type	Description
Description	Section 3.4.4, “AnyOtherLanguageContent (Complex Type)”	Human-readable information.
AppInfo	Section 3.4.3, “AnyOtherContent (Complex Type)”	Machine-readable information.

3.4.3 AnyOtherContent (Complex Type)

Container for any XML content which is in a different namespace to the Schema’s target namespace.

Content Model: (ANY[##other]*)

Mixed Content: No

3.4.4 AnyOtherLanguageContent (Complex Type)

Container for any human-readable XML content which is in a different namespace to the Schema’s target namespace.

Extension of: Section 3.4.3, “AnyOtherContent (Complex Type)”

Content Model: (ANY[##other]*)

Mixed Content: No

Attributes:

Attribute	Usage	Type	Description
xml:lang	optional		Language for the human-readable XML content.

3.4.5 CodeList (Element)

Top-level (root) element for a genericode code list definition.

A code list definition defines the details of a particular (version of a) code list.

Complex Type: Section 3.4.6, “CodeListDocument (Complex Type)”

3.4.6 CodeListDocument (Complex Type)

Document type for genericode code list definitions.

Rules:

Rule 1 [document] :
A code list must have at least one key, unless it is a metadata-only definition without a 'SimpleCodeList' element.

Content Model: ((Annotation? , Identification) , (ColumnSet | ColumnSetRef) , ((SimpleCodeList))?)

Mixed Content: No

Elements:

Element	Type	Description
Annotation (from Section 3.4.26, “DocumentHeader (Model Group)”)	Section 3.4.2, “Annotation (Complex Type)”	User annotation information. **DocumentHeader:** General information (metadata) for the code list.
Identification (from Section 3.4.26, “DocumentHeader (Model Group)”)	Section 3.4.30, “Identification (Complex Type)”	Identification and location information (metadata). **DocumentHeader:** General information (metadata) for the code list.
ColumnSet (from Section 3.4.18, “ColumnSetChoice (Model Group)”)	Section 3.4.17, “ColumnSet (Complex Type)”	Definition of a column set (columns and keys for the code list). **ColumnSetChoice:** A choice between a column set definition and a column set reference.
ColumnSetRef (from Section 3.4.18, “ColumnSetChoice (Model Group)”)	Section 3.4.21, “ColumnSetRef (Complex Type)”	Reference to a column set defined in an external column set or code list document. **ColumnSetChoice:** A choice between a column set definition and a column set reference.
SimpleCodeList (from Section 3.4.47, “SimpleCodeListSequence (Model Group)”)	Section 3.4.46, “SimpleCodeList (Complex Type)”	Simple (explicit) code list definition. **SimpleCodeListSequence:** Details of a simple code list definition. **OuterCodeListChoice:** The only choice is a simple (explicit) code list definition. Not used if the code list definition contains code list metadata only.

Attributes:

Attribute	Usage	Type	Description
xml:base	optional		Base URL which applies to relative location URIs. **Rules:**

Attribute	Usage	Type	Description
			Rule 2 [application] : xml:base does not apply to canonical URIs.

3.4.7 CodeListRef (Complex Type)

Reference to a code list, possibly defined in an external document.

Rules:

Rule 3 [application] :

The code list reference must be valid.

An application may use the CanonicalVersionUri to select a local copy of the code list.

If there is no CanonicalVersionUri, the CanonicalUri may be used to select a local copy of the code list.

Otherwise the LocationUri value(s) may be tried in order, until a valid code list document is retrieved.

An application must signal an error to the user if it is not able to retrieve a code list document to match the code list reference.

Content Model: (Annotation? , CanonicalUri , CanonicalVersionUri? , LocationUri*)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the referenced code list.
CanonicalUri	xsd:anyURI	Canonical URI which uniquely identifies all versions (collectively) of the referenced code list. **Rules:** **Rule 4 [document] :** Must be an absolute URI, must not be relative. **Rule 5 [application] :** Must not be used as a de facto location URI.
CanonicalVersionUri	xsd:anyURI	Canonical URI which uniquely identifies a specific version of the referenced code list. **Rules:**

Element	Type	Description
		Rule 6 [document] : Must be an absolute URI, must not be relative. **Rule 7 [application] :** Must not be used as a de facto location URI.
LocationUri	xsd:anyURI	Suggested retrieval location for this code list, in genericode format. **Rules:** **Rule 8 [application] :** If the CanonicalVersionUri has been defined, the LocationUri must reference a genericode CodeList document. If the CanonicalVersionUri is undefined, the LocationUri must reference a genericode CodeList Metadata document. An application must signal an error to the user if a LocationUri does not reference the appropriate type of genericode document. **Rule 9 [application] :** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Attributes:

Attribute	Usage	Type	Description
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 10 [application] :** xml:base does not apply to canonical URIs.

3.4.8 CodeListSet (Element)

Top-level element for the definition of a code list set.

Complex Type: Section 3.4.10, “CodeListSetDocument (Complex Type)”

3.4.9 CodeListSetChoice (Model Group)

A choice between a code list reference, an inline code list set, or a code list set reference.

Content Model: (CodeListRef | CodeListSet | CodeListSetRef)

Elements:

Element	Type	Description
CodeListRef	Section 3.4.7, “CodeListRef (Complex Type)”	
CodeListSet	Section 3.4.10, “CodeListSetDocument (Complex Type)”	
CodeListSetRef	Section 3.4.11, “CodeListSetRef (Complex Type)”	

3.4.10 CodeListSetDocument (Complex Type)

Document type for the definition of a set of code lists.

Content Model: ((Annotation? , Identification) , (CodeListRef | CodeListSet | CodeListSetRef)*)

Mixed Content: No

Elements:

Element	Type	Description
Annotation (from Section 3.4.26, “DocumentHeader (Model Group)”)	Section 3.4.2, “Annotation (Complex Type)”	User annotation information. **DocumentHeader:** General document information for the code list set.
Identification (from Section 3.4.26, “DocumentHeader (Model Group)”)	Section 3.4.30, “Identification (Complex Type)”	Identification and location information (metadata). **DocumentHeader:** General document information for the code list set.
CodeListRef (from Section 3.4.9, “CodeListSetChoice (Model Group)”)	Section 3.4.7, “CodeListRef (Complex Type)”	**CodeListSetChoice:** Contents of the code list set. If the code list set does not have any contents, it is a CodeListSet Metadata definition.
CodeListSet (from Section 3.4.9, “CodeListSetChoice (Model Group)”)	Section 3.4.10, “CodeListSetDocument (Complex Type)”	**CodeListSetChoice:** Contents of the code list set. If the code list set does not have

Element	Type	Description
		any contents, it is a CodeListSet Metadata definition.
CodeListSetRef (from Section 3.4.9, "CodeListSetChoice (Model Group)")	Section 3.4.11, "CodeListSetRef (Complex Type)"	**CodeListSetChoice:** Contents of the code list set. If the code list set does not have any contents, it is a CodeListSet Metadata definition.

Attributes:

Attribute	Usage	Type	Description
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 11 [application] :** xml:base does not apply to canonical URIs.

3.4.11 CodeListSetRef (Complex Type)

Reference to a code list set, possibly defined in an external document.

Rules:

Rule 47 [application] :

The code list set reference must be valid.

