



Universal Business Language (UBL) Naming and Design Rules

15 September 2004

Document identifier:

cd-UBL-NDR-1.0

Location:

<http://docs.oasis-open.org/ubl/cd-UBL-NDR-1.0>

Naming and Design Rules Subcommittee Co-chairs

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

Mark Crawford, LMI <mcrawford@lmi.org>

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

Lead Editor:

Mark Crawford, LMI <mcrawford@lmi.org>

Contributors:

Bill Burcham, Sterling Commerce

Fabrice Desré, France Telecom

Matt Gertner, Schemantix

Jessica Glace, LMI

Arofan Gregory, Aeon LLC

Michael Grimley, US Navy

Eduardo Gutentag, Sun Microsystems

Sue Probert, CommerceOne

Gunther Stuhec, SAP

Paul Thorpe, OSS Nokalva

Jim Wilson, CIDX

Past Chair

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

Abstract:

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

Status:

This document has been approved as an OASIS UBL TC Committee Draft

Copyright © 2001, 2002, 2003, 2004 The Organization for the Advancement of Structured Information Standards [OASIS]

37	Table of Contents	
38	1 Introduction	5
39	1.1 Audiences	6
40	1.2 Scope	6
41	1.3 Terminology and Notation	6
42	1.4 Guiding Principles	7
43	1.4.1 Adherence to General UBL Guiding Principles	7
44	1.4.2 Design For Extensibility	9
45	1.4.3 Code Generation	10
46	1.5 Choice of schema language	10
47	2 Relationship to ebXML Core Components	12
48	2.1 Mapping Business Information Entities to XSD	14
49	3 General XML Constructs	18
50	3.1 Overall Schema Structure	18
51	3.1.1 Root Element	19
52	3.2 Constraints	20
53	3.2.1 Naming Constraints	20
54	3.2.2 Modeling Constraints	21
55	3.3 Reusability Scheme	22
56	3.4 Namespace Scheme	24
57	3.4.1 Declaring Namespaces	24
58	3.4.2 Namespace Uniform Resource Identifiers	25
59	3.4.3 Schema Location	26
60	3.4.4 Persistence	26
61	3.5 Versioning Scheme	26
62	3.6 Modularity	29
63	3.6.1 UBL Modularity Model	30
64	3.6.2 Internal and External schema modules	34
65	3.6.3 Internal schema modules	34
66	3.6.4 External schema modules	35
67	3.7 Annotation and Documentation	39
68	3.7.1 Schema Annotation	40
69	3.7.2 Embedded documentation	40
70	4 Naming Rules	45
71	4.1 General Naming Rules	45
72	4.2 Type Naming Rules	47
73	4.2.1 Complex Type Names for CCTS Aggregate Business	
74	Information Entities	48

75	4.2.2	Complex Type Names for CCTS Basic Business Information Entity	
76		Properties	48
77	4.2.3	Complex Type Names for CCTS Unspecialised Datatypes	49
78	4.2.4	Complex Type Names for CCTS Core Component Types	49
79	4.2.5	Simple Type Names for CCTS Core Component Types	50
80	4.3	Element Naming Rules	50
81	4.3.1	Element Names for CCTS Aggregate Business Information Entities	50
82	4.3.2	Element Names for CCTS Basic Business Information Entity Properties	51
83	4.3.3	Element Names for CCTS Association Business Information Entities	52
84	4.4	Attribute Naming Rules	52
85	5	Declarations and Definitions	53
86	5.1	Type Definitions	53
87	5.1.1	General Type Definitions	53
88	5.1.2	Simple Types	53
89	5.1.3	Complex Types	54
90	5.2	Element Declarations	58
91	5.2.1	General Element Declarations	58
92	5.2.2	Elements Bound to Complex Types	58
93	5.2.3	Code List Import	59
94	5.2.4	Empty Elements	59
95	5.2.5	Global Elements	59
96	5.2.6	XSD:Any	60
97	5.3	Attribute Declarations	60
98	5.3.1	User Defined Attributes	60
99	5.3.2	Global Attributes	60
100	5.3.3	Supplementary Components	61
101	5.3.4	DatatypeSchema Location	61
102	5.3.5	XSD:Nil	61
103	5.3.6	XSD:Any	61
104	6	Code Lists	62
105	7	Miscellaneous XSD Rules	64
106	7.1	XSD Simple Types	64
107	7.2	Namespace Declaration	64
108	7.3	XSD:Substitution Groups	64
109	7.4	XSD:Final	64
110	7.5	XSD: Notation	64
111	7.6	XSD:All	65
112	7.7	XSD:Choice	65

