



1

2 **Universal Business Language (UBL)** 3 **Naming and Design Rules**

4 **19 November 2003**

5 **Document identifier:**

6 wd-ublndrsc-ndrdoc-V1pt0Draft1 (Word)

7 **Location:**

8 <http://www.oasis-open.org/committees/ubl/ndrsc/drafts/>

9 **Naming and Design Rules Subcommittee Co-chairs**

10 Mavis Cournane, Cognitran Ltd <mavis.cournane@cognitran.com>

11 Mark Crawford, LMI <mcrawford@lmi.org>

12 Lisa Seaburg, Aeon LLC <lseaburg@aeon-llc.com>

13 **Lead Editor:**

14 Mark Crawford, LMI <mcrawford@lmi.org>

15 **Contributors:**

16 Bill Burcham, Sterling Commerce

17 Fabrice Desré, France Telecom

18 Matt Gertner, Schemantix

19 Jessica Glace, LMI

20 Arofan Gregory, Aeon LLC

21 Michael Grimley, US Navy

22 Eduardo Gutentag, Sun Microsystems

23 Sue Probert, CommerceOne

24 Gunther Stuhec, SAP

25 Paul Thorpe, OSS Nokalva

26 Jim Wilson, CIDX

27 **Past Chair**

28 Eve Mailer, Sun Microsystems <eve.maler@sun.com>

29 **Abstract:**

30 This specification documents the naming and design rules and guidelines for the
31 construction of XML components from ebXML Core Components

32 **Status:**

33 *This is a draft document under consideration by the OASIS UBL TC for approval*
34 *as a TC and OASIS standard.*
35

36 Copyright © 2001, 2002, 2003 The Organization for the Advancement of Structured
37 Information Standards [OASIS]
38

38 **Table of Contents**

39	1	Introduction	7
40	1.1	Audiences	7
41	1.2	Scope	8
42	1.3	Terminology and Notation	8
43	1.4	Guiding Principles	9
44	1.4.1	Adherence to General UBL Guiding Principles	9
45	1.4.2	Design For Extensibility	11
46	1.4.3	Code Generation	12
47	1.5	Choice of schema language	12
48	2	Relationship to ebXML Core Components	13
49	2.1	Mapping Business Information Entities to XSD	16
50	3	General XML Constructs	19
51	3.1	Overall Schema Structure	19
52	3.1.1	Root Element	22
53	3.2	Constraints	23
54	3.2.1	Naming Constraints	24
55	3.2.2	Modeling Constraints	24
56	3.3	Reusability Scheme	25
57	3.3.1	Managing by Types	26
58	3.4	Namespace Scheme	29
59	3.4.1	Declaring Namespaces	30
60	3.4.2	Namespace Uniform Resource Identifiers	30
61	3.4.3	Schema Location	31
62	3.4.4	Persistence	32
63	3.5	Versioning Scheme	32
64	3.6	Modularity	35
65	3.6.1	UBL Modularity Model	36
66	3.6.2	Internal and External schema modules	39
67	3.6.3	Internal schema modules	39
68	3.6.4	External schema modules	40
69	3.7	Documentation	44
70	3.7.1	Embedded documentation	44
71	3.7.2	Schema Annotation	50
72	4	Naming Rules	51
73	4.1	General Naming Rules	51

74	4.2	Type Naming Rules	53
75	4.2.1	Complex Type Names for CCTS Aggregate Business Information Entities	54
76		54	
77	4.2.2	Complex Type Names for CCTS Basic Business Information Entities	54
78	4.2.3	Complex Type Names for CCTS Representation Terms	55
79	4.2.4	Complex Type Names for CCTS Core Component Types	55
80	4.3	Element Naming Rules	56
81	4.3.1	Element Names for CCTS Aggregate Business Information Entities	56
82	4.3.2	Element Names for CCTS Basic Business Information Entities	56
83	4.3.3	Element Names for CCTS Association Business Information Entities	57
84	4.4	Attribute Naming Rules	57
85	5	Declarations and Definitions	59
86	5.1	Type Definitions	59
87	5.1.1	General Type Definitions	59
88	5.1.2	Simple Types	59
89	5.1.3	Complex Types	60
90	5.2	Element Declarations	66
91	5.2.1	General Element Declarations	66
92	5.2.2	Elements Bound to Complex Types	66
93	5.2.3	Code List Import	67
94	5.2.4	Empty Elements	67
95	5.2.5	XSD:Any	67
96	5.3	Attribute Declarations	67
97	5.3.1	User Defined Attributes	68
98	5.3.2	Global Attributes	68
99	5.3.3	Supplementary Components	68
100	5.3.4	Schema Location	68
101	5.3.5	XSD:Nil	69
102	5.3.6	XSD:Any	69
103	6	Code Lists	70
104	7	Miscellaneous XSD Rules	73
105	7.1	XSD Simple Types	73
106	7.2	Namespace Declaration	73
107	7.3	XSD:Substitution Groups	73
108	7.4	XSD:Final	73
109	7.5	XSD: Notations	73
110	7.6	XSD:All	74
111	7.7	XSD:Choice	74

112	7.8 XSD:Include	74
113	7.9 XSD:Union	74
114	7.10 XSD:Appinfo	74
115	7.11 Extension and Restriction	75
116	8 Instance Documents	76
117	8.1 Root Element	76
118	8.2 Validation	76
119	8.3 Character Encoding	76
120	8.4 Schema Instance Namespace Declaration	77
121	8.5 Empty Content.	77
122	Appendix A. UBL NDR Checklist	78
123	Table A1 — Code List Rules	79
124	Table A2. Constraint Rules	80
125	Modeling Constraints	80
126	Naming Constraints	80
127	Table A3 — Declarations Rules	81
128	Element Declarations	81
129	Attribute Declarations	81
130	Table A4. Documentation Rules	83
131	Table A5. General XSD Rules	89
132	Table A6 —Instance Documents	92
133	Table A7 — Naming Rules	93
134	General Naming rules	93
135	Specific Naming Rules	94
136	Element Naming Rules	94
137	Attribute Naming Rules	94
138	Type Naming Rules	94
139	Table A8 — Namespace Rules	96
140	Table A9 — Root Element Declaration Rules	98
141	Table A10 — Schema Structure Modularity Rules	99
142	Table A11 — Standards Adherence Rules	101
143	Table A12 — Type Definition Rules	102
144	General Type Definitions	102
145	Simple Type Definitions	102
146	Table A13 — Versioning Rules	105
147	Appendix B. Approved Acronyms and Abbreviations	107
148	Appendix C. Technical Terminology	108
149	Appendix D. References	114

150 Appendix E. Notices

115

151

152 1 Introduction

153 XML is often described as the lingua franca of e-commerce. The implication is that by
154 standardizing on XML, enterprises will be able to trade with anyone, any time, without
155 the need for the costly custom integration work that has been necessary in the past. But
156 this vision of XML-based “plug-and-play” commerce is overly simplistic. Of course
157 XML can be used to create electronic catalogs, purchase orders, invoices, shipping
158 notices, and the other documents needed to conduct business. But XML by itself doesn't
159 guarantee that these documents can be understood by any business other than the one that
160 creates them. XML is only the foundation on which additional standards can be defined
161 to achieve the goal of true interoperability. The Universal Business Language (UBL)
162 initiative is the next step in achieving this goal.

163 The task of creating a universal XML business language is a challenging one. Most large
164 enterprises have already invested significant time and money in an e-business
165 infrastructure and are reluctant to change the way they conduct electronic business.
166 Furthermore, every company has different requirements for the information exchanged in
167 a specific business process, such as procurement or supply-chain optimization. A
168 standard business language must strike a difficult balance, adapting to the specific needs
169 of a given company while remaining general enough to let different companies in
170 different industries communicate with each other.

171 The UBL effort addresses this problem by building on the work of the electronic business
172 XML (ebXML) initiative. EbXML, currently continuing development in the Organization
173 for the Advancement of Structured Information Standards (OASIS), is an initiative to
174 develop a technical framework that enables XML and other payloads to be utilized in a
175 consistent manner for the exchange of all electronic business data. UBL is organized as
176 an OASIS Technical Committee to guarantee a rigorous, open process for the
177 standardization of the XML business language. The development of UBL within OASIS
178 also helps ensure a fit with other essential ebXML specifications. UBL will be promoted
179 to the level of international standard.

180 This specification documents the rules and guidelines for the naming and design of XML
181 components for the UBL library. It contains only rules that have been agreed on by the
182 OASIS UBL Naming and Design Rules Subcommittee (NDR SC). Proposed rules, and
183 rationales for those that have been agreed on, appear in the accompanying NDR SC
184 position papers, which are available at [http://www.oasis-](http://www.oasis-open.org/committees/ubl/ndrsc/)
185 [open.org/committees/ubl/ndrsc/](http://www.oasis-open.org/committees/ubl/ndrsc/).

186 1.1 Audiences

187 This document has several primary and secondary targets that together constitute its
188 intended audience. Our primary target audience is the UBL Library Content
189 Subcommittee. Specifically, the UBL Library Content Subcommittee will use this
190 document to create normative form schema for business transactions. External

191 developers will use this document to extend and restrict UBL schema in a fashion that
192 will ensure conformance to the UBL design rules and guarantee compatibility with
193 existing UBL schema. Other developers implementing ebXML Core Components may
194 find the rules contained herein sufficiently useful to merit adoption as, or infusion into,
195 their own approaches to ebXML Core Component based XML schema development. All
196 other XML Schema developers may find the rules contained herein sufficiently useful to
197 merit consideration for adoption as, or infusion into, their own approaches to XML
198 schema development.

199 1.2 Scope

200 This specification conveys a normative set of XML schema design rules and naming
201 conventions for the creation of business based XML schema for transactions being
202 exchanged between two parties using objects developed in accordance with the ebXML
203 Core Components Technical Specification.

204 1.3 Terminology and Notation

205 The key words **MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,**
206 **SHOULD NOT, RECOMMENDED, MAY,** and **OPTIONAL** in this document are to
207 be interpreted as described in Internet Engineering Task Force (IETF) Request for
208 Comments (RFC) 2119. Non-capitalized forms of these words are used in the regular
209 English sense.

210 [Definition] – A formal definition of a term. Definitions are normative.

211 [Example] – A representation of a definition or a rule. Examples are informative.

212 [Note] – Explanatory information. Notes are informative.

213 [RRRn] - Identification of a rule that requires conformance to ensure that an XML
214 Schema is UBL conformant. The value RRR is a prefix to categorize the type of
215 rule where the value of RRR is as defined in Table 1 and n (1..n) indicates the
216 sequential number of the rule within its category. In order to ensure continuity
217 across versions of the specification, rule numbers that are deleted in future
218 versions will not be re-issued, and any new rules will be assigned the next higher
219 number - regardless of location in the text. Future versions will contain an
220 appendix that lists deleted rules and the reason for their deletion. Only rules are
221 normative; all other text is explanatory.

222

Figure 1 - Rule Prefix Token Value

Rule Prefix Token	Value
ATD	Attribute Declaration
ATN	Attribute Naming
CDL	Code List
CTD	ComplexType Definition
DOC	Documentation
ELD	Element Declaration
ELN	Element Naming
GNR	General Naming

GTD	General Type Definition
GXS	General XML Schema
IND	Instance Document
MDC	Modeling Constraints
NMC	Naming Constraints
NMS	Namespace
RED	Root Element Declaration
SSM	Schema Structure Modularity
STD	SimpleType Definition
VER	Versioning

223 **Bold** - The bolding of words is used to represent example names or parts of names taken
224 from the library.

225 **Courier** – All words appearing in **courier font** are values, objects, and
226 keywords.

227 *Italics* – All words appearing in italics, when not titles or used for emphasis, are special
228 terms defined in Appendix A.

229 The terms “W3C XML Schema” and “XSD” are used throughout this document. They
230 are considered synonymous; both refer to XML Schemas that conform to Parts 1 and 2 of
231 the W3C *XML Schema Definition Language (XSD) Recommendations*. See Appendix A
232 for additional term definitions.

233 1.4 Guiding Principles

234 The UBL guiding principles encompass three areas:

- 235 ◆ General UBL guiding principles
- 236 ◆ Extensibility
- 237 ◆ Code generation

238 1.4.1 Adherence to General UBL Guiding Principles

239 The UBL Technical Committee has approved a set of high-level guiding principles. The
240 UBL Naming and Design Rules Subcommittee (NDRSC) has followed these high-level
241 guiding principles for the design of UBL NDR. These guiding principles are:

- 242 1. Internet Use - UBL shall be straightforwardly usable over the Internet.
- 243 2. Interchange and Application Use–UBL is intended for interchange and
244 application use.
- 245 3. Tool Use and Support - The design of UBL will not make any
246 assumptions about sophisticated tools for creation, management, storage,

- 247 or presentation being available. The lowest common denominator for tools
248 is incredibly low (for example, Notepad) and the variety of tools used is
249 staggering. We do not see this situation changing in the near term.
- 250 4. Legibility - UBL documents should be human-readable and reasonably
251 clear.
- 252 5. Simplicity - The design of UBL must be as simple as possible (but no
253 simpler).
- 254 6. 80/20 Rule - The design of UBL should provide the 20% of features that
255 accommodate 80% of the needs.
- 256 7. Component Reuse -The design of UBL document types should contain as
257 many common features as possible. The nature of e-commerce
258 transactions is to pass along information that gets incorporated into the
259 next transaction down the line. For example, a purchase order contains
260 information that will be copied into the purchase order response. This
261 forms the basis of our need for a core library of reusable components.
262 Reuse in this context is important, not only for the efficient development
263 of software, but also for keeping audit trails.
- 264 8. Standardization - The number of ways to express the same information in
265 a UBL document is to be kept as close to one as possible.
- 266 9. Domain Expertise - UBL will leverage expertise in a variety of domains
267 through interaction with appropriate development efforts.
- 268 10. Customization and Maintenance - The design of UBL must facilitate
269 customization and maintenance.
- 270 11. Context Sensitivity - The design of UBL must ensure that context-
271 sensitive document types aren't precluded.
- 272 12. Prescriptiveness - UBL design will balance prescriptiveness in any single
273 usage scenario with prescriptiveness across the breadth of usage scenarios
274 supported. Having precise, tight content models and datatypes is a good
275 thing (and for this reason, we might want to advocate the creation of more
276 document type "flavors" rather than less; see below). However, in an
277 interchange format, it is often difficult to get the prescriptiveness that
278 would be desired in any single usage scenario.
- 279 13. Content Orientation - Most UBL document types should be as "content-
280 oriented" (as opposed to merely structural) as possible. Some document
281 types, such as product catalogs, will likely have a place for structural
282 material such as paragraphs, but these will be rare.

- 283 14. XML Technology - UBL design will avail itself of standard XML
284 processing technology wherever possible (XML itself, XML Schema,
285 XSLT, XPath, and so on). However, UBL will be cautious about basing
286 decisions on “standards” (foundational or vocabulary) that are works in
287 progress.
- 288 15. Relationship to Other Namespaces - UBL design will be cautious about
289 making dependencies on other namespaces. UBL does not need to reuse
290 existing namespaces wherever possible. For example, XHTML might be
291 useful in catalogs and comments, but it brings its own kind of processing
292 overhead, and if its use is not prescribed carefully it could harm our goals
293 for content orientation as opposed to structural markup.
- 294 16. Legacy formats - UBL is not responsible for catering to legacy formats;
295 companies (such as ERP vendors) can compete to come up with good
296 solutions to permanent conversion. This is not to say that mappings to and
297 from other XML dialects or non-XML legacy formats wouldn't be very
298 valuable.
- 299 17. Relationship to xCBL - UBL will not be a strict subset of xCBL, nor will
300 it be explicitly compatible with it in any way.

301 1.4.2 Design For Extensibility

302 Many e-commerce document types are, broadly speaking, useful but require minor
303 structural modifications for specific tasks or markets. When a truly common XML
304 structure is to be established for e-commerce, it needs to be easy and inexpensive to
305 modify.

306 Many data structures used in e-commerce are very similar to “standard” data structures,
307 but have some significant semantic difference native to a particular industry or process.
308 In traditional Electronic Data Interchange (EDI), there has been a gradual increase in the
309 number of published components to accommodate market-specific variations. Handling
310 these variations are a requirement, and one that is not easy to meet. A related EDI
311 phenomenon is the overloading of the meaning and use of existing elements, which
312 greatly complicates interoperation.

313 To avoid the high degree of cross-application coordination required to handle structural
314 variations common to EDI and Document Type Definition (DTD) based systems - it is
315 necessary to accommodate the required variations in basic data structures without either
316 overloading the meaning and use of existing data elements, or requiring wholesale
317 addition of new data elements. This can be accomplished by allowing implementers to
318 specify new element types that inherit the properties of existing elements, and to also
319 specify exactly the structural and data content of the modifications.

320 This can be expressed by saying that extensions of core elements are driven by context.¹
321 Context driven extensions should be renamed to distinguish them from their parents, and
322 designed so that only the new elements require new processing.

323 Similarly, data structures should be designed so that processes can be easily engineered to
324 ignore additions that are not needed.

325 1.4.3 Code Generation

326 UBL has developed two code generation tools that automatically convert the UBL
327 library into UBL NDR conformant XSD. These two tools have been developed for UBL
328 internal use only. In conformance with UBL guiding principle 3, the UBL design process
329 has scrupulously avoided establishing any NDR that sub-optimize the XSD in favor of
330 automatic generation. In conformance with UBL guiding principle 8, The NDR are
331 sufficiently rigorous to avoid requiring human judgment at generation time.

332 1.5 Choice of schema language

333 The W3C XML Schema Definition Language has become the generally accepted
334 schema language that is experiencing the most widespread adoption. Although
335 other schema languages exist that have their own pro's and con's, UBL has
336 determined that the best approach for developing an international XML business
337 standard is to base its work on W3C XSD.
338

339 [STA1] All UBL schema design rules MUST be based on the W3C XML Schema
340 Recommendations: XML Schema Part 1: Structures and XML Schema
341 Part 2: Datatypes.

342 A W3C technical specification holding recommended status represents consensus within
343 the W3C and has the W3C Director's stamp of approval. Recommendations are
344 appropriate for widespread deployment and promote W3C's mission. Before the Director
345 approves a recommendation, it must show an alignment with the W3C architecture. By
346 aligning with W3C specifications holding recommended status, UBL can ensure that its
347 products and deliverables are well suited for use by the widest possible audience with the
348 best availability of common support tools.

349 [STA2] All UBL schema and messages MUST be based on the W3C suite of
350 technical specifications holding recommendation status.

¹ ebXML, Core Components Technical Specification – Part 8 of the ebXML Technical Framework, V2.0, 11 August 2003

351 2 Relationship to ebXML Core Components

352 UBL employs the methodology and model described in *Core Components Technical*
353 *Specification, Part 8 of the ebXML Technical Framework, Version 2.0 (Second Edition)*
354 of 15 November 2003 (CCTS) to build the UBL Component Library. The Core
355 Components work is a continuation of work that originated in, and remains a part of, the
356 ebXML initiative. The Core Components concept defines a new paradigm in the design
357 and implementation of reusable syntactically neutral information building blocks. Core
358 Components are intended to form the basis of business information standardization
359 efforts and to be realized in syntactically specific instantiations such as electronic data
360 interchange and XML.

361 The essence of the Core Components specification is captured in context neutral and
362 context specific building blocks. The context neutral components are defined as Core
363 Components (`ccts:CoreComponents`). Context neutral `ccts:CoreComponents` are
364 defined in CCTS as “A building block for the creation of a semantically correct and
365 meaningful information exchange package. It contains only the information pieces
366 necessary to describe a specific concept.”² Figure 2-1 illustrates the various pieces of the
367 overall `ccts:CoreComponents` metamodel.

368 The context specific components are defined as Business Information Entities
369 (`ccts:BusinessInformationEntities`).³ Context specific `ccts:Business`
370 `InformationEntities` are defined in CCTS as “A piece of business data or a group of
371 pieces of business data with a unique *Business Semantic* definition.”⁴ Figure 2-2
372 illustrates the various pieces of the overall `ccts:BusinessInformationEntity`
373 metamodel and their relationship with the `ccts:CoreComponents` metamodel.

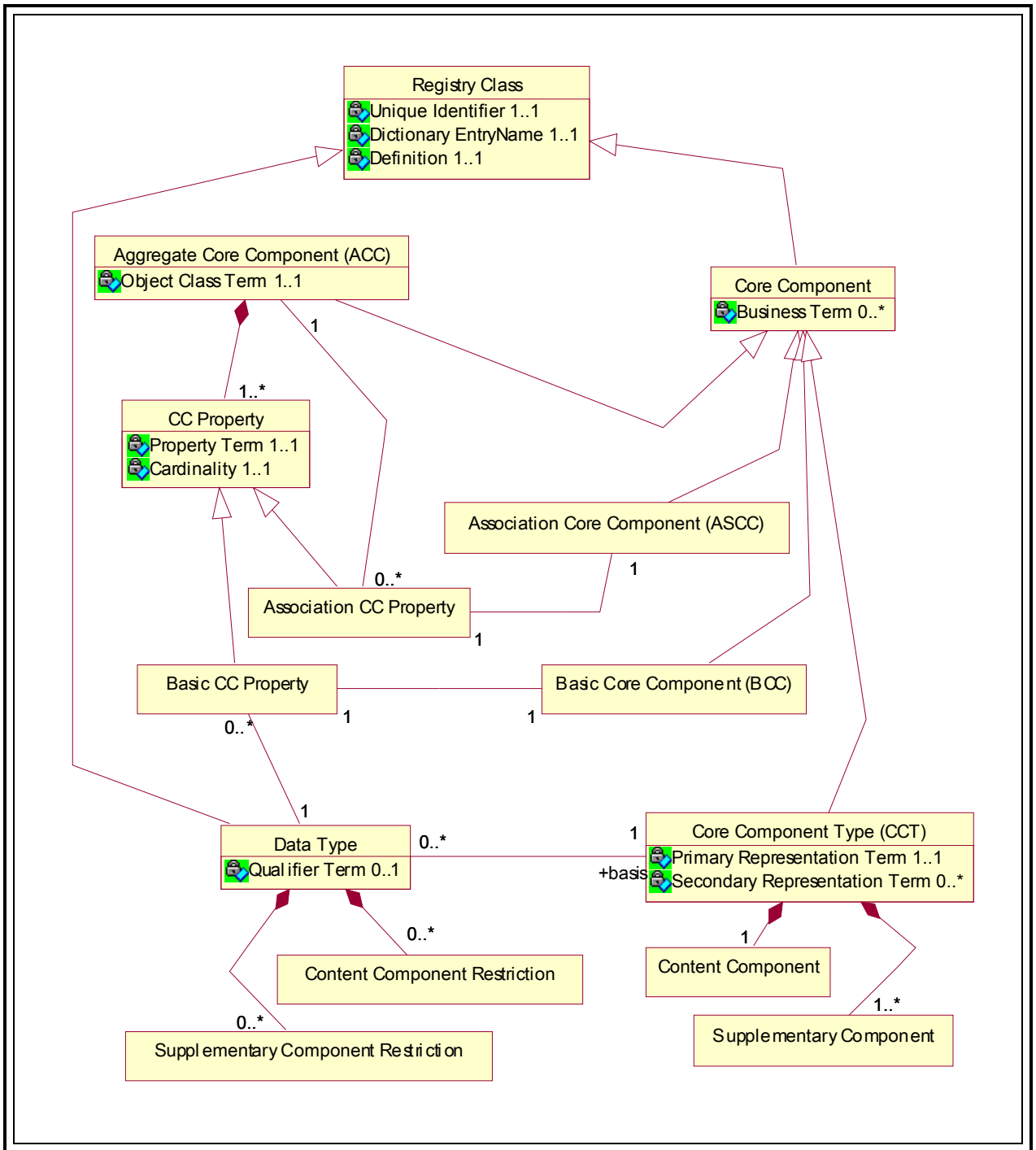
374 As shown in Figure 2-2, there are different types of `ccts:CoreComponents` and
375 `ccts:BusinessInformationEntities`. Each type of `ccts:CoreComponent` and
376 `ccts:BusinessInformationEntity` has specific relationships between and
377 amongst the other components and entities. The context neutral `ccts:Core`
378 `Components` are the linchpin that establishes the formal relationship between the various
379 context-specific `ccts:BusinessInformationEntities`.