An application may use the CanonicalVersionUri to select a local copy of the code list set.

If there is no CanonicalVersionUri, the CanonicalUri may be used to select a local copy of the code list set.

Otherwise the LocationUri value(s) may be tried in order, until a valid code list set document is retrieved.

An application must signal an error to the user if it is not able to retrieve a code list set document to match the code list set reference.

Content Model: (Annotation? , CanonicalUri , CanonicalVersionUri? , LocationUri*)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, "Annotation (Complex Type)"	User annotation for the referenced code list set.
CanonicalUri	xsd:anyURI	Canonical URI which uniquely identifies all versions (collective-

Element	Type	Description
		ly) of the referenced code list set. **Rules:** **Rule 48 [document] :** Must be an absolute URI, must not be relative. **Rule 49 [application] :** Must not be used as a de facto location URI.
CanonicalVersionUri	xsd:anyURI	Canonical URI which uniquely identifies a specific version of the referenced code list set. **Rules:** **Rule 50 [document] :** Must be an absolute URI, must not be relative. **Rule 51 [application] :** Must not be used as a de facto location URI.
LocationUri	xsd:anyURI	Suggested retrieval location for this code list set, in genericode format. **Rules:** **Rule 52 [application] :** If the CanonicalVersionUri has been defined, the LocationUri must reference a genericode CodeListSet document. If the CanonicalVersionUri is undefined, the LocationUri must reference a genericode CodeListSet Metadata document. An application must signal an error to the user if a LocationUri does not reference the appropriate type of genericode document. **Rule 53 [application] :**

Element	Type	Description
		An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Attributes:

Attribute	Usage	Type	Description
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 54 [application] :** xml:base does not apply to canonical URIs.

3.4.12 Column (Complex Type)

Definition of a column.

Each column of a code list defines a piece of metadata that can be specified for each item in the code list.

Content Model: (Annotation? , (ShortName , LongName*) , (CanonicalUri , CanonicalVersionUri?)? , Data)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User information about the column.
ShortName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.45, “ShortName (Complex Type)”	Short name (without whitespace). **NameSet:** Name(s) of the column.
LongName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.38, “LongName (Complex Type)”	Human-readable name. **NameSet:** Name(s) of the column.
CanonicalUri (from Section 3.4.32, “IdentificationVersionUriSet (Model Group)”)	xsd:anyURI	Canonical URI which uniquely identifies all versions collectively. **Rules:** Must be an absolute URI, must not be relative.

Element	Type	Description
		Must not be used as a de facto location URI. **IdentificationVersionUriSet:** URIs used to identify the column and/or the version of the column.
CanonicalVersionUri (from Section 3.4.32, "IdentificationVersionUriSet (Model Group)")	xsd:anyURI	Canonical URI which uniquely identifies this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationVersionUriSet:** URIs used to identify the column and/or the version of the column.
Data	Section 3.4.22, "Data (Complex Type)"	Data type of the column.

Attributes:

Attribute	Usage	Type	Description
Id (from Section 3.4.29, "IdDefinition (Attribute Group)")	required	xsd:ID	Unique ID within the document. **IdDefinition:** ID which identifies the column within the document.
Use (from Section 3.4.43, "RequiredUseDefinition (Attribute Group)")	required	Section 3.4.49, "UseType (Simple Type)"	Whether the usage is required or optional. **RequiredUseDefinition:** Whether the column is required or optional.

3.4.13 ColumnChoice (Model Group)

A choice between a column definition and a column reference.

Content Model: (Column | ColumnRef)

Elements:

Element	Type	Description
Column	Section 3.4.12, "Column (Complex Type)"	Definition of a column.

Element	Type	Description
ColumnRef	Section 3.4.14, “ColumnRef (Complex Type)”	Reference to a column defined in an external column set or code list.

3.4.14 ColumnRef (Complex Type)

Reference to a column defined in an external column set or code list.

Rules:

Rule 12 [application] :
The column reference must be valid. An application may use the CanonicalVersionUri to select a local copy of the code list or column set which contains the column definition. Otherwise the LocationUri value(s) may be tried in order, until a valid code list or column set document (containing the necessary column definition) is retrieved. An application must signal an error to the user if it is not able to retrieve a code list or column set document which contains the necessary column definition.

Content Model: (Annotation? , (CanonicalVersionUri , LocationUri*) , Data?)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the referenced column.
CanonicalVersionUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Canonical URI which serves as a unique identifier for this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationRefUriSet:** Identification of the external column set or code list document which contains the column set definition.
LocationUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:**

Element	Type	Description
		An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format. **IdentificationRefUriSet:** Identification of the external column set or code list document which contains the column set definition.
Data	Section 3.4.23, “DataRestrictions (Complex Type)”	Restrictions to the data type of the referenced column.

Attributes:

Attribute	Usage	Type	Description
Id (from Section 3.4.29, “IdDefinition (Attribute Group)”)	required	xsd:ID	Unique ID within the document. **IdDefinition:** ID which identifies the column within this document.
ExternalRef (from Section 3.4.27, “ExternalReference (Attribute Group)”)	required	xsd:NCName	Unique ID within the external document. **Rules:** The external reference must not be prefixed with a ‘#’ symbol. **ExternalReference:** ID which identifies which identifies the column within the external column set or code list.
Use (from Section 3.4.41, “OptionalUseDefinition (Attribute Group)”)	optional	Section 3.4.49, “UseType (Simple Type)”	Whether the usage is required or optional. **OptionalUseDefinition:** Whether the column is required or optional. **Rules:** **Rule 13 [application] :** If specified, this overrides the usage specified in the external col-

Attribute	Usage	Type	Description
			umn set or code list document.
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 14 [application] :** xml:base does not apply to canonical URIs.

3.4.15 ColumnReference (Attribute Group)

Attribute set for referring to a column definition.

Attributes:

Attribute	Usage	Type	Description
ColumnRef	optional	xsd:IDREF	Reference to a column ID in the document.

3.4.16 ColumnSet (Element)

Top-level element for the definition of a column set.

Complex Type: Section 3.4.20, “ColumnSetDocument (Complex Type)”

3.4.17 ColumnSet (Complex Type)

Definition of a column set (columns and keys for a code list).

Content Model: ((Column | ColumnRef)* , (Key | KeyRef)*)

Mixed Content: No

Elements:

Element	Type	Description
Column (from Section 3.4.13, “ColumnChoice (Model Group)”)	Section 3.4.12, “Column (Complex Type)”	Definition of a column. **ColumnChoice:** A choice between a column definition and a column reference. **ColumnSetContent:** Column set definitions.
ColumnRef (from Section 3.4.13, “ColumnChoice (Model Group)”)	Section 3.4.14, “ColumnRef (Complex Type)”	Reference to a column defined in an external column set or code list.

Element	Type	Description
		ColumnChoice: A choice between a column definition and a column reference. **ColumnSetContent:** Column set definitions.
Key (from Section 3.4.34, “KeyChoice (Model Group)”)	Section 3.4.33, “Key (Complex Type)”	Definition of a key. **KeyChoice:** A choice between a key definition and a key reference. **ColumnSetContent:** Column set definitions.
KeyRef (from Section 3.4.34, “KeyChoice (Model Group)”)	Section 3.4.36, “KeyRef (Complex Type)”	Reference to a key defined in an external column set or code list. **KeyChoice:** A choice between a key definition and a key reference. **ColumnSetContent:** Column set definitions.

