Business Document Naming and Design Rules (BDNDR) Version 1.1

Committee Specification 01

08 November 2021

This stage:
https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/Business-Document-NDR-v1.1-cs01.xml (Authoritative)
https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/Business-Document-NDR-v1.1-cs01.html
https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/Business-Document-NDR-v1.1-cs01.pdf

Previous stage:
http://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/csd04/Business-Document-NDR-v1.1-csd04.xml (Authoritative)
http://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/csd04/Business-Document-NDR-v1.1-csd04.html
http://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/csd04/Business-Document-NDR-v1.1-csd04.pdf

Latest stage:
https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/Business-Document-NDR-v1.1.html
https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/Business-Document-NDR-v1.1.pdf

Technical Committee:
OASIS Universal Business Language (UBL) TC

Chairs:
Kenneth Bengtsson (kbengtsson@efact.pe), Individual
G. Ken Holman (gkholman@CraneSoftwrights.com), Crane Softwrights Ltd.

Editors:
Kenneth Bengtsson (kbengtsson@efact.pe), Individual
Erlend Klakegg Bergheim (erlend.klakegg.bergheim@difi.no), Norwegian Digitalisation Agency
G. Ken Holman (gkholman@CraneSoftwrights.com), Crane Softwrights Ltd.

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

- https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/Business-Document-NDR-v1.1-cs01.zip

Related work:
This specification supersedes:

[**Business-Document-NDR-v1.0**] *Business Document Naming and Design Rules Version 1.0.* Edited by Tim McGrath, Andy Schoka and G. Ken Holman. 18 January 2017.

OASIS Standard. http://docs.oasis-open.org/ubl/Business-Document-NDR/v1.0/os/Business-Document-NDR-v1.0-os.html. Latest version: http://docs.oasis-open.org/ubl/Business-Document-NDR/v1.0/Business-Document-NDR-v1.0.html.

Abstract:

This specification prescribes a set of naming and design rules used to create document model validation artefacts associated with abstract information bundles formally described using the Core Component Technical Specification 2.01 [**CCTS**] in either or both of XML documents using W3C Schema XSD files and OASIS Context/value association files, or/and JSON documents using expressions.

Status:

This document was last revised or approved by the OASIS Universal Business Language TC 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=ubl#technical.

TC members should send comments on this specification 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/ubl/.

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/ubl/ipr.php).

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 at https://www.oasis-open.org/committees/ubl/ipr.php.

See Appendix A, *(informative) Release notes* for more information regarding this release package.

Citation format:

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

[**Business-Document-NDR-v1.1**] *Business Document Naming and Design Rules (BDNDR) Version 1.1.* Edited by Kenneth Bengtsson, Erlend Klakegg Bergheim and G. Ken Holman. 08 November 2021. OASIS Committee Specification 01. https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/Business-Document-NDR-v1.1-cs01.html. Latest stage: https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/Business-Document-NDR-v1.1.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 Basic concepts

1.1.1 Naming and design rules

The Open-edi Reference Model **[ISO 14662]** defines an information bundle as the abstract description of the structure and semantics of the information exchanged between parties.

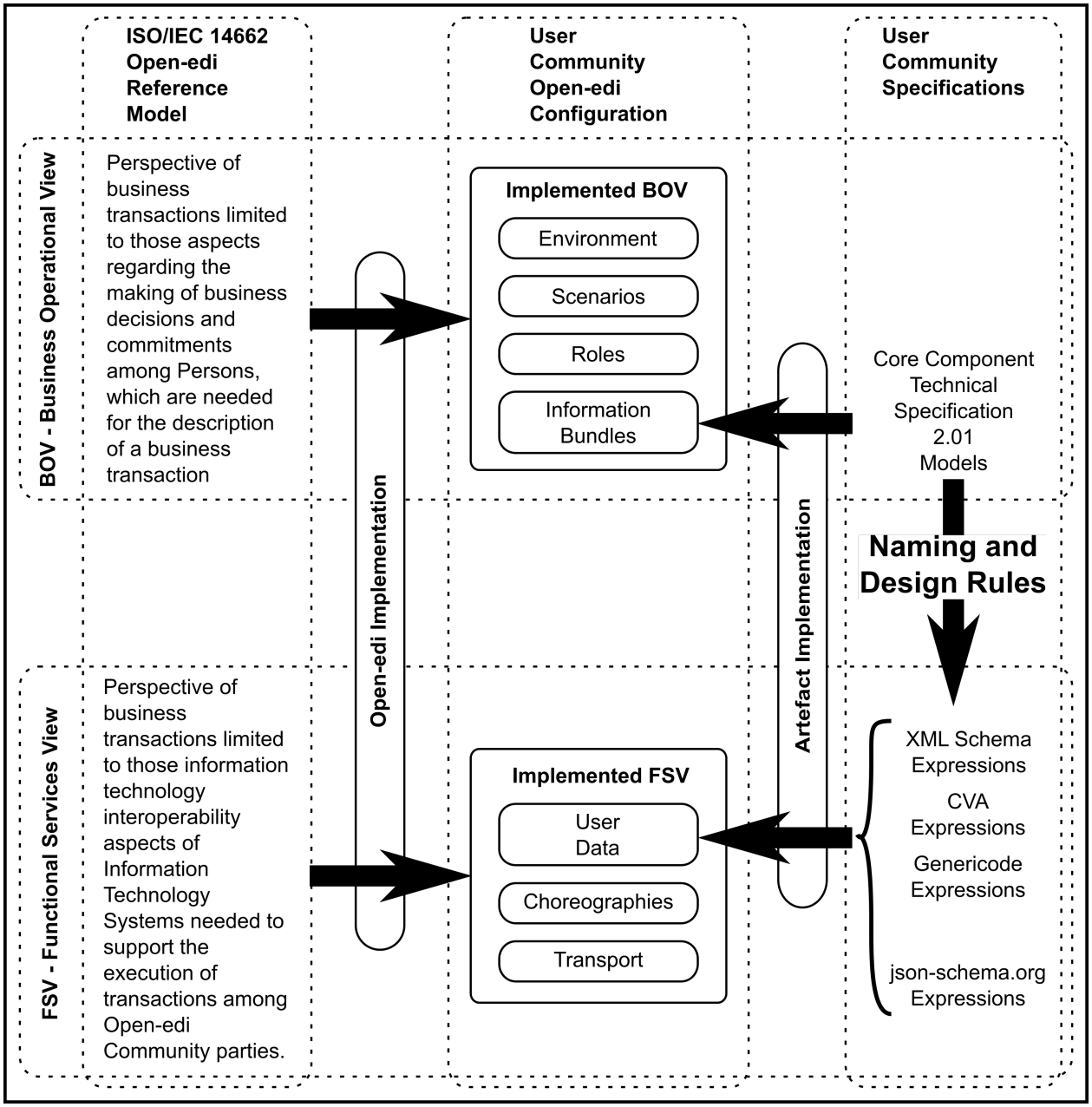
An information bundle can be represented as a logical semantic model. This logical semantic model can be used to produce a physical syntax model that defines the structure and syntax of the Open-edi user data actually exchanged in business transactions. These naming and design rules formalize a method of representing the semantic components of information bundles using the ebXML Core Components Technical Specification **[CCTS]**.

These semantic models can then be used to produce equivalent W3C Schema XSD files [XSD schema] and OASIS Context/value Association **[CVA]** expressions suitable for defining and validating XML documents **[XML]** used to convey Open-edi user data.

Also, or alternatively, these semantic models can then be used to produce equivalent JSON schema **[JSON Schema]** expressions suitable for defining and validating JSON documents **[ISO 21778 - ECMA JSON]** used to convey Open-edi user data.

This is illustrated in Figure 1, “Naming and Design Rules in an Open-edi Application”.

Figure 1. Naming and Design Rules in an Open-edi Application



The rules presume the reader is already familiar with the following specifications:

- United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Core Components Technical Specification 2.01 – Part 8 of the ebXML Framework **[CCTS]**;
- ISO/IEC 11179 Data Elements **[ISO 11179]**;
- W3C Schema [XSD schema] XSD files for XML document constraint specification;
- OASIS Context/value association using genericode **[CVA]** for code list association;
- OASIS genericode **[genericode]** for code list representation;
- JSON data interchange format **[ISO 21778 - ECMA JSON]**; and
- JSON Schema **[JSON Schema]**.

Note

The OASIS Universal Business Language (UBL) can be considered as a reference implementation of these naming and design rules and the examples used in this specification are primarily taken from the UBL 2.3 specifications.

Note

Other validation artefacts (using RelaxNG and ASN.1) are also provided for UBL [**UBL-2.1**]. The rules for producing these are outside the scope of this work product.

Note

The direction taken regarding the JSON implementation of these naming and design rules was developed in the discussion paper [**CCTS-and-JSON**] where the principles are described with instance examples.

1.1.2 Modeling concepts

Information bundles (describing information to be exchanged) are modeled as a collection of semantic components.

Each semantic component in an information bundle corresponds to one of the following types of Business Information Entity (BIE) in a CCTS Document Model:

- a single irreducible semantic component (referred to as a Basic Business Information Entity or BBIE);
- a structured aggregation of other semantic components (referred to as an Aggregate Business Information Entity or ABIE); or
- an association between ABIEs (referred to as an Association Business Information Entity or ASBIE).

The CCTS semantic model is not dependent on the syntax of the Open-edi user data actually exchanged.

The Open-edi user data exchanged between parties is a machine-readable instance of an information bundle (CCTS Document Model).

This specification assumes XML and/or JSON are the machine-readable syntaxes used for exchanging the Open-edi user data. When using an XML document for exchanging Open-edi user data each BIE corresponds to an XML element. When using a JSON document for exchanging Open-edi user data each BIE corresponds to a JSON object.

The relationships between these various concepts are described in Table 1, “Modeling concepts”

Table 1. Modeling concepts

Open-edi Reference Model (ISO 14882)	Semantic Model (ebXML CCTS)	Physical representations (W3C XML; ECMA JSON)
Information bundle	Semantic model	Document schema
Semantic component	Business Information Entity	XML element; JSON object
Open-edi user data	Document	XML document; JSON document

	Document ABIE	XML document element schema definition with only element children, and the XML document element itself; JSON document object schema definition with only object children, and the top-most object itself
	Library ABIE	XML element schema definitions (but not a document element schema definition) with only element children; JSON object schema definition (but not a top-level object schema definition) with only object array children
	ASBIE	XML element (but not a document element) with only element children, defined by a Library ABIE; JSON object (but not a top-level object) with only object array children, defined by a Library ABIE
	BBIE	XML element with only text characters as children, no elements as children, as defined by a BBIE; JSON object with only object array children, defined by a BBIE

1.1.3 Business Information Entities

Following the conventions of the ebXML Core Component Technical Specification **[CCTS]** a Business Information Entity (BIE) is piece of business data or a group of pieces of business data with a unique business semantic definition.

A BIE is the result of using a core component within a specific business context. As such each BIE must be based on a core component. The definition of core components is outside the scope of this specification.

It is the Business Information Entities that provide the structure for the semantic components of the body of the business document.

The semantic components used to create the Open-edi user data validation artefacts (following these naming and design rules) are to be applied in a specific business context. Therefore it is the contextualized Business Information Entities to which these rules apply and not the core components from which they have been derived.

1.2 Terminology

1.2.1 Key words

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this document are to be interpreted as described in **[RFC 2119]**. Note that for reasons of style, these words are not capitalized in this document.

1.2.2 Terms and Definitions

Business Information Entity (BIE)

> A piece of business data or a group of pieces of business data with a unique Business Semantic definition, see **[CCTS]**.

Context/value Association (CVA)

> The association of value constraints imposed on information found in a particular document context, see **[CVA]**.

document ABIE

> The apex ABIE of an information bundle.

document element

The apex element of information in an XML document, see [**XML**].

extension collection

The set of extension elements found in an XML document, constrained by an extension schema, that supplements the base information that is constrained by the published document model.

extension item

A single instance of structured supplemental information and its associated metadata distinguishing it from other extension items.

extension point

The apex element of structured supplemental information described by its metadata.

information bundle

The formal description of the semantics of the recorded information to be exchanged, as defined in [**ISO 14662**].

JSON schema

A JSON vocabulary to annotate and validate JSON documents [**JSON Schema**].

naming and design rules

A set of rules governing the expression of information bundles using [**CCTS**], and the creation of either or both the associated XML document model validation artefacts using [**CVA**] and [XSD schema], or/and JSON schema model artefacts using [**JSON Schema**].

object class

A set of ideas, abstractions, or things in the real world that are identified with explicit boundaries and meaning and whose properties and behavior follow the same rules [**ISO 11179**] (definition 3.3.22).

Open-edi user data

Machine-readable instance of the content of information bundles or components of information bundles (as semantic components), see [**ISO 14662**].

property

A characteristic common to all members of an object class [**ISO 11179**] (definition 3.3.29).

semantic component

A unit of information unambiguously defined in the context of the business goal of the business transaction. A semantic component may be atomic or composed of other semantic components, see [**ISO 14662**].

XSD schema

An XML document model definition conforming to the W3C XML Schema language [**XSD1**] [**XSD2**].

The terms Core Component (CC), Basic Core Component (BCC), Aggregate Core Component (ACC), Association Core Component (ASCC), Business Information Entity (BIE), Basic Business Information Entity (BBIE), and Aggregate Business Information Entity (ABIE) are used in this specification with the meanings given in [**CCTS**].

The terms Object Class, Property Term, Representation Term, and Qualifier are used in this specification with the meanings given in **[ISO 11179]**.

1.2.3 Symbols and Abbreviations

ABIE

Aggregate Business Information Entity **[CCTS]**

ASBIE

Association Business Information Entity **[CCTS]**

BIE

Business Information Entity **[CCTS]**

BBIE

Basic Business Information Entity **[CCTS]**

CCTS

Core Component Technical Specification **[CCTS]**

CVA

Context/value Association **[CVA]**

JSON

JavaScript Object Notation **[ISO 21778 - ECMA JSON]**

NDR

Naming and Design Rules naming and design rules

TC

Technical Committee

XSD

W3C XML Schema Definition XSD schema

1.3 Normative References

[CCTS] *UN/CEFACT Core Component Technical Specification - Part 8 of the ebXML Framework)* 15 November 2003Version 2.01*http://www.unece.org/fileadmin/DAM/cefact/codesfortrade/CCTS/CCTS_V2-01_Final.pdf*

[CVA] *OASIS Context/value association using genericode 1.0.* 15 April 2010. Committee Specification 01. http://docs.oasis-open.org/codelist/cs01-ContextValueAssociation-1.0/doc/context-value-association.html.

[genericode] *OASIS Code List Representation (Genericode) Version 1.0.* 28 December 2007. Committee Specification 01. http://docs.oasis-open.org/codelist/cs-genericode-1.0/doc/oasis-code-list-representation-genericode.html.