² *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition)*, UN/CEFACT, 15 November 2003

³ See CCTS Section 6.2 for a detailed discussion of the ebXML context mechanism.

⁴ *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition)*, UN/CEFACT, 15 November 2003

381

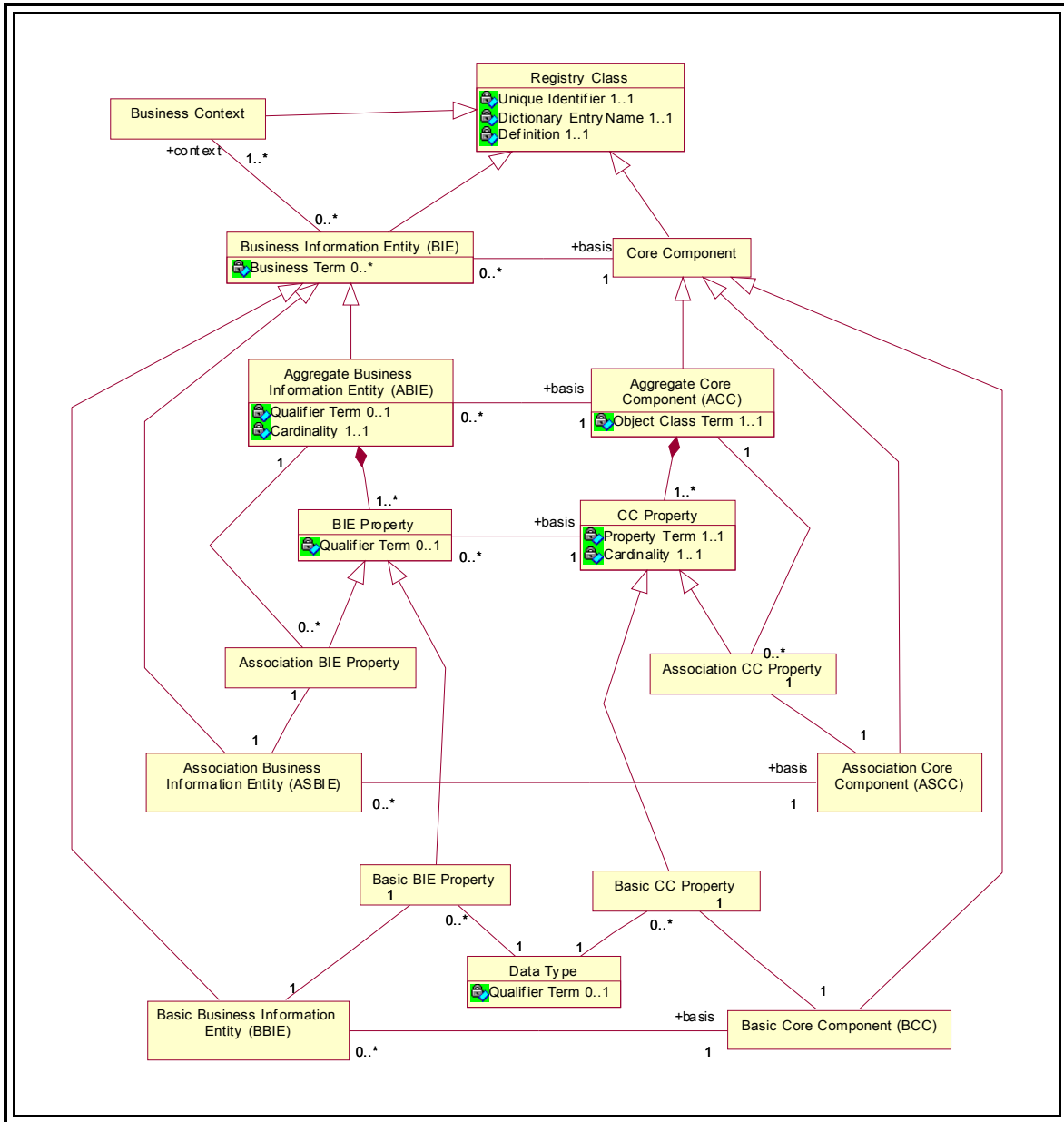


382

383

⁵ *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition)*, UN/CEFACT, 15 November 2003

Figure 2-2. Business Information Entities Basic Definition Model



384

385 Multiple `ccts:BusinessInformationEntities`, each expressing a different context,
 386 can be associated to a single `ccts:CoreComponent`. A collection of
 387 `ccts:BusinessInformationEntities` will constitute a business document. A
 388 larger collection of `ccts:BusinessInformationEntities` will constitute a library
 389 of reusable components.

390 UBL is developing a library of reusable components for XML syntactic expressions, as
 391 well as the syntactic expressions themselves in the form of normative schema. In
 392 keeping with the tenants of the CCTS, the UBL component library will consist of
 393 `ccts:BusinessInformationEntities`. More specifically, the UBL component

394 library consists of Aggregate Business Information Entities (`ccts:Aggregate`
395 `BusinessInformationEntities`), their underlying Basic Business Information
396 Entities (`ccts:BasicBusinessInformationEntities`], and Association Business
397 Information Entities (`ccts:AssociationBusinessInformationEntities`)
398 developed in the context of the business process.

399 UBL is committed to contributing its library of reusable components for harmonization
400 and inclusion in an ebXML Core Component and Business Information library and
401 registry. Since UBL is concerning itself only with the development of
402 `ccts:BusinessInformationEntities` and their realization in XML, the UBL
403 metamodel is that subset of Figure 2-2 that consists of the `ccts:Business`
404 `InformationEntity` concepts. The UBL methodology defines no
405 `ccts:CoreComponents`. Since UBL will not be defining `ccts:CoreComponents`,
406 UBL will leave it to the ebXML library and registry owners to define the relationships
407 between the UBL developed `ccts:BusinessInformationEntities` and their
408 underlying `ccts:CoreComponents`.

409 2.1 Mapping Business Information Entities to XSD

410 UBL has defined how each of the `ccts:BusinessInformationEntity` components
411 map to an XSD construct (See figure 2-3). In defining this mapping, UBL has analyzed
412 the CCTS metamodel and determined the optimal usage of XSD to express the various
413 `ccts:BusinessInformationEntity` components. As stated above, a
414 `ccts:BusinessInformationEntity` can be an `ccts:AggregateBusiness`
415 `InformationEntity`, `ccts:BasicBusinessInformationEntity`, or
416 `ccts:AssociationBusinessInformationEntity`. In understanding the logic of
417 the UBL binding of `ccts:BusinessInformationEntities` to XSD expressions, it is
418 important to understand the basic constructs of the `ccts:AggregateBusiness`
419 `InformationEntities` and their relationships as shown in Figure 2-2.

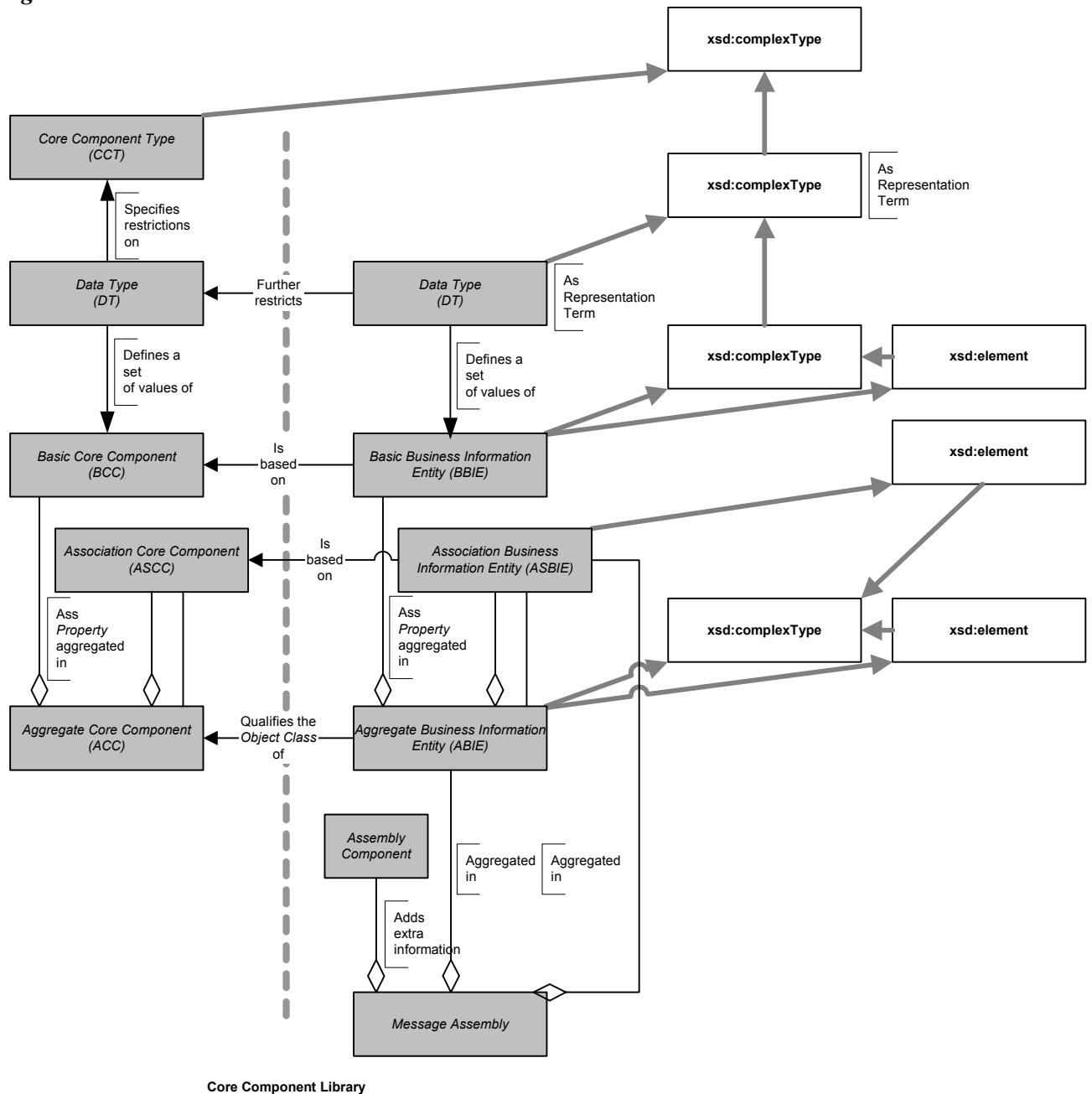
420 Both Aggregate and Basic Business Information Entities must have a unique name
421 (Object Class Term). Both are treated as objects and both are defined as
422 `xsd:ComplexTypes`.

423 There are two kinds of Business Information Entity Properties - Basic and Association. A
424 Basic Business Information Entity Property represents an *intrinsic* property of an
425 Aggregate Business Information Entity. Basic Business Information Entity properties are
426 linked to a data type and expressed as either a primary or secondary Representation
427 Term. Since data types are not expressed directly, UBL does not define an `xsd` structure
428 for data types unless the data type defines a restriction to the facets of the corresponding
429 `ccts:ContentComponent` or `ccts:SupplementaryComponent`.

430 CCTS pre-defines an approved set of primary and secondary representation terms. Since
431 these terms are fixed, UBL defines each primary and secondary representation term in a
432 reusable `xsd:schemaModule`. In that schema module, all representation terms are

433 defined as an `xsd:complexType` with the exception of those representation terms that
 434 directly map to an XSD built-in datatype, which are defined as `xsd:simpleTypes`.

435 **Figure 2-3. UBL Metamodel**



436 CCTS enables users to define their own data types for their syntax neutral constructs. A
 437 `ccts:DataType` “defines the set of valid values that can be used for a particular *Basic*
 438 *Core Component Property* or *Basic Business Information Entity Property*. It is defined by
 439 specifying restrictions on the *Core Component Type* from which the *Data Type* is
 440

441 derived.”⁶ Thus `ccts:DataTypes` allow UBL to identify facets for elements when
442 restrictions to the corresponding `ccts:ContentComponent` or
443 `ccts:SupplementaryComponent` is required. These facets will be defined as
444 `ubl:DataTypes` and will be represented by

445 A `ccts:AssociationBusinessInformationEntityProperty` represents an
446 *extrinsic* property – in other words an association from one `ccts:Aggregate`
447 `BusinessInformationEntityProperty` instance to another `ccts:Aggregate`
448 `BusinessInformationEntityProperty` instance. It is the `ccts:Aggregate`
449 `BusinessInformationEntityProperty` that expresses the relationship between
450 `ccts:AggregateBusinessInformationEntities`. Due to their unique extrinsic
451 association role, `ccts:AssociationBusinessInformationEntities` are not
452 defined as `xsd:complexType`, rather they are declared as elements that are then bound
453 to the `xsd:complexType` of the associated `ccts:AggregateBusiness`
454 `InformationEntity`.

455 As stated above, `ccts:BasicBusinessInformationEntities` define the intrinsic
456 structure of a `ccts:AggregateBusinessInformationEntity`. These
457 `ccts:BasicBusinessInformationEntities` are the “leaf” types in the system in
458 that they contain no `ccts:AssociationBusinessInformationEntity`
459 `Properties`. A `ccts:BasicBusinessInformationEntity` must have a
460 `ccts:CoreComponentType`. `Ccts:CoreComponentTypes` are low-level types, such
461 as Identifiers and Dates. A `Ccts:CoreComponentType` describes these low-level types
462 for use by `ccts:CoreComponents`, and (in parallel) a `ccts:DataType` –
463 corresponding to that `ccts:CoreComponentType`, describes these low-level types for
464 use by `ccts:BusinessInformationEntities`. Every `ccts:CoreComponentType`
465 have a single `ccts:ContentComponent` and one or more `ccts:Supplementary`
466 `Components`. A `ccts:ContentComponent` is of some `Primitive Type`. All
467 `ccts:CoreComponentTypes` and their corresponding content and supplementary
468 components are pre-defined in the CCTS. UBL, in partnership with the Open
469 Applications Group has developed an `xsd:schemaModule` that defines each of the pre-
470 defined `ccts:CoreComponentTypes` as `xsd:complexType`s and declares
471 `ccts:SupplementaryComponents` as `xsd:attributes`.

⁶ *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition)*, UN/CEFACT, 15 November 2003

472 3 General XML Constructs

473 This chapter defines UBL rules related to general XML constructs to include:

- 474 ◆ Overall Schema Structure
- 475 ◆ Naming and Modeling Constraints
- 476 ◆ Reusability Scheme
- 477 ◆ Namespace Scheme
- 478 ◆ Versioning Scheme
- 479 ◆ Modularity Strategy
- 480 ◆ Schema Documentation Requirements

481 3.1 Overall Schema Structure

482 A key aspect of developing standards is to ensure consistency in their development.
483 Since UBL is envisioned to be a collaborative standards development effort, with liberal
484 developer customization opportunities through use of the `xsd:extension` and
485 `xsd:restriction` mechanisms, it is essential to provide a mechanism that will
486 guarantee that each occurrence of a UBL conformant schema will have the same look and
487 feel.

488 [GXS1] UBL Schema MUST conform to the following physical layout as applicable:

489 XML Declaration

490 `<!-- ===== Copyright Notice ===== -->`

491 “Copyright © 2001-2004 The Organization for the Advancement of Structured
492 Information Standards (OASIS). All rights reserved.

493 `<!-- ===== xsd:schema Element With Namespaces Declarations ===== -->`

494 `xsd:schema` element to include version attribute and namespace declarations in the
495 following order:

496 `xmlns:xsd`

497 Target namespace

498 Default namespace

499 `CommonAggregateComponents`

500 `CommonBasicComponents`

501 CoreComponentTypes
 502 Datatypes
 503 Identifier Schemes
 504 Code Lists
 505 Attribute Declarations – elementFormDefault=”qualified”
 506 attributeFormDefault=”unqualified”
 507 <!-- ===== Imports ===== -->CommonAggregateComponents schema module
 508 CommonBasicComponents schema module
 509 Representation Term schema module (to include CCT module)
 510 Common Basic Types schema module
 511 Common Aggregate Types schema module
 512 <!-- ===== Global Attributes ===== -->
 513 Global Attributes and Attribute Groups
 514 <!-- ===== Root Element ===== -->
 515 Root Element Declaration
 516 Root Element Type Definition
 517 <!-- ===== Element Declarations ===== -->
 518 alphabetized order
 519 <!-- ===== Type Definitions ===== -->
 520 All type definitions segregated by basic and aggregates as follows
 521 <!-- ===== Aggregate Business Information Entity Type Definitions ===== -->
 522 alphabetized order of ccts:AggregateBusinessInformationEntity xsd:TypeDefinitions
 523 <!-- =====Basic Business Information Entity Type Definitions ===== -->
 524 alphabetized order of ccts:BasicBusinessInformationEntities
 525 <!-- ===== Copyright Notice ===== -->
 526 Required OASIS full copyright notice.

527 **Example:**

528 **XML Schema**

```

529 <?xml version="1.0" encoding="UTF-8"?>
530 <!--===== Copyright Information =====-->
531 <!--
532     UBL XML Schema of Core Components
533
534     Copyright (C) OASIS-UBL (2003). All Rights Reserved.
535
536     This document and translations of it may be copied and furnished to
537     others, and derivative works that comment on or otherwise explain it
538     or assist in its implementation may be prepared, copied, published
  
```

539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567

```
and distributed, in whole or in part, without restriction of any
kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing
the copyright notice or references to UBL except as required to
translate it into languages other than English.

The limited permissions granted above are perpetual and will not be
revoked by UBL or its successors or assigns. This document and
the information contained herein is provided on an "AS IS" basis and
UN/CEFACT DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

Distribution of this document is unlimited.

=====

For our absent friend, Michael J. Adcock - il miglior fabbro
=====

UBL - XML Naming and Design Rules

Document Type:    <Status>
Generated On:     <Date>
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

568 **Namespaces**

569
570
571
572
573
574
575
576
577
578
579
580

```
... Namespace of Core Component Types ...
... Namespace of Representation Terms ...
... Namespace of Data Types ...
... Namespaces of reusable BBIEs ...
... Namespaces of reusable ABIEs ...
... Namespaces of Identifier Schemes ...
... Namespaces of Code Lists ...

elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

581 **Imports**

582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602

```
<!--==== Imports =====>
<!--=====-->
<!--==== Import of Core Component Types (CCT) =====>
... Import of Core Component Types (CCT) ...

<!--==== Import of Representation Terms (RT) =====>
... Import of Representation Terms (RT) ...

<!--==== Import of Data Types (DT) =====>
... Import of Data Types (DT) ...

<!--==== Imports of Identifier Schemes =====>
... Imports of Identifier Schemes in alphabetized order ...

<!--==== Imports of Code Lists =====>
... Imports of Code Lists in alphabetized order ...

<!--==== Import of reusable BBIEs =====>
... Import of reusable Basic Business Information Entities
(BBIEs) in alphabetized order ...
```

603
604
605

```
<!--==== Import of reusable ABIEs =====>
... Import of reusable Aggregation Business Information Entities
(ABIEs) in alphabetized order ...
```

606 Global Attributes

607
608
609

```
<!--==== Global Attributes =====>
... Global Attributes and Attribute Groups ...
```

610 Type Definitions

611
612
613
614
615
616
617
618
619
620

```
<!--==== Type Definitions =====>
<!--==== Type Definitions for local BBIEs =====>
... Type Definitions of Basic Business Information Entities
in alphabetized order ...

<!--==== Type Definitions for local ABIEs =====>
... Type Definitions of Aggregate Business Information Entities
in alphabetized order ....

</xs:schema>
```

621 XML Global Element Declaration

622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639

```
<!--==== Global Element Declarations =====>
<!--==== Root Element =====>
... Root Element ...

<!--==== Basic Business Information Entity Element Declarations
=====>
... Element Declarations of Basic Business Information Entities
in alphabetized order ....

<!--==== Aggregate Business Information Entity Element
Declarations
=====>
... Element Declarations of Aggregate Business Information
Entities
in alphabetized order ....

</xsd:schema>
```

640 [Ed. Note – Examples require changes to conform to the rules. Gunther has for action]

641 3.1.1 Root Element

642 Per XML 1.0, “There is exactly one element, called the **root**, or document element, no
643 part of which appears in the content of any other element.” XML 1.0 further states “The
644 [root element](#) of any document is considered to have signaled no intentions as regards
645 application space handling, unless it provides a value for this attribute or the attribute is
646 declared with a default value.” W3C XSD allows for any globally declared element to be
647 the document root element. To keep consistency in the instance documents and to adhere
648 to the underlying process model that supports each UBL Schema, it is desirable to have
649 one and only one element function as the root element. Since UBL follows a global
650 element declaration scheme (See Rule ELD2), each UBL Schema will identify one
651 element declaration in each schema as the document root element. This will be

652 accomplished through an `xsd:annotation` child element for that element in accordance
653 with the following rule:

654 [ELD1] Each `UBL:ControlSchema` MUST identify one global element declaration
655 that defines the overall business process being conveyed in the Schema
656 expression. That global element MUST include an `xsd:annotation` child
657 element which MUST further contain an `xsd:documentation` child
658 element that declares *"This element MUST be conveyed as the*
659 *root element in any instance document based on this*
660 *Schema expression."*

661 **[Definition] Control Schema –**

662 The overarching schema within a specific namespace that conveys the business
663 document functionality of that namespace. The control schema declares a target
664 namespace and is likely to pull in by including internal schema modules or importing
665 external schema modules. Each namespace will have one, and only one, control
666 schema.

667 Example:

```
668 <!--==== Root Element =====>  
669 <xsd:element name="Order" type="OrderType">  
670   <xsd:annotation>  
671     <xsd:documentation>  
672       <ccts:CategoryCode>ABIE</ccts:CategoryCode>  
673       <ccts:DictionaryEntryName>  
674         Order. Details</ccts:DictionaryEntryName>  
675       <ccts:Definition>A document that contains  
676 information directly relating to the economic event of ordering products.  
677 </ccts:Definition>  
678       <ccts:ObjectClass>Order</ccts:ObjectClass>  
679       <ccts:PropertyTerm>Details</ccts:PropertyTerm>  
680  
681       <ccts:RepresentationTerm>Details</ccts:RepresentationTerm>  
682       <ccts:BusinessTerm>PurchaseOrder</ccts:BusinessTerm>  
683     </xsd:documentation>  
684   </xsd:annotation>  
685 </xsd:element>
```

686 [Ed. Note – ensure example changed to reflect agreed upon documentation rules]

687 Additional root element rules are contained in Section 8.

688 3.2 Constraints

689 A key aspect of UBL is to base its work on process modeling and data analysis as
690 precursors to developing the UBL library. In determining how best to affect this work,
691 several constraints have been identified that directly impact on both the process modeling
692 and data analysis, as well as on the resultant UBL Schema.