Attributes:

Attribute	Usage	Type	Description
DatatypeLibrary (from Section 3.4.25, “DefaultDatatypeLibrary (Attribute Group)”)	optional	xsd:anyURI	URI which uniquely identifies the default datatype library for the column set. If not provided, defaults to the URI for W3C XML Schema datatypes. **DefaultDatatypeLibrary:** Identification of the default datatype library for the column set.
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 15 [application] :**

Attribute	Usage	Type	Description
			xml:base does not apply to canonical URIs.

3.4.18 ColumnSetChoice (Model Group)

A choice between a column set definition and a column set reference.

Content Model: (ColumnSet | ColumnSetRef)

Elements:

Element	Type	Description
ColumnSet	Section 3.4.17, "ColumnSet (Complex Type)"	Definition of a column set (columns and keys for the code list).
ColumnSetRef	Section 3.4.21, "ColumnSetRef (Complex Type)"	Reference to a column set defined in an external column set or code list document.

3.4.19 ColumnSetContent (Model Group)

Specific details of a column set.

Content Model: ((Column | ColumnRef)* , (Key | KeyRef)*)

Elements:

Element	Type	Description
Column (from Section 3.4.13, "ColumnChoice (Model Group)")	Section 3.4.12, "Column (Complex Type)"	Definition of a column. **ColumnChoice:** A choice between a column definition and a column reference.
ColumnRef (from Section 3.4.13, "ColumnChoice (Model Group)")	Section 3.4.14, "ColumnRef (Complex Type)"	Reference to a column defined in an external column set or code list. **ColumnChoice:** A choice between a column definition and a column reference.
Key (from Section 3.4.34, "KeyChoice (Model Group)")	Section 3.4.33, "Key (Complex Type)"	Definition of a key. **KeyChoice:** A choice between a key definition and a key reference.
KeyRef (from Section 3.4.34, "KeyChoice (Model Group)")	Section 3.4.36, "KeyRef (Complex Type)"	Reference to a key defined in an external column set or code list. **KeyChoice:** A choice between a key definition and a key reference.

3.4.20 ColumnSetDocument (Complex Type)

Document type for the definition of a column set, which is a set of code list columns and/or keys.

Content Model: ((Annotation? , Identification) , ((Column | ColumnRef)* , (Key | KeyRef)*))

Mixed Content: No

Elements:

Element	Type	Description
Annotation (from Section 3.4.26, "DocumentHeader (Model Group)")	Section 3.4.2, "Annotation (Complex Type)"	User annotation information. **DocumentHeader:** General document information for the column set.
Identification (from Section 3.4.26, "DocumentHeader (Model Group)")	Section 3.4.30, "Identification (Complex Type)"	Identification and location information (metadata). **DocumentHeader:** General document information for the column set.
Column (from Section 3.4.13, "ColumnChoice (Model Group)")	Section 3.4.12, "Column (Complex Type)"	Definition of a column. **ColumnChoice:** A choice between a column definition and a column reference. **ColumnSetContent:** Details of the column set.
ColumnRef (from Section 3.4.13, "ColumnChoice (Model Group)")	Section 3.4.14, "ColumnRef (Complex Type)"	Reference to a column defined in an external column set or code list. **ColumnChoice:** A choice between a column definition and a column reference. **ColumnSetContent:** Details of the column set.
Key (from Section 3.4.34, "KeyChoice (Model Group)")	Section 3.4.33, "Key (Complex Type)"	Definition of a key. **KeyChoice:** A choice between a key definition and a key reference. **ColumnSetContent:** Details of the column set.
KeyRef (from Section 3.4.34, "KeyChoice (Model Group)")	Section 3.4.36, "KeyRef (Complex Type)"	Reference to a key defined in an external column set or code list.

Element	Type	Description
		KeyChoice: A choice between a key definition and a key reference. **ColumnSetContent:** Details of the column set.

Attributes:

Attribute	Usage	Type	Description
DatatypeLibrary (from Section 3.4.25, "DefaultDatatypeLibrary (Attribute Group)")	optional	xsd:anyURI	URI which uniquely identifies the default datatype library for the column set. If not provided, defaults to the URI for W3C XML Schema datatypes. **DefaultDatatypeLibrary:** Identification of the default datatype library for the column set.
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 16 [application] :** xml:base does not apply to canonical URIs.

3.4.21 ColumnSetRef (Complex Type)

Reference to a column set defined in an external column set or code list document.

Rules:

Rule 17 [application] :

The column set reference must be valid.

An application may use the CanonicalVersionUri to select a local copy of the column set or code list.

Otherwise the LocationUri value(s) may be tried in order, until a valid column set or code list document is retrieved.

An application must signal an error to the user if it is not able to retrieve a column set or code list document to match the column set reference.

Content Model: (Annotation? , (CanonicalVersionUri , LocationUri*))

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the referenced column set.
CanonicalVersionUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Canonical URI which serves as a unique identifier for this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationRefUriSet:** Identification of the external column set or code list document which contains the column set definition.
LocationUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format. **IdentificationRefUriSet:** Identification of the external column set or code list document which contains the column set definition.

Attributes:

Attribute	Usage	Type	Description
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 18 [application] :** xml:base does not apply to canonical URIs.

3.4.22 Data (Complex Type)

Data type definition.

Content Model: (Annotation? , Parameter*)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the datatype.
Parameter	Section 3.4.24, “DatatypeFacet (Complex Type)”	Facet parameter which refines the datatype.

Attributes:

Attribute	Usage	Type	Description
Type	required	xsd:token	Unique ID for the datatype within its datatype library. **Rules:** **Rule 19 [document] :** The datatype ID must not include a namespace prefix. For the W3C XML Schema datatypes, possible datatype IDs are 'string', 'token', 'boolean', 'decimal', etc. **Rule 20 [document] :** If the data is complex (i.e. XML), this value is set to the root element name for the XML value, or '*' if the root element name is not restricted.
DatatypeLibrary	optional	xsd:anyURI	URI which uniquely identifies the datatype library. **Rules:** **Rule 21 [application] :**

Attribute	Usage	Type	Description
			If this URI not explicitly provided, the datatype library for the enclosing column set is used. **Rule 22 [document] :** If the data is complex (i.e. XML), this value is set to the namespace URI for the XML, or '*' if the namespace URI is not restricted.
Lang (from Section 3.4.37, "Language (Attribute Group)")	optional	xsd:language	Language code which accepts the same values as 'xml:lang'. Unlike 'xml:lang', the scope of the language definition is not restricted to the XML content within the element where the 'lang' attribute appears. **Language:** Language from which the data is taken or derived.

3.4.23 DataRestrictions (Complex Type)

Restrictions to a data type.

Rules:

Rule 23 [document] :

The 'gc:lang' attribute may be specified only if no language is already set for the data type that is being restricted.

Content Model: (Parameter*)

Mixed Content: No

Elements:

Element	Type	Description
Parameter	Section 3.4.24, "DatatypeFacet (Complex Type)"	Facet parameter which refines the datatype.