[ISO 11179] *ISO/IEC 11179-1:2004 Information technology — Specification and standardization of data elements — Part 1: Framework for the specification and*

standardization of data elements *http://standards.iso.org/ittf/PubliclyAvailableStandards/c035343_ISO_IEC_11179-1_2004(E).zip*

[**ISO 14662**] *ISO/IEC 14662:2004 Information technology — Open-edi reference model* *http://standards.iso.org/ittf/PubliclyAvailableStandards/*

[**ISO 21778 - ECMA JSON**] *ISO/IEC 21778 Information technology — The JSON data interchange format* *http://standards.iso.org/ittf/PubliclyAvailableStandards/*, *ECMA 404 The JSON data interchange format* *https://www.ecma-international.org/publications/standards/Ecma-404.htm*

[**JSON Schema**] *JSON Schema Validation: A Vocabulary for Structural Validation of JSON*, A. Wright, H. Andrews, G. Luff, Editors, *http://json-schema.org/latest/json-schema-validation.html*

[**RFC 2119**] S. Bradner, , *Key words for use in RFCs to Indicate Requirement Levels*, BCP 14, RFC 2119, March 1997, <*http://www.rfc-editor.org/info/rfc2119*>.

[**XML**] *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, Editors, W3C Recommendation, 26 November 2008, *http://www.w3.org/TR/2008/REC-xml-20081126/*. Latest version available at *http://www.w3.org/TR/xml*.

[**XPath 1.0**] *XML Path Language (XPath) Version 1.0*, J. Clark, S. DeRose, Editors, W3C Recommendation, 16 November 1999, *http://www.w3.org/TR/1999/REC-xpath-19991116/*. Latest version available at *http://www.w3.org/TR/xpath*.

[**XSD1**] *XML Schema Part 1: Structures Second Edition*, H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, Editors, W3C Recommendation, 28 October 2004, *http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/*. Latest version available at *http://www.w3.org/TR/xmlschema-1*.

[**XSD2**] *XML Schema Part 2: Datatypes Second Edition*, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 28 October 2004, *http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/*. Latest version available at *http://www.w3.org/TR/xmlschema-2*.

1.4 Non-Normative References

[**CCTS-and-JSON**] *CCTS and JSON discussion paper*, 06 December 2016, Kenneth Bengtsson, Erlend Bergheim, G. Ken Holman, Editors, *https://www.oasis-open.org/committees/document.php?document_id=59528*.

[**UBL-2.1**] *Universal Business Language Version 2.1.* 04 November 2013. OASIS Standard. http://docs.oasis-open.org/ubl/os-UBL-2.1/UBL-2.1.html.

[**xmldsig**] *XML-Signature Syntax and Processing Version 1.1*, D. E. Eastlake, J. Reagle, D. Solo, F. Hirsch, M. Nyström, T. Roessler, K. Yiu, Editors, W3C Recommendation, 11 April 2013, *http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/*. Latest version available at *http://www.w3.org/TR/xmldsig-core1*.

2 Context of use and application of these rules

These XML Naming and Design Rules may be used to create a collection of artefacts for defining and validated a set of extensible XML document types and extensible JSON schema definitions.

They describe processes for:

A. expressing the semantic components of information bundles using the ebXML Core Components Technical Specification **[CCTS]**, and

B. producing XML definition and validation artefacts based on those semantic components, specifically:

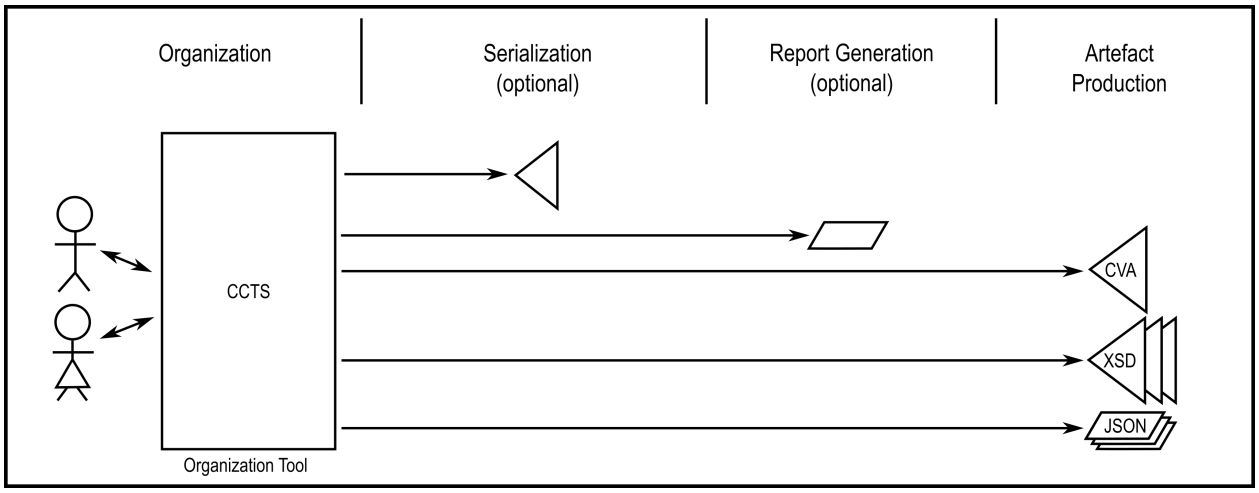
   I. XML document structural constraints of elements and attributes (for example, nesting and cardinality) using W3C Schema [XSD schema],

   II. XML data value constraints using OASIS Context/value Association **[CVA]** expressions of values **[genericode]** (for example, coded value domains or code lists) with XML document locations using XPath **[XPath 1.0]**, and

   III JSON expression constraints using JSON Schema **[JSON Schema]** expressions.
   .

These processes are depicted in Figure 2, “Generic NDR processes to create validation artefacts”.

Figure 2. Generic NDR processes to create validation artefacts



Designers (and implementers) may choose to adopt other, optional processes to produce additional artefacts. For example, the serialization of the information bundle (as a CCTS model) into a form suitable for further processing or the production of reports useful in the design and review of the model. See Appendix C, *(informative) Additional production processes* for more details.

3 Information modeling

3.1 Document ABIEs

A Document ABIE structures the apex of the information bundle to be exchanged between parties.

MOD01 Document ABIE
The apex of the information bundle shall be structured as a single top-level ABIE, referred to in this specification as a Document ABIE. **Note** The rationale is that the Document ABIE is always considered a one-member collection in and of itself with no other members in the collection.

3.2 ABIE library

The ABIE library is a collection of common, reusable ABIEs available to be referenced by ASBIEs.

The ABIE library does not include any Document ABIEs.

MOD02 ABIE library contents
A Document ABIE shall not be referenced by any ASBIE. **Note** The rationale is that Document ABIEs are identified in syntax implementations separately from other collections of ABIEs.

MOD03 ABIE library ordering
The ABIE library shall have all ABIEs defined in either alphabetical order or Unicode code point order of the ABIE's dictionary entry name. **Note** The rationale is that the library can be very large and a reader new to the library may be unfamiliar with any arbitrary ordering of the ABIEs. Designers and implementers can navigate a collection of ABIEs more easily when they are in either alphabetical order or Unicode code point order. The difference between the two is that in Unicode code point order upper-case letters are ordered in advance of lower-case letters, as found in acronyms. There may be restrictions on some tools to use Unicode code point order instead of alphabetical order.

3.3 Extensions

Wherever possible semantic components within an information bundle should be expressed using the BIEs found in an existing Document ABIE or the ABIE Library. However there may be implementations where supplementary information is required to be exchanged in the information bundle in a way that does not interfere with existing BIEs.

The extension mechanism in this specification provides for including additional semantic components that may augment a standardized information bundle with customized additional information. This mechanism is made available at the beginning of every Document ABIE and Library ABIE.

Extensions can contain new BIEs or can reference BIEs from existing ABIEs but used in a different context. Extensions can also include foreign constructs not defined as BIEs.

Extensions may be horizontal in nature in that they are available to use for all information bundles. An example might be the structure of digital signatures[xmldsig].

Extensions may also be vertical in nature, in that they are applicable only to a single information bundle. For example, additional invoice line detail information to augment an invoice for a specific industry.

An extension collection provides a placeholder for the extensions to be used with a set of Open-edi user data. Within each collection there may be a number of extensions, each with metadata properties regarding the extension and each with a single extension point (the apex structure of the supplemental information).

MOD04 Extension availability
Each document shall allow for optional augmentation with a collection of information not conceptually described by existing BIEs.

When using an XML document for exchanging Open-edi user data, extension schema fragments augment the document's schema created for a document ABIE and all library ABIEs.

3.4 Model revisions

When a vocabulary is modified to become a new revision, care must be taken to ensure that all revisions are backward compatible with all previous revisions. That is, that every possible instance of all previous revisions remain valid instances of any new revision. This is guaranteed by constraining how the cardinality of any existing BBIE and ASBIE can be changed, and how any new BBIE or ASBIE can be created.

MOD05 Revision existing BBIE and ASBIE cardinality
The cardinality of an existing BBIE or ASBIE can change in a new revision only to be a superset of the old model. That is, the constraints are as follows: • the new minimum cardinality shall be lower than or equal to the old minimum cardinality, and • the new maximum cardinality shall be higher than or equal to the old maximum cardinality. **Note** This rule permits a new revision to make a mandatory BIE optional but not make an optional BIE mandatory.

MOD06 Revision new BBIE and ASBIE cardinality
The cardinality of a new BBIE or ASBIE in a new model revision is based on where the BIE is being added: • a BBIE or ASBIE that is newly-added to an existing ABIE shall have a minimum cardinality of zero

MOD06 Revision new BBIE and ASBIE cardinality
Note This rule imposes no constraint on any BBIE or ASBIE in a newly-created ABIE in the revised model.

3.5 Model subsets

When a vocabulary is modified to become a subset of an existing revision, care must be taken to ensure that every possible instance of the subset remains a valid instance of the revision. This is guaranteed by constraining the cardinality of every existing BBIE and ASBIE as used in the subset.

MOD07 Subset existing BBIE and ASBIE cardinality
The cardinality of an existing BBIE or ASBIE can change in a model subset only to be a subset of the old model. That is, the constraints are as follows: • the new minimum cardinality can be higher than or equal to the old minimum cardinality, and • the new maximum cardinality can be lower than or equal to the old maximum cardinality. **Note** This rule permits a subset to make an optional BIE mandatory but not make a mandatory BIE optional. **Note** This rule permits a subset to omit an optional BIE entirely from the model.

4 Dictionary information

4.1 Dictionary information overview

A BIE is described by the values of its dictionary information as specified by the Core Components Technical Specification 2.01 [CCTS].

To facilitate the definition of the appropriate name for a BBIE, the CCTS property term is specified as the concatenation of two contributing dictionary information values. The optional property term possessive noun and the mandatory property term primary noun are used. When the possessive noun is used the values are separated by a space character.

To facilitate the definition of the appropriate name for a ASBIE, the CCTS property term is specified as the concatenation of two contributing dictionary information values. The optional associated object class qualifier and the mandatory associated object class are used. When the associated object class qualifier is used the qualifier value is suffixed with an underscore character and the values are separated by a space character.

4.2 Dictionary information values

Certain rules govern the creation and expression of dictionary information values for all BIEs defined in the semantic model of an information bundle.

COM01 Dictionary information values
The text describing dictionary information values shall be a string value of Unicode characters without embedded hierarchical structure. The value itself may be structured in its syntax within the string. **Note** The rationale is that the dictionary information values may be constrained in its expression, such as is true in an XML attribute.

COM02 Dictionary information value prohibited characters and character sequences
The following characters shall not be contained in any dictionary information value: • the characters “<”, “>”, “&” • white-space characters other than the “ ” (space) character • the character sequence “--” **Note** The rationale is that prohibiting these characters and sequences will allow the dictionary information values to be processed more simply in different XML contexts without special handling. **Note** The constraint on white-space characters prohibits values from being structured as paragraphs.

4.3 Abbreviations

It may be convenient to abbreviate complex terms or to use commonly-accepted abbreviations in dictionary information values.

COM03 Controlled list of abbreviations in BIE names
Abbreviations for terms used in BIE names shall be taken from a controlled list of abbreviations agreed for use within the semantic model. **Examples** “Identifier” is abbreviated as “ID”. “Universally Unique Identifier” is abbreviated as “UUID”. **Note** The rationale is that some common or complex terms have commonly-accepted abbreviations suitable for shortening the length of BIE names.

COM04 Controlled list of abbreviations in dictionary entry name information values
Abbreviations for terms used in dictionary entry name information values shall be taken from a controlled list of commonly-agreed abbreviations. **Examples** “XML Path Language” is abbreviated as “XPath”. “Card Verification Value” is abbreviated as “CV2”. **Note** The rationale is that some common terms have widely-accepted abbreviations in general use or in particular use within the information domain. Inconsistent use of abbreviations may lead to semantic ambiguity.

COM05 List of equivalent terms in BIE names
Equivalent terms used in BIE names shall be taken from a list of commonly-agreed equivalent terms. **Example** The property term primary noun “URI” is considered equivalent to the representation term “Identifier”. **Note** The rationale is that some common terms wholly include the concepts presented in other terms and so should be considered equivalent in order to prevent duplication.

4.4 Dictionary information for BIEs

4.4.1 Component type for BIEs

COM06 Component Type for a BIE
Each BIE component shall be typed as being one of either an ABIE, BBIE or ASBIE.

4.4.2 Dictionary information for ABIEs

COM07 Minimum set of dictionary information values describing an ABIE

Each ABIE shall have the following set of dictionary information values:
• Component Type (mandatory)
  • shall be the value “ABIE”
• Definition (mandatory)
  • this value shall describe the ABIE using complete natural language sentences in a single paragraph
• Alternative Business Terms (optional)
  • this value shall list any other commonly used terms that are synonyms for the ABIE
• Object Class Qualifier (optional)
  • this value shall qualify the object class for a specific context
• Object Class (mandatory)
  • this value shall identify the object of interest within an information bundle; it is the class to which the ABIE’s BIEs belong
• Name (mandatory)
  • this value shall be the concatenation of Object Class Qualifier and the Object Class without any spaces, abbreviating the values as required
Example (using an XPath expression)
Given the following:
$OCQ = Object Class Qualifier
$OC = Object Class
C:ABBREV(arg) = custom function to return the abbreviation of an argument, or the argument itself if no abbreviation, and all spaces removed
`concat( C:ABBREV($OCQ),`
`C:ABBREV($OC)`
`)`
• Dictionary Entry Name (mandatory)
  • this value shall be the concatenation of the Object Class Qualifier, followed by an underscore and space when defined, followed by the Object Class, followed by a period and space, followed by the word “Details”

COM07 Minimum set of dictionary information values describing an ABIE

Example (using an XPath expression)

Given the following:

$OCQ = Object Class Qualifier
$OC = Object Class

```
concat( if( $OCQ )
          then concat($OCQ,'_ ') else '',
        $OC,
        '. Details'
      )
```

Example ABIE

Fixed value:

Component Type="ABIE"

Entered values:

Definition="A class to define common information related to an address."
Object Class="Address"

Derived values:

Name="Address"
Dictionary Entry Name="Address. Details"

4.4.3 Dictionary information for BBIEs

COM08 Minimum set of dictionary information values describing a BBIE

Each BBIE shall have the following set of dictionary information values:

- Component Type (mandatory)
  - shall be the value "BBIE"
- Cardinality (mandatory)
  - shall be one of:
    - "1" (required and not repeatable),
    - "0..1" (optional and not repeatable),
    - "0..n" (optional and repeatable) or
    - "1..n" (required and repeatable)
- Definition (mandatory)
  - this value shall describe the BBIE using complete natural language sentences in a single paragraph
- Alternative Business Terms (optional)