693 3.2.1 Naming Constraints

694 A primary component of the UBL library documentation is its dictionary. The entries in
695 the dictionary fully define the pieces of information available for use in UBL business
696 messages. These entries contain fully conformant CCTS dictionary entry names as well
697 as truncated UBL XML element names developed in conformance with the rules in
698 section XX. The dictionary entry name ties the information to its standardized semantics,
699 while the name of the corresponding XML element or attribute is only shorthand for this
700 full name. The rules for element and attribute naming and dictionary entry naming are
701 different.

702 [NMC1] Each dictionary entry name MUST define one and only one fully qualified
703 path (FQP) for an element or attribute.

704 The fully qualified path anchors the use of that construct to a particular location in a
705 business message. The dictionary definition identifies any semantic dependencies that the
706 FQP has on other elements and attributes within the UBL library that are not otherwise
707 enforced or made explicit in its structural definition. The dictionary serves as a traditional
708 data dictionary, and also serves *some* of the functions of traditional implementation
709 guides.

710 3.2.2 Modeling Constraints

711 In keeping with UBL guiding principles, modeling constraints are limited to those
712 necessary to ensure consistency in development.

713 3.2.2.1 Defining Classes

714 UBL is based on instantiating ebXML `ccts:CoreComponents`. UBL models and the
715 XML expressions of those models are class driven. Specifically, classes are defined for
716 each `ccts:BasicBusinessInformationEntity` and `ccts:AggregateBusiness
717 InformationEntity` defined.

718 [MDC1] UBL Models MUST define classes based on ebXML
719 `ccts:BasicBusinessInformationEntities` and
720 `ccts:AggregateBusinessInformationEntities`.

721 **Example:**

722 **Basic Business Information Entity**

723

724 **Aggregate Business Information Entity**

725

726 [Ed. Note – need to have text based examples for one ccts:BBIE and one ccts:ABIE.
727 Volunteers?]

728 3.2.2.2 Core Component Types

729 Each `ccts:BasicBusinessInformationEntity` has an associated
730 `ccts:CoreComponentType`. The CCTS specifies an approved set of
731 `ccts:CoreComponentTypes`. To ensure conformance, UBL is limited to using this
732 approved set.

733 [MDC2] UBL Libraries and Schemas MUST only use ebXML Core Component
734 approved `ccts:CoreComponentTypes`.

735 3.2.2.3 Consistent Business Function in Customizations

736 A key aspect of UBL customization methodology is to ensure that customizations are
737 consistent with the original UBL document purpose.

738 [MDC3] If a UBL document is customized it MUST retain the business function of the
739 original UBL document.

740 3.2.2.4 Mixed Content Models

741 UBL documents are designed to effect data-centric electronic commerce. Including
742 mixed content in business documents is undesirable because business transactions are
743 based on exchange of discrete pieces of data that must be clearly unambiguous. The
744 white space aspects of mixed content makes processing unnecessarily difficult and adds a
745 layer of complexity not desirable in business exchanges.

746 [MDC4] Mixed content MUST NOT be used except where contained in an
747 `xsd:documentation` element.

748 3.3 Reusability Scheme

749 A fundamental question in determining UBL's approach to developing a reusable library
750 requires a decision on managing by types, or managing by types and elements. Put
751 another way, can UBL effectively manage its library through unique complex types, or
752 should UBL also concern itself with ensuring that all element declarations are unique
753 across the breadth of the UBL library. Put another way, should UBL elements be
754 declared globally or locally? Many questions surround this issue, and given its relative
755 importance, an understanding of the key factors in UBL's decision is important.

756 3.3.1 Managing by Types

757 Type information availability is unreliable in a distributed environment, since it
758 nominally requires an extra input (the schema) and since XML instance documents are
759 often passed around solo. In addition, type information (in the form of the PSVI, or post
760 schema-validation info) is so far standardized only in the most abstract sense – there is
761 no standard for an XML-based serialization of type information or an API that accesses
762 it. The existence of the PSVI in the XSD specification is frequently and strongly
763 criticized by many in the XML developer community for its complexity and its lack of
764 processing-pipeline clarity. While some sophisticated software is starting to emerge that
765 takes advantage of the PSVI, such as "data-binding" software that compiles schemas into
766 ready-to-use program classes that create and manipulate XML data in a type-aware
767 fashion, it is far from being the constant companion of XML programmers so far.

768 XPath and SAX both operate on well-formed XML instance documents just fine without
769 the presence of additional inputs, such as a schema that provides type information; in
770 fact, they don't even have access to type information without extra instance
771 transformations (for example, adding `xsi:type` attributes to every element). The typical
772 and natural way for them to operate on XML documents is primarily by name (possibly
773 qualified with a namespace), and not by type or by `xsi:type` attribute value.

774 3.3.1.1 Achieving the Assembly Use Case with Reusable Types

775 Assume the following scenario: The standard UBL notion of "Address" is perfectly
776 usable for a new message type called Foo. In this scenario, the developer doesn't want to
777 change Address; they just want to use it. One of the motivations for using pieces of the
778 UBL Library is that there are some software modules and stylesheets available that
779 support them already. The developer is willing to modify this software a little bit, but
780 would obviously like to do as little as possible in this regard.

781 For simplicity, let's say that UBL has only one `<Address>` element (remember that, with
782 locally declared elements, UBL could have many elements with the same name, although
783 all of these same-named elements typically have identical types in our case) and that this
784 element is locally defined in `PartyType`. With local unqualified elements, the relevant
785 definitions look like this (embedded documentation is stripped out for clarity):

```
786 <xsd:complexType name="PartyType">  
787   <xsd:element name="PartyID" type="IdentifierType"/>  
788   <xsd:element name="Name" type="NameType"/>  
789   <xsd:element name="Address" type="AddressType" minOccurs="0"/>  
790   ...  
791 </xsd:complexType>  
792  
793 <xsd:complexType name="AddressType">  
794   <xsd:element name="Identifier" type="IdentifierType" minOccurs="0"/>  
795   <xsd:element name="Street" type="StreetType" minOccurs="0"/>  
796   ...
```

797 `</xsd:complexType>`

798 The software we want to leverage by reusing UBL happens not to be type-aware. In
799 particular, there is a stylesheet that has templates with XPaths like these:

800 `//Party/Address`

801 `//Address`

802 There are two choices for reuse:

- 803 1. Bind the UBL `AddressType` type to an element
- 804 2. Bind the UBL `PartyType` type to an element and then use the actual UBL element
805 `<Address>` in the message

806 Choice #1 would look like this:

807 `<xsd:element name="FooAddress" type="ubl:AddressType"/>`

808 Instances conforming to the derived schema would contain this sort of markup:

```
809 <foo:Foo
810   xmlns:foo="some_namespace_name_for_foo">
811   ...
812   <foo:FooAddress>
813     <Identifier>...</Identifier> <!-- real UBL element -->
814     <Street>...</Street> <!-- real UBL element -->
815   </foo:FooAddress>
816   ...
817 </foo:Foo>
```

818 Any `//Address` XPaths in stylesheets would have to be changed to

819 `//foo:FooAddress` XPaths.

820 Choice #2 would look like this:

821 `<xsd:element name="FooParty" type="ubl:PartyType"/>`

822 Instances conforming to this derived schema would have real UBL `<Address>` elements
823 but would also require usage of the overall content model in which `<Address>` was
824 defined:

```
825 <foo:Foo
826   xmlns:foo="some_namespace_name_for_foo">
827   ...
828   <foo:FooParty> <!-- unwanted outer wrapper for Address -->
829     <PartyID>...</PartyID> <!-- real UBL element; undesired -->
830     <Name>...</Name> <!-- real UBL element; undesired -->
831     <Address> <!-- real UBL element; desired -->
832       <Identifier>...</Identifier>
833       <Street>...</Street>
```

```
834     ...
835     </Address>
836   </foo:FooParty>
837   ...
838 </foo:Foo>
```

839 The developer can use any existing `//Address` XPath in stylesheets, but if they didn't
840 want UBL's Party content model, there is a problem. They can not use UBL's real
841 `<Address>` element without the Party model coming along. Either way, a local
842 `<Address>` element is not truly a reusable component, and software reuse is unsatisfying
843 as well.

844 3.3.1.2 Reusable Elements

845 If UBL elements are global and qualified, rather than local and unqualified, then the
846 `<Address>` element will be directly reusable as a modular component and some software
847 can be used without modification. The UBL schema will look like this, creating
848 `<ubl:Party>` and `<ubl:Address>` elements:

```
849 <xsd:element name="Party" type="PartyType">
850
851 <xsd:complexType name="PartyType">
852   <xsd:element ref="PartyID"/>
853   <xsd:element ref="Name"/>
854   <xsd:element ref="Address" minOccurs="0"/>
855   ...
856 </xsd:complexType>
857
858 <xsd:element name="Address" type="AddressType">
859
860 <xsd:complexType name="AddressType">
861   <xsd:element ref="Identifier" minOccurs="0"/>
862   <xsd:element ref="Street" minOccurs="0"/>
863   ...
864 </xsd:complexType>
```

865 [Ed Note: Above example should be changed to reflect current schema. Lisa has for
866 action.]

867 XPaths will look like this:

```
868 //ubl:Party/ubl:Address
869 //ubl:Address
```

870 The `<Address>` element will be reused like this:

```
871 <xsd:element name="Foo" type="ubl:FooType"/>
872
873 <xsd:complexType name="FooType">
```

```
874 <xsd:element ref="ubl:Address"/>
875 ...
876 </xsd:complexType>
```

877 Instances conforming to this derived schema will look like this since qualified elements
878 are fully prefixed:

```
879 <foo:Foo
880   xmlns:foo="some_namespace_name_for_foo"
881   xmlns:ubl="ubl_namespace_name">
882   ...
883   <ubl:Address>
884     <ubl:Identifier>...</ubl:Identifier>
885     <ubl:Street>...</ubl:Street>
886     ...
887   </ubl:Address>
888   ...
889 </foo:Foo>
```

890 Software written to work with UBL's standard library will work with new assemblies of
891 the same components since global elements will remain consistent and unchanged.. The
892 globally declared <Address> element is fully reusable without regard to the reusability of
893 types and provides a solid mechanism for ensuring that extensions to the UBL core
894 library will provide consistency and semantic clarity regardless of its placement within a
895 particular type.

896 The only cases where locally declared elements are seen to be advantageous are in the
897 case of Identifiers and Code. Since identification schemes are often every specific to
898 trading partner and small communities, these constructs require specific processing and
899 can not be generically treated in software. There is no reuse benefit to declaring them as
900 global elements. Codes are treated as a special case in UBL which is also highly
901 configurable according to trading partner or community preference.

902 [ELD2] All element declarations MUST be global with the exception of ID and Code 903 which MUST be local.
--

904 3.4 Namespace Scheme

905 The concept of XML namespaces is defined in the W3C XML namespaces technical
906 specification.⁷ XML namespace features are available in the W3C XML Schema (XSD).
907 A namespace is declared in the root element of a Schema using a namespace identifier.
908 Namespace declarations can also identify an associated prefix – shorthand identifier –
909 that allows for compression of the namespace name. It is common for an instance
910 document to carry namespace declarations, so that it might be validated.

⁷ See Note 4.

911 3.4.1 Declaring Namespaces

912 Neither XML 1.0 or XSD require the use of Namespaces. However the use of
913 namespaces is essential to managing the complex UBL library. UBL will use UBL-
914 defined (schemas created by UBL) and UBL-used (schemas created by external
915 activities) and both require a consistent approach to namespace declarations.

916 [NMS1] Every UBL-defined or -used schema module MUST have a namespace
917 declared using the `xsd:targetNamespace` attribute.

918 Namespaces provide a mechanism for ensuring consistency and harmonization between
919 schema versions. Each UBL schema module consists of a logical grouping of lower level
920 artifacts that together comprise an association that will be able to be used in a variety of
921 UBL schemas. These schema modules are grouped into a schema set/collection. Each
922 schema set is assigned a namespace that identifies that group of schema modules. As
923 constructs are changed, new versions will be created. The schema set is the versioned
924 entity, all schema modules within that package are of the same version, and each version
925 has a unique namespace.

926 Definition. Schema Set

927 A collection of schema instances that together comprise the names in a specific UBL
928 namespace.

929 Schema validation ensures that an instance conforms to its declared schema. There are
930 never two (different) schemas with the same namespace URI. In keeping with Rule
931 NMS1, each UBL schema module will be part of a versioned namespace.

932 [NMS2] Every UBL defined or used schema set version MUST have its own unique
933 namespace.

934 UBL extension methodology will encourage a wide variety in the number of schema
935 modules that are created as derivations from UBL schema modules. Clarity and
936 consistency requires that customized schema not be confused with those developed by
937 UBL.

938 [NMS3] UBL namespaces MUST only contain UBL developed schema modules.

939 3.4.2 Namespace Uniform Resource Identifiers

940 This namespace identifier must be a Uniform Resource Identifier (URI) reference that
941 conforms to the Internet Engineering Task Force (IETF) request for comments (RFC)

942 2396, *Uniform Resource Identifiers: Generic Syntax*.⁸ There are two types of URIs:
943 Uniform Resource Locator (URL) and the Uniform Resource Name (URN).

944 As defined in RFC 2396, a URI is a “compact string of characters for identifying an
945 abstract or physical resource.” A URI scheme can be “a locator, a name, or both.” A URI
946 locator scheme is in the form of a URL, and a URI name scheme is of the form of a URN.
947 URLs generally define a location, but are not required to be a resolvable Internet or World
948 Wide Web address. URNs are required to provide a globally unique and persistent
949 reference even if the URL subset of the URI scheme ceases to exist.

950 UBL has adopted the URN scheme as the standard for URIs for UBL namespaces. UBL
951 namespaces must be consistent with the UBL versioning rules identified in Section 3.5.

952 Rule NMS2 requires separate namespaces for each UBL schema set. The UBL
953 versioning rules differentiate between committee draft and OASIS Standard status. For
954 each schema holding draft status, a UBL namespace must be declared and named.

955 [NMS4] The namespace names for UBL Schemas holding committee draft status
956 MUST be of the form:
957 `urn:oasis:names:tc:ubl:schema:<name>:<major>:<minor>[<revision>]`

958 Note:

959 [] = optional.

960 <> = variable

961 Definitions for optional and variable values are contained in Section 3.5.

962 For each UBL schema holding OASIS Standard status, a UBL namespace must be
963 declared and named using the same notation.

964 [NMS5] The namespace names for UBL Schemas holding OASIS Standard status
965 MUST be of the form:
966 `urn:oasis:names:specification:ubl:schema:<name>:<major>:<
967 minor>`
968

969 3.4.3 Schema Location

970 UBL schemas use a URN namespace scheme. Schema locations are typically defined as a
971 URL. UBL schema must be available both at design time and run time. As such, the UBL
972 schema locations will differ from the UBL namespace declarations. UBL, as an OASIS
973 TC, will utilize an OASIS URL for hosting UBL schemas.

⁸ T. Berners-Lee, R. Fielding, L. Masinter; *Internet Engineering Task Force (IETF) RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax, Internet Society, August 1998.*

974 [NMS6] UBL Schema modules MUST be hosted under the UBL committee directory:
975 <http://www.oasis-open.org/committees/ubl/schema/<schema->
976 [mod-name>.xsd](http://www.oasis-open.org/committees/ubl/schema/<schema-mod-name>.xsd)

977 3.4.4 Persistence

978 A key differentiator in selecting URNs for UBL namespaces is URN persistence. UBL
979 namespaces must never violate this functionality by subsequently changing a namespace
980 once it has been declared. Conversely, any changes to a schema will result in a new
981 namespace declaration. Thus a published schema version and its namespace association
982 will always be inviolate.

983 [NMS7] UBL published namespaces MUST never be changed.

984 3.5 Versioning Scheme

985 A UBL namespace URI is divided into three parts. First is the standard OASIS
986 namespace information, Second is the description of the purpose of the namespace.
987 Third is the version information. The version information will in turn be divided into
988 major and minor fields. The minor field has an optional revision extension. For
989 example, the namespace URI for the draft Invoice domain has this form:

990 urn:oasis:names:tc:ubl:schema:Invoice:<major-version>:<minor-
991 version>[<revision>]

992 The *major-version* field is “1” for the first release of a namespace. Subsequent major
993 releases increment the value by 1. For example, the first namespace URI for the first
994 major release of the Invoice domain has the form:

995 urn:oasis:names:tc:ubl:schema:Invoice:1:0

996 The second major release will have a URI of the form:

997 urn:oasis:names:tc:ubl:schema:Invoice:2:0

998 The distinguished value “0” (zero) is used in the *minor-version* position when defining a
999 new major version. In general, the namespace URI for every major release of the Invoice
1000 domain has the form:

1001 urn:oasis:names:tc:ubl:schema:Invoice:<major-number>:0:[<revision>]
1002

1003 [VER1] Every UBL Schema and schema module major version committee draft
1004 MUST have the URI of:

1005 `urn:oasis:names:tc:ubl:schema:<name>:<major>:0:[<revision>]`

1006 [VER2] Every UBL Schema and schema module major version OASIS Standard
1007 MUST have the URI of:

1008 `urn:oasis:names:specification:ubl:schema:<name>:<major>:0`

1009 In UBL, the major-version field of a namespace URI must be changed in a release that
1010 breaks compatibility with the previous release of that namespace. If a change does not
1011 break compatibility then only the minor version need change. Subsequent minor releases
1012 begin with *minor-version* 1.

1013 **Example:**

1014 Example

1015
1016 The namespace URI for the first minor release of the Invoice domain has this
1017 form:

1018
1019 `urn:oasis:names:tc:ubl:schema:Invoice:major-number:1`

1020

1021 [VER3] The first minor version release of a UBL Schema or schema module
1022 committee draft MUST have the URI of:

1023 `urn:oasis:names:tc:ubl:schema:<name>:<major-number>:<non-`
1024 `zero>:[<revision>]`

1025 [VER4] The first minor version release of a UBL schema or schema module OASIS
1026 Standard MUST have the URI of:

1027 `urn:oasis:names:specification:ubl:schema:name:major-number:non-`
1028 `zero`

1029

1030 Any change to any schema module mandates association to a new namespace, The
1031 implication is because once a schema and its associated namespaces are published by
1032 UBL they shall not change.

1033 [VER5] For UBL Minor version changes, the name of the version construct MUST
1034 NOT change (short name not qualified name), unless the intent of the change
1035 is to rename the construct.

1036 UBL is composed of a number of interdependent namespaces. For instance, namespaces
1037 whose URI's start with `urn:oasis:names:tc:ubl:schema:Invoice:*` are
1038 dependent upon the common basic and aggregate namespaces, whose URI's have the
1039 form `urn:oasis:names:tc:ubl:schema:CommonBasicComponents:*` and
1040 `urn:oasis:names:tc:ubl:schema:CommonAggregateComponents:*` respectively. If

1041 either of the common namespaces change then its namespace URI must change. If its
1042 namespace URI changes then any schema that imports the *new version* of the namespace
1043 must also change (to update the namespace declaration). And since the importing schema
1044 changes, its namespace URI in turn must change. The outcome is twofold:

1045 ◆ There is never ambiguity at the point of reference in a namespace declaration
1046 or version identification. A dependent schema imports precisely the version
1047 of the namespace that is needed. The dependent never needs to account for
1048 the possibility that the imported namespace can change.

1049 ◆ When a dependent is upgraded to import a new version of a schema, the
1050 dependent's version (in its namespace URI) must change.

1051 Version numbers are based on a logical progression. All major and minor version
1052 numbers will be based on positive integers. Version numbers will never move in a non-
1053 negative fashion.

1054 [VER4] Every UBL Schema and schema module major version number MUST be
1055 sequentially assigned, incremental number greater than zero.

1056 [VER5] Every UBL Schema and schema module minor version number MUST be a
1057 sequentially assigned, incremental non-negative integer.

1058 In keeping with rules NMS1 and NMS2, each schema minor version will be assigned a
1059 separate namespace.

1060 [VER6] Each UBL minor version MUST be given a separate namespace.

1061 A minor revision (of a namespace) *imports* the schema module for the previous version.
1062 For instance, the schema module defining:

1063 `urn:oasis:names:tc:ubl:schema:Invoice:1:2`

1064 *Must* import the namespace:

1065 `urn:oasis:names:tc:ubl:schema:Invoice:1:1`

1066 The `version 1:2` revision may define new complex types by extending or restricting
1067 `version 1:1` types. It may define brand new complex types and elements by
1068 composition. It must not use the XSD `redefine` element to change the definition of a type
1069 or element in the `1:1` version.

1070 The opportunity exists in the `version 1:2` revision to rename derived types. For
1071 instance if `version 1:1` defines `Address` and `version 1:2` specializes `Address` it
1072 would be possible to give the derived `Address` a new name, e.g. `NewAddress`. This is
1073 not required since namespace qualification suffices to distinguish the two distinct types.
1074 The minor revision may give a derived type a new name only if the semantics of the two
1075 types are distinct.

1076 For a particular namespace, the minor versions of a major version form a linearly-linked
1077 family. Each successive minor version imports the schema module of the preceding
1078 minor version. The process is bootstrapped by the first minor version importing the
1079 namespace defining the major version of interest.

1080 Example

```
1081 urn:oasis:names:tc:ubl:schema:Invoice:1:2 imports  
1082 urn:oasis:names:tc:ubl:schema:Invoice:1:1 which imports  
1083 urn:oasis:names:tc:ubl:schema:Invoice:1:0.  
1084
```

1086 [VER10] A UBL minor version control schema MUST import its immediately
1087 preceding minor version control schema.

1088 To ensure that backwards compatibility through polymorphic processing of minor
1089 versions within a major version, minor versions must be limited to certain allowed
1090 changes. This guarantee of backward compatibility is built into the `xsd:extension`
1091 mechanism. Thus, backward incompatible version changes can not be expressed using
1092 this mechanism.

1093 [VER8] UBL Schema and schema module minor version changes MUST be limited to
1094 the use of `xsd:extension` or `xsd:restriction` to alter existing types or
1095 add new constructs.

1096 In addition to polymorphic processing considerations, semantic compatibility across
1097 minor versions (as well as major versions) is essential.

1098 [VER9] UBL Schema and schema module minor version changes MUST not break
1099 semantic compatibility with prior versions.

1100

1101 3.6 Modularity

1102 There are many possible mappings of XML schema constructs to namespaces and to
1103 operating system files. As with other significant software artifacts, schemas can become
1104 large. In addition to the logical taming of complexity that namespaces provide, dividing
1105 the physical realization of schema into multiple operating system files-schema modules-
1106 provides a mechanism whereby reusable components can be imported as needed without
1107 the need to import overly complex complete schema.