Attributes:

Attribute	Usage	Type	Description
Lang (from Section 3.4.37, “Language (Attribute Group)”)	optional	xsd:language	Language code which accepts the same values as ‘xml:lang’. Unlike ‘xml:lang’, the scope of the language definition is not restricted to the XML content within the element where the ‘lang’ attribute appears. **Language:** Language from which the data is taken or derived.

3.4.24 DatatypeFacet (Complex Type)

Facet information for refining a datatype.

Extension of: xsd:string

Attributes:

Attribute	Usage	Type	Description
ShortName	required	xsd:token	Short name (token) for the datatype facet.
LongName	optional	xsd:normalizedString	Long name for the datatype facet.

3.4.25 DefaultDatatypeLibrary (Attribute Group)

Identification of the default datatype library for a column set.

Attributes:

Attribute	Usage	Type	Description
DatatypeLibrary	optional	xsd:anyURI	URI which uniquely identifies the default datatype library for the column set. If not provided, defaults to the URI for W3C XML Schema datatypes.

3.4.26 DocumentHeader (Model Group)

General document information (metadata).

Content Model: (Annotation? , Identification)

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation information.
Identification	Section 3.4.30, “Identification (Complex Type)”	Identification and location information (metadata).

3.4.27 ExternalReference (Attribute Group)

Attribute set used to identify a definition within an external document.

Attributes:

Attribute	Usage	Type	Description
ExternalRef	required	xsd:NCName	Unique ID within the external document. **Rules:** **Rule 24 [document] :** The external reference must not be prefixed with a '#' symbol.

3.4.28 GeneralIdentifier (Complex Type)

An identifier value. Typically not a long or short name.

Extension of: xsd:normalizedString

3.4.29 IdDefinition (Attribute Group)

Attribute set used to identify a definition within the document.

Attributes:

Attribute	Usage	Type	Description
Id	required	xsd:ID	Unique ID within the document.

3.4.30 Identification (Complex Type)

Identification and location information (metadata).

Content Model: ((ShortName , LongName*) , Version , CanonicalUri , (CanonicalVersionUri , LocationUri* , AlternateFormatLocationUri*) , Agency?)

Mixed Content: No

Elements:

Element	Type	Description
ShortName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.45, “ShortName (Complex Type)”	Short name (without white-space).

Element	Type	Description
		NameSet: Various names.
LongName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.38, “LongName (Complex Type)”	Human-readable name. **NameSet:** Various names.
Version	xsd:token	Version identifier.
CanonicalUri	xsd:anyURI	Canonical URI which uniquely identifies all versions (collectively). **Rules:** **Rule 25 [document] :** Must be an absolute URI, must not be relative. **Rule 26 [application] :** Must not be used as a de facto location URI.
CanonicalVersionUri (from Section 3.4.53, “VersionLocationUriSet (Model Group)”)	xsd:anyURI	Canonical URI which uniquely identifies this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **VersionLocationUriSet:** Identification and location URIs for the version.
LocationUri (from Section 3.4.53, “VersionLocationUriSet (Model Group)”)	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format. **VersionLocationUriSet:** Identification and location URIs for the version.

Element	Type	Description
AlternateFormatLocationUri (from Section 3.4.53, “VersionLocationUriSet (Model Group)”)	Section 3.4.39, “MimeTypedUri (Complex Type)”	Suggested retrieval location for this version, in a non-genericode format. Such alternative formats are intended only as additional renditions of the code list information, not as a replacements nor as alternatives for use in application processing. **VersionLocationUriSet:** Identification and location URIs for the version.
Agency	Section 3.4.1, “Agency (Complex Type)”	Agency that is responsible for publication and/or maintenance of the information.

3.4.31 IdentificationRefUriSet (Model Group)

Identification and location URIs.

Content Model: (CanonicalVersionUri , LocationUri*)

Elements:

Element	Type	Description
CanonicalVersionUri	xsd:anyURI	Canonical URI which serves as a unique identifier for this version. **Rules:** **Rule 27 [document] :** Must be an absolute URI, must not be relative. **Rule 28 [application] :** Must not be used as a de facto location URI.
LocationUri	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:** **Rule 29 [application] :** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

3.4.32 IdentificationVersionUriSet (Model Group)

URIs used as unique identifiers.

Content Model: (CanonicalUri , CanonicalVersionUri?)

Elements:

Element	Type	Description
CanonicalUri	xsd:anyURI	Canonical URI which uniquely identifies all versions collectively. **Rules:** **Rule 30 [document] :** Must be an absolute URI, must not be relative. **Rule 31 [application] :** Must not be used as a de facto location URI.
CanonicalVersionUri	xsd:anyURI	Canonical URI which uniquely identifies this version. **Rules:** **Rule 32 [document] :** Must be an absolute URI, must not be relative. **Rule 33 [application] :** Must not be used as a de facto location URI.

3.4.33 Key (Complex Type)

Definition of a key.

A key is a set of one or more columns whose values together provide a unique identification of each item in a code list.

Rules:

Rule 34 [document] :

Only required columns can be used for keys.

Content Model: (Annotation? , (ShortName , LongName*) , (CanonicalUri , CanonicalVersionUri?)? , ColumnRef+)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the key.
ShortName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.45, “ShortName (Complex Type)”	Short name (without white-space). **NameSet:** Name(s) of the key.
LongName (from Section 3.4.40, “NameSet (Model Group)”)	Section 3.4.38, “LongName (Complex Type)”	Human-readable name. **NameSet:** Name(s) of the key.
CanonicalUri (from Section 3.4.32, “IdentificationVersionUriSet (Model Group)”)	xsd:anyURI	Canonical URI which uniquely identifies all versions collectively. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationVersionUriSet:** URIs used to identify the key and/or the version of the key.
CanonicalVersionUri (from Section 3.4.32, “IdentificationVersionUriSet (Model Group)”)	xsd:anyURI	Canonical URI which uniquely identifies this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationVersionUriSet:** URIs used to identify the key and/or the version of the key.
ColumnRef	Section 3.4.35, “KeyColumnRef (Complex Type)”	Reference to the document ID of a column in the key.

Attributes:

Attribute	Usage	Type	Description
Id (from Section 3.4.29, “IdDefinition (Attribute Group)”)	required	xsd:ID	Unique ID within the document. **IdDefinition:**

Attribute	Usage	Type	Description
			ID which identifies the key within the document.

3.4.34 KeyChoice (Model Group)

A choice between a key definition and a key reference.

Content Model: (Key | KeyRef)

Elements:

Element	Type	Description
Key	Section 3.4.33, “Key (Complex Type)”	Definition of a key.
KeyRef	Section 3.4.36, “KeyRef (Complex Type)”	Reference to a key defined in an external column set or code list.

3.4.35 KeyColumnRef (Complex Type)

Reference to a column which forms part of a key.

Content Model: (Annotation?)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the column.

Attributes:

Attribute	Usage	Type	Description
Ref	required	xsd:IDREF	Reference to the ID of the column within the document.

3.4.36 KeyRef (Complex Type)

Reference to a key defined in an external column set or code list.

Rules:

Rule 35 [application] :
The key reference must be valid. An application may use the CanonicalVersionUri to select a local copy of the code list or column set which contains the key definition. Otherwise the LocationUri value(s) may be tried in order, until a valid code list or column set document (containing the necessary key definition) is retrieved.