COM08 Minimum set of dictionary information values describing a BBIE

- this value shall list any other commonly used terms that are synonyms for the ABIE

- Object Class Qualifier (optional)
  - this value shall qualify the object class for a specific context
- Object Class (mandatory)
  - this value shall identify the object class of the ABIE to which the BBIE belongs
- Property Term Qualifier (optional)
  - this value shall qualify the property term for a specific context
- Property Term Possessive Noun (optional)
  - this value shall identify a distinguishing nature of the characteristic of the object class
- Property Term Primary Noun (mandatory)
  - this value shall identify the principle nature of the characteristic of the object class
- Property Term (mandatory)
  - this value shall identify a characteristic of the object class as the concatenation of Property Term Possessive Noun followed by a space should it exist, followed by the Property Term Primary Noun

Example (using an XPath expression)

Given the following:

$PTPsN = Property Term Possessive Noun
$PTPrN = Property Term Primary Noun

```
concat( $PTPSN,
        if( $PTPSN )
          then ' ' else '',
        $PTPRN
      )
```

- Representation Term (mandatory)
  - this value shall identify the form of the value domain and shall be selected from the set of primary and secondary representation terms specified in CCTS Table 8.3 Permissible Representation Terms (ordered by primary term with secondary terms in parentheses):
    - Amount
    - Binary Object (Graphic, Picture, Sound, Video)
    - Code
    - Date Time (Date, Time)
    - Identifier

COM08 Minimum set of dictionary information values describing a BBIE

- Indicator
- Measure
- Numeric (Value, Rate, Percent)
- Quantity
- Text (Name)

- Name (mandatory)
  - this value shall be the concatenation of Property Term Qualifier, Property Term Possessive Noun and, when the Property Term Primary Noun is not “Text” or it is “Text” and both the Property Term Qualifier and the Property Term Possessive Noun are not defined, then the Property Term Primary Noun (abbreviating it as required) and, when the Representation Term is not “Text” and the Property Term Primary Noun is not equivalent to the Representation Term, then also the Representation Term component (abbreviating it as required), all without intervening spaces

Example (using an XPath expression)

Given the following:

$PTQ = Property Term Qualifier
$PTPsN = Property Term Possessive Noun
$PTPrN = Property Term Primary Noun
$RT = Representation Term
C:ABBREV(arg) = custom function to return the abbreviation of an argument, or the argument itself if no abbreviation, and all spaces removed, or the argument itself if no abbreviation, and all spaces removed
C:EQUIVALENT(noun,term) = custom function to return TRUE/FALSE if the primary noun and representation term are to be considered equivalent

```
concat( C:ABBREV($PTQ),
        C:ABBREV($PTPSN),
        if( $PTPRN!='Text' OR
            ( not($PTQ) AND not($PTPSN) ) )
          then C:ABBREV($PTPRN) else '',
        if( $RT!='Text' AND not(C:EQUIVALENT($PTPRN,$RT)))
          then C:ABBREV($RT) else ''
      )
```

- Dictionary Entry Name (mandatory)
  - this value shall be the concatenation of the Object Class Qualifier, followed by an underscore and space when defined, followed by the Object Class, followed by a period and space, followed by the Property Term Qualifier, followed by an underscore and space when defined, followed by the Property Term, and then, when either the Property Term Qualifier is defined or the Property Term is not equal to the Representation Term, followed by a period and space and the Representation Term

Example (using an XPath expression)

Given the following:

COM08 Minimum set of dictionary information values describing a BBIE

$OCQ = Object Class Qualifier
$OC = Object Class
$PTQ = Property Term Qualifier
$PT = Property Term
$RT = Representation Term

```
concat( $OCQ,
        if( $OCQ )
          then '_ ' else '',
        $OC,
        '. ',
        $PTQ,
        if( $PTQ )
          then '_ ' else '',
        $PT,
        if( $PTQ OR ( $PT != $RT ) )
          then concat( '. ',$RT) else ''
      )
```

- Data Type Qualifier (optional)
  - this value shall distinguish particular restrictions on a data type from the use of a data type with other (or no) restrictions
- Data Type (mandatory)
  - this value shall be the concatenation of the Data Type Qualifier followed by an underscore and space when it exists, the Representation Term, followed by a period and space, followed by the word “Type”

Example (using an XPath expression)

Given the following:

$DTQ = Data Type Qualifier
$RT = Representation Term

```
concat( $DTQ,
        if( $DTQ )
          then '_ ' else '',
        $RT,
        '. Type'
      )
```

Example BBIE

Fixed value:

Component Type=“BBIE”

Entered values:

Cardinality=“0..1”
Definition=“An additional street name used to further clarify the address.”
Object Class=“Address”
Property Term Qualifier=“Additional”
Property Term Possessive Noun=“Street”
Property Term Primary Noun=“Name”

COM08 Minimum set of dictionary information values describing a BBIE
Representation Term=“Name” Derived values: Name=“AdditionalStreetName” Dictionary Entry Name=“Address. Additional_ Street Name. Name” Property Term=“Street Name” Data Type=“Name. Type”

4.4.4 Dictionary information for ASBIEs

COM09 Minimum set of dictionary information values describing an ASBIE

Each ASBIE shall have the following set of dictionary information values:
• Component Type (mandatory)
  • shall be the value “ASBIE”
• Cardinality (mandatory)
  • shall be one of:
  • “1” (required and not repeatable),
  • “0..1” (optional and not repeatable),
  • “0..n” (optional and repeatable) or
  • “1..n” (required and repeatable)
• Definition (mandatory)
  • this value shall describe the BBIE using complete natural language sentences in a single paragraph
• Alternative Business Terms (optional)
  • this value shall list any other commonly used terms that are synonyms for the ABIE
• Object Class Qualifier (optional)
  • this value shall qualify the object class for a specific context
• Object Class (mandatory)
  • this value shall identify the object class of the ABIE to which the BBIE belongs
• Associated Object Class Qualifier (optional)
  • this value shall qualify the object class of the associated ABIE for a specific context
• Associated Object Class (mandatory)
  • this value shall identify the object class of the ABIE the ASBIE associates to
  • the ABIE must exist in the model with the same qualification (or lack thereof) as the ASBIE’s associated object class qualifier

COM09 Minimum set of dictionary information values describing an ASBIE

- Property Term Qualifier (optional)
  - this value shall qualify the property term for a specific context
- Property Term (mandatory)
  - this value shall be the concatenation of the Associated Object Class Qualifier, an underscore and a space when defined, and the Associated Object Class

Example (using an XPath expression)

Given the following:

$AOCQ = Associated Object Class Qualifier
$AOC = Associated Object Class

```
concat( $AOCQ,
        if( $AOCQ )
          then '_ ' else '',
        $AOC
      )
```

- Representation Term (mandatory)
  - this value shall be the same as the Property Term
- Name (mandatory)
  - this value shall be the concatenation of Property Term Qualifier and the Property Term without any spaces or underscore, abbreviating the values as required

Example (using an XPath expression)

Given the following:

$PTQ = Property Term Qualifier
$PT = Property Term
C:ABBREV(arg) = custom function to return the abbreviation of an argument, or the argument itself if no abbreviation, and all spaces removed, or the argument itself if no abbreviation, and all spaces removed

```
concat( C:ABBREV($PTQ),
        C:ABBREV($PT)
      )
```

- Dictionary Entry Name (mandatory)
  - this value shall be the concatenation of the Object Class Qualifier, followed by an underscore and space when defined, followed by the Object Class, followed by a period and space, followed by the Property Term Qualifier, followed by and underscore and space when defined, followed by the Property Term, and then when the Property Term Qualifier is defined, followed by a period and space and the Representation Term

Example (using an XPath expression)

Given the following:

COM09 Minimum set of dictionary information values describing an ASBIE

$OCQ = Object Class Qualifier
$OC = Object Class
$PTQ = Property Term Qualifier
$PT = Property Term
$RT = Representation Term

```
concat( $OCQ,
        if( $OCQ )
          then '_ ' else '',
        $OC,
        '. ',
        $PTQ,
        if( $PTQ )
          then '_ ' else '',
        $PT,
        if( $PTQ )
          then concat('. ',$RT) else ''
      )
```

Example ASBIE

Fixed value:

Component Type="ASBIE"

Entered values:

Cardinality="0..1"
Definition="The buyer of the item."
Object Class="Forecast"
Associated Object Class="Customer Party"
Property Term Qualifier="Buyer"

Derived values:

Name="BuyerCustomerParty"
Dictionary Entry Name="Forecast. Buyer_ Customer Party. Customer Party"
Property Term="CustomerParty"
Representation Term="CustomerParty"

4.4.5 Dictionary entry names

COM10 Dictionary entry name uniqueness

All dictionary entry names in a semantic model shall be unique.

Note

The rationale is that a BIE describes a unique semantic component of business data within a specific business context.

COM11 CCTS dictionary information item name value prohibited characters

All information items contributing to a component's dictionary entry name shall be void of all sensitive dictionary entry name markup characters.

The following characters must not be used in any dictionary information values that contribute to the dictionary entry name:

COM11 CCTS dictionary information item name value prohibited characters
• the characters “.” (period) and “_” (underscore) • leading, trailing or multiple sequential “ ” (space) characters **Note** The rationale is that prohibiting these characters in name values prevents ambiguity when assembled into the dictionary entry name using these characters to provide structure.

COM12 Use of leading upper case letter in dictionary entry name values
All words that are not abbreviated in name values contributing to the dictionary entry name shall have a leading upper-case letter and all other letters in lower-case.

4.5 Structure of an ABIE

COM13 ABIE contents cannot be empty
Every ABIE shall contain at least one BIE.

COM14 ABIE children ordering
Every ABIE shall have all of its BBIE children listed before its ASBIE children.

5 XML validation artefacts

5.1 XML validation artefacts overview

These NDR provide for expressing the semantic model of an information bundle as XML artefacts that provide for definition and validation of the structure and content of XML Documents.

There are two types of validation artefacts required for XML Documents:

1. W3C Schemas are used to define and validate *the structure* of elements and data content, and

2. OASIS CVA expressions and OASIS genericode files are used to define and validate *the values* of data content.

These rules do not require these artefacts to be located in any particular directory structure.

5.2 XML Namespaces

An important aspect of identifying and distinguishing the names used for information in XML documents is by using namespaces. Like-named constructs are distinguished by having different namespaces.

Note

Namespace abbreviations (also used here as namespace prefixes) in these examples are not mandatory and have been selected solely for convenience and consistency. Implementations of these NDR and the documents governed by these NDR are welcome to use any namespace abbreviation or namespace prefix for any of the namespaces defined or referenced.

NAM01 Namespaces for information found in information bundles
BIEs expressed in XML documents shall use the following set of namespaces: • one namespace for each Document ABIE **Note** These namespaces are not abbreviated in these examples as they are not imported or included. • one namespace for all BBIE components **Note** This namespace is abbreviated in these examples as “cbc” for “common basic components”. • one namespace for all Library ABIE components **Note** This namespace is abbreviated in these examples as “cac” for “common aggregate components”. • one namespace for the extension collection and extension metadata elements

NAM01 Namespaces for information found in information bundles

> **Note**
>
> This namespace is abbreviated in these examples as “ext”.

Each extension has a number of namespaces distinct from the document, library and other extensions.

NAM02 Namespaces for an extension

BIEs expressed in extensions shall use the following set of namespaces:

- one namespace for the apex ABIE of the extension

> **Note**
>
> This namespace is abbreviated in these examples as “myext1”.

- one namespace for all BBIE extension components that are not existing library components

> **Note**
>
> This namespace is abbreviated in these examples as “mycbc1”.

- one namespace for all ABIE extension components that are not existing Library ABIE components

> **Note**
>
> This namespace is abbreviated in these examples as “mycac1”.

> **Note**
>
> The structure of an extension parallels that of a Document ABIE with the distinct apex namespace, BBIE namespace and Library ABIE namespace. Where possible the extension should reuse existing library components that have their own namespaces. This parallel approach makes it easier to consider incorporating extension elements in future versions of the library simply by changing the namespace.

> **Note**
>
> In these abbreviations “my” is used as in the first-person possessive pronoun, and “1” implies one of multiple extension namespaces

In addition to the namespaces for the elements in XML documents, the dictionary information regarding BBIE data types is also distinguished using namespaces.

NAM03 Namespaces for BBIE data types

The expression of data type information supporting BBIE definitions shall use the following set of namespaces:

- one namespace for all qualified data types

NAM03 Namespaces for BBIE data types
Note This namespace is abbreviated in these examples as “qdt”. • one namespace for all unqualified data types (permissible representation terms) **Note** This namespace is abbreviated in these examples as “udt”. • one namespace for CCTS Core Component Type definitions **Note** This namespace is abbreviated in these examples as “ccts-cct”. **Note** These namespaces are used only for dictionary information definition and not for BIEs themselves. As such they never appear in the XML document and are therefore never declared in an XML artefact governed by these NDR.

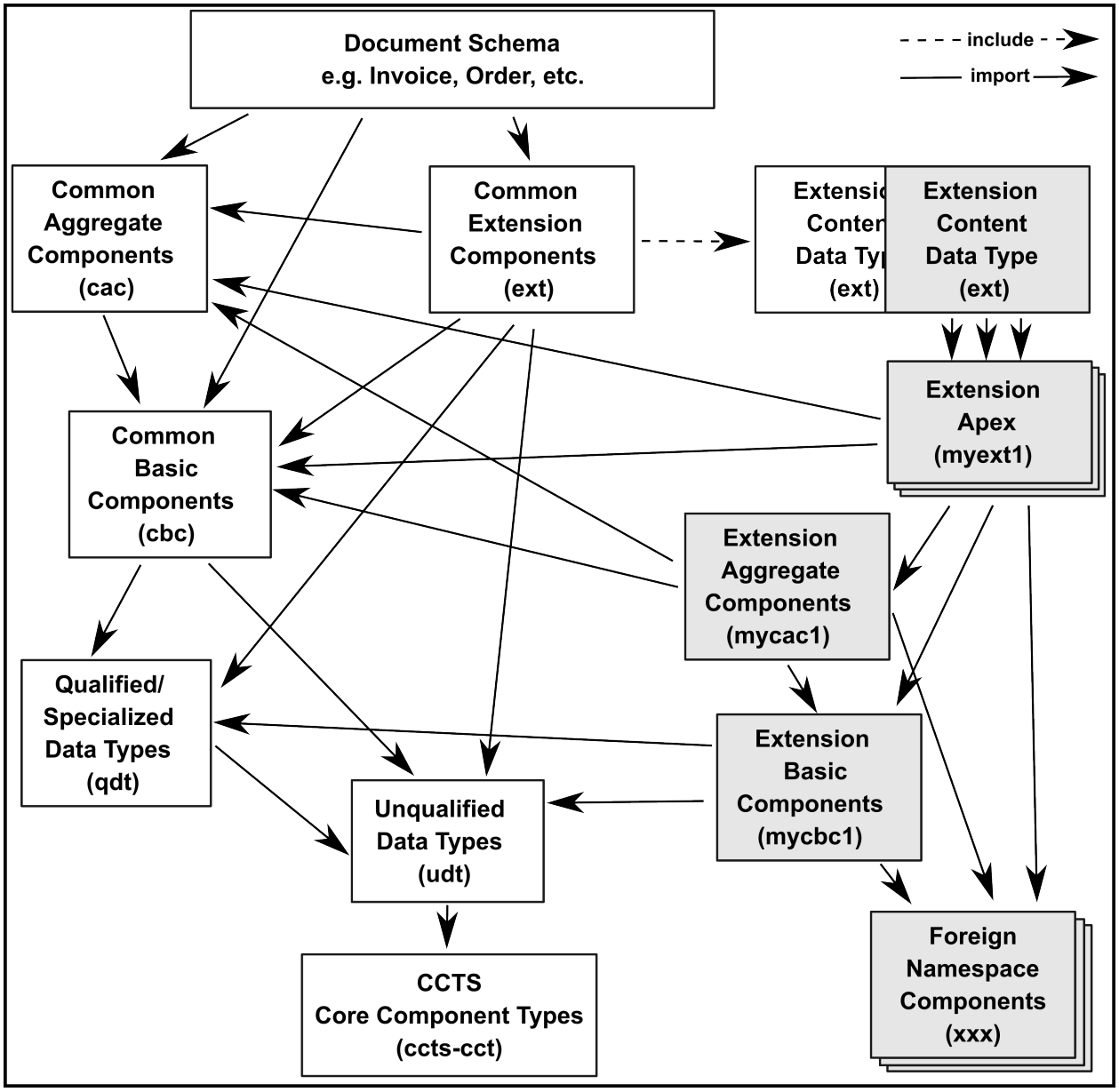
An implementation of these NDR may choose to have additional namespaces for the expression of type information.