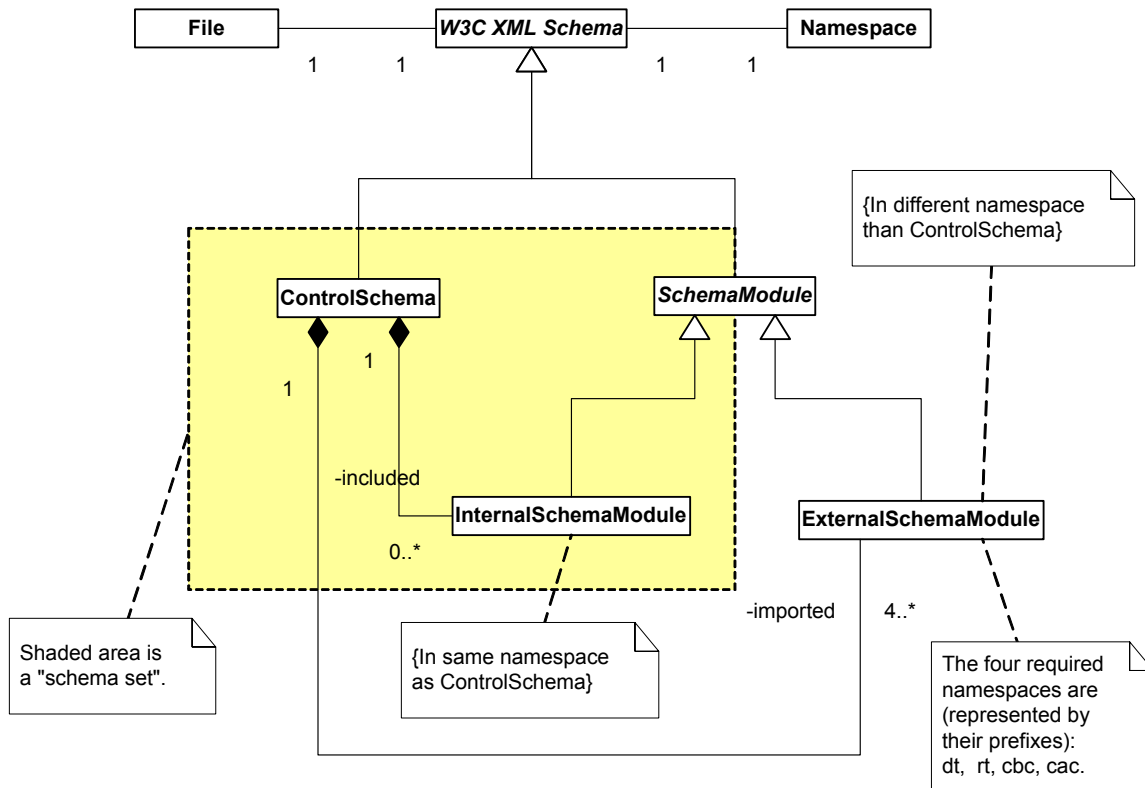
1108 [SSM1] UBL Schema expressions MAY be split into multiple schema modules.

1109 [Definition] schema module: A schema document containing type definitions and
1110 element declarations intended to be reused in multiple schemas.

1111 **3.6.1 UBL Modularity Model**

1112 UBL relies extensively on modularity in schema design. The UBL modularity approach
 1113 is structured so that a user can import individual components without getting the whole.
 1114 The UBL schema modularity model ensures that logical associations exist between root
 1115 and internal schema modules and that individual modules can be reused to the maximum
 1116 extent possible. This is accomplished through the use of control and internal schema
 1117 modules as shown in Figure 3-1.

1118 **Figure 3-1. UBL Schema Modularity Model**



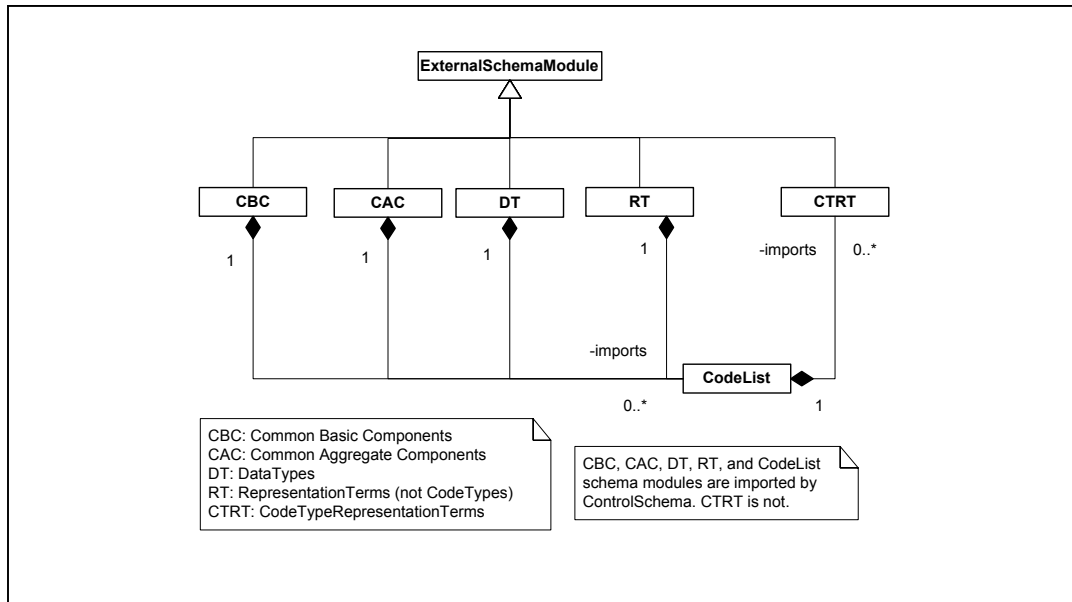
1119
 1120 If the contents of a namespace are small enough then they can be completely specified
 1121 within the control schema.
 1122 A namespace is an indivisible grouping of types. A “piece” of a namespace can never be
 1123 used without all its pieces. For larger namespaces, schema modules – internal schema
 1124 modules – may be defined. UBL control schema may have zero or more internal modules
 1125 that it includes. The control schema for that namespace then includes those internal
 1126 modules.

1127 **[Definition] Internal schema module:** A schema instance that is part of a schema set
 1128 within a specific namespace.

1129 [Ed. Note – Is this definition correct?]

1130 Figure 3-1 shows the 1-1 correspondence between control schemas and namespaces. It
 1131 also shows the 1-1 correspondence between files and schema modules. Another way to
 1132 visualize the structure is by example. Figure 3-2 depicts instances of the various classes
 1133 from the previous diagram.

1134 **Figure 3-2 Classes**

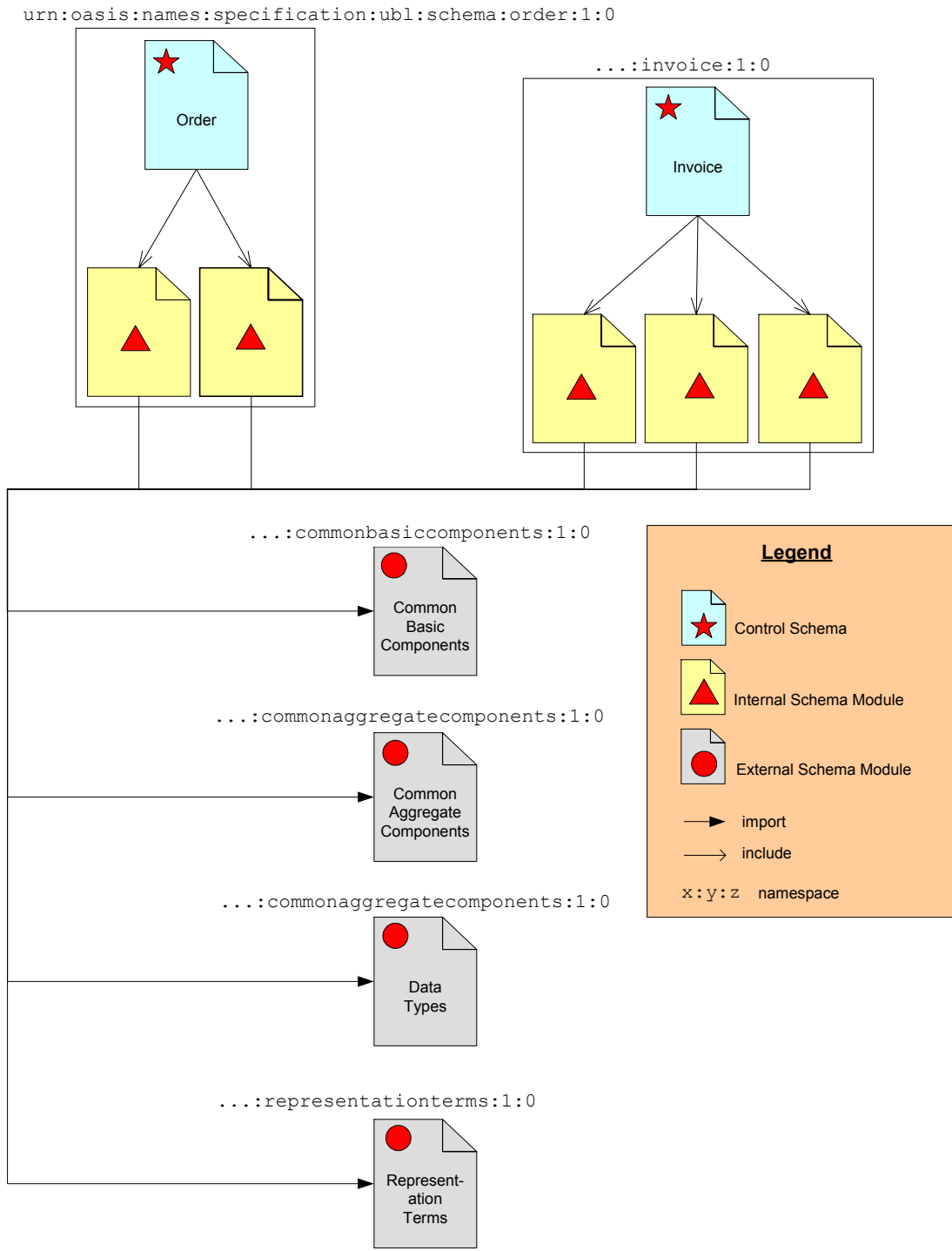


1135

1136 Figure 3-3 shows how the order and invoice control schemas import the
 1137 "CommonAggregateComponents" and "CommonBasicComponents" external schema
 1138 modules. It also shows how the order control schema includes various internal modules –
 1139 modules local to that namespace. The clear boxes show how the various schema modules
 1140 are grouped into namespaces.

1141 Any UBL schema module, be it a control schema or an internal module may import other
 1142 control schemas from other namespaces.

1143 **Figure 3-3 Order and Invoice Schema Import of Common Component Schema Modules**



1144

1145 **3.6.1.1 Limitations on Import**

1146 If two namespaces are mutually dependent then clearly, importing one will cause the
 1147 other to be imported as well. For this reason there must not exist circular dependencies
 1148 between UBL schema modules. By extension, there must not exist circular dependencies

1149 between namespaces. A namespace “A” dependent upon type definitions or element
1150 declaration defined in another namespace “B” must import “B’s” control schema.

1151 [SSM2] A control schema in one UBL namespace that is dependent upon type
1152 definitions or element declaration defined in another namespace MUST only
1153 import the control schema from that namespace.

1154 To ensure there is no ambiguity in understanding this rule, an additional rule is necessary
1155 to address potentially circular dependencies as well –schema A must not import internal
1156 schema modules of schema B.

1157 [SSM3] A UBL control schema in one UBL namespace that is dependant upon type
1158 definitions or element declarations defined in another namespace MUST NOT
1159 import internal schema modules from that namespace.

1160 3.6.1.2 Module Conformance

1161 UBL has defined a set of naming and design rules that are carefully crafted to ensure
1162 maximum interoperability and standardization.

1163 [SSM4] Imported schema modules MUST be fully conformant with UBL naming and
1164 design rules.

1165 3.6.2 Internal and External schema modules

1166 As defined in Rule SSM7, UBL will create schema modules. As illustrated in Figure 3-1
1167 and Figure 3-2, UBL schema modules will either be located in the same namespace as the
1168 corresponding control schema, or in a separate namespace.

1169 [SSM5] UBL schema modules MUST either be treated as external schema modules or
1170 as internal schema modules of the control schema.

1171 3.6.3 Internal schema modules

1172 UBL internal schema modules do not declare a target namespace, but instead reside in the
1173 namespace of their parent schema. All internal schema modules will be accessed using
1174 `xsd:include`.

1175 [SSM6] All UBL internal schema modules MUST be in the same namespace as their
1176 corresponding control schema.

1177 UBL internal schema modules will necessarily have semantically meaningful names.
1178 Internal schema module names will identify the parent schema module, the internal
1179 schema module function, and the schema module itself.

1180 [SSM7] Each UBL internal schema module MUST be named
1181 {ParentSchemaModuleName}{InternalSchemaModuleFunction}{sc
1182 hema module}

1183 [Ed. Note – need example here].

1184 3.6.4 External schema modules

1185 UBL is dedicated to maximizing reuse. As the complex types and global element
1186 declarations will be reused in multiple UBL schemas, a logical modularity approach is to
1187 create UBL schema modules based on collections of reusable types and elements.

1188 [SSM8] A UBL schema module MAY be created for reusable components.

1189 As identified in rule SSM2, UBL will create external schema modules. These external
1190 schema modules will be based on logical groupings of contents. At a minimum, UBL
1191 schema modules will be comprised of:

- 1192 ◆ UBL CommonAggregateComponents
- 1193 ◆ UBL CommonBasicComponents
- 1194 ◆ CCTS Core Component Types
- 1195 ◆ CCTS Representation Terms
- 1196 ◆ CCTS Code Type Representation Term
- 1197 ◆ CCTS Data Types

1198 3.6.4.1 UBL CommonAggregateTypes schema module

1199 The UBL library will also contain a wide variety of
1200 `ccts:AggregateBusinessInformationEntities`. As defined in rule MDC1, each
1201 of these will constitute a class in UBL. As defined in rule CTD1, each of these
1202 `ccts:AggregateBusinessInformationEntity` classes will be defined as an
1203 `xsd:complexType`. Although some of these `xsd:complexType`s may be used on
1204 only one UBL Schema, many will be reused in multiple UBL schema modules. An
1205 aggregation of all of the `ccts:AggregateBusinessInformationEntity`
1206 `xsd:ComplexType` definitions that are used in multiple UBL schema modules into a
1207 single schema module of common aggregate types will provide for maximum ease of
1208 reuse.

1209 [SSM9] A schema module defining all `ubl:CommonAggregateComponents` MUST
1210 be created.

1211 The normative name for this `xsd:ComplexType` schema module will be based on its
1212 `ccts:AggregateBusinessInformationEntity` content.

1213 [SSM10] The `ubl:CommonAggregateComponents` schema module MUST be named
1214 *"ubl:CommonAggregateTypes Schema Module"*

1215 ***3.6.4.1.1 UBL CommonAggregateTypes schema module Namespace***

1216 In keeping with the overall UBL namespace approach, a singular namespace must be
1217 created for storing the `ubl:CommonBasicTypes` schema module.

1218 [NMS8] The `ubl:CommonAggregateComponents` schema module MUST reside in
1219 its own namespace.

1220 To ensure consistency in expressing this module, a normative token that will be used in
1221 consistently in all UBL Schema must be defined.

1222 [NMS9] The `ubl:CommonAggregateComponents` schema module MUST be
1223 represented by the token “cac”.

1224 ***3.6.4.2 UBL CommonBasicTypes schema module***

1225 The UBL library will contain a wide variety of
1226 `ccts:BasicBusinessInformationEntities`. As defined in rule MDC1, each
1227 of these will constitute a class in UBL. As defined in rule CTD1, each of these
1228 `ccts:BasicBusinessInformationEntity` classes will be defined as an
1229 `xsd:ComplexType`. Although some of these `xsd:ComplexTypes` may be used on
1230 only one UBL Schema, many will be reused in multiple UBL schema modules. An
1231 aggregation of all of the `ccts:BasicBusinessInformationEntity`
1232 `xsd:ComplexType` definitions that are used in multiple UBL schema modules into a
1233 single schema module of common basic types will provide for maximum ease of reuse.

1234 [SSM11] A schema module defining all `ubl:CommonBasicComponents` MUST be
1235 created.

1236 The normative name for this schema module will be based on its
1237 `ccts:BasicBusinessInformationEntity` `xsd:ComplexType` content.

1238 [SSM12] The `ubl:CommonBasicComponents` schema module MUST be named
1239 “*ubl:CommonBasicComponents Schema Module*”

1240 ***3.6.4.2.1 UBL CommonBasicTypes schema module Namespace***

1241 In keeping with the overall UBL namespace approach, a singular namespace must be
1242 created for storing the `ubl:CommonBasicComponents` schema module.

1243 [NMS10] The `ubl:CommonBasicComponents` schema module MUST reside in its
1244 own namespace.

1245 To ensure consistency in expressing the `ubl:CommonBasicComponents` schema
1246 module, a normative token that will be used in consistently in all UBL Schema must be
1247 defined.

1248 [NMS11] The `UBL:CommonBasicComponents` schema module MUST be
1249 represented by the token “cbc”.

1250 3.6.4.3 CCTS Core Component Type schema module

1251 The CCTS defines an authorized set of Core Component Types
1252 (`ccts:CoreComponentTypes`) that convey content and supplementary information
1253 related to exchanged data. As the basis for all higher level CCTS models, the
1254 `ccts:CoreComponentTypes` are reusable in every UBL schema. An external
1255 schema module consisting of a complex type definition for each
1256 `ccts:CoreComponentType` is essential to maximize reusability.

1257 [SSM13] A schema module defining all `ccts:CoreComponentTypes` MUST be
1258 created.

1259 The normative name for the `ccts:CoreComponentType` schema module will be based
1260 on its content.

1261 [SSM14] The `ccts:CoreComponentType` schema module MUST be named
1262 “*ccts:CoreComponentType Schema Module*”

1263 By design, `ccts:CoreComponentTypes` are generic in nature. As such,
1264 restrictions are not appropriate. Such restrictions will be applied through the application
1265 of data types. Accordingly, the `xsd:facet` feature must not be used in the `ccts:CCT`
1266 schema module.

1267 [SSM15] The `xsd:facet` feature MUST not be used in the
1268 `ccts:CoreComponentType` schema module.

1269 3.6.4.3.1 Core Component Type schema module Namespace

1270 In keeping with the overall UBL namespace approach, a singular namespace must be
1271 created for storing the `ccts:CoreComponentType` schema module.

1272 [NMS12] The `ccts:CoreComponentType` schema module MUST reside in its own
1273 namespace.

1274 To ensure consistency in expressing the `ccts:CoreComponentType` schema module, a
1275 normative token that will be used in consistently in all UBL Schema must be defined.

1276 [NMS13] The `ccts:CoreComponentType` schema module namespace MUST be
1277 represented by the token “cct”.

1278 3.6.4.4 CCTS Representation Term schema module

1279 The CCTS defines an authorized set of primary and secondary Representation Terms
1280 (`ccts:RepresentationTerms`) that describes the form of every
1281 `ccts:BusinessInformationEntity`. These `ccts:RepresentationTerms` are

1282 reusable in every UBL schema. An external schema module consisting of a complex
1283 type definition for each `ccts:RepresentationTerm` is essential to maximize
1284 reusability. However, since UBL is also using code list schema modules that themselves
1285 import the `ccts:RepresentationTerms` schema module, a separate schema module for
1286 `ccts:CodeTypeRepresentationTerm` is also required, to avoid circular dependencies.

1287 [SSM16] A schema module defining all `ccts:PrimaryRepresentationTerms` and
1288 `ccts:SecondaryRepresentationTerms` with the exception of
1289 `ccts:CodeType` MUST be created

1290 [SSM17] A schema module defining the `ccts:CodeType`
1291 `ccts:RepresentationTerm` MUST be created.

1292 The normative name for the `ccts:RepresentationTerm` schema module will be
1293 based on its content.

1294 [SSM17] The `ccts:RepresentationTerm` schema module MUST be named
1295 "*ccts:RepresentationTerm Schema Module*"

1296 [SSM19] The `ccts:CodeTypeRepresentationTerm` schema module MUST be
1297 named "*ccts:CodeTypeRepresentationTerm Schema Module*"

1298 ***3.6.4.4.1 CCTS Representation Term schema module Namespace***

1299 In keeping with the overall UBL namespace approach, a singular namespace must be
1300 created for storing the `ccts:RepresentationTerm` and
1301 `ccts:CodeTypeRepresentationTerm` schema module.

1302 [NMS14] The `ccts:RepresentationTerm` schema module MUST reside in its own
1303 namespace.

1304 [NMS15] The `ccts:CodeTypeRepresentationTerm` schema module MUST reside
1305 in the `ccts:RepresentationTerm` namespace.

1306 To ensure consistency in expressing the `ccts:RepresentationTerm` schema
1307 module, a normative token that will be used in consistently in all UBL Schema must be
1308 defined.

1309 [NMS15] The `ccts:RepresentationTerm` schema module namespace MUST be
1310 represented by the token "rt".

1311 ***3.6.4.5 UBL Datatypes***

1312 CCTS stipulates Datatypes (`ccts:Datatypes`) will be defined for
1313 `ccts:BasicBusinessInformationEntity` properties. The `ccts:Datatype`
1314 defines the set of valid values that can be used for its associated
1315 `ccts:BasicBusinessInformationEntity` Property. The `ccts:Datatype` is
1316 defined by specifying restrictions on the `ccts:CoreComponentType` that forms the
1317 basis of the `ccts:DataType`. As datatypes are defined by individual users, they should
1318 be identified by those users. To ensure consistency of UBL datatypes

1319 (ubl:Datatypes)with the UBL modularity and reuse goals requires creating a single
1320 schema module that defines all ubl:Datatypes.

1321 [SSM18] A schema module defining all ubl:Datatypes MUST be created.

1322 The ubl:Datatypes schema module name must follow the UBL module naming
1323 approach.

1324 [SSM19] The UBL:Datatypes schema module MUST be named “ubl:Datatypes
1325 schema module”

1326 *3.6.4.5.1 UBL Datatype schema module Namespace*

1327 In keeping with the overall UBL namespace approach, a singular namespace must be
1328 created for storing the ubl:Datatypes schema module.

1329 [NMS16] The ubl:Datatypes schema module MUST reside in its own namespace.

1330 To ensure consistency in expressing the ubl:Datatypes schema module, a
1331 normative token that will be used in all UBL schemas must be defined.

1332 [NMS17] The ubl:Datatypes schema module namespace MUST be represented by
1333 the token “dt”.

1334 *3.7 Documentation*

1335 The UBL documentation also includes definitions of:

- 1336 ◆ XSD complex and simple types in the UBL library, including whether and
1337 how that type maps to a core component type
- 1338 ◆ The top-level whole message elements in UBL
- 1339 ◆ Global attributes
- 1340 ◆ Summaries of Code Lists
- 1341 ◆ UBL-specific Core Component Types
- 1342 ◆ UBL-specific representation terms

1343 The UBL documentation should be automatically generated to the extent possible, using
1344 embedded documentation fields in the structural definitions from the UBL library.

1345 *3.7.1 Embedded documentation*

1346 The information about each UBL BIE is in the library spreadsheets. UBL spreadsheets
1347 contain all necessary information to produce fully annotated Schema. Fully annotated
1348 Schema are valuable tools to implementers to assist in understanding the nuances of the

1349 information contained therein. UBL annotations will consist of information currently
1350 required by Section 7 of the CCTS and supplemented by necessary information identified
1351 by LCSC.

1352 The absence of an optional annotation inside the structured set of annotations in the
1353 documentation element implies the use of the default value. For example, there are
1354 several annotations relating to context such as BusinessTermContext or IndustryContext
1355 whose absence implies that their value is "all contexts".

1356 The following rule describes the documentation requirements for each Data Type
1357 definition.

1358 [DOC1] Every Data Type definition MUST contain a structured set of annotations in the
1359 following patterns:
1360

- 1361 ▪ UniqueIdentifier (mandatory): The identifier that references a Data
1362 Type instance in a unique and unambiguous way.
- 1363 ▪ CategoryCode (mandatory): The category to which the object
1364 belongs. For example, BBIE, ABIE, ASBIE, RT (Representation
1365 Term).
- 1366 ▪ DictionaryEntryName (mandatory): The official name of a Data
1367 Type.
- 1368 ▪ Definition (mandatory): The semantic meaning of a Data Type.
- 1369 ▪ Version (mandatory): An indication of the evolution over time of a
1370 Data Type instance.
- 1371 ▪ QualifierObjectClass (optional): The qualifier for the object class.
- 1372 ▪ ObjectClass: The Object Class represented by the Data Type.
- 1373 ▪ Qualifier Term (mandatory): A semantically meaningful name that
1374 differentiates the Data Type from its underlying Core Component
1375 Type.
- 1376 ▪ Usage Rule (optional, repetitive): A constraint that describes specific
1377 conditions that are applicable to the Data Type.