An application must signal an error to the user if it is not able to retrieve a code list or column set document which contains the necessary key definition.

Content Model: (Annotation? , (CanonicalVersionUri , LocationUri*))

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the referenced key.
CanonicalVersionUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Canonical URI which serves as a unique identifier for this version. **Rules:** Must be an absolute URI, must not be relative. Must not be used as a de facto location URI. **IdentificationRefUriSet:** Identification of the external column set or code list which contains the key definition.
LocationUri (from Section 3.4.31, “IdentificationRefUriSet (Model Group)”)	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format. **IdentificationRefUriSet:** Identification of the external column set or code list which contains the key definition.

Attributes:

Attribute	Usage	Type	Description
Id (from Section 3.4.29, “IdDefinition (Attribute Group)”)	required	xsd:ID	Unique ID within the document. **IdDefinition:** ID which identifies the key within this document.

Attribute	Usage	Type	Description
ExternalRef (from Section 3.4.27, "ExternalReference (Attribute Group)")	required	xsd:NCName	Unique ID within the external document. **Rules:** The external reference must not be prefixed with a '#' symbol. **ExternalReference:** ID which identifies which identifies the key within the external column set or code list.
xml:base	optional		Base URL which applies to relative location URIs. **Rules:** **Rule 36 [application] :** xml:base does not apply to canonical URIs.

3.4.37 Language (Attribute Group)

Attributes which describe the language of a piece of text.

Attributes:

Attribute	Usage	Type	Description
Lang	optional	xsd:language	Language code which accepts the same values as 'xml:lang'. Unlike 'xml:lang', the scope of the language definition is not restricted to the XML content within the element where the 'lang' attribute appears.

3.4.38 LongName (Complex Type)

A human-readable name.

Extension of: xsd:normalizedString

3.4.39 MimeTypedUri (Complex Type)

URI for a resource, with support for specifying the MIME type.

Extension of: xsd:anyURI

Attributes:

Attribute	Usage	Type	Description
MimeType	optional	xsd:normalizedString	MIME type of the resource which can be retrieved from the URI.

3.4.40 NameSet (Model Group)

Various names.

Content Model: (ShortName , LongName*)

Elements:

Element	Type	Description
ShortName	Section 3.4.45, "ShortName (Complex Type)"	Short name (without white-space).
LongName	Section 3.4.38, "LongName (Complex Type)"	Human-readable name.

3.4.41 OptionalUseDefinition (Attribute Group)

Attribute set which defines the usage (optional attribute).

Attributes:

Attribute	Usage	Type	Description
Use	optional	Section 3.4.49, "UseType (Simple Type)"	Whether the usage is required or optional.

3.4.42 OuterCodeListChoice (Model Group)

A choice which currently only allows a simple (explicit) code list definition.

Content Model: ((SimpleCodeList))

Elements:

Element	Type	Description
SimpleCodeList (from Section 3.4.47, "SimpleCodeListSequence (Model Group)")	Section 3.4.46, "SimpleCodeList (Complex Type)"	Simple (explicit) code list definition. **SimpleCodeListSequence:** Details of a simple code list definition.

3.4.43 RequiredUseDefinition (Attribute Group)

Attribute set which defines the usage (required attribute).

Attributes:

Attribute	Usage	Type	Description
Use	required	Section 3.4.49, “Use-Type (Simple Type)”	Whether the usage is required or optional.

3.4.44 Row (Complex Type)

Row which represents an individual item in a code list.

Content Model: (Annotation? , Value+)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the row.
Value	Section 3.4.50, “Value (Complex Type)”	Column value for the row. **Rules:** **Rule 37 [document] :** A value must be provided for each required column. A value does not need to be provided for a column if the column is optional. **Rule 38 [document] :** If a value does not have an explicit column reference, the column is taken to be the column following the column of the preceding value in the row, or the first column if the value is the first value of the row.

3.4.45 ShortName (Complex Type)

A short name without whitespace that is suitable for use in generating names for software artifacts.

Rules:

Rule 39 [document] :

Must not contain whitespace characters.

Extension of: xsd:token

Attributes:

Attribute	Usage	Type	Description
xml:lang	optional		The language from which the short name is taken or derived.

3.4.46 SimpleCodeList (Complex Type)

Simple (explicit) code list definition.

Rules:

Rule 40 [application] :
Applications must not have any dependency on the ordering of the rows.

Content Model: (Annotation? , Row*)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the code list.
Row	Section 3.4.44, “Row (Complex Type)”	Row which represents an individual item in the code list.

3.4.47 SimpleCodeListSequence (Model Group)

Details of a simple code list definition.

Content Model: (SimpleCodeList)

Elements:

Element	Type	Description
SimpleCodeList	Section 3.4.46, “SimpleCodeList (Complex Type)”	Simple (explicit) code list definition.

3.4.48 SimpleValue (Complex Type)

Simple textual value.

Extension of: xsd:string

3.4.49 UseType (Simple Type)

Indicates whether the usage is required or optional.

Restriction of: xsd:token

Allowed Values:

· optional

· required

3.4.50 Value (Complex Type)

An individual code list metadata value.

Content Model: (Annotation? , (SimpleValue | ComplexValue)?)

Mixed Content: No

Elements:

Element	Type	Description
Annotation	Section 3.4.2, “Annotation (Complex Type)”	User annotation for the value.
SimpleValue (from Section 3.4.51, “ValueChoice (Model Group)”)	Section 3.4.48, “SimpleValue (Complex Type)”	Simple textual value. **Rules:** The value must be valid with respect to the datatype and restrictions of the matching column. **ValueChoice:** A choice between a simple textual value and a complex (structured) XML value. If the value is undefined, then neither choice is used.
ComplexValue (from Section 3.4.51, “ValueChoice (Model Group)”)	Section 3.4.3, “AnyOtherContent (Complex Type)”	Complex (structured) XML value. **Rules:** The names of all direct child elements of the ‘ComplexValue’ element must match the datatype ID for the matching column, unless that ID is set to ‘*’. The namespace URIs of all direct child elements of the ‘ComplexValue’ element must match the datatype library URI for the matching column, unless that URI is set to ‘*’. **ValueChoice:** A choice between a simple textual value and a complex (structured) XML value. If the value is undefined, then neither choice is used.

Attributes:

Attribute	Usage	Type	Description
ColumnRef (from Section 3.4.15, “ColumnReference (Attribute Group)”)	optional	xsd:IDREF	Reference to a column ID in the document. **ColumnReference:** Reference to the column with which this value is associated.

3.4.51 ValueChoice (Model Group)

A choice between a simple textual value and a complex (structured) XML value.

Content Model: (SimpleValue | ComplexValue)

Elements:

Element	Type	Description
SimpleValue	Section 3.4.48, “SimpleValue (Complex Type)”	Simple textual value. **Rules:** **Rule 41 [document] :** The value must be valid with respect to the datatype and restrictions of the matching column.
ComplexValue	Section 3.4.3, “AnyOtherContent (Complex Type)”	Complex (structured) XML value. **Rules:** **Rule 42 [document] :** The names of all direct child elements of the 'ComplexValue' element must match the datatype ID for the matching column, unless that ID is set to '*'. **Rule 43 [document] :** The namespace URIs of all direct child elements of the 'ComplexValue' element must match the datatype library URI for the matching column, unless that URI is set to '*'.