5.3 XSD import/include tree

The relationships between the XSD schema fragments that are described in this section are shown in Figure 3, “Possible import/include hierarchy of XSD schema expressions”.

For reference purposes each box is a schema fragment and the parenthesized name tokens identify the namespace associated with the schema fragment.

Figure 3. Possible import/include hierarchy of XSD schema expressions



In this diagram the unshaded boxes represent fragments of the common schema. Once created there is no need for implementers of the schema to modify these fragments. To ensure they are not inadvertently modified, these may be marked as read-only files in the file system.

The shaded boxes represent fragments that extend the common schema. The Extension Content Data Type defines which schemas are in play for the extension point of an extension item in the extension collection. This fragment is distinguished from other fragments in that it is initially created by the designers of the schema and subsequently may be replaced by implementers.

5.4 Schema fragment annotations

There are no constraints regarding what annotation information may be added to the W3C Schema expressions in XSD files.

Good practice suggests augmenting the schema fragments with dictionary information and governance information using W3C Schema declaration annotations and XML comments. This information may be useful to the human reader or to tools exploiting the schema information in providing func-

tionality to operators, but it does not impact on the interpretation of constraints imposed on XML documents being validated with the XSD documents.

W3C Schema annotations are typically defined for and with each of the many declarations in the XSD file. Good practice suggests providing a version of the published XSD files without annotations so as not to burden runtime processing. A runtime schema processor has no use of informational annotations and may incur unnecessary processing time ingesting and accommodating the information.

Separate from the concept of W3C Schema annotations are simple XML comments that annotate schema. Such comments are ignored by schema processors and do not burden their processing. The information in these comments may be useful to implementers and, in some cases, may be required for intellectual property reasons imposed by the designers. Good practice suggests that such information includes module identification, module revision metadata and copyright declarations. The information in these comments may be useful to implementers and may by required by licensing conditions on the use of the files.

The W3C Schema version annotation (the “version” attribute of the xsd:schema element) may be used to record the release version of the collection of schema fragments.

5.5 XML schema fragments and declarations

5.5.1 XML schema fragments for Document ABIEs

FRG01 Document ABIE schema fragments
There shall be one schema fragment created for each Document ABIE.

FRG02 Document ABIE element declaration
Each Document ABIE schema fragment shall include a single element declaration, that being for the Document ABIE.

FRG03 Document ABIE type declaration
Each Document ABIE schema fragment shall include a single type declaration, that being for the content of the Document ABIE.

5.5.2 XML schema fragment for Library ABIEs

FRG04 Library ABIE schema fragment
There shall be one common schema fragment created to contain all ASBIEs (that is, from every Document ABIE and every Library ABIE) and all Library ABIEs. **Example** In UBL 2.2 the `UBL-CommonAggregateComponents-2.2.xsd` fragment serves this purpose.

FRG05 Library ABIE element declarations
The common Library ABIE schema fragment shall include an element declaration for every ASBIE in the model (that is, from every Document ABIE and every Library ABIE) and for every Library ABIE.

FRG06 Library ABIE type declarations
The common Library ABIE schema fragment shall include a type declaration for every Library ABIE, each being for the content of each Library ABIE.

There are no constraints on the order of the ABIE declarations in the schema expression.

Note

This lack of an order constraint may seem in conflict with the semantic model library constraint of alphabetical order of ABIE components. Whereas the semantic ABIE components are organized for the benefit of the human reader, the order of the schema component declarations does not affect the schema processor. However, using alphabetical order in the schema fragment may be a convenience for the human reader of the schema expressions.

5.5.3 XML schema fragment for BBIEs

FRG07 BBIE schema fragment
There shall be one common schema fragment created to describe all BBIEs in the model (that is, from every Document ABIE and every Library ABIE). **Example** In UBL 2.2 the `UBL-CommonBasicComponents-2.2.xsd` fragment serves this purpose.

FRG08 BBIE element declarations
The common BBIE schema fragment shall include an element declaration for every BBIE in the model (that is, from every Document ABIE and every Library ABIE) describing the content of each BBIE.

FRG09 Library ABIE type declarations
The one BBIE schema fragment shall include a type declaration for every BBIE in the model (that is, from every Document ABIE and every Library ABIE), each being for the content of each BBIE.

There are no constraints on the order of the BBIE declarations in the schema expression.

Note

The order of the schema component declarations does not affect the schema processor. However, using alphabetical order in the schema fragment may be a convenience for the human reader of the schema expressions.

5.5.4 XML schema declarations for all BIEs

DCL01 Element declarations
Every BIE element declaration shall be global.

DCL02 Element declaration references
Every BIE element in an ABIE type definition shall be declared by reference.

DCL03 Type declarations
Every BIE type declaration shall be global.

5.5.5 XML schema declarations for ABIEs

DCL04 ABIE element declaration

Every ABIE element shall be declared with the ABIE name as the element name and the ABIE name suffixed with “Type” as the type.

Example

```
<xsd:element name="ApplicationResponse"
             type="ApplicationResponseType"/>
```

DCL05 ABIE type declaration

Every ABIE complex type name shall be declared with the name of the ABIE suffixed with “Type” as the name.

Example

```
<xsd:complexType name="ApplicationResponseType">
 (... contents ...}
</xsd:complexType>
```

DCL06 Library ABIE type declaration content order

The members of a Library ABIE shall be ordered first with a reference to the extension collection element, followed by the sequence (in the order the BIEs appear in the semantic model of the ABIE) of all BBIE element references first, followed by all ASBIE references.

Example

```
<xsd:complexType name="CatalogueRequestLineType">
 <xsd:sequence>
  <xsd:element ref="ext:UBLExtensions"
               minOccurs="0" maxOccurs="1"/>
  <xsd:element ref="cbc:ID" minOccurs="1" maxOccurs="1"/>
  <xsd:element ref="cbc:ContractSubdivision"
               minOccurs="0" maxOccurs="1"/>
  <xsd:element ref="cbc:Note" minOccurs="0"
               maxOccurs="unbounded"/>
  <xsd:element ref="cac:LineValidityPeriod"
               minOccurs="0" maxOccurs="1"/>
  <xsd:element ref="cac:RequiredItemLocationQuantity"
               minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="cac:Item" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

DCL07 Document ABIE type declaration content order

The members of a Document ABIE shall be ordered first with a reference to the extension collection element, followed by the sequence (in the order the BIEs appear in the semantic model of the ABIE) of all BBIE element references first, followed by all ASBIE references.

DCL07 Document ABIE type declaration content order

Example

```
<xsd:complexType name="ApplicationResponseType">
  <xsd:sequence>
     <xsd:element ref="ext:UBLExtensions"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:UBLVersionID"
                      minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:CustomizationID"
                      minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:ProfileID"
                      minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:ProfileExecutionID"
                      minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:ID"
                     minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="cbc:UUID"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:IssueDate"
                     minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="cbc:IssueTime"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:ResponseDate"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:ResponseTime"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cbc:Note"
                     minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element ref="cbc:VersionID"
                     minOccurs="0" maxOccurs="1"/>
     <xsd:element ref="cac:Signature"
                     minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element ref="cac:SenderParty"
                      minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="cac:ReceiverParty"
                     minOccurs="1" maxOccurs="1"/>
     <xsd:element ref="cac:DocumentResponse"
                     minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Note

The rationale for positioning extension information at the very start of the XML instance is to allow processing applications acting sequentially on the document to consume and cache all non-modeled extension information in preparation for consuming any of the modeled document information. Should extension information follow modeled information, the sequential processing of the modeled information would have passed before recognizing the need to associate extension information. In essence, such a sequential processing application would have to cache the entire document, thus losing the benefit of sequential processing.

Note

That the rules DCL06 and DCL07 now read the same is due to the evolution of this specification. The rules for Library ABIEs and for Document ABIEs historically were different. To accommodate the future case where may be distinctions, it was decided not to replace the two rules with a common rule for all ABIEs.

DCL08 Document ABIE extension element cardinality

In the content type for every Document ABIE the extension collection element cardinality shall be declared as optional and not repeatable.

Example

```
    <xsd:element ref="ext:UBLExtensions"
                 minOccurs="0" maxOccurs="1"/>
```

5.5.6 XML schema declarations for ASBIEs

DCL09 ASBIE schema element declaration

Every ASBIE element shall be declared with the ASBIE name as the element name and the ABIE name of the associated object class suffixed with “Type” as the type.

Example

```
<xsd:element name="Party" type="PartyType"/>
```

Example

```
<xsd:element name="AgentParty" type="PartyType"/>
```

5.5.7 XML schema declarations for BBIEs

DCL10 BBIE element declaration

Every BBIE element shall be declared with the BBIE name as the element name and the concatenation of the BBIE name and “Type” as the type.

Example

```
<xsd:element name="SourceCurrencyCode"
             type="SourceCurrencyCodeType"/>
```

DCL11 BBIE unqualified type declaration

Every BBIE element type with an unqualified data type shall be declared as simple content restricted from a base of the corresponding unqualified data type without the addition of any additional attributes.

Example

```
<xsd:complexType name="SourceCurrencyBaseRateType">
   <xsd:simpleContent>
      <xsd:restriction base="udt:RateType"/>
   </xsd:simpleContent>
</xsd:complexType>
```

DCL12 BBIE qualified type declaration
Every BBIE element type with a qualified data type shall be declared as simple content restricted from a base of the corresponding qualified data type without the addition of any additional attributes.

Example

```
<xsd:complexType name="SourceCurrencyCodeType">
   <xsd:simpleContent>
      <xsd:restriction base="qdt:CurrencyCodeType"/>
   </xsd:simpleContent>
</xsd:complexType>
```

5.5.8 XML schema declarations for Qualified Data Types

DCL13 Qualified data type declaration
Every qualified data type shall be declared as simple content restricted from a base of the corresponding unqualified data type without the addition of any additional attributes.

Example

```
<xsd:complexType name="CurrencyCodeType">
   <xsd:simpleContent>
      <xsd:restriction base="udt:CodeType"/>
   </xsd:simpleContent>
</xsd:complexType>
```

5.6 Extension XML schema fragments and declarations

5.6.1 Extension information in XML

The content type for a Document ABIE contains a single optional extension collection element in order to provide for the inclusion of data in the XML that is in addition to the data of the information bundle for the document. Such data may include content designed by other organizations (e.g. signature information) as well as augmentations of the semantic model.

An extension collection element contains one or more extension elements. Each extension element has a suite of metadata elements used to describe the extension. The extension content may reuse existing ABIEs or BBIEs and may contain XML content not modeled as BIEs.

The extension collection and metadata are XML elements implemented as schema fragments and constructs independent of the semantic model.

5.6.2 Extension collection schema fragments and declarations

EXT01 Extension collection schema fragment
The extension collection schema fragment shall include the declarations of the extension collection element, the extension element, the extension content element, the extension metadata elements and any required type information for metadata elements that are not BIEs in the Document ABIEs and Library ABIEs.

EXT01 Extension collection schema fragment

Example

In UBL 2.2 the `UBL-CommonExtensionComponents-2.2.xsd` fragment serves this purpose.

EXT02 Extension content element declaration

The extension collection schema fragment shall include the declaration of the mandatory extension content element, but not the type information for the extension content element.

Note

The rationale for not including the type information for the extension point element is that the type is subject to change, while the extension collection, the extension item and extension item metadata element and type information is not. This separation allows the extension collection and item schema fragment to be deployed as read-only, while the extension point data type schema fragment can be deployed as writable in order to be defined by users.

EXT03 Extension collection element content

The document's extension collection shall have one or more extension elements as its content.

Example

```
<xsd:complexType name="UBLExtensionsType">
  <xsd:sequence>
    <xsd:element ref="UBLExtension"
          minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

Note

The rationale is that different users of a document may have different extension items added to the content. Also, different extensions may be thematically distinguished (e.g. the digital signature extension is semantically separate from an extension augmenting invoice line content).

EXT04 Extension element content ordering

The extension element shall declare all available metadata elements (if any) in advance of a last mandatory single extension content element being the extension point under which the extension information is added to the document.

Example

```
<xsd:complexType name="UBLExtensionType">
  <xsd:sequence>
    <xsd:element ref="cbc:ID"
                 minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="cbc:Name"
                 minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionAgencyID"
```


EXT04 Extension element content ordering

```
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionAgencyName"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionVersionID"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionAgencyURI"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionURI"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionReasonCode"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionReason"
                  minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="ExtensionContent"
                  minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

Note

There are no constraints on the selection, name, definition or cardinality of the extension metadata elements.

5.6.3 XML schema fragment for the extension content data type declaration

EXT05 Extension content data type schema fragment

The extension content element type schema fragment shall include the declaration of the content type for the extension content element and any import statements defining constraints on recognized constructs.

Example

In UBL 2.2 the `UBL-ExtensionContentDataType-2.2.xsd` fragment serves this purpose.

EXT06 Extension content data type declaration

The extension content element type schema fragment shall contain only a single complex type declaration being a sequence of exactly one element in a namespace other than the extension namespace to be processed with lax validation.

Example

```
  <xsd:complexType name="ExtensionContentType">
    <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax"
               minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
```

EXT06 Extension content data type declaration

Note

The rationale for the lax validation is to allow for the extension point to contain, without error, any element for which there are no constraints in any schema fragment imported or included in the validation step.

EXT07 Extension content data type imports

The extension content element type schema fragment may contain import directives for the expected data content of an extension.

Example

```
<xsd:import namespace="urn:oasis:names:specification:
ubl:schema:xsd:CommonSignatureComponents-2"
  schemaLocation="UBL-CommonSignatureComponents-2.2.xsd"/>
```

Note

The rationale for including import directives is to validate those constructs found in an extension that are expected.

Note

There is no order to the import directives.

5.7 Qualified data types XML schema fragment and declarations

QDT01 Qualified data types schema fragment

The qualified data types schema fragment shall include the declarations of any qualified data types referenced in the schema fragment for BBIEs.