1378

1379 [DOC2] A Data Type definition MAY contain one or more Content Component
1380 Restrictions to provide additional information on the relationship between the
1381 Data Type and its corresponding Core Component Type. If used the Content
1382 Component Restrictions must contain a structured set of annotations in the
1383 following patterns:

- 1384 ▪ RestrictionType (mandatory): Defines the type of format restriction
1385 that applies to the Content Component.

1386
1387
1388
1389

- RestrictionValue (mandatory): The actual value of the format restriction that applies to the Content Component.
- ExpressionType (optional): Defines the type of the regular expression of the restriction value.

1390
1391
1392
1393
1394

[DOC3] A Data Type definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the Data Type and its corresponding Core Component Type. If used the Supplementary Component Restrictions must contain a structured set of annotations in the following patterns:

1395
1396
1397
1398
1399

- SupplementaryComponentName (mandatory): Identifies the Supplementary Component on which the restriction applies.
- RestrictionValue (mandatory, repetitive): The actual value(s) that is (are) valid for the Supplementary Component

1400
1401

The following rule describes the documentation requirements for each Basic Business Information Entity definition.

1402
1403

[DOC4] Every Basic Business Information Entity definition MUST contain a structured set of annotations in the following patterns:

1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424

- Unique Identifier (mandatory): The identifier that references a Basic Business Information Entity instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be BBIE.
- Dictionary Entry Name (mandatory): The official name of a Basic Business Information Entity.
- Version (mandatory): An indication of the evolution over time of a Basic Business Information Entity instance.
- Definition (mandatory): The semantic meaning of a Basic Business Information Entity.
- Cardinality (mandatory): Indication whether the Basic Business Information Entity Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Aggregate Business Information Entity.
- QualifierTerm (optional): Qualifies the Property Term of the associated Core Component Property in the associated Aggregate Core Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Basic Business Information Entity.

1425
1426
1427
1428
1429
1430
1431
1432
1433

- ConstraintLanguage (optional, repetitive): A formal description of a way the Basic Business Information Entity is derived from the corresponding stored Core Component and stored Business Context.
- BusinessTerm (optional, repetitive): A synonym term under which the Basic Business Information Entity is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a Basic Business Information Entity.

1434 The following rule describes the documentation requirements for each Aggregate
1435 Business Information Entity definition.

1436 [DOC5] Every Aggregate Business Information Entity definition MUST contain a
1437 structured set of annotations in the following patterns:

1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462

- UniqueIdentifier (mandatory): The identifier that references an Aggregate Business Information Entity instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ABIE.
- Version (mandatory): An indication of the evolution over time of an Aggregate Business Information Entity instance.
- DictionaryEntryName (mandatory): The official name of an Aggregate Business Information Entity.
- Definition (mandatory): The semantic meaning of an Aggregate Business Information Entity.
- QualifierTerm (mandatory): Qualifies the Object Class Term of the associated Aggregate Core Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Aggregate Business Information Entity.
- ConstraintLanguage (optional, repetitive): A formal description of a way the Aggregate Business Information Entity is derived from the corresponding stored Core Component and stored Business Context.
- BusinessTerm (optional, repetitive): A synonym term under which the Aggregate Business Information Entity is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an Aggregate Business Information Entity.

1463 The following rule describes the documentation requirements for each Association
1464 Business Information Entity definition.

1465 [DOC6] Every Association Business Information Entity definition MUST contain a
1466 structured set of annotations in the following patterns:

- 1467 ▪ UniqueIdentifier (mandatory): The identifier that references an
1468 Association Business Information Entity instance in a unique and
1469 unambiguous way.
- 1470 ▪ CategoryCode (mandatory): The category to which the object
1471 belongs. In this case the value will always be ASBIE.
- 1472 ▪ DictionaryEntryName (mandatory): The official name of an
1473 Association Business Information Entity.
- 1474 ▪ Definition (mandatory): The semantic meaning of an Association
1475 Business Information Entity.
- 1476 ▪ Version (mandatory): An indication of the evolution over time of an
1477 Association Business Information Entity instance.
- 1478 ▪ Cardinality (mandatory): Indication whether the Association
1479 Business Information Entity Property represents a not-applicable,
1480 optional, mandatory and/or repetitive characteristic of the Aggregate
1481 Business Information Entity.
- 1482 ▪ QualifierTerm (optional): Qualifies the Property Term of the
1483 associated Core Component Property in the associated Aggregate
1484 Core Component.
- 1485 ▪ UsageRule (optional, repetitive): A constraint that describes specific
1486 conditions that are applicable to the Association Business
1487 Information Entity.
- 1488 ▪ ConstraintLanguage (optional, repetitive): A formal description of a
1489 way the Association Business Information Entity is derived from the
1490 corresponding stored Core Component and stored Business Context.
- 1491 ▪ BusinessTerm (optional, repetitive): A synonym term under which
1492 the Association Business Information Entity is commonly known
1493 and used in the business.
- 1494 ▪ Example (optional, repetitive): Example of a possible value of an
1495 Association Business Information Entity.

1496 The following rule describes the documentation requirements for each Core Component
1497 definition.

1498 [DOC7] Every Core Component definition MUST contain a structured set of annotations
1499 in the following patterns:

- 1500 ▪ UniqueIdentifier (mandatory): The identifier that references a Core
1501 Component instance in a unique and unambiguous way.

1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517

- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be CCT.
- DictionaryEntryName (mandatory): The official name of a Core Component.
- Definition (mandatory): The semantic meaning of a Core Component.
- ObjectClass: The Object Class represented by the type.
- PropertyTerm: The Property Term represented by the type.
- Version (mandatory): An indication of the evolution over time of a Core Component instance.
- Usage Rule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Basic Business Information Entity.
- Business Term (optional, repetitive): A synonym term under which the Basic Business Information Entity is commonly known and used in the business.

1518 The following rule describes the documentation requirements for each element
1519 declaration.

1520
1521
1522
1523
1524

[DOC8] Every element declaration MUST contain an annotation as follows:

<Documentation>Dictionary Entry Name</Documentation> where
Dictionary Entry Name is the complete name (not the tag name) that is the
unique official name of the element in the UBL library.

1525 The following rule describes the documentation requirements for each UBL construct
1526 containing a code.

1527
1528
1529

[DOC9] For each UBL construct containing a code, the UBL documentation MUST
identify the zero or more code lists that MUST be minimally supported when
the construct is used.

1530
1531
1532
1533
1534
1535

- Prefix (mandatory): The code prefix, for example "cnt" for Country Code List.
- CodeListQualifier (mandatory): The qualifier for the codelist, for example "ISO 3166-1".
- CodeListAgency: The maintainer of the codelist, for example "6".
- CodeListVersion: The version of the codelist, for example "0.3".

1536 **3.7.2 Schema Annotation**

1537 An annotated schema will be provided to facilitate greater understanding of the schema
1538 module and its components. Additionally, a bare-bones schema will be provided that can
1539 be used at run-time if required to meet any resource constraints.

1540	[GXS2]	UBL MUST provide two normative schemas for each transaction. One
1541		schema shall be a run-time schema devoid of documentation. One schema
1542		shall be fully annotated.

1543

1544 4 Naming Rules

1545 The rules in this section make use of the following special concepts related to XML
1546 elements and attributes:

- 1547 ◆ Top-level element: An element that encloses a whole UBL business message.
1548 Note that UBL business messages might be carried by messaging transport
1549 protocols that themselves have higher-level XML structure. Thus, a UBL top-
1550 level element is not necessarily the root element of the XML document that
1551 carries it.
- 1552 ◆ Lower-level element: An element that appears inside a UBL business
1553 message.
- 1554 ◆ Intermediate element: An element not at the top level that is of a complex
1555 type, only containing other elements and attributes.
- 1556 ◆ Leaf element: An element containing only character data (though it may also
1557 have attributes). Note that, because of the XSD mechanisms involved, a leaf
1558 element that has attributes must be declared as having a complex type, but a
1559 leaf element with no attributes may be declared with either a simple type or a
1560 complex type.
- 1561 ◆ Common attribute: An attribute that has identical meaning on the multiple
1562 elements on which it appears. A common attribute might or might not
1563 correspond to an XSD global attribute.

1564 4.1 General Naming Rules

1565 The CCTS contains specific ISO/IEC 11179 based naming rules for each CCTS
1566 construct. The UBL component library, as a syntax-neutral representation, is fully
1567 conformant to those rules. The UBL syntax-specific XSD instantiation of the UBL
1568 component library, in some cases modifies the CCTS naming rules to leverage the
1569 capabilities of XML and XSD. Specifically, truncation rules are applied to allow for
1570 reuse of element names across parent element environments and to maintain brevity and
1571 clarity.

1572 In keeping with CCTS, UBL will use English as its normative language. If the UBL
1573 Library is translated into other languages for localization purposes, these additional
1574 languages might require additional restrictions. Such restrictions are expected be
1575 formulated as additional rules and published as appropriate.

1576 [GNR1] UBL XML element, attribute and type names MUST be in the English 1577 language, using the primary English spellings provided in the Oxford English 1578 Dictionary.

1579 UBL fully supports the concepts of data standardization contained in ISO 11179. CCTS,
1580 as an implementation of 11179, furthers its basic tenants of data standardization into
1581 higher level constructs as expressed by the CCTS dictionary entry names of those
1582 constructs – such as those for `ccts:BasicBusinessInformationEntities` and
1583 `ccts:AggregateBusinessInformationEntities`. Since UBL is an implementation of CCTS,
1584 UBL uses CCTS dictionary entry names as the basis for UBL XML artifact names.

1585 [GNR2] UBL XML element, attribute and type names MUST be taken from CCTS
1586 conformant dictionary entry names.

1587 The ISO 11179 specifies, and the CCTS uses, periods, spaces, other separators, and other
1588 characters not allowed by W3C XML. As such, these separators and characters are not
1589 appropriate for UBL XML component names.

1590 [GNR3] UBL XML element, attribute and type names constructed from
1591 `ccts:DictionaryEntryNames` MUST NOT include periods, spaces,
1592 other separators, or characters not allowed by W3C XML 1.0 for XML names.

1593 Acronyms and abbreviations impact on semantic interoperability and as such are to be
1594 avoided to the maximum extent practicable. Since some abbreviations will inevitably be
1595 necessary, UBL will maintain a normative list of authorized acronyms and abbreviations.
1596 Appendix B provides the current list of permissible acronyms, abbreviations and word
1597 truncations. The intent of this restriction is to facilitate the use of common semantics and
1598 greater understanding. Appendix B is a living document and will be updated to reflect
1599 growing requirements.

1600 [GNR4] UBL XML Element, attribute, and Simple and complex type names MUST
1601 NOT use acronyms, abbreviations, or other word truncations, except those in
1602 the list of exceptions published in Appendix B.

1603 UBL does not desire a proliferation of acronyms and abbreviations. Appendix B is an
1604 exception list and will be tightly controlled by UBL. Any additions will only occur after
1605 careful scrutiny to include assurance that any addition is critically necessary, and that any
1606 addition will not in any way create semantic ambiguity.

1607 [GNR5] Acronyms and abbreviations MUST only be added to the UBL approved
1608 acronym and abbreviation list after careful consideration for maximum
1609 understanding and reuse.

1610 Appendix B abbreviations and acronyms are taken directly from the most up-to-date
1611 version of the Pocket Oxford English Dictionary. Appendix B abbreviations and
1612 acronyms are taken directly from the most up-to-date version of the Pocket Oxford
1613 English Dictionary.

1614 [GNR6] Acronyms and abbreviations added to the UBL approved list MUST only be
1615 taken from the latest version of the Pocket Oxford English Dictionary. The
1616 first occurrence listed for a word MUST be used.

1617 Once an acronym or abbreviation has been approved, it is essential to ensuring semantic
1618 clarity and interoperability that the acronym or abbreviation is ***always*** used.

1619 [GNR7] The acronyms and abbreviations listed in Appendix B **MUST** always be used.

1620 Generally speaking the names for UBL XML constructs must always be singular, the
1621 only exception permissible is where the concept itself is pluralized.

1622 [GNR8] UBL XML element, attribute and type names **MUST** be in singular form
1623 unless the concept itself is plural (example: Goods).

1624 XML is case sensitive. Consistency in the use of case for a specific XML component
1625 (element, attribute, type) is essential to ensure every occurrence of a component is treated
1626 as the same. This is especially true in a business-based data-centric environment as is
1627 being addressed by UBL. Additionally, the use of visualization mechanisms such as
1628 capitalization techniques assist in ease of readability and ensure consistency in
1629 application and semantic clarity. The ebXML architecture document specifies a standard
1630 use of camel case for expressing XML elements and attributes.⁹ UBL will adhere to the
1631 ebXML standard. Specifically, UBL element and type names will be in UpperCamelCase
1632 (UCC).

1633 [GNR9] The UpperCamelCase (UCC) convention **MUST** be used for naming elements
1634 and types.

1635 Example:

1636 TransportEquipmentSeal
1637 ChargeIndicator

1639

1640 UBL attribute names will be in lowerCamelCase (LCC).

1641 [GNR10] The lowerCamelCase (LCC) convention **MUST** be used for naming attributes.

1642 Example:

1643 AmountCurrencyCodeListVersionIdentifier
1644 binaryObjectFilenameText

1645

1646 4.2 Type Naming Rules

1647 UBL identifies several categories of naming rules for types, namely for complex types
1648 based on Aggregate Business Information Entities, Basic Business Information Entities,
1649 Primary Representation Terms, Secondary Representation Terms and the Core
1650 Component Type.

⁹ *ebXML*, ebXML Technical Architecture Specification v1.0.4, 16 February 2001

1651 4.2.1 Complex Type Names for CCTS Aggregate Business
1652 Information Entities

1653 A ccts:AggregateBusinessInformationEntity ccts:DictionaryEntryName is a fully
1654 qualified construct based on ISO 11179. As such, this name conveys explicit semantic
1655 clarity with respect to the data being described. Accordingly, this
1656 ccts:DictionaryEntryName provides a mechanism for ensuring that UBL type names are
1657 semantically unambiguous, and that there is no duplication of UBL type names for
1658 different type constructs.

1659 [CTN1] A UBL xsd:complexType name based on an
1660 ccts:AggregateBusinessInformationEntity MUST be the
1661 ccts:DictionaryEntryName with the separators removed and with the
1662 “Details” suffix replaced with “Type”.

1663 **Example:**

1664 Ccts:AggregateBusinessInformationEntity UBL xsd:complexType

1665

```
1666 <!--==== Aggregate Business Information Entity Type Definitions  
1667 =====>  
1668 <xsd:complexType name="TransportEquipmentSealType">  
1669     ...  
1670 </xsd:complexType>
```

1671 The element thus created is useful for reuse in the building of new business messages.
1672 The complex type thus created is useful for both reuse and customization, in the building
1673 of both new and contextualized business messages.

1674 4.2.2 Complex Type Names for CCTS Basic Business Information
1675 Entities

1676 [CTN2] A UBL xsd:complexType name based on a ccts:BBIE MUST be the
1677 ccts:DictionaryEntryName property term and qualifiers and representation
1678 term, with the separators removed and with the “Type” suffix appended after
1679 the representation term.

1680

1681 **Example:**

```
1682 <!--==== Basic Business Information Entity Type Definitions =====>  
1683 ->  
1684 <xsd:complexType name="ChargeIndicatorType">  
1685     ...  
1686 </xsd:complexType>
```

1687

1688

1689 4.2.3 Complex Type Names for CCTS Representation Terms

1690

1691 [CTN3] A UBL xsd:complexType name based on a primary representation term used
1692 in the UBL model MUST be the name of the corresponding ccts:CCT, with
1693 the separators removed and with the “Type” suffix appended after the primary
1694 representation term name.

1695

1696 **Example:**

```
1697 <!-- ===== Primary Representation Term: AmountType ===== -->  
1698 <xsd:complexType name="AmountType">  
1699 ...  
1700 </xsd:complexType>
```

1701

1702 [CTN4] A UBL xsd:complexType name based on a secondary representation term
1703 used in UBL model MUST be the name of the secondary representation term,
1704 with the separators removed and with the “Type” suffix appended after the
1705 secondary representation term name.

1706 **Example:**

```
1707 <!-- ===== Secondary Representation Term: GraphicType ===== -->  
1708 <xsd:complexType name="GraphicType">  
1709 ...  
1710 </xsd:complexType>
```

1711

1712 4.2.4 Complex Type Names for CCTS Core Component Types

1713

1714 [CTN5] A UBL xsd:complexType name based on a ccts:CCT MUST be the
1715 Dictionary entry name of the ccts:CCT, with the separators removed.

1716

1717 **Example:**

```
1718 <!-- ===== CCT: QuantityType ===== -->  
1719 <xsd:complexType name="QuantityType">  
1720 ...  
1721 </xsd:complexType>
```

1722 4.3 Element Naming Rules

1723 As defined in the UBL Model (See Figure 2-3), UBL elements will be created for
1724 ccts:AggregateBusinessInformationEntities, ccts:BasicBusinessInformationEntities, and
1725 ccts:AssociationBusinessInformationEntities. UBL element names will reflect this
1726 relationship in full conformance with ISO11179 element naming rules.

1727 4.3.1 Element Names for CCTS Aggregate Business Information 1728 Entities

1729 [ELN1] A UBL global element name based on an ccts:ABIE MUST be the same as
1730 the name of the corresponding xsd:complexType to which it is bound,
1731 with the word "Type" removed.

1732 Example:

1733 For a ccts:AggregateBusinessInformationEntity of Party. Details,
1734 Rule CTN1 states that the Party. Details object class becomes PartyType
1735 xsd:ComplexType. Rule ELD3 states that for the PartyType
1736 xsd:ComplexType, a corresponding global element must be declared. Rule
1737 ELN1 states that the name of this corresponding global element must be Party.
1738

```
1739 <!--==== Aggregate Business Information Entity Type Definitions  
1740 =====>  
1741 <xsd:complexType name="TransportEquipmentSealType">  
1742     ...  
1743 </xsd:complexType>  
1744     ...  
1745 <!--==== Aggregate Business Information Entity Element  
1746 Declarations  
1747 =====>  
1748 <xsd:element name="TransportEquipmentSeal"  
1749             type="TransportEquipmentSealType"/>
```

1750 4.3.2 Element Names for CCTS Basic Business Information 1751 Entities

1752 The same naming concept applies to ccts:BasicBusinessInformationEntities

1753 [ELN2] A UBL global element name based on a ccts:BBIE MUST be the same as
1754 the name of the corresponding xsd:complexType to which it is bound,
1755 with the word "Type" removed.

1756 Example:

```
1757 <!--==== Basic Business Information Entity Type Definitions =====>  
1758 ->  
1759 <xsd:complexType name="ChargeIndicatorType">  
1760     ...  
1761 </xsd:complexType>  
1762     ...  
1763 <!--==== Basic Business Information Entity Element Declarations  
1764 =====>
```

1765

```
<xsd:element name="ChargeIndicator" type="ChargeIndicatorType"/>
```

1766 4.3.3 Element Names for CCTS Association Business Information 1767 Entities

1768 A `ccts:AssociationBusinessInformationEntity` is not a class like
1769 `ccts:AggregateBusinessInformationEntities` and `ccts:BasicBusiness`
1770 `InformationEntities` are. Rather, it is an association between two classes. As such,
1771 an element representing the `ccts:AssociationBusinessInformationEntity` does
1772 not have its own unique `xsd:ComplexType`, rather the element is bound to the
1773 `xsd:complexType` of its associated `ccts:AssociationBusinessInformation`
1774 `Entity`.

1775 [ELN4] A UBL global element name based on an `ccts:ASBIE` MUST be the
1776 `ccts:ASBIE` dictionary entry name property term and qualifiers; and the
1777 object class term and qualifiers of its associated `ccts:ABIE`. All
1778 `ccts:DictionaryEntryName` separators MUST be removed.
1779 Redundant words in the `ccts:ASBIE` property term or qualifiers and the
1780 associated `ccts:ABIE` object class term or qualifiers MUST be dropped.

1781

1782 **Example:**

1783 [Ed. Note – need to insert example here]

1784 4.4 Attribute Naming Rules

1785 UBL, as a transactional based XML exchange format, has chosen to significantly restrict
1786 the use of attributes. This restriction is in keeping with the Attribute usage is relegated to
1787 supplementary components only; all “primary” business data appears exclusively in
1788 element content.

1789 [ATN1] Each `CCT:SupplementaryComponent` `xsd:attribute` “name”
1790 MUST be the `ccts:SupplementaryComponent` dictionary entry name
1791 property term and representation term, with the separators removed.

1792 Example:

<code>ccts:SupplementaryComponent</code>	<code>ubl:attribute</code>

1793

1794 **Example:**

1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812

For the preceding example, the attribute declarations would look like this:

```
<!-- ===== CCT: QuantityType ===== -->
<xsd:complexType name="QuantityType">
  ...
  <xsd:attribute name="unitCode"
type="xsd:token" use="optional"/>
  <xsd:attribute name="unitCodeListID"
type="xsd:token"
          use="optional"/>
  <xsd:attribute name="unitCodeListAgencyID"
type="xsd:token"
          use="optional"/>
  <xsd:attribute name="unitCodeListAgencyName"
type="xsd:token"
          use="optional"/>
  ...
</xsd:complexType>
```

1813

1814 [Ed. Note – this is wrong! The attribute can only be a pre-defined
1815 ccts:SupplementaryComponent]

1816 5 Declarations and Definitions

1817 In W3C XML Schema, elements are defined in terms of complex or simple types and
1818 attributes are defined in terms of simple types. The rules in this section govern the
1819 consistent structuring of these type constructs and the manner for unambiguously and
1820 thoroughly documenting them in the UBL Library

1821 5.1 Type Definitions

1822 5.1.1 General Type Definitions

1823 Since UBL elements and types are intended to be reusable, all types must be named. This
1824 permits other types to establish elements that reference these types, and also supports the
1825 use of extensions for the purposes of versioning and customization.

1826 [GTD1] All types MUST be named.

1827 Example:

```
1828 <xsd:complexType name="QuantityType">  
1829   ...  
1830 </xsd:complexType>
```

1831 UBL disallows the use of `xsd:any`, because it permits the introduction of potentially
1832 unknown elements into an XML instance. UBL intends that all constructs within the
1833 instance be described by the schemas describing that instance - `xsd:any` is seen as
1834 working counter to the requirements of interoperability.

1835 [GTD2] The `xsd:any` Type MUST NOT be used.

1836

1837 5.1.2 Simple Types

1838 The Core Components Specification provides a set of constructs for the modeling of
1839 basic data, Core Component Types. These are represented in UBL with a library of
1840 complex types, with the effect that most "simple" data is represented as property sets
1841 defined according to the CCTs, made up of content components and supplementary
1842 components. In most cases, the supplementary components are expressed as XML
1843 attributes, and the content component becomes element content, and the CCT is
1844 represented with an `xsd:complexType`. There are exceptions to this rule in those cases
1845 where all of a CCTs properties can be expressed without the use of attributes. In these
1846 cases, an `xsd:simpleType` is used.