3.4.52 ValueIdentification (Attribute Group)

Information which identifies one of a set of alternate values.

Attributes:

Attribute	Usage	Type	Description
Identifier	optional	xsd:normalizedString	A string which identifies one of a set of alternate values.
xml:lang	optional		The language from which the value is taken or derived.

3.4.53 VersionLocationUriSet (Model Group)

Identification and location URIs for a version.

Content Model: (CanonicalVersionUri , LocationUri* , AlternateFormatLocationUri*)

Elements:

Element	Type	Description
CanonicalVersionUri	xsd:anyURI	Canonical URI which uniquely identifies this version. **Rules:** **Rule 44 [document] :** Must be an absolute URI, must not be relative. **Rule 45 [application] :** Must not be used as a de facto location URI.
LocationUri	xsd:anyURI	Suggested retrieval location for this version, in genericode format. **Rules:** **Rule 46 [application] :** An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.
AlternateFormatLocationUri	Section 3.4.39, “MimeTypedUri (Complex Type)”	Suggested retrieval location for this version, in a non-genericode format. Such alternative formats are intended only as additional renditions of the code list information, not as a replacements nor as alternatives for use in application processing.

4 Conformance

4.1 Auxiliary Rules

An XML instance conforms to the OASIS Code List Representation genericode document model if it does not violate any constraints expressed in the genericode.xsd schema associated with this version of the specification, including auxiliary rules marked as “document” rules.

An application conforms to the OASIS Code List Representation genericode processing rules if, in addition, it does not violate any of auxiliary rules marked as “application” rules.

4.2 Category: document

Note

Some of these document-related auxiliary rules can be programmatically tested using ISO/IEC 19757-3 Schematron as described in Appendix D, *Testing select document conformance rules (Non-Normative)*.

Rule 1 [document:: complexType CodeListDocument]

A code list must have at least one key, unless it is a metadata-only definition without a 'SimpleCodeList' element.

Rule 4 [document:: element CanonicalUri in complexType CodeListRef]

Must be an absolute URI, must not be relative.

Rule 6 [document:: element CanonicalVersionUri in complexType CodeListRef]

Must be an absolute URI, must not be relative.

Rule 19 [document:: attribute Type in complexType Data]

The datatype ID must not include a namespace prefix. For the W3C XML Schema datatypes, possible datatype IDs are 'string', 'token', 'boolean', 'decimal', etc.

Rule 20 [document:: attribute Type in complexType Data]

If the data is complex (i.e. XML), this value is set to the root element name for the XML value, or '*' if the root element name is not restricted.

Rule 22 [document:: attribute DatatypeLibrary in complexType Data]

If the data is complex (i.e. XML), this value is set to the namespace URI for the XML, or '*' if the namespace URI is not restricted.

Rule 23 [document:: complexType DataRestrictions]

The 'gc:lang' attribute may be specified only if no language is already set for the data type that is being restricted.

Rule 24 [document:: attribute ExternalRef in attributeGroup ExternalReference]

The external reference must not be prefixed with a '#' symbol.

Rule 25 [document:: element CanonicalUri in complexType Identification]

Must be an absolute URI, must not be relative.

Rule 27 [document:: element CanonicalVersionUri in modelGroup IdentificationRefUriSet]

Must be an absolute URI, must not be relative.

Rule 30 [document:: element CanonicalUri in modelGroup IdentificationVersionUriSet]

Must be an absolute URI, must not be relative.

Rule 32 [document:: element CanonicalVersionUri in modelGroup IdentificationVersionUriSet]

Must be an absolute URI, must not be relative.

Rule 34 [document:: complexType Key]

Only required columns can be used for keys.

Rule 37 [document:: element Value in complexType Row]

A value must be provided for each required column. A value does not need to be provided for a column if the column is optional.

Rule 38 [document:: element Value in complexType Row]

If a value does not have an explicit column reference, the column is taken to be the column following the column of the preceding value in the row, or the first column if the value is the first value of the row.

Rule 39 [document:: complexType ShortName]

Must not contain whitespace characters.

Rule 41 [document:: element SimpleValue in modelGroup ValueChoice]

The value must be valid with respect to the datatype and restrictions of the matching column.

Rule 42 [document:: element ComplexValue in modelGroup ValueChoice]

The names of all direct child elements of the 'ComplexValue' element must match the datatype ID for the matching column, unless that ID is set to '*'.

Rule 43 [document:: element ComplexValue in modelGroup ValueChoice]

The namespace URIs of all direct child elements of the 'ComplexValue' element must match the datatype library URI for the matching column, unless that URI is set to '*'.

Rule 44 [document:: element CanonicalVersionUri in modelGroup VersionLocationUriSet]

Must be an absolute URI, must not be relative.

Rule 48 [document:: element CanonicalUri in complexType CodeListSetRef]

Must be an absolute URI, must not be relative.

Rule 50 [document:: element CanonicalVersionUri in complexType CodeListSetRef]

Must be an absolute URI, must not be relative.

4.3 Category: application

Rule 2 [application:: attribute xml:base in complexType CodeListDocument]

xml:base does not apply to canonical URIs.

Rule 3 [application:: complexType CodeListRef]

The code list reference must be valid. An application may use the CanonicalVersionUri to select a local copy of the code list. If there is no CanonicalVersionUri, the CanonicalUri may be used to select a local copy of the code list. Otherwise the LocationUri value(s) may be tried in order, until a valid code list document is retrieved. An application must signal an error to the user if it is not able to retrieve a code list document to match the code list reference.

Rule 5 [application:: element CanonicalUri in complexType CodeListRef]

Must not be used as a de facto location URI.

Rule 7 [application:: element CanonicalVersionUri in complexType CodeListRef]

Must not be used as a de facto location URI.

Rule 8 [application:: element LocationUri in complexType CodeListRef]

If the CanonicalVersionUri has been defined, the LocationUri must reference a genericode CodeList document. If the CanonicalVersionUri is undefined, the LocationUri must reference a genericode CodeList Metadata document. An application must signal an error to the user if a LocationUri does not reference the appropriate type of genericode document.

Rule 9 [application:: element LocationUri in complexType CodeListRef]

An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Rule 10 [application:: attribute xml:base in complexType CodeListRef]

xml:base does not apply to canonical URIs.

Rule 11 [application:: attribute xml:base in complexType CodeListSetDocument]

xml:base does not apply to canonical URIs.

Rule 12 [application:: complexType ColumnRef]

The column reference must be valid. An application may use the CanonicalVersionUri to select a local copy of the code list or column set which contains the column definition. Otherwise the LocationUri value(s) may be tried in order, until a valid code list or column set document (containing the necessary column definition) is retrieved. An application must signal an error to the user if it is not able to retrieve a code list or column set document which contains the necessary column definition.

Rule 13 [application:: attribute Use (from) in complexType ColumnRef]

If specified, this overrides the usage specified in the external column set or code list document.

Rule 14 [application:: attribute xml:base in complexType ColumnRef]

xml:base does not apply to canonical URIs.

Rule 15 [application:: attribute xml:base in complexType ColumnSet]

xml:base does not apply to canonical URIs.