Example

In UBL 2.2 the `UBL-QualifiedDataTypes-2.2.xsd` fragment serves this purpose.

QDT02 Qualified data type declaration name

Every qualified data type shall be declared using the name of the data type qualifier followed by the unqualified data type name.

QDT03 Qualified data type declaration basis

Every qualified data type shall be based on an unqualified data type, imposing whatever qualifications are required to be expressed using XSD schema semantics.

Note

In UBL 2.2 there are no qualifications expressed using XSD schema semantics.

QDT04 Qualified data type declaration constraint
Every qualified data type declaration shall be such that every possible instance of the declared type is also an instance of the base type. **Note** This constraint prevents additions of anything that is not part of the base type, such as the introduction of any new attributes or sub-elements, or any less-constrained element or attribute values.

5.8 Unqualified data types XML schema fragment and declarations

UDT01 Unqualified data types schema fragment
The unqualified data types schema fragment shall include the declarations of all unqualified data types referenced in the schema fragment for BBIEs. **Example** The `BDNDR-UnqualifiedDataTypes-1.1.xsd` is an example fragment that serves this purpose.

UDT02 Unqualified data types declaration inclusions
An unqualified data type shall be declared for every one of the permitted Primary Representation Terms and Secondary Representation Terms defined as Permissible Representation Terms in the Core Component Technical Specification **[CCTS]**.

UDT03 Unqualified data types declaration exclusions
Unqualified data types shall only be declared for the permitted Primary Representation Terms and Secondary Representation Terms defined as Permissible Representation Terms in the Core Component Technical Specification **[CCTS]**.

UDT04 Unqualified data types declaration base
Every unqualified data type shall be either a restriction on one of the approved Core Component Types defined in the Core Component Technical Specification **[CCTS]**, or an extension of a base XSD data type. **Note** This constraint may be accomplished by either restricting a declaration imported from the Core Component Types schema or by wholly replacing the corresponding Core Component Types schema declaration. **Note** The rational for having UDT declarations is to impose some XSD syntax semantics on top of the more general decimal and string lexical syntax defined in the CCTS specification of Core Component Types. For example, the CCTS Core Component Type for date and time is a simple string without constraint. Such lax structuring of the field value does not serve users in that no particular format is obligated. The UDT can impose, for example,

UDT04 Unqualified data types declaration base

the xsd:dateTime lexical syntax on all date and time values, overriding the CCTS definition.

Note

This rule does allow an optional supplementary component defined in the CCTS Core Component Type not to be available in the associated unqualified data type. For example, if the UDT implements a built-in XSD data type for the component then there is no use of a format supplementary component and associated attribute and so the format attribute declaration can be omitted and, thereby, be unavailable for use for that data type.

Example 1

In this example the unqualified data type uses the base type without change:

```
<xsd:complexType name="NumericType">
  <xsd:simpleContent>
    <xsd:restriction base="ccts-cct:NumericType"/>
  </xsd:simpleContent>
</xsd:complexType>
```

Example 2

In this example the unqualified data type redeclares the base type's optional attribute as mandatory:

```
<xsd:complexType name="MeasureType">
  <xsd:simpleContent>
    <xsd:restriction base="ccts-cct:MeasureType">
      <xsd:attribute name="unitCode"
                     type="xsd:normalizedString"
                     use="required"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

Example 3

In this example the unqualified data type replaces the base type with no attributes and with an XSD built-in type for content:

```
<xsd:complexType name="DateTimeType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:dateTime"/>
  </xsd:simpleContent>
</xsd:complexType>
```

UDT05 Unqualified data types declaration constraint

Every unqualified data type declaration that is not a formal restriction of one of the Core Component Type declarations defined in the Core Component Technical Specification [CCTS] schema fragment shall be such that every possible instance of the declared type is also an instance of one of the Core Component Types as defined in CCTS.

UDT05 Unqualified data types declaration constraint

Note

This constraint prevents additions of anything that is not part of the base Core Component Type, such as the introduction of any new attributes or sub-elements, or any less-constrained element or attribute values.

Example

In this example the unqualified data type for a date is based on the XSD data type for date and all instances of the date data type are also instances of the string-based `DateTimeType` data type in CCTS:

```
<xsd:complexType name="DateType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:date"/>
  </xsd:simpleContent>
</xsd:complexType>
```

5.9 CCTS Core Component Types XML schema

All primitive data types correspond to the 10 CCTS Primary Representation Terms defined in **[CCTS]** Section 8-3 “Permissible Representation Terms”.

CCT01 CCTS Core Component Type schema

The core component type schema of primitive data types for primary representation terms on which the unqualified data types are based shall be an unmodified copy of the schema fragment published by UN/CEFACT with the following embedded title and metadata:

```
CCTS Core Component Type Schema Module

Module of Core Component Type
Agency: UN/CEFACT
VersionID: 1.1
Last change: 14 January 2005
```

Example

The `BDNDR-CCTS_CCT_SchemaModule-1.1.xsd` is an example fragment that serves this purpose.

5.10 XML attribute names

ATT01 Leading name part in attribute names

Every attribute’s derived name shall be composed with the leading name part entirely in lower-case, even when that name part is an agreed-upon abbreviation.

Example

The data type of the BBIE known as “Binary Object. Uniform Resource. Identifier” is represented with the attribute named “uri”

ATT01 Leading name part in attribute names

Note

Terms used in attribute names of supplementary components of CCTS Table 8-2 “Approved Core Component Type Content and Supplementary Components” are subject to abbreviation.

ATT02 Non-leading abbreviations in attribute names

When an attribute’s derived name is composed with an agreed-upon abbreviation in other than the leading name part, the abbreviation shall be used unchanged.

Examples

The data type of the BBIE known as “Amount Currency. Identifier” is represented with the attribute named “currencyID”
The data type of the BBIE known as “Amount Currency. Code List Version. Identifier” is represented with the attribute named “currencyCodeListVersionID”

Note

Terms used in attribute names of supplementary components of CCTS Table 8-2 “Approved Core Component Type Content and Supplementary Components” are subject to abbreviation.

5.11 Data type qualifications in XML

Data types may be qualified to define additional constraints on the possible values of the BBIEs of that data type. These constraints may be subject to change over time and so should be applied in a manner that allows modification of the data type qualifications without impacting the schema.

Code lists and identifier lists are examples of controlled sets of values (e.g. currency codes, country codes, product identifiers, etc.).

For some communities of users (e.g. those with business-oriented XML documents) the management of controlled vocabularies presents particular challenges for document modeling. While communities may standardize document structures, trading partners within the community have their own constraints that may change on an hourly basis yet must work within the community framework.

Externalizing the list in a genericode file expresses the enumeration as a resource available to the application for data entry. However, the choice of genericode file may vary on a per-information item basis due to the item’s document context, or perhaps vary again for particular trading partners. Expressing the appropriate mappings in a colloquial fashion inhibits interoperability and the sharing of resources and program code.

A context/value association file specifies the relationship from information items found in different document contexts to one or more external genericode files for each item. With these relationships a directed authoring environment can precisely direct the editing of individual information items. Different context/value association files can then be used to create instances for different purposes that have different constraints on the enumerations used therein.

DTQ01 Data type qualification CVA file

Data type qualifications that are not expressed as qualified data types using XSD schema semantics may be expressed using the OASIS Context Value Association **[CVA]** XML vocabulary.

DTQ01 Data type qualification CVA file

Note

The CVA file provides for users to associate value validation constraint expressions and/or coded domain value enumerations with different CVA contexts. A value validation constraint is expressed using XPath. A coded domain value enumeration is expressed using one or the union of more than one OASIS genericode file.

Note

The CVA expressions typically are not used at runtime. More likely the CVA expressions and their associated genericode files would be processed or compiled into a runtime validation artefact. These rules do not constrain where this runtime artefact would be kept, but good practice suggests a location separate from the XSD schemas.

Example

In UBL 2.2 the CVA expression is the `cva/UBL-DefaultDTQ-2.2.cva` file, the associated runtime artefact is the `val/UBL-DefaultDTQ-2.2.xsl` XSLT stylesheet and the referenced genericode files are located in the `cl/` subdirectory of each of the UBL 2.2 distribution, the UBL 2.1 distribution and the UBL 2.0 distribution.

DTQ02 Data type element content qualifications

A CVA context shall be created for every BBIE with a non-empty value in the CCTS dictionary information for the data type qualifier.

Example

Entered dictionary information values:

Name=“ChannelCode”
Data Type Qualifier=“Channel”
Representation Term=“Code”
Data Type=“Channel_ Code. Type”

In this example the constraints on the value of the `cbc:ChannelCode` element are described by the union of three constraint expressions identified by “`Channel-2.0`”, “`Channel-2.1`” and “`Channel-2.2`”:

```
<Context values="Channel-2.0 Channel-2.1 Channel-2.2"
         metadata="cctsV2.01-code"
         address="cbc:ChannelCode"/>
```

Example

In this example the constraints on the value of the `cbc:PayableAmount` element child of the `cac:LegalMonetaryTotal` element are described by the single constraint expression identified by “maxValue”:

```
<Context values="maxValue"
         address="cac:LegalMonetaryTotal/
                  cbc:PayableAmount"/>
```


DTQ03 Data type attribute content qualifications

A CVA context shall be created for every CCTS supplementary component to be validated.

Example

In this example the constraints on the value of the `currencyID` attribute are described by the union of two constraint expressions identified by "`Currency-2.0`", "`Currency-2.1`" and "`Currency-2.2`":

```
<Context values="Currency-2.0 Currency-2.1 Currency-2.2"
         metadata="cctsV2.01-amount"
         address="@currencyID"/>
```

DTQ04 Value test constraints

A CVA value test constraint shall be written as an XPath expression.

Example

In this example the constraint on the value is that its numeric value be less than 10,000:

```
<ValueTest xml:id="maxValue"
           test=". &lt; 10000"/>
```

DTQ05 Value list constraints

A CVA value list constraint shall reference an OASIS genericode **[genericode]** file.

Example