1847 [STD1] For every `ccts:CCT` whose supplementary components are equivalent to the
1848 properties of a built-in `xsd:datatype`, the
1849 `CCT:SupplementaryComponents` MUST NOT be expressed as
1850 attributes, and the `ccts:CCT` MUST be defined as a named `simpleType` in
1851 the `ccts:CCT` schema module.

1852 [ED NOTE: Proposed wording change to the rule above: "For every `ccts:CCT`
1853 whose supplementary components map directly onto the properties of a
1854 built-in `xsd:datatype`, the `ccts:CCT` MUST be represented as a named
1855 `xsd:simpleType` in the `ccts:CCT` Schema Module."]

1856

1857 [STD2] Each `ccts:CCT` `simpleType` definition name MUST be the `ccts:CCT`
1858 dictionary entry name with the separators removed.

1859 **Example:**

```
1860 <!-- ===== CCT: DateTimeType ===== -->  
1861 <xsd:simpleType name="DateTimeType">  
1862   ...  
1863   <xsd:restriction base="cct:DateTimeType"/>  
1864 </xsd:simpleType>
```

1865 Because CCTs represent primitives, they are not allowed to be restrictions of other types.

1866 [STD3] `xsd:simpleType` restriction MUST NOT be used for `ccts:CCTs`.

1867 [ED NOTE: The word "restriction" above is somewhat unclear - proposed re-
1868 wording: "`xsd:simpleType` MUST NOT be used to represent `ccts:CCTs`."]

1869 5.1.3 Complex Types

1870 Since even simple data types are modeled as property sets in most cases, the XML
1871 expression of these models primarily employs `xsd:complexType`. To facilitate reuse,
1872 versioning, and customization, all complex types are named. The main exception to this
1873 form of representation concerns Aggregate Business Information Entities, which
1874 represent the relationship between an aggregate "parent" object and its aggregate
1875 properties, or children.

1876 [CTD1] For every class identified in the UBL model, a named `xsd:complexType`
1877 MUST be defined.

1878 [ED NOTE: This is ambiguous - aggregation business entities are also
1879 expressed as classes in the model, but are not represented by complex
1880 types. We should list out the types of classes in the model that are
1881 represented as complex types.]

1882 **Example:**

```
1883 <xsd:complexType name="BasePriceType">  
1884   ...  
1885 </xsd:complexType>
```

1886 5.1.3.1 Aggregate Business Information Entities

1887 The relationship expressed by an Aggregate Business Information Entity is not directly
1888 represented with a class. Instead, this relationship is captured in UBL with a containment
1889 relationship, expressed in the content model of the parent object's type with a sequence
1890 of elements. (Sequence facilitates the use of `xsd:extension` for versioning and
1891 customization.) The members of the sequence – elements which are themselves defined
1892 by reference to complex types – are the properties of the containing type.

1893 [CTD4] Every `ccts:ABIE` `xsd:complexType` definition content model MUST
1894 use the `xsd:sequence` element with appropriate global element references,
1895 or local element declarations in the case of `ID` and `Code`, to reflect each
1896 property of its class as defined in the corresponding UBL model.

1897 Example:

```
1898 <xsd:complexType name="ContactType">  
1899   ...  
1900   <xsd:sequence>  
1901     ...  
1902     <xsd:element ref="Name" minOccurs="0">  
1903       ...  
1904     </xsd:element>  
1905     <xsd:element ref="Phone" minOccurs="0">  
1906       ...  
1907     </xsd:element>  
1908     ...  
1909   </xsd:sequence>  
1910 </xsd:complexType>
```

1911

1912 5.1.3.2 Basic Business Information Entities

1913 Basic Business Information Entities (BBIEs), in accordance with the Core Components
1914 Technical Specification, always have a primary representation term, and may have
1915 secondary representation terms, which describes their structural representation. These
1916 representation terms are bound to a Core Component Types that describe their structure.
1917 There are a set of rules concerning the way these relationships are expressed in the UBL
1918 XML library. BBIEs are represented with complex types. Within these are `simpleContent`
1919 elements that extend representation terms, which are themselves represented by types.
1920 This extension is a re-naming

1921 [CTD5] Every `ccts:BBIE` `xsd:complexType` definition content model MUST
1922 use the `xsd:simpleContent` element.

1923

1924 [CTD6] Every `ccts:BBIE` `ComplexType` content model
1925 `xsd:simpleContent` element MUST consist of an `xsd:extension`
1926 element.

1927
1928
1929

[ED NOTE: Arofan feels rule CTD7 is unneeded because you **cannot** have a valid extension element without providing a value for the "base" attribute. Delete.]

1930

1931 [CTD7] Every `ccts:BBIE xsd:complexType` content model
1932 `xsd:extension` element MUST use the `xsd:base` attribute to define the
1933 basis of each primary or secondary representation term.

1934

1935 [CTD8] Every `ccts:BBIE xsd:complexType` content model `xsd:base`
1936 attribute value MUST be the `ccts:CCT` of the primary representation term
1937 or the datatype of the secondary representation term as appropriate.

1938 **Example:**

1939
1940
1941
1942
1943

```
<xsd:complexType name="AdditionalStreetNameType">  
  <xsd:simpleContent>  
    <xsd:extension base="cct:NameType"/>  
  </xsd:simpleContent>  
</xsd:complexType>
```

1944

1945 5.1.3.3 Representation Terms

1946 Representation terms describe the representation of a BBIE. In the UBL XML Library,
1947 they are expressed as complex types. The relationship between primary and secondary
1948 representation terms is that the secondary representation terms are specializations of
1949 primary representation terms. Thus, each representation term is expressed as a separate
1950 complex type.

1951 [CTD2] For every primary representation term used in the UBL model, a named
1952 `xsd:complexType` MUST be defined.

1953 **Example:**

1954
1955
1956
1957

```
<!-- ===== Primary Representation Term: MeasureType ===== -->  
<xsd:complexType name="MeasureType">  
  ...  
</xsd:complexType>
```

1958

1959 [CTD3] For every secondary representation term used in the UBL model, a named
1960 `xsd:complexType` MUST be defined.

1961 **Example:**

1962
1963
1964

```
<!-- ===== Secondary Representation Term: NameType ===== -->  
<xsd:complexType name="NameType">  
  ...
```

1965

```
</xsd:complexType>
```

1966

1967 [Gunthers Note: we have simpleTypes for restricted representation types by using
1968 specific built-in data-types] [ED NOTE: Arofan feels we will need to add some wording
1969 – must ask Gunther about this.]

1970 5.1.3.4 Core Component Types

1971 A CCT consists of a “content component” which may be supported by a set of properties
1972 referred to as “supplementary components”. CCTs may be expressed as a simple type
1973 (where possible), but may require expression as a complex type. Content components are
1974 expressed as extensions of the set of built-in xsd data types. Supplementary components
1975 are expressed either as extensions of built-in data types, or user-defined simple types.

1976 [CTD9] For every `ccts:CCT` whose supplementary components are not equivalent to
1977 the properties of a built-in `xsd:datatype`, the `ccts:CCT` MUST be
1978 defined as a named `xsd:complexType` in the `ccts:CCT` schema
1979 module.

1980 CCTs complex types always have `xsd:simpleContent`, which is an extension of a built-in
1981 `xsd` data type.

1982 [CTD10] Each `ccts:CCT` `xsd:complexType` definition MUST contain one
1983 `xsd:simpleContent` element

1984

1985 [CTD11] The `ccts:CCT` `xsd:complexType` definition `xsd:simpleContent`
1986 element MUST contain one `xsd:extension` element. This
1987 `xsd:extension` element MUST include an `xsd:base` attribute that
1988 defines the specific `xsd:built-inDatatype` required for the
1989 `ccts:ContentComponent` of the `ccts:CCT`.

1990 Example:

1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007

```
<!-- ===== CCT: QuantityType ===== -->
<xsd:complexType name="QuantityType">
  ...
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="unitCode"
type="xsd:token" use="optional"/>
      <xsd:attribute name="unitCodeListID"
type="xsd:token"
use="optional"/>
      <xsd:attribute name="unitCodeListAgencyID"
type="xsd:token"
use="optional"/>
      <xsd:attribute name="unitCodeListAgencyName"
type="xsd:token"
use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

2008
2009

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

2010 5.1.3.5 Supplementary Components

2011 Supplementary components are expressed with references to either built-in xsd data
2012 types, or to user-defined simple types.

2013 [CTD12] Each `CCT:SupplementaryComponent` `xsd:attribute` “type” MUST
2014 define the specific `xsd:built-in Datatype` or the user defined
2015 `xsd:simpleType` for the `ccts:SupplementaryComponent` of the
2016 `ccts:CCT`.

2017 **Example:**

```
<xsd:attribute name="unitCode"
type="xsd:token" use="optional"/>
```

2020

2021 [CTD13] Each `ccts:SupplementaryComponent` `xsd:attribute` user-defined
2022 `xsd:simpleType` MUST only be used when the
2023 `ccts:SupplementaryComponent` is based on a standardized code list for
2024 which a UBL conformant code list schema module has been created.

2025 **Example:**

```
<xsd:complexType name="AmountType">
...
<xsd:simpleContent>
<xsd:restriction base="cct:AmountType">
<xsd:attribute name="currencyID"
type="iso4217:CurrencyCodeContent"
use="required"/>
...
</xsd:restriction>
</xsd:simpleContent>
</xsd:complexType>
```

2037 <Gunthers Notes: The codelist inclusion in attributes must need some further rules>

2038 [ED NOTE: Add text here to explain].

2039 [ED NOTE: Is the wording of CTD14 mangled? This is confusing, and either needs more
2040 explanation or re-wording or both.]

2041 [CTD14] Each `ccts:SupplementaryComponent` `xsd:attribute` user
2042 defined `xsd:simpleType` MUST be the same `xsd:simpleType` from
2043 the appropriate UBL conformant code list schema module for that type.

2044 **Example:**

```
2045 <xsd:complexType name="AmountType">
2046   ...
2047   <xsd:simpleContent>
2048     <xsd:restriction base="cct:AmountType">
2049       <xsd:attribute name="currencyID"
2050         type="iso4217:CurrencyCodeContent"
2051       use="required"/>
2052     ....
2053   </xsd:restriction>
2054 </xsd:simpleContent>
2055 </xsd:complexType>
```

2056 The same simple Type from the appropriate UBL conformant code list Schema Module
2057 for CurrencyCodeContent:

```
2058 <xsd:simpleType name="CurrencyCodeContent">
2059   <xsd:restriction base="xsd:token">
2060     <xsd:maxLength value="3"/>
2061     ...
2062   </xsd:restriction>
2063 </xsd:simpleType>
```

2064 Supplementary components are either required or optional, based on the description of
2065 CCTs in the Core Components Technical Specification.

2066 [CTD15] Each ccts:Supplementary Component xsd:attribute “use”
2067 MUST define the occurrence of that ccts:SupplementaryComponent
2068 as either “required”, or “optional”.

2069 **Example:**

```
2070 <!-- ===== CCT: AmountType ===== -->
2071 <xsd:complexType name="AmountType">
2072   ...
2073   <xsd:simpleContent>
2074     <xsd:extension base="xsd:decimal">
2075       <xsd:attribute name="currencyID"
2076       type="xsd:token"
2077         use="required"/>
2078       <xsd:attribute name="codeListVersionID"
2079       type="xsd:token"
2080         use="optional"/>
2081     </xsd:extension>
2082   </xsd:simpleContent>
2083 </xsd:complexType>
```

2084 [Gunthers Notes: The ‘not allowed’ supplementary components are defined as
2085 ‘prohibited’ in the representation term schema module]

2086 [5.2 Element Declarations](#)

2087 [5.2.1 General Element Declarations](#)

2088 [5.2.2 Elements Bound to Complex Types](#)

2089 The binding of UBL elements to their `xsd:complexType` is based on the associations
2090 identified in the UBL model. For the `ccts:BasicBusinessInformationEntities`
2091 and `ccts:AggregateInformationEntities`, the UBL elements will be directly
2092 associated to its corresponding `xsd:complexType`.

2093 [ELD3] For every class identified in the UBL model, a global element bound to the
2094 corresponding `xsd:complexType` MUST be declared.

2095 **Example:**

2096 For the Party.Details object class, a complex type/global element declaration
2097 pair is created through the declaration of a Party element that is of type
2098 PartyType.

2099 The element thus created is useful for reuse in the building of new business messages.
2100 The complex type thus created is useful for both reuse and customization, in the building
2101 of both new and contextualized business messages. [TBD: point to a context
2102 methodology document or section from here.]

2103 **Example:**