Rule 16 [application:: attribute xml:base in complexType ColumnSetDocument]

xml:base does not apply to canonical URIs.

Rule 17 [application:: complexType ColumnSetRef]

The column set reference must be valid. An application may use the CanonicalVersionUri to select a local copy of the column set or code list. Otherwise the LocationUri value(s) may be tried in order, until a valid column set or code list document is retrieved. An application must signal an error to the user if it is not able to retrieve a column set or code list document to match the column set reference.

Rule 18 [application:: attribute xml:base in complexType ColumnSetRef]

xml:base does not apply to canonical URIs.

Rule 21 [application:: attribute DatatypeLibrary in complexType Data]

If this URI not explicitly provided, the datatype library for the enclosing column set is used.

Rule 26 [application:: element CanonicalUri in complexType Identification]

Must not be used as a de facto location URI.

Rule 28 [application:: element CanonicalVersionUri in modelGroup IdentificationRefUriSet]

Must not be used as a de facto location URI.

Rule 29 [application:: element LocationUri in modelGroup IdentificationRefUriSet]

An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Rule 31 [application:: element CanonicalUri in modelGroup IdentificationVersionUriSet]

Must not be used as a de facto location URI.

Rule 33 [application:: element CanonicalVersionUri in modelGroup IdentificationVersionUriSet]

Must not be used as a de facto location URI.

Rule 35 [application:: complexType KeyRef]

The key reference must be valid.

An application may use the CanonicalVersionUri to select a local copy of the code list or column set which contains the key definition.

Otherwise the LocationUri value(s) may be tried in order, until a valid code list or column set document (containing the necessary key definition) is retrieved.

An application must signal an error to the user if it is not able to retrieve a code list or column set document which contains the necessary key definition.

Rule 36 [application:: attribute xml:base in complexType KeyRef]

xml:base does not apply to canonical URIs.

Rule 40 [application:: complexType SimpleCodeList]

Applications must not have any dependency on the ordering of the rows.

Rule 45 [application:: element CanonicalVersionUri in modelGroup VersionLocationUriSet]

Must not be used as a de facto location URI.

Rule 46 [application:: element LocationUri in modelGroup VersionLocationUriSet]

An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Rule 47 [application:: complexType CodeListSetRef]

The code list set reference must be valid.

An application may use the CanonicalVersionUri to select a local copy of the code list set.

If there is no CanonicalVersionUri, the CanonicalUri may be used to select a local copy of the code list set.

Otherwise the LocationUri value(s) may be tried in order, until a valid code list set document is retrieved.

An application must signal an error to the user if it is not able to retrieve a code list set document to match the code list set reference.

Rule 49 [application:: element CanonicalUri in complexType CodeListSetRef]

Must not be used as a de facto location URI.

Rule 51 [application:: element CanonicalVersionUri in complexType CodeListSetRef]

Must not be used as a de facto location URI.

Rule 52 [application:: element LocationUri in complexType CodeListSetRef]

If the CanonicalVersionUri has been defined, the LocationUri must reference a genericode CodeList-Set document.

If the CanonicalVersionUri is undefined, the LocationUri must reference a genericode CodeListSet Metadata document.

An application must signal an error to the user if a LocationUri does not reference the appropriate type of genericode document.

Rule 53 [application:: element LocationUri in complexType CodeListSetRef]

An application must signal an error to the user if a document retrieved using a LocationUri is not in genericode format.

Rule 54 [application:: attribute xml:base in complexType CodeListSetRef]

xml:base does not apply to canonical URIs.

Appendix A Release Notes (Non-Normative)

A.1 Availability

Online and downloadable versions of the latest OASIS release of this package are available from:

- https://docs.oasis-open.org/codelist/genericode/v1.0/cs02/

Online and downloadable versions of the latest ISO/IEC release of this package are available from:

- http://standards.iso.org/ittf/PubliclyAvailableStandards/

A.2 Package Structure (Non-Normative)

The genericode specification is published as a zip archive in the release directory. Unzipping this archive creates a directory named genericode/v1.0/cs02 containing a "`doc/`" directory with documentation. The authoritative source of the documentation is a master DocBook XML file (genericode-v1.0-cs02.xml), a generated hypertext version of this file (genericode-v1.0-cs02.html), and a generated PDF version of this file (genericode-v1.0-cs02.pdf). The files in other subdirectories, linked to from genericode-v1.0-cs02.xml, genericode-v1.0-cs02.html, and genericode-v1.0-cs02.pdf, contain the various normative and informational pieces of the genericode release. A description of each subdirectory is given below. Note that while the genericode.xml file is the "original" of this specification, it may not be viewable in all currently available web browsers.

`doc/`

Documentation sources and published results

`doc/art/`

HTML presentation artwork

`doc/art/pdfart`

PDF publishing artwork

`doc/db/`

DocBook stylesheets for viewing XML and HTML

`json-example/`

Sample genericode JSON files

`sch/`

Schematron auxiliary rule constraints

`xml/`

Sample genericode XML files

`xsd/`

XML structural constraints

`xslt/`

Sample XSLT transformation (see Appendix B, *Sample transformation to JSON (Non-Normative)*)

Appendix B Sample transformation to JSON (Non-Normative)

Recognizing the custom use of JSON in a tight binding between user-defined processes, the committee sees no purpose served by standardizing a JSON syntax for the genericode vocabulary. Genericode is for the interchange of code list information.

Nevertheless, having the information transformed into JSON may be a convenience to users. Included in the distribution is a non-normative XSLT 2.0 transformation of genericode XML into a colloquial JSON syntax. No JSON schema is provided for validating the colloquial syntax, as it is anticipated users will have the need for their own custom JSON. This transformation may be useful as inspiration for creating one's own transformation.

The sample `xslt/gc2gcj.xsl` stylesheet reads genericode XML and produces well-formed JSON syntax as its output. The stylesheet imports the `xslt/jsonsupport.xsl` fragment which is not run standalone.

Appendix C Sample instances of genericode (Non-Normative)

The distribution directory `xml/` includes the following sample instances copied from the indicated publicly-available sources:

- from `https://docs.oasis-open.org/legalxml-courtfiling/ecf/v5.0/cs01/schema/`
  - `CaseTypeCode.gc`
    - a short list of only codes representing types of cases
- from `https://docs.oasis-open.org/ubl/os-UBL-2.3/cl/gc/default/`
  - `ChannelCode-2.3.gc`
    - a typical list of code, name, and description values representing type available channels for communication
  - `CurrencyCode-2.3.gc`
    - a code list with five columns representing currencies of the world
  - `UnitOfMeasureCode-2.3.gc`
    - a sparsely populated list with seven columns representing units of measure for values

The distribution directory `json-example/` includes the JSON transformations of the examples found in the `xml/` directory. These are created using the `make-sample.sh` script that invokes the transformation described in Appendix C, *Sample instances of genericode (Non-Normative)*.

Appendix D Testing select document conformance rules (Non-Normative)

Some of the document conformance auxiliary rules prescribed in Section 4.2, “Category: document” can be tested using ISO/IEC 19757-3 Schematron.

The sample `sch/genericode.sch` Schematron assertion schema is provided as a sample set of assertions to be converted into testable code.