```
<ValueList xml:id="Channel-2.2"
  uri="../cl/gc/default/ChannelCode-2.2.gc"/>
```

Note

Each genericode file defines the metadata associated with the enumeration of values therein. Therefore, the union of multiple genericode files is required when the constraint includes values from different enumerations associated with distinctive metadata.

DTQ06 Value metadata association

The CVA instance metadata sets shall identify the XML attributes used in Open-edi user data that are associated with the supplementary components of the unqualified data types.

Example

In UBL 2.2 the `UBL-DefaultDTQ-2.2.cva` fragment includes such declarations.

6 JSON validation artefacts

6.1 JSON serialization modes

6.1.1 JSON serialization modes overview

Recognizing there are different applications of the use of JSON syntax in different programming and data scenarios, these NDR for JSON have evolved to embrace the concept of three serialization modes of CCTS information into JSON serialized syntax: legacy mode, schema mode, and instance mode.

Users can assess which serialization mode meets their information requirements in a particular scenario. Each serialization mode is built on the same conventions for naming, so as to promote familiarity across the use of all modes. The modes are not directly interchangeable as each exhibit unique properties in the serialization that have impacts on JSON validation.

6.1.2 JSON legacy-mode serialization

The JSON legacy-mode serialization protects a given serialization of an instance of CCTS content from future changes in cardinality.

In this mode, the JSON array structure of object structures is used in the serialization of each and every BIE in a document instance, regardless of the declared cardinality in the CCTS model. This is reflected in the JSON legacy schema for the CCTS model.

Any given legacy JSON instance is guaranteed to be JSON-schema-valid to the current and all future JSON legacy schemas. This is due to the backward-compatibility of future CCTS models.

These NDR specify the creation of JSON legacy schemas to be used to validate instances following legacy-mode serialization.

6.1.3 JSON model-mode serialization

The JSON model-mode serialization presents a given serialization of an instance of CCTS according to the current version of the CCTS model, respecting the declared cardinality found in the current model.

In this mode, the JSON array structure is used in the serialization only for those BIE constructs with an unbounded maximum cardinality. The JSON array structure is not used for the serialization of any BIE construct with a maximum cardinality of 1. The JSON object structure is used for the serialization of any BIE construct with a maximum cardinality of 1. This is reflected in the JSON model schema for the particular version of the CCTS model.

Any given version's JSON instance is guaranteed to be schema-valid only to that version's JSON model schema and not necessarily to any past or future version's JSON model schema. This is due to the possibility that the current or future version of the CCTS model may have increased the upper bound of the cardinality of any given BIE. Such would change the serialization from a single object structure to an array structure of object structures, and so the current or future version JSON model schema would not successfully validate the older version JSON model-mode serialization.

These NDR specify the creation of version-specific JSON model schemas to be used to validate instances following model-mode serialization.

6.1.4 JSON instance-mode serialization

The JSON instance-mode serialization is the most parsimonious of the three serialization modes. A string value is used for the content of a BBIE that does not have any supplementary components

associated with the content. An object structure is used for the content of a singleton BBIE that does have supplementary components associated with the content. An object structure is used for the content of a singleton ASBIE. An array construct of object constructs is used for an adjacent sequence of like-named BBIEs and for an adjacent sequence of like-named ASBIEs.

These NDR do not specify any creation of instance-mode JSON schemas to be used for validation purposes. It is up to the ingesting application to determine the structural validity and content of a given construct on-the-fly.

6.1.5 JSON serialization mode schema declaration differences

Legacy-mode and model-mode schemas are predominantly identical except only for the applications of NDR DCL24 ASBIE property declaration in an ABIE object and DCL26 BBIE property declaration in an ABIE object that differ based on creating an array of objects or a single object.

6.2 JSON validation artefacts overview

These NDR provide for expressing the semantic model of an information bundle as two sets of JSON Schema **[JSON Schema]** artefacts that support the definition and validation of the structure and content of JSON documents in support of legacy-mode serialization and model-mode serialization..

These rules do not require these artefacts to be located in any particular directory structure.

6.3 JSON preservation of XML namespaces

An important aspect of identifying and distinguishing the names used for information in XML documents is by using namespaces. Like-named constructs are distinguished by having different namespaces. These NDR provide for preserving in the JSON structure namespaces from an XML document transliterated to JSON following these rules. Such preservation supports round-tripping, that is, converting the transliterated JSON instance back to an XML document in a lossless fashion, but without consideration for foreign extension content. See Section A.3, “Sample schema and code fragments” for an example of an XSLT stylesheet that converts an instance of XML into JSON.

NAM21 Namespaces for information found in information bundles
A JSON instance shall provide for preserving the following XML document namespaces for BIEs: • one namespace for each Document ABIE • one namespace for all BBIE components • one namespace for all Library ABIE components • one namespace for the extension collection and extension metadata elements

A name used in any given group is not prohibited from also being used in another group. For example, in the UBL vocabulary there is a Library ABIE named “Location” and a BBIE named “Location”. The naming and design rules provide a mechanism, not based on namespace URI strings, to disambiguate names that might be sibling name/value pairs in an object.

When two name/value pairs in a given JSON object have the same name but one of them is an ASBIE and the other is a BBIE, each name must be suffixed with, respectively, either “_A” or “_B” as appropriate and other names not in conflict shall not be suffixed. If all names of name/value pairs in a given JSON object are distinct, then all of the names shall have no suffix.

Note

A consequence of these rules is that when a CCTS-defined business vocabulary has no ABIE where a member ASBIE and a member BBIE have the same name, every possible JSON serialization will have no name suffixes.

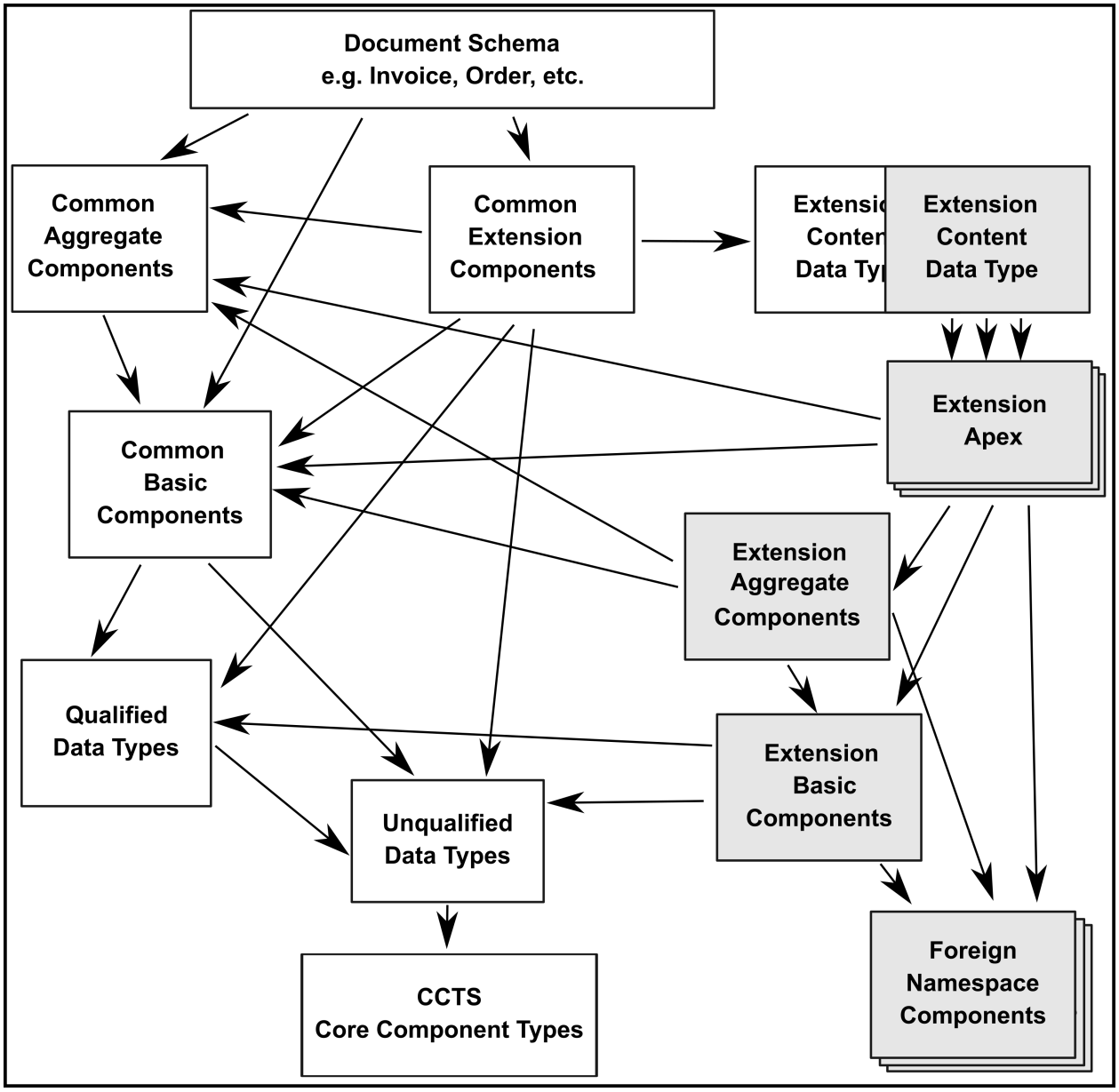
Note

Users of a CCTS-defined business vocabulary will know where these rules may come into play before writing the application and so, for any given version of the vocabulary, an application writer will know when it may be necessary to check for a given name with either a suffixed name or a name without a suffix. If a future version of the vocabulary introduces a potential name collision, legacy applications may not be in a position to recognize the new use of a suffix. However, from a maintenance perspective, the application developer need only look for the use of those names where the collision has been introduced.

6.4 JSON schema file references

The JSON schema expressions are fragmented in order that like-named items in each role can be declared without the use of prefixes or suffixes. This fragmentation mimics that found in the XML schemas illustrated in Figure 3, “Possible import/include hierarchy of XSD schema expressions”.

Figure 4. JSON schema fragment reference tree



6.5 JSON schema fragment annotations

JSON schema expressions provide only for title and description annotations in addition to the value constraint properties. The top-level title and description of each fragment shall include an overall description title for that fragment and any additional global information such as source and copyright details.

Within the fragments, these annotations shall be present only on those declarations of Document and Library ABIE definitions and each of their constituent ASBIE and BBIE members as arrays, not on the declarations of the content of the ASBIE and BBIE objects that define the members of the arrays. The "`title`" annotation shall be CCTS Dictionary Entry Name for the BIE. The "`description`" annotation shall be the CCTS definition value.

6.6 JSON Schema fragments and declarations

6.6.1 JSON schema fragments for Document ABIEs

FRG21 Document ABIE JSON schema fragments
There shall be one JSON schema fragment created for each Document ABIE, declared to be a "JSON draft v4 schema" with the appropriate "`$schema`" string property defined as "`http://json-schema.org/draft-04/schema#`" in addition to the annotation title and description for the fragment.

FRG22 Document ABIE object namespace declarations
Each Document ABIE JSON schema fragment shall include a root schema object declaration including declarations for four properties as string values, all optional, those being for namespaces of record providing for transliteration with XML instances if desired. The string values shall be named: • "`_D`" for the Document ABIE namespace • "`_A`" for the Library ABIE namespace used for ASBIEs • "`_B`" for the BBIE namespace • "`_E`" for the extension scaffolding namespace

FRG23 Document ABIE object reference declaration
The Document ABIE JSON schema fragment root schema object declaration shall include a single required property declaration for a array of maximum and minimum one item, named for the Document ABIE, referencing its definition locally in the file under the "`definitions`" object. **Note** While the prescribed referencing provides no additional facility for a Document ABIE when compared to an inline object definition, this declaration pattern mimics the declaration pattern that is required for Library ABIEs, and thus is required here simply for consistency. Generation tools may find this a convenience.

FRG24 Document ABIE object definition declaration
Each Document ABIE schema fragment shall include a "`definitions`" object that contains a single object definition declaration, that being for the content of the Document ABIE.

6.6.2 JSON schema fragment for Library ABIEs

FRG25 Library ABIE JSON schema fragment
There shall be one common schema fragment created to contain all ASBIEs (that is, from every Document ABIE and every Library ABIE) and all Library ABIEs. This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

FRG25 Library ABIE JSON schema fragment
Example In UBL 2.2 the `UBL-CommonAggregateComponents-2.2.json` fragment serves this purpose.

FRG26 Library ABIE object reference declarations
The one Library ABIE schema fragment shall include in its "`definitions`" object an object reference declaration for every ASBIE in the model (that is, from every Document ABIE and every Library ABIE) that is not named the same as its corresponding Library ABIE. Each such declaration shall reference its corresponding Library ABIE without any title or description properties.

FRG27 Library ABIE object definition declarations
The one Library ABIE schema fragment shall include in its "`definitions`" object an object definition declaration for every Library ABIE, each being for its content.

There are no constraints on the order of the ABIE declarations in the schema expression.

Note

This lack of an order constraint may seem in conflict with the semantic model library constraint of alphabetical order of ABIE components. Whereas the semantic ABIE components are organized for the benefit of the human reader, the order of the schema component declarations does not affect the schema processor. However, using alphabetical order in the schema fragment may be a convenience for the human reader of the schema expressions.

6.6.3 JSON schema fragment for BBIEs

FRG28 BBIE JSON schema fragment
There shall be one common schema fragment created to describe all BBIEs in the model (that is, from every Document ABIE and every Library ABIE). This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment. **Example** In UBL 2.2 the `UBL-CommonBasicComponents-2.2.json` fragment serves this purpose.

FRG29 BBIE object definition declarations
The one BBIE schema fragment shall include a "`definitions`" object that contains an object definition declaration for every BBIE in the model (that is, from every Document ABIE and every Library ABIE). Each such declaration shall reference its corresponding qualified or unqualified data type without any title or description information.

There are no constraints on the order of the BBIE declarations in the schema expression.

Note

The order of the schema component declarations does not affect the schema processor. However, using alphabetical order in the schema fragment may be a convenience for the human reader of the schema expressions.

6.6.4 JSON schema declarations for ABIEs

DCL21 ABIE object declaration

Every ABIE shall be declared as an object named by the CCTS Component Name of the ABIE. It shall have as its title the CCTS Dictionary Entry Name. It shall have as its description the CCTS Definition. It shall declare as the "`required`" property the names of the ASBIE and BBIE children whose model cardinality has a minimum bound of 1. Its properties shall be the ASBIE and BBIE declarations of the ABIE's children. It shall declare that additional properties are not permitted.

Example

```
"CatalogueRequestLine": {
  "title": "Catalogue Request Line. Details",
  "description": "A class to define a line describing a
  request for a catalogue line.",
  "required": [
    "ID",
    "Item"
    ],
  "properties": {...},
  "additionalProperties": false,
  "type": "object"
  },
```

DCL22 Library ABIE object declaration content order

The members of a Library ABIE shall be ordered as the sequence (in the order the BIEs appear in the semantic model of the ABIE) of all BBIE children references first, followed by all ASBIE children references.

Example

```
"CatalogueRequestLine": {
  "title": "Catalogue Request Line. Details",
  "description": "A class to define a line describing a
  request for a catalogue line.",
  "required": [
    "ID",
    "Item"
    ],
  "properties": {
    "UBLExtensions": {
      "title": "UBLExtensions",
      "description":
"An optional set of extensions to the committee model",
      "items": {
        "$ref":
"../common/UBL-CommonExtensionComponents-2.2.json#/
definitions/UBLExtensions" },
      "maxItems": 1,
```

DCL22 Library ABIE object declaration content order

```
      "minItems": 1,
      "type": "array"
      },
    "ID": {..."items": {
        "$ref":
"UBL-CommonBasicComponents-2.2.json#/definitions/ID"
        },...},
    "ContractSubdivision": {..."items": {
        "$ref":
"UBL-CommonBasicComponents-2.2.json#/definitions/
ContractSubdivision"
        },...},
    "Note": {..."items": {
        "$ref":
"UBL-CommonBasicComponents-2.2.json#/definitions/Note"
        },...},
    "LineValidityPeriod": {..."items": {
        "$ref": "#/definitions/Period"
        },...},
    "RequiredItemLocationQuantity": {..."items": {
        "$ref": "#/definitions/ItemLocationQuantity"
        },...},
    "Item": {..."items": {
        "$ref": "#/definitions/Item"
        },...}
    },
  "additionalProperties": false,
  "type": "object"
  },
```

Note

Although the order of properties of a JSON object is not relevant, ordering the properties as prescribed is consistent with both the CCTS model and the XML serialization of the model. For the human reader of the JSON schema, having this order in the schema declarations should be helpful.

DCL23 Document ABIE object declaration content order

The members of a Document ABIE shall be ordered first with a reference to the extension collection object array, followed by the sequence (in the order the BIEs appear in the semantic model of the ABIE) of all BBIE children references first, followed by all ASBIE children references. The extension collection array shall have a minimum and maximum number of items of 1 and shall not allow additional properties. The array shall not be listed in the "`required`" property.

Example

```
"ApplicationResponse": {
  "title": "Application Response. Details",
  "description": "A document to indicate the
application's response to a transaction. This may be a
business response initiated by a user or a technical
response sent automatically by an application.",
  "required": [
    "ID",
    "IssueDate",
```


DCL23 Document ABIE object declaration content order

```
    "SenderParty",
    "ReceiverParty"
    ],
  "properties": {

    "UBLExtensions": {
      "title": "UBLExtensions",
      "description":
"An optional set of extensions to the committee model",
      "items": {
        "$ref":
"../common/UBL-CommonExtensionComponents-2.2.json#/
definitions/UBLExtensions"
        },
      "maxItems": 1,
      "minItems": 1,
      "type": "array"
      },

    "UBLVersionID": {...},
    "CustomizationID": {...},
    "ProfileID": {...},
    "ProfileExecutionID": {...},
    "ID": {...},
    "UUID": {...},
    "IssueDate": {...},
    "IssueTime": {...},
    "ResponseDate": {...},
    "ResponseTime": {...},
    "Note": {...},
    "VersionID": {...},
    "Signature": {...},
    "SenderParty": {...},
    "ReceiverParty": {...},
    "DocumentResponse": {...}
    },
  "additionalProperties": false,
  "type": "object"
  }
```

Note

Although the order of properties of a JSON object is not relevant, ordering the properties as prescribed is consistent with both the CCTS model and the XML serialization of the model. For the human reader of the JSON schema, having this order in the schema declarations should be helpful.

6.6.5 JSON schema declarations for ASBIEs

DCL24 ASBIE property declaration in an ABIE object

Legacy-mode declaration:

Every ASBIE child of an ABIE shall be declared as an array named by the CCTS Component Name of the ASBIE. It shall have as its title the CCTS Dictionary Entry Name. It shall have as its description the CCTS Definition. It shall have a minimum number of items as 1. If the cardinality has a maximum bound of 1, then

DCL24 ASBIE property declaration in an ABIE object
the declaration shall have a maximum number of items as 1, otherwise there shall be no constraint on the maximum number of items. It shall declare that additional properties are not permitted. The items of the array shall be declared by referencing the ASBIE in the Library ABIE schema fragment.

Example

```
"SenderParty": {
  "title": "Application Response. Sender_ Party. Party",
  "description": "The party sending this document.",
  "items": {
    "$ref":
"../common/UBL-CommonAggregateComponents-2.2.json#/
definitions/SenderParty"
    },
  "maxItems": 1,
  "minItems": 1,
  "type": "array",
  },
```

Model-mode declaration:

All ASBIE children with an unbounded maximum cardinality of ‘n’ are declared as arrays in the manner used for a legacy-mode declaration.

All ASBIE children with a bounded maximum cardinality of “1” are declared as an object referencing the ASBIE in the Library ABIE schema fragment.

Example

```
"SenderParty": {
  "title": "Application Response. Sender_ Party. Party",
  "description": "The party sending this document.",
  "$ref":
"../common/UBL-CommonAggregateComponents-2.2.json#/
definitions/SenderParty"
  },
```

DCL25 ASBIE declaration in the Library ABIE JSON schema fragment
In the Library ABIE schema fragment, every ASBIE child of an ABIE shall be declared by referencing the ASBIE’s associated ABIE within the same fragment. There shall be no title, description or other properties.

Example