```
2104 <xsd:element name="BuyerParty" type="BuyerPartyType"/>  
2105 <xsd:complexType name="BuyerPartyType">  
2106 ...  
2107 </xsd:complexType>
```

2108

2109 [ELD4] `ccts:CCT` simple and `xsd:complexType`s MUST only be bound to
2110 elements that represent a BCC or a BBIE.

2111 [Ed Note: This is not correct for the following reasons:

2112 1) BBIEs are bound to the types in the representation term schema module

2113 2) ASBIEs are bound to types that represent their associated ABIE]

2114 [5.2.2.1 Elements Representing AS BIEs](#)

2115 A `ccts:AssociationBusinessInformationEntity` is not a class like
2116 `ccts:AggregateBusinessInformationEntities` and `ccts:BasicBusiness`
2117 `InformationEntities` are. Rather, it is an association between two classes. As such,

2118 the element declaration will reference the `xsd:complexType` of the associated
2119 `ccts:AggregateBusinessInformationEntity`.

2120 [ELN3] A UBL global element name based on a `ccts:AssociationBusiness`
2121 `InformationEntity` MUST be declared and bound to the
2122 `xsd:complexType` of its associated `ccts:AggregateBusiness`
2123 `InformationEntity`.

2124 [Ed. Note – Rule ELN3 must be reclassified as an element declaration rule]

2125 5.2.2.2 Elements Bound to Representation Terms

2126

2127 5.2.2.3 Elements Bound to Core Component Types

2128 [ELD5] For each `ccts:CCT simpleType`, an `xsd:restriction` element
2129 MUST be declared.

2130 5.2.3 Code List Import

2131 [ELD6] The code list `xsd:import` element MUST contain the namespace and
2132 schema location attributes.

2133 [Gunthers Notes: The namespace rules for code lists missing.]

2134 5.2.4 Empty Elements

2135 [ELD7] Empty elements MUST not be declared.

2136 5.2.5 XSD:Any

2137 [ELD8] The `xsd:any` element MUST NOT be used.

2138 5.3 Attribute Declarations

2139 Attributes are W3C Schema constructs associated with elements that provide further
2140 information regarding elements. While elements can be thought of as containing data,
2141 attributes can be thought of as containing metadata. Unlike elements, attributes cannot be
2142 nested within each other—there are no “subattributes.” Therefore, attributes cannot be
2143 extended as elements can. Attribute order is not enforced by XML processors—that is, if
2144 the attribute order in an XML instance document is different than the order in which the
2145 attributes are declared in the schema to which the XML instance document conforms, no
2146 error will result. UBL has determined that these limitations dictate that UBL restrict the
2147 use of attributes to either XSD built-in attributes, or to Supplementary Components
2148 which by their nature within the CCTS metamodel only carry metadata.

2149 **5.3.1 User Defined Attributes**

2150 [ATD1] User defined attributes SHOULD NOT be used. When used, user defined
2151 attributes MUST only convey `CCT:SupplementaryComponent`
2152 information.

2153 **5.3.2 Global Attributes**

2154

2155 [ATD6] If a UBL `xsd:SchemaExpression` contains one or more common
2156 attributes that apply to all UBL elements contained or included or imported
2157 therein, the common attributes MUST be declared as part of a global attribute
2158 group.

2159

2160 [ATD2] If a `ccts:SupplementaryComponent` `xsd:attribute` is common
2161 to all UBL elements, it MUST be declared as part of a global attribute group
2162 in the `ccts:CCT` schema module.

2163 **5.3.3 Supplementary Components**

2164 [ATD3] Within the `ccts:CCT` `xsd:extension` element an `xsd:attribute`
2165 MUST be declared for each `ccts:SupplementaryComponent`
2166 pertaining to that `ccts:CCT`.

2167

2168 [ATD4] For each `ccts:CCT` `simpleType` `xsd:Restriction` element, an
2169 `xsd:base` attribute MUST be declared.

2170

2171 [ATD5] Each `ccts:CCT` `simpleType` `xsd:Restriction` element
2172 `xsd:base` attribute value MUST be set to the appropriate `xsd:datatype`.

2173 **5.3.4 Schema Location**

2174

2175 [ATD7] Each `xsd:schemaLocation` attribute declaration MUST contain a
2176 persistent and resolvable URL.

2177

2178 [ATD8] Each `xsd:schemaLocation` attribute declaration URL MUST contain an
2179 absolute path.

2180 To identify schema modules relative paths are not allowed. Although this may cause a
2181 problem with mirror sites, this is outside the scope of UBL.

2182 5.3.5 XSD:nil

2183 [ATD9] The `xsd` built in nillable attribute MUST NOT be used for any UBL declared
2184 element.

2185 5.3.6 XSD:Any

2186 [ATD10] The `xsd:any` attribute MUST NOT be used.

2187

6 Code Lists

2188 UBL has determined that the best approach for code lists is to handle them as schema
2189 modules. In recognition of the fact that most code lists are maintained by external
2190 agencies, UBL has determined that if code list owners all used the same normative form
2191 schema module, all users of those code lists could avoid a significant level of code list
2192 maintenance. By having each code list owner develop, maintain, and make available via
2193 the internet their code lists using the same normative form schema, code list users would
2194 be spared the unnecessary and duplicative efforts required for incorporation in the form
2195 of enumeration of such code lists into Schema, and would subsequently avoid the
2196 maintenance of such enumerations since code lists are handled as imported schema
2197 modules rather than cumbersome enumerations. To make this mechanism operational,
2198 UBL has defined a number of rules. To avoid enumeration of codes, UBL has
2199 determined that:

2200 [CDL1] All UBL Codes MUST be part of a UBL or externally maintained Code List.

2201 Because the majority of code lists are owned and maintained by external agencies, UBL
2202 will make maximum use of such external code lists where they exist.

2203 [CDL2] The UBL Library SHOULD identify and use external standardized code lists
2204 rather than develop its own UBL-native code lists.

2205 In some cases the UBL Library may extend an existing code list to meet specific business
2206 requirements. In others cases the UBL Library may have to create and maintain a code
2207 list where a suitable code list does not exist in the public domain. Both of these type of
2208 code lists would be considered UBL-internal code lists.

2209 [CDL3] The UBL Library MAY design and use an internal code list where an existing
2210 external code list needs to be extended, or where no suitable external code list
2211 exists.

2212 UBL-internal code lists will be designed with maximum re-use in mind to facilitate
2213 maximum use by others.

2214 [CDL4] If a UBL code list is created, the lists SHOULD be globally scoped (designed
2215 for reuse and sharing, using named types and namespaced Schema Modules)
2216 rather than locally scoped (not designed for others to use and therefore hidden
2217 from their use).

2218 To guarantee consistency within all code list schema modules all ubl-internal code lists
2219 and externally used code lists will use the UBL Code List Schema Module. This schema
2220 module will contain an enumeration of code list values.

2221 [CDL5] All UBL maintained or used Code Lists MUST be enumerated using the UBL
2222 Code List Schema Module.

2223 To guarantee consistency of code list schema module naming, the name of each UBL
2224 Code List Schema Module will adhere to a prescribed form.

2225 [CDL6] The name of each UBL Code List Schema Module MUST be of the form:
2226 {Owning Organization}[Code List Name]{Code List Schema Module}

2227 Each code list used in the UBL schema MUST be imported individually.

2228 [CDL7] An `xsd:Import` element MUST be declared for every code list required in a
2229 UBL schema.

2230 The UBL library allows partial implementations of code lists which may required by
2231 customizers.

2232 [CDL8] Users of the UBL Library MAY identify any subset they wish from an
2233 identified code list for their own trading community conformance
2234 requirements.

2235

2236 The following rule describes the requirements for the namespace of each UBL Code List
2237 Schema Module. The URN consists of some fixed tokens, the name of the code list and
2238 the supplementary components of the code list datatype.

2239 [CDLX] The namespace name of each UBL Code List Schema Module MUST conform
2240 to the following pattern:

2241 urn:oasis:ubl:codeList:<Code List.Identification.Identifier>:<Code
2242 List.Name.Text>:<Code List.Version.Identifier>:<Code
2243 List.Agency.Identifier>:<Code List. AgencyName.Text>

2244 The first three levels are fixed by Uniform Resource Name (URN) as defined in the RFC
2245 specification.

- 2246 ◆ urn: The leading token of URNs
- 2247 ◆ oasis: The registered namespace ID "oasis"
- 2248 ◆ ubl: The registered namespace ID "ubl" (optional)

2249 The values of the following tokens are determined by the code list being used.

- 2250 ◆ codeList: This identifies the OASIS/UBL Code List Schema Module.
- 2251 ◆ Code List. Identification. Identifier: This identifies a list of the respective
2252 corresponding codes.
- 2253 ◆ ListID: This is only unique within the agency that manages this code list.
- 2254 ◆ Code List. Name. Text : The name of a list of codes.

- 2255 ◆ Code List. Version. Identifier: This identifies the version of a code list.
- 2256 ◆ Code List. Agency. Identifier: This identifies the agency that manages a
2257 code
- 2258 ◆ List: The default agencies used are those from DE 3055. However, roles
2259 defined in DE 3055 MUST NOT be used.
- 2260 ◆ Code List. Agency Name.Text - The name of the agency that maintains
2261 the code list.

2262 An example URN is provided below.

2263 urn:oasis:ubl:codeList:3055:AgencyCode:d.02a:6:unece

2264 The following rule describes the requirements for the tokens contained in the URN.

2265 [CDLXX] The tokens comprising the URN MUST adhere to the following guidelines

- 2266 ▪ Whitespace MUST NOT be used within the URN.
- 2267 ▪ Special characters MUST NOT be used within the URN. Special
2268 characters are those characters outside of the range 0-9 or a-z or A-Z.
- 2269 ▪ Ed.Note what is the rule on using lowercase letters
- 2270 ▪ If the code list version identifies a minor version then the major and
2271 minor version of the code list MUST be separated by a period (.).

2272

2273 The following rule describes the requirements for the xsd:schemaLocation for the
2274 importation of the code lists into a UBL business document.

2275 [CDLXXX] The xsd:schemaLocation MUST include the complete URI used to
2276 identify the relevant code list schema.

2277

2278 [NMS19] Each UBL:CodeList schema module MUST be maintained in a separate
2279 namespace.

2280 7 Miscellaneous XSD Rules

2281 UBL, as a business standard vocabulary, requires consistency in its development. The
2282 number of UBL Schema developers will expand over time. To ensure consistency, it is
2283 necessary to address the optional features in XSD that are not addressed elsewhere.

2284 7.1 XSD Simple Types

2285 UBL guiding principles require maximum reuse. XSD provides for forty four built in
2286 data types expressed as simple types. In keeping with the maximize reuse guiding
2287 principle, these built-in `xsd:SimpleTypes` should be used wherever possible.

2288 [GXS3] Built-in XSD Simple Types SHOULD be used wherever possible.

2289 7.2 Namespace Declaration

2290

2291 [GXS4] All W3C XML Schema constructs in UBL Schema and schema modules
2292 MUST contain the following namespace declaration on the `xsd` schema
2293 element:

2294 `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

2295 7.3 XSD:Substitution Groups

2296 The `xsd:SubstitutionGroups` feature enables a type definition to identify substitution
2297 elements in a group. This feature is inconsistent with guiding principle

2298 [GXS5] The `xsd:SubstitutionGroups` feature MUST NOT be used.

2299 7.4 XSD:Final

2300

2301 [GXS6] The `xsd:final` attribute MUST be used to control extensions.

2302 7.5 XSD: Notations

2303

2304 [GXS7] `xsd:notations` MUST NOT be used.

2305 [Ed. Note – do we mean `xsd:notation` datatype?]

2306 **7.6 XSD:All**

2307 The `xsd:all` compositor requires occurrence indicators of `minOccurs = 0` and `maxOccurs`
2308 `= 1`. The `xsd:all` compositor allows for elements to occur in any order. The result is that
2309 in an instance document, elements can occur in any order, are always optional, and never
2310 occur more than once. Such restrictions are inconsistent with data-centric scenarios such as
2311 UBL.

2312 [GXS8] The `xsd:all` element MUST NOT be used.

2313 **7.7 XSD:Choice**

2314 The `xsd:choice` compositor allows for any element declared inside it to occur in the
2315 instance document, but only one. As with the `xsd:all` compositor, this feature is
2316 inconsistent with business transaction exchanges and is not allowed in UBL.

2317 [GXS9] The `xsd:choice` element MUST NOT be used.

2318 **7.8 XSD:Include**

2319 The `xsd:include` feature provides a mechanism for bringing in schemas that reside in the
2320 same namespace. UBL employs multiple schema modules within a namespace. To
2321 avoid circular references, this feature will not be used except by the control schema.

2322

2323 [GXS10] The `xsd:include` feature MUST only be used within a control schema.

2324 **7.9 XSD:Union**

2325 The `xsd:union` feature provides a mechanism whereby a datatype is created as a union
2326 of two or more existing datatypes. With UBL's strict adherence to the use of
2327 `cts:Datatypes` that are explicitly declared in the UBL library, this feature is inappropriate
2328 except for codelists. In some cases external customizers may choose to use this technique
2329 for Codelists and as such the use of the union technique may prove beneficial for
2330 customizers.

2331

2332 [GXS11] The `xsd:union` technique MUST NOT be used except for Code Lists. The
2333 `xsd:union` technique MAY be used for Code Lists.

2334 **7.10 XSD:Appinfo**

2335 The `xsd:appinfo` feature is used by schema to convey processing instructions to a
2336 processing application, Stylesheet, or other tool. Some users of UBL have determined
2337 that this technique poses a security risk and have employed techniques for stripping
2338 `xsd:appinfo` from schema. As UBL is committed to ensuring the widest possible

2339 target audience for its XML library, this feature is not used – except to convey non-
2340 normative information whose removal will not result in non-normative schema.

2341

2342 [GXS12] UBL designed schema SHOULD NOT use `xsd:appinfo`. If used,
2343 `xsd:appinfo` MUST only be used to convey non-normative information.

2344 7.11 Extension and Restriction

2345 UBL fully recognizes the value of supporting extension and restriction of its core library
2346 by customizers.

2347 [GXS13] Complex Type extension or restriction MAY be used where appropriate.

2348 8 Instance Documents

2349 Consistency in UBL instance documents is essential in a trade environment. UBL has
2350 defined several rules to help affect this consistency.

2351 8.1 Root Element

2352 UBL has chosen a global element approach. In XSD, every global element is eligible to
2353 act as a root element in an instance document. Rule ELD1 requires the identification of a
2354 single global element in each UBL schema to be carried as the root element in the
2355 instance document. UBL business documents (UBL instances) must have a single root
2356 element as defined in the corresponding UBL XSD.

2357 [RED1] Every UBL business document MUST have a single root element.

2358 The root element must properly identify the business process being expressed in the UBL
2359 business document.

2360 [RED2] Every root element in a UBL document MUST be named according to the
2361 portion of the business process that it initiates.

2362 Examples:

2363

2364 Order, OrderResponse, ChangeOrder, AdvanceShipNotice,
2365 AdvanceShipNoticeResponse.

2366

2367 8.2 Validation

2368 The UBL library and supporting schema are targeted at supporting business information
2369 exchanges. Business information exchanges require a high degree of precision to ensure
2370 that application processing and corresponding business cycle actions are reflective of the
2371 purpose, intent, and information content agreed to by both trading partners. Schema
2372 provide the necessary mechanism for ensuring that instance documents do in fact support
2373 these requirements.

2374 [IND1] All UBL instance documents MUST validate to a corresponding schema.

2375 8.3 Character Encoding

2376 XML supports a wide variety of character encoding. Processors must understand which
2377 character encoding is employed in each XML document. XML 1.0 supports a default
2378 value of UTF-8 for character encoding, but best practice is to always identify the
2379 character encoding scheme being employed.

2380 [IND2] All UBL instance documents MUST always identify their character encoding
2381 with the XML declaration.

2382
2383
2384

Example:

UTF-8; ISO-8859-1; EUC-JP

2385 UBL, as an OASIS TC, is obligated to conform to agreements OASIS has entered into.
2386 OASIS is a liaison member of the ISO/IETF/ITU/UNCEFACT Memorandum of
2387 Understanding Management Group (MOUMG). Resolution 01/08 (MOU/MG01n83)
2388 requires the use of UTF-8.

2389
2390
2391
2392

[IND3] In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83) as agreed to by OASIS, all UBL XML SHOULD be expressed using UTF-8.

2393
2394
2395
2396

Example:

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

2397 8.4 Schema Instance Namespace Declaration

2398 The W3C XSD specification defines

2399
2400

[IND4] All UBL instance documents MUST contain the following namespace declaration in the root element:

2401

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

2402 8.5 Empty Content.

2403 Usage of empty elements within XML instance documents are a source of controversy
2404 for a variety of reasons. An empty element does not simply represent data that is
2405 missing. It may express data that is not applicable for some reason, trigger the expression
2406 of an attribute, denote all possible values instead of just one, mark the end of a series of
2407 data, or appear as a result of an error in XML file generation. In converse, missing data
2408 elements can also have meaning - data not provided by a trading partner. In information
2409 exchange environments, different Trading Partners may allow, require and ban empty
2410 elements. UBL has determined that empty elements do not provide the level of assurance
2411 necessary for business information exchanges and as such will not be used.

2412
2413

[IND5] UBL conformant instance documents MUST NOT contain an element devoid of content.

2414
2415

To ensure that no attempt is made to circumvent rule IND5, UBL also prohibits attempting to convey meaning by not conveying an element.

2416
2417

[IND6] The absence of a construct or data in a UBL instance document MUST NOT carry meaning.

2418

2419 **Appendix A. UBL NDR Checklist**

2420 The following checklist constitutes all UBL XML naming and design rules as defined in
2421 *UBL Naming and Design Rules version 1.0*, xx November 2003. The checklist is in
2422 alphabetical sequence as follows:

2423 Table A1 — Code List Rules (CDL)

2424 Table A2 — Constraint Rules

2425 ◆ Modeling Constraints (MDC)

2426 ◆ Naming Constraints {NMC}

2427 Table A3 — Declaration Rules

2428 ▪ Element Declarations (ELD)

2429 ▪ Attribute Declarations (ATD)

2430 Table A4 — Documentation Rules (DOC)

2431 Table A5 — General XSD Rules (GXS)

2432 Table A6 — Instance Document Rules (IND)

2433 Table A7 — Naming Rules

2434 General Naming Rules (GNR)

2435 Specific Naming Rules

2436 ▪ Element Naming Rules (ELN)

2437 ▪ Attribute Naming Rules (ATN)

2438 ▪ Type Naming Rules (CTN)

2439 Table A8 — Namespace Rules (NMS)

2440 Table A9 — Root Element Declaration Rules (RED)

2441 Table A10 —

2442

2442 **Table A1 — Code List Rules**

Rule Number	Rule
[CDL1]	All UBL Codes MUST be part of a UBL or External maintained Code List.
[CDL2]	The UBL Library SHOULD identify and use external standardized code lists rather than develop its own UBL-native code lists.
[CDL3]	The UBL Library MAY design and use an internal code list where an existing external code list needs to be extended, or where no suitable external code list exists.
[CDL4]	If a UBL code list is created, the lists SHOULD be globally scoped (designed for reuse and sharing, using named types and namespaced schema modules) rather than locally scoped (not designed for others to use and therefore hidden from their use).
[CDL5]	All UBL maintained or used Code Lists MUST be enumerated using the UBL Code List schema module.
[CDL6]	The name of each UBL Code List schema module MUST be of the form: {Owning Organization}{Code List Name} {Code List schema module}
[CDL7]	An <code>xsd:Import</code> element MUST be declared for every code list required in a UBL schema.
[CDL8]	Users of the UBL Library may identify any subset they wish from an identified code list for their own trading community conformance requirements.

2443

2444

2445

2445 Table A2. Constraint Rules

Rule Number	Rule
Modeling Constraints	
[MDC1]	UBL Models MUST define classes based on ebXML <code>ccts:BasicBusinessInformationEntities</code> and <code>ccts:AggregateBusinessInformationEntities</code> .
[MDC2]	UBL Libraries and Schemas MUST only use ebXML Core Component approved <code>ccts:CoreComponentTypes</code> .
[MDC3]	If a UBL document is extended it MUST retain the business function of the original UBL document.
[MDC4]	Mixed content MUST NOT be used except where contained in an <code>xsd:documentation</code> element.
Naming Constraints	
[NMC1]	Each dictionary entry name MUST define one and only one fully qualified path (FQP) for an element or attribute.

2446

2447

Table A3 — Declarations Rules

Rule Number	Rule
Element Declarations	
[ELD1]	Each <code>UBL:ControlSchema</code> MUST identify one global element declaration that defines the overall business process being conveyed in the Schema expression. That global element MUST include an <code>xsd:annotation</code> child element which MUST further contain an <code>xsd:documentation</code> child element that declares <i>"This element MUST be conveyed as the root element in any instance document based on this Schema expression."</i>
[ELD2]	All element declarations MUST be global with the exception of <code>ID</code> and <code>Code</code> which MUST be local.
[ELD3]	For every class identified in the UBL model, a global element bound to the corresponding <code>xsd:complexType</code> MUST be declared.
[ELD4]	<code>ccts:CCT simple</code> and <code>xsd:complexTypes</code> MUST only be bound to elements that represent a BCC or a BBIE.
[ELD5]	For each <code>ccts:CCT simpleType</code> , an <code>xsd:restriction</code> element MUST be declared.
[ELD6]	The code list <code>xsd:import</code> element MUST contain the namespace and schema location attributes.
[ELD7]	Empty elements MUST not be declared.
[ELD8]	The <code>xsd:any</code> element MUST NOT be used.
Attribute Declarations	
[ATD1]	User defined attributes SHOULD NOT be used. When used, user defined attributes MUST only convey <code>CCT:SupplementaryComponent</code> information.

[ATD2]	If a <code>ccts:SupplementaryComponent</code> <code>xsd:attribute</code> is common to all UBL elements, it MUST be declared as part of a global attribute group in the <code>ccts:CCT</code> schema module.
[ATD3]	Within the <code>ccts:CCT</code> <code>xsd:extension</code> element an <code>xsd:attribute</code> MUST be declared for each <code>ccts:SupplementaryComponent</code> pertaining to that <code>ccts:CCT</code> .
[ATD4]	For each <code>ccts:CCT</code> <code>simpleType</code> <code>xsd:Restriction</code> element, an <code>xsd:base</code> attribute MUST be declared.
[ATD5]	Each <code>ccts:CCT</code> <code>simpleType</code> <code>xsd:Restriction</code> element <code>xsd:base</code> attribute value MUST be set to the appropriate <code>xsd:datatype</code> .
[ATD6]	If a UBL <code>xsd:SchemaExpression</code> contains one or more common attributes that apply to all UBL elements contained or included or imported therein, the common attributes MUST be declared as part of a global attribute group.
[ATD7]	Each <code>xsd:schemaLocation</code> attribute declaration MUST contain a persistent and resolvable URL.
[ATD8]	Each <code>xsd:schemaLocation</code> attribute declaration URL MUST contain an absolute path. To identify schema modules relative paths are not allowed. Although this may cause a problem with mirror sites, this is outside the scope of UBL.
[ATD9]	The <code>xsd</code> built in nillable attribute MUST NOT be used for any UBL declared element.
[ATD10]	The <code>xsd:any</code> attribute MUST NOT be used.
[ATD11]	The <code>xsd:version</code> attribute MUST be used to convey the version of the schema. Its value MUST be identical to the portion of the namespace declaration schema version information. The <code>xsd:version</code> attribute MUST NOT be considered normative if different from the version information contained in the namespace declaration. FIX

2448 Table A4. Documentation Rules

Rule Number	Rule
[DOC1]	<p>Every Data Type definition MUST contain a structured set of annotations in the following patterns:</p> <ul style="list-style-type: none"> ▪ UniqueIdentifier (mandatory): The identifier that references a Data Type instance in a unique and unambiguous way. ▪ CategoryCode (mandatory): The category to which the object belongs. For example, BBIE, ABIE, ASBIE, RT (Representation Term). ▪ DictionaryEntryName (mandatory): The official name of a Data Type. ▪ Definition (mandatory): The semantic meaning of a Data Type. ▪ Version (mandatory): An indication of the evolution over time of a Data Type instance. ▪ QualifierObjectClass (optional): The qualifier for the object class. ▪ ObjectClass: The Object Class represented by the Data Type. ▪ Qualifier Term (mandatory): A semantically meaningful name that differentiates the Data Type from its underlying Core Component Type. ▪ Usage Rule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Data Type.

<p>[DOC2]</p>	<p>A Data Type definition MAY contain one or more Content Component Restrictions to provide additional information on the relationship between the Data Type and its corresponding Core Component Type. If used the Content Component Restrictions must contain a structured set of annotations in the following patterns:</p> <ul style="list-style-type: none"> ▪ RestrictionType (mandatory): Defines the type of format restriction that applies to the Content Component. ▪ RestrictionValue (mandatory): The actual value of the format restriction that applies to the Content Component. ▪ ExpressionType (optional): Defines the type of the regular expression of the restriction value.
<p>[DOC3]</p>	<p>A Data Type definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the Data Type and its corresponding Core Component Type. If used the Supplementary Component Restrictions must contain a structured set of annotations in the following patterns:</p> <ul style="list-style-type: none"> ▪ SupplementaryComponentName (mandatory): Identifies the Supplementary Component on which the restriction applies. ▪ RestrictionValue (mandatory, repetitive): The actual value(s) that is (are) valid for the Supplementary Component

[DOC4]

Every Basic Business Information Entity definition MUST contain a structured set of annotations in the following patterns:

- Unique Identifier (mandatory): The identifier that references a Basic Business Information Entity instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be BBIE.
- Dictionary Entry Name (mandatory): The official name of a Basic Business Information Entity.
- Version (mandatory): An indication of the evolution over time of a Basic Business Information Entity instance.
- Definition (mandatory): The semantic meaning of a Basic Business Information Entity.
- Cardinality (mandatory): Indication whether the Basic Business Information Entity Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Aggregate Business Information Entity.
- QualifierTerm (optional): Qualifies the Property Term of the associated Core Component Property in the associated Aggregate Core Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Basic Business Information Entity.
- ConstraintLanguage (optional, repetitive): A formal description of a way the Basic Business Information Entity is derived from the corresponding stored Core Component and stored Business Context.
- BusinessTerm (optional, repetitive): A synonym term under which the Basic Business Information Entity is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of a Basic Business Information Entity.

Every Aggregate Business Information Entity definition MUST contain a structured set of annotations in the following patterns:

- UniqueIdentifier (mandatory): The identifier that references an Aggregate Business Information Entity instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ABIE.
- Version (mandatory): An indication of the evolution over time of an Aggregate Business Information Entity instance.
- DictionaryEntryName (mandatory): The official name of an Aggregate Business Information Entity.
- Definition (mandatory): The semantic meaning of an Aggregate Business Information Entity.
- QualifierTerm (mandatory): Qualifies the Object Class Term of the associated Aggregate Core Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Aggregate Business Information Entity.
- ConstraintLanguage (optional, repetitive): A formal description of a way the Aggregate Business Information Entity is derived from the corresponding stored Core Component and stored Business Context.
- BusinessTerm (optional, repetitive): A synonym term under which the Aggregate Business Information Entity is commonly known and used in the business.

Every Association Business Information Entity definition MUST contain a structured set of annotations in the following patterns:

- UniqueIdentifier (mandatory): The identifier that references an Association Business Information Entity instance in a unique and unambiguous way.
- CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ASBIE.
- DictionaryEntryName (mandatory): The official name of an Association Business Information Entity.
- Definition (mandatory): The semantic meaning of an Association Business Information Entity.
- Version (mandatory): An indication of the evolution over time of an Association Business Information Entity instance.
- Cardinality (mandatory): Indication whether the Association Business Information Entity Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Aggregate Business Information Entity.
- QualifierTerm (optional): Qualifies the Property Term of the associated Core Component Property in the associated Aggregate Core Component.
- UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Association Business Information Entity.
- ConstraintLanguage (optional, repetitive): A formal description of a way the Association Business Information Entity is derived from the corresponding stored Core Component and stored Business Context.
- BusinessTerm (optional, repetitive): A synonym term under which the Association Business Information Entity is commonly known and used in the business.
- Example (optional, repetitive): Example of a possible value of an Association Business Information Entity.

<p>DOC7</p>	<p>Every Core Component definition MUST contain a structured set of annotations in the following patterns:</p> <ul style="list-style-type: none"> ▪ UniqueIdentifier (mandatory): The identifier that references a Core Component instance in a unique and unambiguous way. ▪ CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be CCT. ▪ DictionaryEntryName (mandatory): The official name of a Core Component. ▪ Definition (mandatory): The semantic meaning of a Core Component. ▪ ObjectClass: The Object Class represented by the type. ▪ PropertyTerm: The Property Term represented by the type. ▪ Version (mandatory): An indication of the evolution over time of a Core Component instance. ▪ Usage Rule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Basic Business Information Entity. ▪ Business Term (optional, repetitive): A synonym term under which the Basic Business Information Entity is commonly known and used in the business.