Appendix E Considerations of life cycle meta-data (Non-Normative)

Data life cycle management relies on having useful life cycle metadata with which to record properties of data and histories of changes in that data. Such life-cycle concepts readily apply to code list values. A few example concepts include but are not limited to dates and times indicating when a value in a coded domain is accepted to be used by users or is considered a valid value, a signal of deprecation indicating when a value may still be valid but is recommended not to be used, and jurisdictional constraints within which a value may be used to the exclusion of other jurisdictional boundaries.

Custodians of code lists should consider all such metadata issues from the beginning of planning and creating a new list of values in a coded domain.

The flexibility of genericode to declare columns of value-level metadata supports describing life-cycle metadata in addition to any other kind of metadata or other information that may be associated with values in a coded domain.

Per the semantics of genericode, all of the information associated with an individual coded value or part thereof is captured in the <Row> element. Human-oriented information about the coded value or part in general that is not intended to be processed by a computer is put into the <Annotation> child of <Row>. Computer-processable information is put into a set of <Value> elements as metadata. Human-oriented information about a particular metadata value is put into the <Annotation> child of <Value>. Computer-processable information is put into either a <SimpleValue> child of <Value> for a monolithic string value, or into a <ComplexValue> child of <Value> for a value that, itself, is allowed to be described richly using markup.

The semantics and constraints of each metadata <Value> are described to an arbitrary extent in the <ColumnKey> element identified with the <Value> element's columnRef= attribute. Custodians of code lists should consider how richly described each of the metadata columns serves the reader or user of the code list when they inspect the content of <ColumnKey>. It is entirely up to the list's custodian to establish and to document the semantics and constraints of code-level metadata values in <ColumnKey>.

Accordingly, it is not part of genericode to standardize any semantics beyond the semantics of the genericode specification itself of how genericode markup is used to represent values in a coded domain. There are no standardized value-level metadata semantics, and so, there are no standardized life-cycle metadata semantics to be leveraged by genericode users.

While some life-cycle metadata semantics are considered somewhat universal, each custodian may have a different perspective of such semantics that impacts on their decisions regarding how each semantic is to be expressed, constrained, and documented. Therefore, this specification does not attempt to define any standardized code-level metadata of any kind, including data life cycle management metadata.

Appendix F The Open-edi reference model perspective of code lists (Non-Normative)

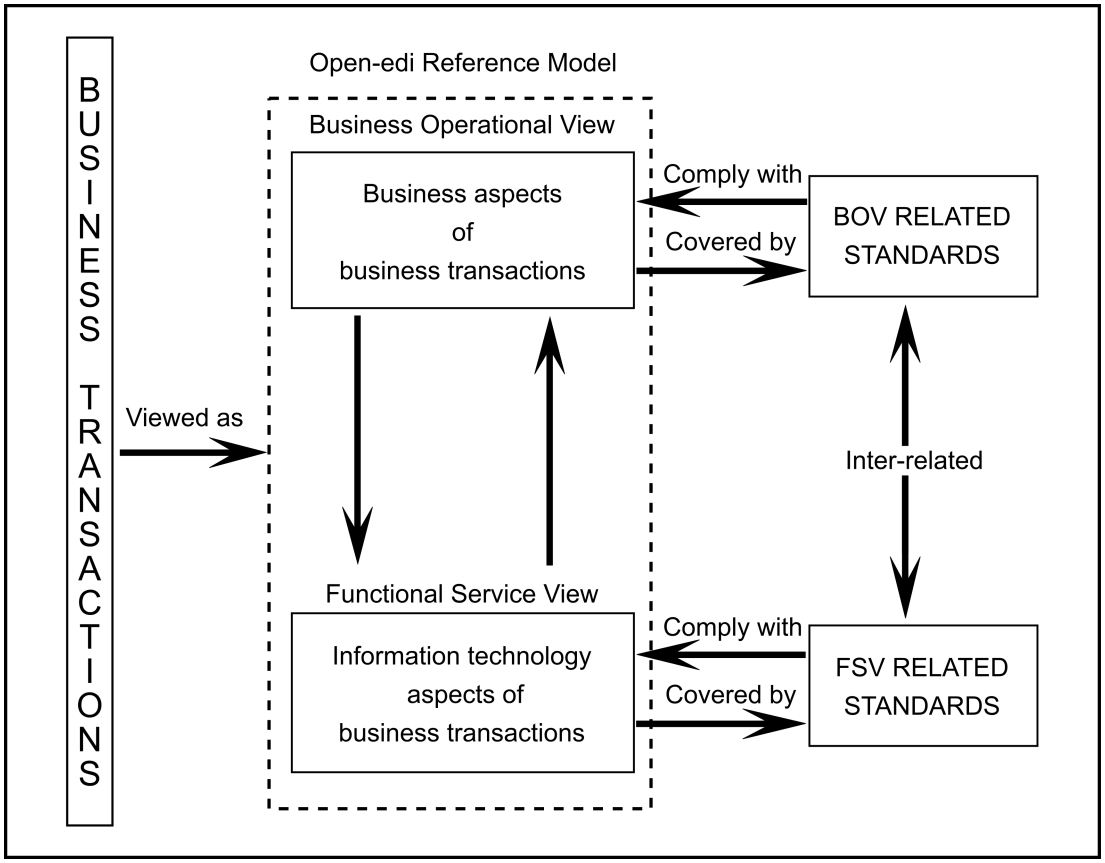
ISO/IEC 14662:2010 Information technology - Open-edi reference model [**Open-edi**] has been developed primarily in order to provide standards required for the inter-working of organizations through interconnected information technology systems. Open-edi lowers barriers to electronic data interchange by introducing standard business scenarios and the necessary services to support them.

The Open-edi Reference Model identifies the required standards for Open-edi and provides a reference for those standards by defining the basic concepts used to develop them.

Figure F.1, “Open-edi Overview” depicts two views to describe the relevant aspects of business transactions:

- the Business Operational View (BOV);
- the Functional Service View (FSV).

Figure F.1. Open-edi Overview



The BOV addresses the aspects of the semantics of business data in business transactions and associated data interchanges which apply to the business needs of Open-edi. The BOV-related standards are tools and rules by which users who understand the operating aspects of a business domain may create scenarios.

The FSV addresses the supporting services meeting the mechanistic needs of Open-edi, focusing on information technology aspects of functional capabilities, service interfaces, and protocols.

Appendix G Acknowledgements (Non-Normative)

The following persons and companies participated as members of the OASIS Code List Representation Technical Committee during the years of its development (2006–2022).

Todd Albers, Federal Reserve Bank of Minneapolis
Kenneth Bengtsson, Individual
Erlend Klakegg Bergheim, The Norwegian Agency for Public and Financial Management (DFO)
Jon Bosak, Individual
James Cabral, Individual
Andrea Caccia, Individual
Ger Clancy, IBM
Anthony Coates, Individual
Ray Denenberg, Library of Congress
Marc Gratacos, Individual
Jim Harris, National Center for State Courts
Philip Helger, Individual
G. Ken Holman, Crane Softwrights Ltd.
Natalie Muric, Publications Office of the European Union
Levine Naidoo, IBM
Paul Spencer, Individual
Pim van der Eijk, Sonnenglanz Consulting
Dennis Weddig, Federal Reserve Bank of Minneapolis