```
    "SenderParty": {
      "$ref": "#/definitions/Party"
      },
```

6.6.6 JSON schema declarations for BBIEs

DCL26 BBIE property declaration in an ABIE object
Legacy-mode declaration:

DCL26 BBIE property declaration in an ABIE object

Every BBIE child of an ABIE shall be declared as an array named by the CCTS Component Name of the BBIE. It shall have as its title the CCTS Dictionary Entry Name. It shall have as its description the CCTS Definition. It shall have a minimum number of items as 1. If the cardinality has a maximum bound of 1, then the declaration shall have a maximum number of items as 1, otherwise there shall be no constraint on the maximum number of items. It shall declare that additional properties are not permitted. The items of the array shall be declared by referencing the BBIE declaration in the BBIE schema fragment using the CCTS Component Name of the BBIE.

Example

```
"ResponseDate": {
  "title": "Application Response. Response Date. Date",
  "description": "The date on which the information in
the response was created.",
  "items": {
    "$ref":
"../common/UBL-CommonBasicComponents-2.2.json#/
definitions/ResponseDate"
    },
  "maxItems": 1,
  "minItems": 1,
  "type": "array",
  },
```

Model-mode declaration:

All BBIE children with an unbounded maximum cardinality of ‘n’ are declared as arrays in the manner used for a legacy-mode declaration.
All BBIE children with a bounded maximum cardinality of “1” are declared as an object referencing the BBIE declaration in the BBIE schema fragment using the CCTS Component Name of the BBIE.

Example

```
"ResponseDate": {
  "title": "Application Response. Response Date. Date",
  "description": "The date on which the information in
the response was created.",
  "$ref":
"../common/UBL-CommonBasicComponents-2.2.json#/
definitions/ResponseDate"
  },
```

DCL27 BBIE declaration in the BBIE JSON schema fragment

In the BBIE schema fragment, every BBIE shall be declared by referencing the BBIE’s type’s declaration in either the qualified data type fragment or the unqualified data type fragment as required. There shall be no title, description or other properties.

Example of a BBIE declaration with a qualified data type

```
"SourceCurrencyCode": {
  "$ref":
```


DCL27 BBIE declaration in the BBIE JSON schema fragment

```
"UBL-QualifiedDataTypes-2.2.json#/definitions/
Currency_CodeType"
  },
```

Example of a BBIE declaration with an unqualified data type

```
"SourceCurrencyBaseRate": {
  "$ref":
"BDNDR-UnqualifiedDataTypes.json#/definitions/
RateType"
  },
```

6.6.7 JSON schema declarations for Qualified Data Types

DCL28 Qualified data type declaration in the qualified data type JSON schema fragment

Every qualified data type shall be declared with its name being the type's Dictionary Entry Name compressed with all periods and spaces removed. It shall have as its title the type's Dictionary Entry Name. It shall reference the associated unqualified data type in the unqualified data type schema fragment. There shall be no description or other properties.

Example

```
"Currency_CodeType": {
  "title": "Currency_ Code. Type",
  "$ref":
"BDNDR-UnqualifiedDataTypes.json#/definitions/
CodeType"
  },
```

6.7 Extension JSON schema fragments and declarations

6.7.1 Extension information in JSON

The content type for a Document ABIE contains a single optional extension collection object defined as an array of exactly one extension property in order to provide for the inclusion of data in the JSON expression that is in addition to the data of the information bundle for the document. Such data may include content designed by other organizations as well as augmentations of the semantic model.

The extension property is declared as an array of one or more extension objects. Optionally, each extension object has a suite of metadata properties used to describe the extension. Each extension object must have a required extension content array of exactly one extension content object. The extension metadata and content may reuse existing ABIEs or BBIEs and may contain JSON content not modeled as BIEs.

The extension scaffolding and metadata are JSON objects modeled using JSON schema fragments and constructs independent of the schema fragments created for the semantic model.

6.7.2 Extension collection JSON schema fragments and declarations

EXT21 Extension collection JSON schema fragment

There shall be one common extension collection JSON schema fragment created to include the declarations of the extension collection object, the extension object, the extension content element, the extension metadata objects and any required type information for metadata objects that are not BIEs in the Library ABIE schema fragment. The extension schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

Example

In UBL 2.2 the `UBL-CommonExtensionComponents-2.2.xsd` fragment serves this purpose.

EXT22 Extension content object declaration

The extension collection JSON schema fragment shall include the declaration of the mandatory extension content object, but not the definition information for the extension content object.

Note

The rationale for not including the definition information for the extension point object is that the type is subject to change, while the extension collection, the extension item and extension item metadata object and type information all are not. This separation allows the extension collection and item JSON schema fragment to be deployed as read-only, while the extension point data type JSON schema fragment can be deployed as writable in order to be overridden with a definition created by users.

EXT23 Extension collection object content

The document's extension collection shall be declared as having an array property of one or more extension objects as its content. The array property shall have a minimum number of items of "1". It may have a "`description` property. It shall declare that additional properties are not permitted. The array items shall reference the extension object definition in the same schema fragment.

Example

```
"UBLExtensions": {
  "description": "A container for all extensions present
in the document.",
  "required": [
    "UBLExtension"
    ],
  "properties": {
    "UBLExtension": {
      "description":
      "A single extension for private
use.",
      "items": {
        "$ref": "#/definitions/UBLExtension"
        },
```

EXT23 Extension collection object content

```
      "minItems": 1,
      "type": "array",
      }
    },
  "additionalProperties": false,
  "type": "object"
  },
```

Note

The rationale for providing for multiple extensions is that different users of a document may have different extension items added to the content. Also, different extensions may be thematically distinguished (e.g. the digital signature extension is semantically separate from an extension augmenting invoice line content).

EXT24 Extension object content ordering

The extension object shall declare all available metadata objects (if any) in advance of a single extension content property listed last. The extension content property shall be listed in the "`required`" property as well as any extension metadata that may be mandatory. There are no constraints on the available properties describing extension metadata, but there shall be a declaration of no additional properties.

Example

```
"UBLExtension": {
  "description": "A single extension for private use.",
  "required": [
    "ExtensionContent"
    ],
  "properties": {
    "ID": {...},
    "Name": {...},
    "ExtensionAgencyID": {...},
    "ExtensionAgencyName": {...},
    "ExtensionVersionID": {...},
    "ExtensionAgencyURI": {...},
    "ExtensionURI": {...},
    "ExtensionReasonCode": {...},
    "ExtensionReason": {...},
    "ExtensionContent": {...}
    },
  "additionalProperties": false,
  "type": "object"
  }
```

Note

There are no constraints on the selection, name, definition or cardinality of the extension metadata elements.

EXT25 Extension object property content declarations

The content property and each metadata property of the extension object shall be declared as an array. Each property shall declare a minimum number of items as

EXT25 Extension object property content declarations

"1". The property for the content shall declare a maximum number of items as "1". The array for the content item shall reference the extension content data type JSON schema fragment. The properties for the metadata items with a maximum cardinality of "1" shall declare a maximum number of items as "1".The array for each metadata property shall reference a definition in one of the Library ABIE, BBIE, Qualified Data Type or Unqualified Data Type JSON schema fragments.

Example

```
"ID": {
  "description": "An identifier for the Extension
assigned by the creator of the extension.",
  "items": {
    "$ref":
"UBL-CommonBasicComponents-2.2.json#/definitions/ID"
    },
  "maxItems": 1,
  "minItems": 1,
  "type": "array",
  "additionalProperties": false
  },
"ExtensionReason": {
  "description": "A description of the reason for the
Extension.",
  "items": {
    "$ref":
"UBL-UnqualifiedDataTypes-2.2.json#/definitions/
TextType"
    },
  "maxItems": 1,
  "minItems": 1,
  "type": "array",
  "additionalProperties": false
  },
"ExtensionContent": {
  "description": "The definition of the extension
content.",
  "items": {
    "$ref":
"UBL-ExtensionContentDataType-2.2.json#/definitions/
ExtensionContent"
    },
  "maxItems": 1,
  "minItems": 1,
  "type": "array",
  "additionalProperties": false
  }
},
```

Note

There are no constraints on the selection, name, definition or maximum cardinality of the extension metadata properties.

6.7.3 JSON schema fragment for the extension content data type declaration

EXT26 Extension content data type JSON schema fragment

There shall be one extension content data type schema fragment created to include the declaration of the content type for the extension content object. This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

Example

In UBL 2.2 the `UBL-ExtensionContentDataType-2.2.xsd` fragment serves this purpose.

EXT27 Extension content data type declaration

The extension content element type schema fragment shall contain only a single object declaration comprised of any content without constraint.

Example

```
"ExtensionContent": {
  "description":
"A user-defined repository of additional content",
  "type": "object"
  }
```

Note

The rationale for the lax validation is to allow for the extension point to contain, without error, any information that is supplemental to the business document but not defined by the semantic model.

6.8 Qualified data types JSON schema fragment and declarations

QDT21 Qualified data types JSON schema fragment

There shall be one qualified data types schema fragment created to include the declarations of any qualified data types referenced in the schema fragment for BBIEs. This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

Example

In UBL 2.2 the `UBL-QualifiedDataTypes-2.2.json` fragment serves this purpose.

QDT22 Qualified data type JSON declaration name

Every qualified data type shall be declared using the Dictionary Entry Name name of the data type compressed by removing periods and spaces. The `title` property of the declaration shall be the uncompressed Dictionary Entry Name.

QDT23 Qualified data type JSON declaration basis

Every qualified data type shall be based on an unqualified data type, and may impose as additional constraints whatever qualifications are required to be expressed using JSON schema semantics.

Note

In UBL 2.2 there are no qualifications expressed using JSON schema semantics. Every qualified data type declaration simply makes reference to its associated unqualified data type declaration.

QDT24 Qualified data type JSON declaration constraint

Every qualified data type declaration shall be such that every possible instance of the declared type is also an instance of the base type.

Note

This constraint prevents additions of anything that is not part of the base unqualified data type, such as the introduction of any new properties, or any less-constrained property values than the constraints on the unqualified data type.

6.9 Unqualified data types JSON schema fragment and declarations

UDT21 Unqualified data types JSON schema fragment

There shall be one unqualified data types schema fragment created to include the declarations of all unqualified data types referenced in the schema fragment for BBIEs. This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

Example

The included `BDNDR-UnqualifiedDataTypes-1.1.json` is an example fragment that serves this purpose.

UDT22 Unqualified data types declaration inclusions

An unqualified data type shall be declared for every one of the permitted Primary Representation Terms and the Secondary Representation Terms defined as Permissible Representation Terms in the Core Component Technical Specification **[CCTS]** Section 8-3 "Permissible Representation Terms".

Note

It may be a convenience to implementers to generate this JSON schema fragment by transforming an XSD expression of all possible unqualified data types as any particular CCTS model for documents may not be comprehensively using all possible unqualified data types. For example, the included BDNDR unqualified data type JSON schema fragment was generated from the included `BDNDR-UnqualifiedDataTypes-1.1.xsd` schema fragment

UDT23 Unqualified data types declaration exclusions
Unqualified data types shall only be declared for the permitted Primary Representation Terms and the Secondary Representation Terms defined as Permissible Representation Terms in the Core Component Technical Specification [CCTS] Section 8-3 “Permissible Representation Terms”.

UDT24 Unqualified data types declaration base
Every unqualified data type shall be either a reference to one of the approved Core Component Types defined in the Core Component Technical Specification [CCTS] JSON schema fragment, or be an independent object declaration of constraints that satisfies the intent of the type. The “`title`” property shall be the Dictionary Entry Name and the “`description` shall include the definition. An independent declaration shall have a set of properties based on the content and supplementary components of the type. The name of object properties shall be prefixed with the CCTS Dictionary Entry Name compressed by removing the word “`Type`” from the end and by removing periods and spaces. The name of the content property shall be suffixed with the word “`Content`”. The names of the supplemental component properties shall be suffixed with the CCTS Dictionary Entry Name of the supplementary component compressed by removing periods and spaces. The data types of the subordinate declarations shall be numbers or strings as appropriate to the corresponding XSD declared type. The object shall declare that additional properties are not permitted.

Note

The rational for having UDT declarations is to impose some JSON syntax semantics on top of the more general decimal and string lexical syntax defined in the CCTS specification of Core Component Types. For example, the CCTS Core Component Type for date and time is a simple string without constraint. Such lax structuring of the field value does not serve users in that no particular format is obligated. The UDT can impose, for example, the JSON date primitive type lexical syntax on all date and time values, overriding the CCTS definition.

Note

This rule does allow an optional supplementary component defined in the CCTS Core Component Type not to be available in the associated unqualified data type. For example, if the UDT implements a built-in JSON primitive type for the component then there is no use of a format supplementary component and associated attribute and so the format attribute declaration can be omitted and, thereby, be unavailable for use for that data type.

Example 1

In this example the unqualified data type “`Value. Type`” uses the Core Component Type “`Numeric. Type`” without change:

```
"ValueType": {
  "title": "Value. Type",
  "description": "Numeric information that is assigned
or is determined by calculation, counting, or
sequencing. It does not require a unit of quantity or
unit of measure.",
  "$ref":
"CCTS_CCT_SchemaModule-2.2.json#/definitions/
```


UDT24 Unqualified data types declaration base

```
NumericType"
  },
```

Example 2

In this example the unqualified data type "`Date Time. Type`" replaces the Core Component Type "`Date Time. Type`" with no attributes and with a JSON built-in primitive type for content:

```
"DateTimeType": {
  "title": "Date Time. Type",
  "description": "A particular point in the progression
of time, together with relevant supplementary
information.",
  "properties": {
    "_": {
      "type": "string",
      "format": "date-time"
      },
    "additionalProperties": false,
    "type": "object"
    }
  },
```

UDT25 Unqualified data types declaration constraint

Every unqualified data type declaration that is not a reference to one of the Core Component Type declarations defined in the Core Component Technical Specification **[CCTS]** JSON schema fragment shall be such that every possible instance of the declared type is also an instance of one of the Core Component Types as defined in CCTS.

Note

This constraint prevents additions of anything that is not part of the base Core Component Type, such as the introduction of any new properties, or any less-constrained property values.

6.10 CCTS Core Component Types JSON schema

All data types in the Core Component Technical Specification **[CCTS]** JSON schema fragment correspond to the 10 CCTS Primary Representation Terms defined in **[CCTS]** Section 8-3 "Permissible Representation Terms".

CCT21 CCTS Core Component Type JSON schema fragment

There shall be one core component type schema fragment of primitive data types for primary representation terms on which the unqualified data types are based. This schema fragment shall not have a "`$schema`" property and shall include only a "`definitions`" object in addition to the annotation title and description for the fragment.

This schema fragment's definitions shall be derived from the XSD schema fragment published by UN/CEFACT with the following embedded title and metadata:

```
CCTS Core Component Type Schema Module
```

CCT21 CCTS Core Component Type JSON schema fragment

```
Module of Core Component Type
Agency: UN/CEFACT
VersionID: 1.1
Last change: 14 January 2005
```

Example

The included BDNDR-CCTS_CCT_SchemaModule-1.1.json is an example of the results of such derivation from the UN/CEFACT fragment.

CCT22 Core Component Type property declarations

Every core component type shall be declared as an object named by the Dictionary Entry Name compressed by removing periods and spaces. It shall have as its title the CCTS Dictionary Entry Name. It shall have as its description the CCTS Definition. It shall have as its properties the content declaration as well as a declaration of each of its supplemental components. The name of these properties shall be prefixed with the CCTS Dictionary Entry Name compressed by removing the word "Type" from the end and by removing periods and spaces. The name of the content property shall be the underscore character "_". The names of the supplemental component properties shall be the names of the associated XML attribute names defined in the CCTS Core Component Type Schema Module version 1.1 dated 14 January 2005 BDNDR-CCTS_CCT_SchemaModule-1.1.xsd. The data types of the subordinate declarations shall be numbers or strings as appropriate to the corresponding XSD declared type. The object shall declare that additional properties are not permitted.

Example