<p>DOC8</p>	<p>Every element declaration MUST contain an annotation as follows:</p> <p><Documentation>Dictionary Entry Name</Documentation> where Dictionary Entry Name is the complete name (not the tag name) that is the unique official name of the element in the UBL library.</p>
<p>DOC9</p>	<p>For each UBL construct containing a code, the UBL documentation MUST identify the zero or more code lists that MUST be minimally supported when the construct is used.</p>

2449

2450

2450 Table A5. General XSD Rules

Rule Number	Rule
<p>[GXS1]</p>	<p>UBL Schema MUST conform to the following physical layout as applicable:</p> <p>XML Declaration</p> <pre><!-- ===== Copyright Notice ===== --></pre> <p>“Copyright © 2001-2004 The Organization for the Advancement of Structured Information Standards (OASIS). All rights reserved.</p> <pre><!-- ===== xsd:schema Element With Namespaces Declarations ===== --></pre> <p>xsd:schema element to include version attribute and namespace declarations in the following order:</p> <ul style="list-style-type: none"> xmlns:xsd Target namespace Default namespace CommonAggregateComponents CommonBasicComponents CoreComponentTypes Datatypes Identifier Schemes Code Lists <p>Attribute Declarations – elementFormDefault=”qualified” attributeFormDefault=”unqualified”</p> <pre><!-- ===== Imports ===== --></pre> <ul style="list-style-type: none"> CommonAggregateComponents schema module CommonBasicComponents schema module Representation Term schema module (to include CCT module) Common Basic Types schema module Common Aggregate Types schema module <pre><!-- ===== Global Attributes ===== --></pre> <p>Global Attributes and Attribute Groups</p> <pre><!-- ===== Root Element ===== --></pre>

	<p>Root Element Declaration</p> <p>Root Element Type Definition</p> <p><!-- ===== Element Declarations ===== --></p> <p>alphabetized order</p> <p><!-- ===== Type Definitions ===== --></p> <p>All type definitions segregated by basic and aggregates as follows</p> <p><!-- ===== Aggregate Business Information Entity Type Definitions ===== --></p> <p>alphabetized order of ccts:AggregateBusinessInformationEntity xsd:TypeDefinitions</p> <p><!-- =====Basic Business Information Entity Type Definitions ===== --></p> <p>alphabetized order of ccts:BasicBusinessInformationEntities</p> <p><!-- ===== Copyright Notice ===== --></p> <p>Required OASIS full copyright notice.</p>
[GXS2]	UBL MUST provide two normative schemas for each transaction. One schema shall be a run-time schema devoid of documentation. One schema shall be fully annotated.
[GXS3]	Built-in XSD Simple Types SHOULD be used wherever possible.
[GXS4]	All W3C XML Schema constructs in UBL Schema and schema modules MUST contain the following namespace declaration on the xsd schema element: xmlns:xsd="http://www.w3.org/2001/XMLSchema"
[GXS5]	The xsd:substitutionGroups feature MUST NOT be used.
[GXS6]	The xsd:final attribute MUST be used to control extensions.
[GXS7]	xsd:notations MUST NOT be used.
[GXS8]	The xsd:all element MUST NOT be used.
[GXS9]	The xsd:choice element MUST NOT be used.
[GXS10]	The xsd:include feature MUST only be used within a control schema.

[GXS11]	The <code>xsd:union</code> technique MUST NOT be used except for Code Lists. The <code>xsd:union</code> technique MAY be used for Code Lists.
[GXS12]	UBL designed schema SHOULD NOT use <code>xsd:appinfo</code> . If used, <code>xsd:appinfo</code> MUST only be used to convey non-normative information.
[GXS13]	Complex Type extension or restriction MAY be used where appropriate.

2451

2452

2452 **Table A6 —Instance Documents**

Rule Number	Rule
[IND1]	All UBL instance documents MUST validate to a corresponding schema.
[IND2]	All UBL instance documents MUST always identify their character encoding with the XML declaration.
[IND3]	In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83) as agreed to by OASIS, all UBL XML SHOULD be expressed using UTF-8.
[IND4]	All UBL instance documents MUST contain the following namespace declaration in the root element: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance".
[IND5]	UBL conformant instance documents MUST NOT contain an element devoid of content.
[IND6]	The absence of a construct or data in a UBL instance document MUST NOT carry meaning.

2453

Table A7 — Naming Rules

Rule Number	Rule
General Naming rules	
[GNR1]	UBL XML element, attribute and type names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
[GNR2]	UBL XML element, attribute and type names MUST be taken from CCTS conformant dictionary entry names.
[GNR3]	UBL XML element, attribute and type names constructed from <code>ccts:DictionaryEntryNames</code> MUST NOT include periods, spaces, other separators, or characters not allowed by W3C XML 1.0 for XML names.
[GNR4]	UBL XML Element, attribute, and Simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions published in Appendix B.
[GNR5]	Acronyms and abbreviations MUST only be added to the UBL approved acronym and abbreviation list after careful consideration for maximum understanding and reuse.
[GNR6]	Acronyms and abbreviations added to the UBL approved list MUST only be taken from the latest version of the Pocket Oxford English Dictionary. The first occurrence listed for a word MUST be used.
[GNR7]	The acronyms and abbreviations listed in Appendix B MUST always be used.
[GNR8]	UBL XML element, attribute and type names MUST be in singular form unless the concept itself is plural (example: Goods).
[GNR9]	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.

[GNR10]	The lowerCamelCase (LCC) convention MUST be used for naming attributes.
Specific Naming Rules	
Element Naming Rules	
[ELN1]	A UBL global element name based on an <code>ccts:ABIE</code> MUST be the same as the name of the corresponding <code>xsd:complexType</code> to which it is bound, with the word “Type” removed.
[ELN2]	A UBL global element name based on a <code>ccts:BBIE</code> MUST be the same as the name of the corresponding <code>xsd:complexType</code> to which it is bound, with the word “Type” removed.
[ELN3]	A UBL global element name based on an <code>ccts:ASBIE</code> MUST be declared and bound to the <code>xsd:complexType</code> of its associated <code>ccts:ABIE</code> .
[ELN4]	A UBL global element name based on an <code>ccts:ASBIE</code> MUST be the <code>ccts:ASBIE</code> dictionary entry name property term and qualifiers; and the object class term and qualifiers of its associated <code>ccts:ABIE</code> . All <code>ccts:DictionaryEntryName</code> separators MUST be removed. Redundant words in the <code>ccts:ASBIE</code> property term or qualifiers and the associated <code>ccts:ABIE</code> object class term or qualifiers MUST be dropped.
Attribute Naming Rules	
[ATN1]	Each <code>CCT:SupplementaryComponent</code> <code>xsd:attribute</code> “name” MUST be the <code>ccts:SupplementaryComponent</code> dictionary entry name property term and representation term, with the separators removed.
Type Naming Rules	
[CTN1]	A UBL <code>xsd:complexType</code> name based on an <code>ccts:ABIE</code> MUST be the <code>ccts:DictionaryEntryName</code> with the separators removed and with the “Details” suffix replaced with “Type”.

[CTN2]	A UBL <code>xsd:complexType</code> name based on a <code>ccts:BBIE</code> MUST be the <code>ccts:DictionaryEntryName</code> property term and qualifiers and representation term, with the separators removed and with the “Type” suffix appended after the representation term.
[CTN3]	A UBL <code>xsd:complexType</code> name based on a primary representation term used in the UBL model MUST be the name of the corresponding <code>ccts:CCT</code> , with the separators removed and with the “Type” suffix appended after the primary representation term name.
[CTN4]	A UBL <code>xsd:complexType</code> name based on a secondary representation term used in UBL model MUST be the name of the secondary representation term, with the separators removed and with the “Type” suffix appended after the secondary representation term name.
[CTN5]	A UBL <code>xsd:complexType</code> name based on a <code>ccts:CCT</code> MUST be the Dictionary entry name of the <code>ccts:CCT</code> , with the separators removed.

2454

2455

Table A8 — Namespace Rules

Rule Number	Rule
[NMS1]	Every UBL defined or used schema module MUST have a namespace declared using the <code>xsd:targetNamespace</code> attribute.
[NMS2]	Every UBL defined or used schema set version MUST have its own unique namespace.
[NMS3]	UBL namespaces MUST only contain UBL developed schema modules.
[NMS4]	The namespace names for UBL schemas holding committee draft status MUST be of the form: <code>urn:oasis:names:tc:ubl:schema:<name>:<major>:<minor>[<revision>]</code>
[NMS5]	The namespace names for UBL Schemas holding OASIS Standard status MUST be of the form: <code>urn:oasis:names:specification:ubl:schema:<name>:<major>:<minor></code>
[NMS6]	UBL Schema modules MUST be hosted under the UBL committee directory: <code>http://www.oasis-open.org/committees/ubl/schema/<schema-mod-name>.xsd</code>
[NMS7]	UBL published namespaces MUST never be changed.
[NMS8]	The <code>UBL:CommonAggregateComponents</code> schema module MUST reside in its own namespace.
[NMS9]	The UBL <code>CommonAggregateComponents</code> schema module MUST be represented by the token “cac”.
[NMS10]	The <code>UBL:CommonBasicComponents</code> schema module MUST reside in its own namespace.
[NMS11]	The <code>UBL:CommonBasicComponents</code> schema module MUST be represented by the token “cbc”.

[NMS12]	The <code>ccts:CoreComonentType</code> schema module MUST reside in its own namespace.
[NMS13]	The <code>ccts:CoreComonentType</code> schema module namespace MUST be represented by the token “cct”.
[NMS14]	The <code>ccts:RepresentationTerm</code> schema module MUST reside in its own namespace.
[NMS15]	The <code>ccts:CodeTypeRepresentationTerm</code> schema module MUST reside in the <code>ccts:RepresentationTerm</code> schema module namespace
[NMS16]	The <code>ccts:RepresentationTerm</code> schema module namespace MUST be represented by the token “rt”.
[NMS17]	The <code>UBL:Datatypes</code> schema module MUST reside in its own namespace.
[NMS18]	The <code>UBL:Datatypes</code> schema module namespace MUST be represented by the token “dt”.
[NMS19]	Each <code>UBL:CodeList</code> schema module MUST be maintained in a separate namespace.

2456

2457

2458

2458 Table A9 — Root Element Declaration Rules

Rule Number	Rule
[RED1]	Every UBL business document MUST have a single root element.
[RED2]	Every root element in a UBL document MUST be named according to the portion of the business process that it initiates.

2459

2460

Table A10 — Schema Structure Modularity Rules

Rule Number	Rule
[SSM1]	UBL Schema expressions MAY be split into multiple schema modules.
[SSM2]	A control schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another namespace MUST only import the control schema from that namespace.
[SSM3]	A UBL control schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another namespace MUST NOT import internal schema modules from that namespace.
[SSM4]	Imported schema modules MUST be fully conformant with UBL naming and design rules.
[SSM5]	UBL schema modules MUST either be treated as external schema modules or as internal schema modules of the control schema.
[SSM6]	All UBL internal schema modules MUST be in the same namespace as their corresponding control schema.
[SSM7]	Each UBL internal schema module MUST be named <code>{ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}</code>
[SSM8]	A UBL schema module MAY be created for reusable components.
[SSM9]	A schema module defining all <code>ubl:CommonAggregateComponents</code> MUST be created.
[SSM10]	The <code>ubl:CommonAggregateComponents</code> schema module MUST be named <code>"ubl:CommonAggregateComponents Schema Module"</code>
[SSM11]	A schema module defining all <code>ubl:CommonBasicComponents</code> MUST be created.
[SSM12]	The <code>ubl:CommonBasicComponents</code> schema module MUST be named <code>"ubl:CommonBasicComponmnents Schema Module"</code>

[SSM13]	A schema module defining all <code>ccts:CoreComponentTypes</code> MUST be created.
[SSM14]	The <code>ccts:CoreComponentType</code> schema module MUST be named " <code>ccts:CoreComponentType Schema Module</code> "
[SSM15]	The <code>xsd:facet</code> feature MUST not be used in the <code>ccts:CoreComponentType</code> schema module.
[SSM16]	A schema module defining all <code>ccts:PrimaryRepresentationTerms</code> and <code>ccts:SecondaryRepresentationTerms</code> with the exception of <code>ccts:CodeType</code> MUST be created.
[SSM17]	A schema module defining the <code>ccts:CodeType</code> <code>ccts:RepresentationTerm</code> MUST be created
[SSM18]	The <code>ccts:RepresentationTerm</code> schema module MUST be named " <code>ccts:RepresentationTerm Schema Module</code> "
[SSM19]	The <code>ccts:CodeTypeRepresentationTerm</code> schema module MUST be named " <code>ccts:CodeTypeRepresentationTerm Schema Module</code> "
[SSM20]	A schema module defining all UBL Datatypes MUST be created.
[SSM21]	The <code>UBL:Datatypes</code> schema module MUST be named " <code>ubl:Datatypes schema module</code> "

2461

2462

2462 Table A11 — Standards Adherence Rules

Rule Number	Rule
[STA1]	All UBL schema design rules MUST be based on the W3C XML Schema Recommendations: <i>XML Schema Part 1: Structures</i> and <i>XML Schema Part 2: Datatypes</i> .
[STA2]	All UBL schema and messages MUST be based on the W3C suite of technical specifications holding recommendation status.

2463

2464

2464 Table A12 — Type Definition Rules

Rule Number	Rule
General Type Definitions	
[GTD1]	All types MUST be named.
[GTD2]	The <code>xsd:any</code> Type MUST NOT be used.
Simple Type Definitions	
[STD1]	For every <code>ccts:CCT</code> whose supplementary components are equivalent to the properties of a built-in <code>xsd:datatype</code> , the <code>CCT:SupplementaryComponents</code> MUST NOT be expressed as attributes, and the <code>ccts:CCT</code> MUST be defined as a named <code>simpleType</code> in the <code>ccts:CCT</code> schema module.
[STD2]	<code>xsd:simpleType</code> restriction MUST NOT be used for <code>ccts:CCTs</code> .
[CTD1]	For every class identified in the UBL model, a named <code>xsd:complexType</code> MUST be defined.
[CTD2]	For every primary representation term used in the UBL model, a named <code>xsd:complexType</code> MUST be defined.
[CTD3]	For every secondary representation term used in the UBL model, a named <code>xsd:complexType</code> MUST be defined.
[CTD4]	Every <code>ccts:ABIE</code> <code>xsd:complexType</code> definition content model MUST use the <code>xsd:sequence</code> element with appropriate global element references, or local element declarations in the case of ID and Code, to reflect each property of its class as defined in the corresponding UBL model.
[CTD5]	Every <code>ccts:BBIE</code> <code>xsd:complexType</code> definition content model MUST use the <code>xsd:simpleContent</code> element.

[CTD6]	Every <code>ccts:BBIE</code> <code>ComplexType</code> content model <code>xsd:simpleContent</code> element MUST consist of an <code>xsd:extension</code> element.
[CTD7]	Every <code>ccts:BBIE</code> <code>xsd:complexType</code> content model <code>xsd:extension</code> element MUST use the <code>xsd:base</code> attribute to define the basis of each primary or secondary representation term.
[CTD8]	Every <code>ccts:BBIE</code> <code>xsd:complexType</code> content model <code>xsd:base</code> attribute value MUST be the <code>ccts:CCT</code> of the primary representation term or the datatype of the secondary representation term as appropriate.
[CTD9]	For every <code>ccts:CCT</code> whose supplementary components are not equivalent to the properties of a built-in <code>xsd:datatype</code> , the <code>ccts:CCT</code> MUST be defined as a named <code>xsd:complexType</code> in the <code>ccts:CCT</code> schema module.
[CTD10]	Each <code>ccts:CCT</code> <code>xsd:complexType</code> definition MUST contain one <code>xsd:simpleContent</code> element
[CTD11]	The <code>ccts:CCT</code> <code>xsd:complexType</code> definition <code>xsd:simpleContent</code> element MUST contain one <code>xsd:extension</code> element. This <code>xsd:extension</code> element MUST include an <code>xsd:base</code> attribute that defines the specific <code>xsd:built-inDatatype</code> required for the <code>ccts:ContentComponent</code> of the <code>ccts:CCT</code> .
[CTD12]	Each <code>CCT:SupplementaryComponent</code> <code>xsd:attribute</code> “type” MUST define the specific <code>xsd:built-in Datatype</code> or the user defined <code>xsd:simpleType</code> for the <code>ccts:SupplementaryComponent</code> of the <code>ccts:CCT</code> .
[CTD13]	Each <code>ccts:SupplementaryComponent</code> <code>xsd:attribute</code> user-defined <code>xsd:simpleType</code> MUST only be used when the <code>ccts:SupplementaryComponent</code> is based on a standardized code list for which a UBL conformant code list schema module has been created.
[CTD14]	Each <code>ccts:SupplementaryComponent</code> <code>xsd:attribute</code> user defined <code>xsd:simpleType</code> MUST be the same <code>xsd:simpleType</code> from the appropriate UBL conformant code list schema module for that type.

[CTD15]	Each <code>ccts:SupplementaryComponent</code> <code>xsd:attribute</code> “use” MUST define the occurrence of that <code>ccts:SupplementaryComponent</code> as either “required”, or “optional”.
[CTD16]	Each <code>ccts:CCT</code> <code>simpleType</code> definition name MUST be the <code>ccts:CCT</code> dictionary entry name with the separators removed.

2465

2465 Table A13 — Versioning Rules

Rule Number	Rule
[VER1]	<p>Every UBL Schema and schema module major version committee draft MUST have the URI of:</p> <p>urn:oasis:names:tc:ubl:<name>:<major-number>:0</p>
[VER2]	<p>Every UBL schema and schema module major version OASIS Standard MUST have the URI of:</p> <p>urn:oasis:names:specification:ubl:schema:<name>:<major>:0</p>
[VER3]	<p>The first minor version release of a UBL schema or schema module committee draft MUST have the URI of:</p> <p>urn:oasis:names:tc:ubl:name:<major-number>:<non-zero></p>
[VER4]	<p>The first minor version release of a UBL schema or schema module OASIS Standard MUST have the URI of:</p> <p>urn:oasis:names:tc:ubl:name:<major-number>:<non-zero></p>
[VER5]	<p>For UBL minor version changes, the name of the version construct MUST NOT change (short name not qualified name), unless the intent of the change is to rename the construct.</p>
[VER6]	<p>Every UBL schema and schema module major version number MUST be a sequentially assigned, incremental number greater than zero.</p>
[VER7]	<p>Every UBL schema and schema module minor version number MUST be a sequentially assigned, incremental non-negative number.</p>
[VER8]	<p>Each UBL minor version MUST be given a separate namespace.</p>
[VER9]	<p>A UBL minor version control schema MUST import its immediately preceding minor version control schema.</p>

[VER10]	UBL Schema and schema module minor version changes MUST be limited to the use of <code>xsd:extension</code> or <code>xsd:restriction</code> to alter existing types or add new constructs.
[VER11]	UBL Schema and schema module minor version changes MUST not break semantic compatibility with prior versions.

2466

2467 **Appendix B. Approved Acronyms and Abbreviations**

2468

2469 The following Acronyms and Abbreviations have been approved for UBL use:

2470 ◆ A Dun & Bradstreet number *must* appear as "DUNS". [TBD: need example.]

2471 ◆ "Identifier" *must* appear as "ID".

2472 ◆ "Uniform Resource Identifier" *must* appear as "URI"

2473 ◆ [Example] the "Uniform Resource. Identifier" portion of the **Binary Object.**
2474 **Uniform Resource. Identifier** supplementary component becomes "URI" in
2475 the resulting XML name). The use of URI for Uniform Resource Identifier
2476 takes precedence over the use of "ID" for "Identifier".

Appendix C. Technical Terminology

Ad hoc schema processing	Doing partial schema processing, but not with official schema validator software; e.g., reading through schema to get the default values out of it.
Application-level validation	Adherence to business requirements, such as valid account numbers.
Assembly	Using parts of the library of reusable UBL components to create a new kind of business document type.
Business Context	<p>Defines a context in which a business has chosen to employ an information entity.</p> <p>The formal description of a specific business circumstance as identified by the values of a set of <i>Context Categories</i>, allowing different business circumstances to be uniquely distinguished.</p>
Business Object	<p>An unambiguously identified, specified, referenceable, registerable and re-useable scenario or scenario component of a business transaction.</p> <p>The term business object is used in two distinct but related ways, with slightly different meanings for each usage:</p> <p>In a business model, business objects describe a business itself, and its business context. The business objects capture business concepts and express an abstract view of the business's "real world". The term "modeling business object" is used to designate this usage.</p> <p>In a design for a software system or in program code, business objects reflects how business concepts are represented in software. The abstraction here reflects the transformation of business ideas into a software realization. The term "systems business objects" is used to designate this usage.</p>

business semantic(s)	A precise meaning of words from a business perspective.
Business Term	This is a synonym under which the Core Component or Business Information Entity is commonly known and used in the business. A Core Component or Business Information Entity may have several business terms or synonyms.
class	A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See interface.
class diagram	Shows static structure of concepts, types, and classes. Concepts show how users think about the world; types show interfaces of software components; classes show implementation of software components. (OMG Distilled) A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. (Rational Unified Process)
classification scheme	This is an officially supported scheme to describe a given <i>Context Category</i>
Common attribute	An attribute that has identical meaning on the multiple elements on which it appears. A common attribute might or might not correspond to an XSD global attribute.
component	A physical, replaceable part of a system that packages implementation and conforms to and provides the realization of a set of interfaces. A component represents a physical piece of implementation of a system, including software code (source, binary or executable) or equivalents such as scripts or command files.
context	Defines the circumstances in which a Business Process may be used. This is specified by a set of Context Categories known as Business Context. (See Business

	Context.)
context category	A group of one or more related values used to express a characteristic of a business circumstance.
context driver	Driver information that may be discovered from the Trading Partner Profiles or the Registry Information Model data at the Trading Partner Agreement design time. Eight context categories defined: Business Process, Product Classification, Industry Classification, Geopolitical, Official Constraints, Business Process Role, Supporting Role, System Capabilities.
Control schema	A schema document corresponding to a single namespace, which is likely to pull in (by including or importing) schema modules.
Core Component	A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.
Core Component Catalog	The temporary collection of all metadata about each Core Component that has been discovered during the development and initial testing of this Core Component Technical Specification, pending the establishment of a permanent Registry/Repository.
Core Component Library	The Core Component Library is the part of the registry/repository in which Core Components shall be stored as Registry Classes. The Core Component Library will contain all the Core Component Types, Basic Core Components, Aggregate Core Components, Basic Business Information Entities and Aggregate Business Information Entities.
Core Component Type	A Core Component which consists of one and only one Content Component that carries the actual content plus one or more Supplementary Components giving an essential extra definition to the Content Component. <i>Core Component Types</i> do not have business

	semantics.
Datatype	<p>A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations.</p> <p>Defines the set of valid values that can be used for a particular <i>Basic Core Component Property</i> or <i>Basic Business Information Entity Property</i>. It is defined by specifying restrictions on the <i>Core Component Type</i> that forms the basis of the <i>Data Type</i>.</p>
DTD validation	Adherence to an XML 1.0 DTD.
Generic BIE	A semantic model that has a “zeroed” context. We are assuming that it covers the requirements of 80% of business uses, and therefore is useful in that state.
instance	An individual entity satisfying the description of a class or type.
Instance constraint checking	Additional validation checking of an instance, beyond what XSD makes available, that relies only on constraints describable in terms of the instance and not additional business knowledge; e.g., checking co-occurrence constraints across elements and attributes. Such constraints might be able to be described in terms of Schematron.
Instance root/doctype	This is still mushy. The transitive closure of all the declarations imported from whatever namespaces are necessary. A doctype may have several namespaces used within it.
Intermediate element	An element not at the top level that is of a complex type, only containing other elements and attributes.
Internal schema module:	A schema module that does not declare a target namespace.
Leaf element	An element containing only character data (though it

	may also have attributes). Note that, because of the XSD mechanisms involved, a leaf element that has attributes must be declared as having a complex type, but a leaf element with no attributes may be declared with either a simple type or a complex type.
Lower-level element	An element that appears inside a business message.
Object Class	The logical data grouping (in a logical data model) to which a data element belongs (ISO11179). The <i>Object Class</i> is the part of a <i>Core Component's Dictionary Entry Name</i> that represents an activity or object in a specific <i>Context</i> .
Namespace schema module:	A schema module that declares a target namespace and is likely to pull in (by including or importing) schema modules.
Naming Convention	The set of rules that together comprise how the dictionary entry name for <i>Core Components</i> and <i>Business Information Entities</i> are constructed.
Schema	Never use this term unqualified!
schema module	A “schema document” (as defined by the XSD spec) that is intended to be taken in combination with other such schema documents to be used.
Schema module:	A schema document containing type definitions and element declarations.
Schema Processing	Schema validation checking plus provision of default values and provision of new info set properties.
Schema Validation	Adherence to an XSD schema.
semantic	Relating to meaning in language; relating to the connotations of words.
Top-level element	An element that encloses a whole UBL business

	message. Note that UBL business messages might be carried by messaging transport protocols that themselves have higher-level XML structure. Thus, a UBL top-level element is not necessarily the root element of the XML document that carries it.
type	<p>Description of a set of entities that share common characteristics, relations, attributes, and semantics.</p> <p>A stereotype of class that is used to specify an area of instances (objects) together with the operations applicable to the objects. A type may not contain any methods. See class, instance. Contrast interface.</p>
Syntax Neutral Model	TBD Need definition.
Aggregate Business Information Entity (ABIE)	A collection of related pieces of business information that together convey a distinct business meaning in a specific Business Context. Expressed in modelling terms, it is the representation of an Object Class, in a specific Business Context.
Well-Formedness Checking	Basic XML 1.0 adherence.

2479

2480 **Appendix D. References**

- 2481 [CCTS] *Core Components Technical Specification – Part 8 of the ebXML*
2482 *Technical Framework, Version 2.0 (Second Edition) 15 November*
2483 *2003*
- 2484 [CCFeedback] *Feedback from OASIS UBL TC to Draft Core Components*
2485 *Specification 1.8, version 5.2, May 4, 2002, [http://www.ietf.org/rfc/rfc2119.txt](http://oasis-</i>
2486 <i>open.org/committees/ubl/lcsc/doc/ubl-ctscomments-5p2.pdf.</i></p><p>2487 [GOF] <i>Design Patterns</i>, Gamma, et al. ISBN 0201633612</p><p>2488 [ISONaming] <i>ISO/IEC 11179</i>, Final committee draft, Parts 1-6.</p><p>2489 (RFC) 2119 S. Bradner, <i>Key words for use in RFCs to Indicate Requirement</i>
2490 <i>Levels</i>, <a href=), IETF RFC 2119, March
2491 1997.*
- 2492 [UBLChart] UBL TC Charter, <http://oasis->
2493 [open.org/committees/ubl/charter/ubl.htm](http://oasis-open.org/committees/ubl/charter/ubl.htm)
- 2494 [XML] *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C
2495 Recommendation, October 6, 2000
- 2496 (XSD) *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May
2497 2001.
- 2498
- 2499 (XHTML) *XHTML™ Basic*, W3C Recommendation 19 December 2000:
2500 <http://www.w3.org/TR/2000/REC-xhtml-basic-20001219>
2501

2502

Appendix E. Notices

2503 OASIS takes no position regarding the validity or scope of any intellectual property or
2504 other rights that might be claimed to pertain to the implementation or use of the
2505 technology described in this document or the extent to which any license under such
2506 rights might or might not be available; neither does it represent that it has made any effort
2507 to identify any such rights. Information on OASIS's procedures with respect to rights in
2508 OASIS specifications can be found at the OASIS website. Copies of claims of rights
2509 made available for publication and any assurances of licenses to be made available, or the
2510 result of an attempt made to obtain a general license or permission for the use of such
2511 proprietary rights by implementors or users of this specification, can be obtained from the
2512 OASIS Executive Director.

2513 OASIS invites any interested party to bring to its attention any copyrights, patents or
2514 patent applications, or other proprietary rights which may cover technology that may be
2515 required to implement this specification. Please address the information to the OASIS
2516 Executive Director.

2517 Copyright © The Organization for the Advancement of Structured Information Standards
2518 [OASIS] 2001. All Rights Reserved.

2519 This document and translations of it may be copied and furnished to others, and
2520 derivative works that comment on or otherwise explain it or assist in its implementation
2521 may be prepared, copied, published and distributed, in whole or in part, without
2522 restriction of any kind, provided that the above copyright notice and this paragraph are
2523 included on all such copies and derivative works. However, this document itself does not
2524 be modified in any way, such as by removing the copyright notice or references to
2525 OASIS, except as needed for the purpose of developing OASIS specifications, in which
2526 case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2527 document must be followed, or as required to translate it into languages other than
2528 English.

2529 The limited permissions granted above are perpetual and will not be revoked by OASIS
2530 or its successors or assigns.

2531 This document and the information contained herein is provided on an "AS IS" basis and
2532 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
2533 BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
2534 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2535 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
2536 PURPOSE.

2537