```
"MeasureType": {
  "title": "Measure. Type",
  "description": "A numeric value determined by
measuring an object along with the specified unit of
measure.",
  "properties": {
    "MeasureContent": {
      "type": "number"
      },
    "MeasureUnitCode": {
      "type": "string"
      },
    "MeasureUnitCodeListVersionIdentifier": {
      "type": "string"
      }
    },
  "additionalProperties": false,
  "type": "object"
  },
```

Example

The included BDNDR-CCTS_CCT_SchemaModule-1.1.json is an example of the results of such derivation from the UN/CEFACT fragment.

7 Additional Document Constraints

7.1 Additional Document Constraints Introduction

In addition to the document constraints formally expressed by schemas created according to this specification, there are several other rules governing conforming business document instances that cannot be expressed using W3C Schema. These additional document rules, addressing XML instance **[XML]** validation, character encoding, and empty elements, are specified below.

Note

These rules first appeared in the OASIS UBL 1.0 and UBL 1.0 NDR Standards. To aid in coordinating references between these various publications, the rules below retain their original “IND” labels. The former IND4 was removed in the revision process leading to UBL 2.0.

Additional document constraints do not apply to the arbitrary content of extensions expressed in a business document as described in Section 3.3, “Extensions”.

7.2 Validation

The business document library and document schemas are targeted at supporting business information exchanges. Business information exchanges require a high degree of precision to ensure that application processing and corresponding business actions are reflective of the purpose, intent, and information content agreed to by both trading partners. Schemas provide the base mechanism for ensuring that instance documents do in fact support these requirements.

> **[IND1]** All instance documents MUST validate to a corresponding schema.

The use of these NDRs favours a two-phase approach for validation of rules related to specific data content (such as to check of code list values). For XML, support for this is outlined in Section 5.11, “Data type qualifications in XML”.

7.3 Character Encoding

XML supports a wide variety of character encodings. Processors SHALL understand which character encoding is employed in each XML document. XML 1.0 supports a default value of UTF-8 for character encoding, but best practice is always to identify the character encoding being employed.

> **[IND2]** All instance documents MUST identify their character encoding within the XML declaration.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

OASIS Technical Committees are obligated to conform to agreements OASIS has entered into. OASIS is a liaison member of the ISO IEC ITU UN/CEFACT eBusiness Memorandum of Understanding Management Group (MOUMG). Resolution 01/08 (MOU/MG01n83) requires the use of UTF-8.

> **[IND3]** In conformance with ISO IEC ITU UN/CEFACT eBusiness Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83) as agreed to by OASIS, all instance documents SHOULD be expressed using UTF-8.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

7.4 Empty Elements

Use of empty elements within XML instance documents is a source of controversy for a variety of reasons. An empty element does not simply represent data that is missing. It may express data that is not applicable for some reason, trigger the expression of an attribute, denote all possible values instead of just one, mark the end of a series of data, or appear as a result of an error in XML file generation. Conversely, missing data elements can also have meaning—that the trading partner does not provide that data. In information exchange environments, different trading partners may allow, require, or ban empty elements. These NDRs take the position that empty elements do not provide the level of assurance necessary for business information exchanges and therefore must not be used.

> **[IND5]** Schema-conforming instance documents SHALL NOT contain an element devoid of content or containing null values.

An important implication of this rule is that every container element must contain at least one of its possible constituents even if all of its possible constituents are declared to be optional.

To ensure that no attempt is made to circumvent rule IND5, these NDRs also prohibit attempting to convey meaning by omitting an element (i.e., an optional element may be omitted, but that omission cannot carry a specific meaning upon which an action is conditioned).

> **[IND6]** The absence of a construct or data in an instance document SHALL NOT carry meaning.

These constraints are consistent with the principle of having manifest values, that is, that the recipient must receive all pertinent information manifest in the business document. Relying on the absence of a construct would require the recipient to know of the sender's intention with that construct being absent. For reliable communication this cannot be assumed.

7.5 Natural Language Text Elements

Natural language text elements such as Note and Description appear throughout business document models following these NDRs. They are of the same unstructured Text type as character data fields that are not intended for natural language prose, such as an address line.

All natural language text elements in a business document following these NDRs are repeatable within some container; for example, all documentary note elements would be repeatable as adjacent siblings under a common parent. Despite appearances, these multiple text elements are not intended for the representation of separate paragraphs or divisions within a single parent text; rather, each note element (for example) contains the entire text of the note in one of the languages in which the note is provided. In other words, these NDRs allow 0..n for note or description elements in order to present the same note or description in 0..n languages, not to reflect structures such as paragraphs internal to a text in a single language. Since business document text elements are intended for unstructured sequences of character data, more complex texts should be located in external documents and associated with the business document using document references.

These NDRs enforce this restriction with the following two rules:

> **[IND7]** Where two or more sibling "Text. Type" elements of the same name exist in a document, no two can have the same "languageID" attribute value.

> **[IND8]** Where two or more sibling "Text. Type" elements of the same name exist in a document, no two can omit the "languageID" attribute.

7.6 Empty Supplemental Components

Attributes in XML and properties in JSON are used exclusively for supplemental components of the data types of basic business information entities. An empty component conveys no information but may be the source of confusion for users.

[IND9] BDNDR-conforming instance documents SHALL NOT contain an supplemental component devoid of content or containing null values.

8 Conformance

An information bundle and its associated validation artefacts conforming to these naming and design rules do not violate any rule or requirement expressed in normative sections of this specification related to modeling (clauses 3 and 4) and one's choices of validation artefacts. The XSD and CVA artefacts (clause 5) are for documents expressed in XML syntax according to the model. The JSON schema artefacts (clause 6) are for documents expressed in JSON syntax according to the model.

.2 ATT01 Leading name part in attribute names (5.10)

.3 ATT02 Non-leading abbreviations in attribute names (5.10)

.4 CCT01 CCTS Core Component Type schema (5.9)

.5 CCT21 CCTS Core Component Type JSON schema fragment (6.10)

.6 CCT22 Core Component Type property declarations (6.10)

.7 COM01 Dictionary information values (4.2)

.8 COM02 Dictionary information value prohibited characters and character sequences (4.2)

.9 COM03 Controlled list of abbreviations in BIE names (4.3)

.10 COM04 Controlled list of abbreviations in dictionary entry name information values (4.3)

.11 COM05 List of equivalent terms in BIE names (4.3)

.12 COM06 Component Type for a BIE (4.4.1)

.13 COM07 Minimum set of dictionary information values describing an ABIE (4.4.2)

.14 COM08 Minimum set of dictionary information values describing a BBIE (4.4.3)

.15 COM09 Minimum set of dictionary information values describing an ASBIE (4.4.4)

.16 COM10 Dictionary entry name uniqueness (4.4.5)

.17 COM11 CCTS dictionary information item name value prohibited characters (4.4.5)

.18 COM12 Use of leading upper case letter in dictionary entry name values (4.4.5)

.19 COM13 ABIE contents cannot be empty (4.5)

.20 COM14 ABIE children ordering (4.5)

.21 DCL01 Element declarations (5.5.4)

.22 DCL02 Element declaration references (5.5.4)

.23 DCL03 Type declarations (5.5.4)

.24 DCL04 ABIE element declaration (5.5.5)

.25 DCL05 ABIE type declaration (5.5.5)

.26 DCL06 Library ABIE type declaration content order (5.5.5)

.27 DCL07 Document ABIE type declaration content order (5.5.5)

.28 DCL08 Document ABIE extension element cardinality (5.5.5)

.93 QDT03 Qualified data type declaration basis (5.7)

.94 QDT04 Qualified data type declaration constraint (5.7)

.95 QDT21 Qualified data types JSON schema fragment (6.8)

.96 QDT22 Qualified data type JSON declaration name (6.8)

.97 QDT23 Qualified data type JSON declaration basis (6.8)

.98 QDT24 Qualified data type JSON declaration constraint (6.8)

.99 UDT01 Unqualified data types schema fragment (5.8)

.100 UDT02 Unqualified data types declaration inclusions (5.8)

.101 UDT03 Unqualified data types declaration exclusions (5.8)

.102 UDT04 Unqualified data types declaration base (5.8)

.103 UDT05 Unqualified data types declaration constraint (5.8)

.104 UDT21 Unqualified data types JSON schema fragment (6.9)

.105 UDT22 Unqualified data types declaration inclusions (6.9)

.106 UDT23 Unqualified data types declaration exclusions (6.9)

.107 UDT24 Unqualified data types declaration base (6.9)

.108 UDT25 Unqualified data types declaration constraint (6.9)

Appendix A (informative) Release notes

A.1 Availability

Online and downloadable versions of this release are available from the locations specified at the top of this document.

A.2 Package structure

This Committee Specification 01 is published as a zip archive in the https://docs.oasis-open.org/ubl/Business-Document-NDR/v1.1/cs01/ directory. Unzipping this archive creates a directory tree containing a number of files and subdirectories. Note that while the two XML files comprise the revisable version of this specification, this revisable XML may not be directly viewable in all currently available web browsers.

The base directory has the following files:

`Business-Document-NDR-v1.1-cs01.xml`

The revisable form of the document.

`Business-Document-NDR-v1.1-cs01-summary.ent`

A distillation of the rules, comprising only the rule title and the section number in which the rule is found. The title is linked to the rule and the section number is linked to the section. This XML document is incorporated in the revisable form of the document by way of an entity reference. During the publishing process this file is first distilled from the revisable form and then subsequently incorporated in the revisable form for publishing.

`Business-Document-NDR-v1.1-cs01.html`

An HTML rendering of the document.

`Business-Document-NDR-v1.1-cs01.pdf`

A PDF rendering of the document.

These are the subdirectories in the package:

`art`

Diagrams and illustrations used in this specification

`db`

DocBook stylesheets for viewing in HTML the XML of this work product

`jsonschema`

JSON Schema fragments supporting UN/CEFACT core component types and OASIS BDNDR unqualified data types

`sample`

Illustrative XSLT stylesheet and documentation for converting an instance of XML conforming to BDNDR XML Schema into an instance of JSON conforming to BDNDR JSON Schema.

xsd

XML Schema fragments supporting UN/CEFACT core component types and OASIS BDNDR unqualified data types

A.3 Sample schema and code fragments

The release includes the following sample schema and code fragments for the convenience of some technical users:

- sample/CCTSXML2JSON.xsl - XSLT stylesheet to convert an instance of XML conforming to BDNDR XML Schema into an instance of JSON conforming to BDNDR JSON Schema
- sample/readme-CCTSXML2JSON.html - documentation generated from the embedded constructs, including invocation details
- xsd/BDNDR-CCTS_CCT_SchemaModule-1.1.xsd - copy of the UN/CEFACT core component type XML Schema file
- xsd/BDNDR-UnqualifiedDataTypes-1.1.xsd - example interpretation in XML Schema of the permissible representation terms of the UN/CEFACT core component types as unqualified data types
- jsonschema/BDNDR-CCTS_CCT_SchemaModule-1.1.json - JSON interpretation of the UN/CEFACT core component type XML Schema file
- jsonschema/BDNDR-UnqualifiedDataTypes-1.1.json - JSON interpretation of the XML Schema of the permissible representation terms of the UN/CEFACT core component types as unqualified data types

A.4 Release history

A.4.1 Version 1.0 release

Version 1.0 of the Business Document Naming and Design Rules describes the constraints on the CCTS modeling [**CCTS**] of XML interchange documents [**XML**] and the creation of corresponding W3C schema [XSD schema] and OASIS Context/Value Association documents [**CVA**].

A.4.2 Differences between version 1.1 and version 1.0

Version 1.1 introduces the specification of JSON schemas to govern JSON serializations of Open-edi user data, not available in version 1.0.

Version 1.1 also modifies the availability of extension content in 1.1 also to be available for all Library ABIEs in addition to the 1.0 availability only on Document ABIEs (see Section 5.5.5, “XML schema declarations for ABIEs”).

A.4.3 Differences between version 1.1 csd03 and 1.1 csd01

Version 1.1 csd02 modified the availability of extension content in csd02 also to be available for all Library ABIEs in addition to the csd01 availability only on Document ABIEs (see Section 5.5.5, “XML schema declarations for ABIEs”).

Version 1.1 csd02 repaired some faults in the specification of JSON schema identified during the public review. Specifically, the inappropriate specifications of “`additionalProperties`” in rules DCL23, DCL24, DCL26, and EXT23 have been removed. Additionally, in UDT25 the need for co-constraints on date-time values in order to express individual date and time constraints in Draft-04 of JSON Schema is removed due to the availability in Draft-07 of JSON Schema of individual date and time constraints.

Based on community feedback, version 1.1 csd03 makes some wholesale changes to the JSON serialization in csd02 in order to promote ease of transliteration of instances with the XML serialization. Before the changes were made it was necessary to have some foreknowledge of the document model. With these changes, transliteration results can be inferred entirely from manifest content. These benefits were deemed important enough to use different naming conventions in the JSON serialization as follows:

- the aggregate namespace property name is changed from "_S" (for ASBIE) to "_A" (for ABIE),
- the BBIE content property name is changed from a concatenation of the type and the word "Content" to be the single character "_", and
- the BBIE supplementary component property names are changed from their CCTS name to be the XML attribute name as dictated by the published UN/CEFACT Core Component Type schema fragment.

The sample JSON Schema and XML Schema versions of the BDNDR unqualified data types and UN/CEFACT core component types is added to the package for the convenience of some technical users.

The sample CCTSXML2JSON.xsl XSLT stylesheet is added to the package for the convenience of some technical users.

Appendix B (informative) Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Todd Albers, Federal Reserve Bank of Minneapolis
Rui Barros, Individual
Oriol Bausa Peris, Individual
Kenneth Bengtsson, Individual
Erlend Klakegg Bergheim, Norwegian Digitalisation Agency
Peter Borresen, ClearView Trade
Andrea Caccia, Individual
Ger Clancy, IBM
Kees Duvekot, RFS Holland Holding B.V.
Martin Forsberg, Swedish Association of Local Authorities & Regions
Cecile Guasch, Individual
Philip Helger, Individual
Ken Holman, Crane Softwrights Ltd.
Yves Jordan, Publications Office of the European Union
Kari Korpela, Individual
Ole Madsen, Danish Business Authority
Natalie Muric, Publications Office of the European Union
Levine Naidoo, IBM
Enric Torregrosa, everis, S.L.U.
Matt Vickers, Xero

Appendix C (informative) Additional production processes

C.1 CCTS serialization

An implementation of these naming and design rules may choose to create a serialization of the CCTS information. This can be a useful convenience when processing the CCTS information as a whole. The CCTS collaboration tool is not required to produce a serialization if such is not needed to fulfill its obligation to produce validation artefacts.

There are no formal rules for CCTS serialization.

C.2 Reporting

An implementation of these naming and design rules is not required to produce any supplementary reports. These reports can be useful reference materials for review of the CCTS information maintained for the information bundles.

There are no formal rules for reporting.