113	7.8 XSD:Include	65
114	7.9 XSD:Union	65
115	7.10 XSD:Appinfo	66
116	7.11 Extension and Restriction	66
117	8 Instance Documents	67
118	8.1 Root Element	67
119	8.2 Validation	67
120	8.3 Character Encoding	67
121	8.4 Schema Instance Namespace Declaration	68
122	8.5 Empty Content.	68
123	Appendix A. UBL NDR Checklist	69
124	A.1 Attribute Declaration Rules	70
125	A.2 Attribute Naming Rules	71
126	A.3 Code List Rules	71
127	A.4 ComplexType Definition Rules	72
128	A.5 ComplexType Naming Rules	74
129	A.6 Documentation Rules	75
130	A.7 Element Declaration Rules	80
131	A.8 Element Naming Rules	81
132	A.9 General Naming Rules	82
133	A.10 General Type Definition Rules	83
134	A.11 General XML Schema Rules	83
135	A.12 Instance Document Rules	86
136	A.13 Modeling Constraints Rules	86
137	A.14 Naming Constraints Rules	86
138	A.15 Namespace Rules	87
139	A.0 Root Element Declaration Rules	88
140	A.0 Schema Structure Modularity Rules	88
141	A.0 Standards Adherence rules	90
142	A.0 SimpleType Naming Rules	90
143	A.0 SimpleType Definition Rules	90
144	A.0 Versioning Rules	91
145	Appendix B. Approved Acronyms and Abbreviations	92
146	Appendix C. Technical Terminology	93
147	Appendix D. References	99
148	Appendix E. Notices	100

149 **1 Introduction**

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

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

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

177 The UBL Technical Committee has established the UBL Naming and Design Rules
178 Subcommittee with the charter to "Recommend to the TC rules and guidelines for
179 normative-form schema design, instance design, and markup naming, and write and
180 maintain documentation of these rules and guidelines". Accordingly, this specification
181 documents the rules and guidelines for the naming and design of XML components for
182 the UBL library. It contains only rules that have been agreed on by the OASIS UBL
183 Naming and Design Rules Subcommittee (NDR SC). Proposed rules, and rationales for
184 those that have been agreed on, appear in the accompanying NDR SC position papers,
185 which are available at <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 Technical Committee will use the rules in this
190 document to create normative form schema for business transactions. Developers
191 implementing ebXML Core Components may find the rules contained herein sufficiently
192 useful to merit adoption as, or infusion into, their own approaches to ebXML Core
193 Component based XML schema development. All other XML Schema developers may
194 find the rules contained herein sufficiently useful to merit consideration for adoption as,
195 or infusion into, their own approaches to XML schema development.

196 1.2 Scope

197 This specification conveys a normative set of XML schema design rules and naming
198 conventions for the creation of business based XML schema for business documents
199 being exchanged between two parties using objects defined in accordance with the
200 ebXML Core Components Technical Specification.

201 1.3 Terminology and Notation

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

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

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

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

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

219 *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

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

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

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

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

230 1.4 Guiding Principles

231 The UBL guiding principles encompass three areas:

- 232 ◆ General UBL guiding principles
- 233 ◆ Extensibility
- 234 ◆ Code generation

235 1.4.1 Adherence to General UBL Guiding Principles

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

- 239 ◆ Internet Use – UBL shall be straightforwardly usable over the Internet.

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

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

296 1.4.2 Design For Extensibility

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

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

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

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

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

320 1.4.3 Code Generation

321 The UBL NDR makes no assumptions on the availability or capabilities of tools to
322 generate UBL conformant XSD Schemas. In conformance with UBL guiding principle 3,
323 the UBL NDR design process has scrupulously avoided establishing any naming or
324 design rules that sub-optimizes the XSD in favor of tool generation. Additionally, in
325 conformance with UBL guiding principle 8, the NDR are sufficiently rigorous to avoid
326 requiring human judgment at schema generation time.

327 1.5 Choice of schema language

328 The W3C XML Schema Definition Language has become the generally accepted schema
329 language that is experiencing the most widespread adoption. Although other schema
330 languages exist that offer their own advantages and disadvantages, UBL has determined
331 that the best approach for developing an international XML business standard is to base
332 its work on W3C XSD.

333

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

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

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

344
345

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

346 2 Relationship to ebXML Core Components

347 UBL employs the methodology and model described in *Core Components Technical*
348 *Specification, Part 8 of the ebXML Technical Framework, Version 2.0 (Second Edition)*
349 of 15 November 2003 (CCTS) to build the UBL Component Library. The Core
350 Components work is a continuation of work that originated in, and remains a part of, the
351 ebXML initiative. The Core Components concept defines a new paradigm in the design
352 and implementation of reusable syntactically neutral information building blocks. Core
353 Components are intended to form the basis of business information standardization
354 efforts and to be realized in syntactically specific instantiations such as ANSI ASC X12,
355 UN/EDIFACT and XML.

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

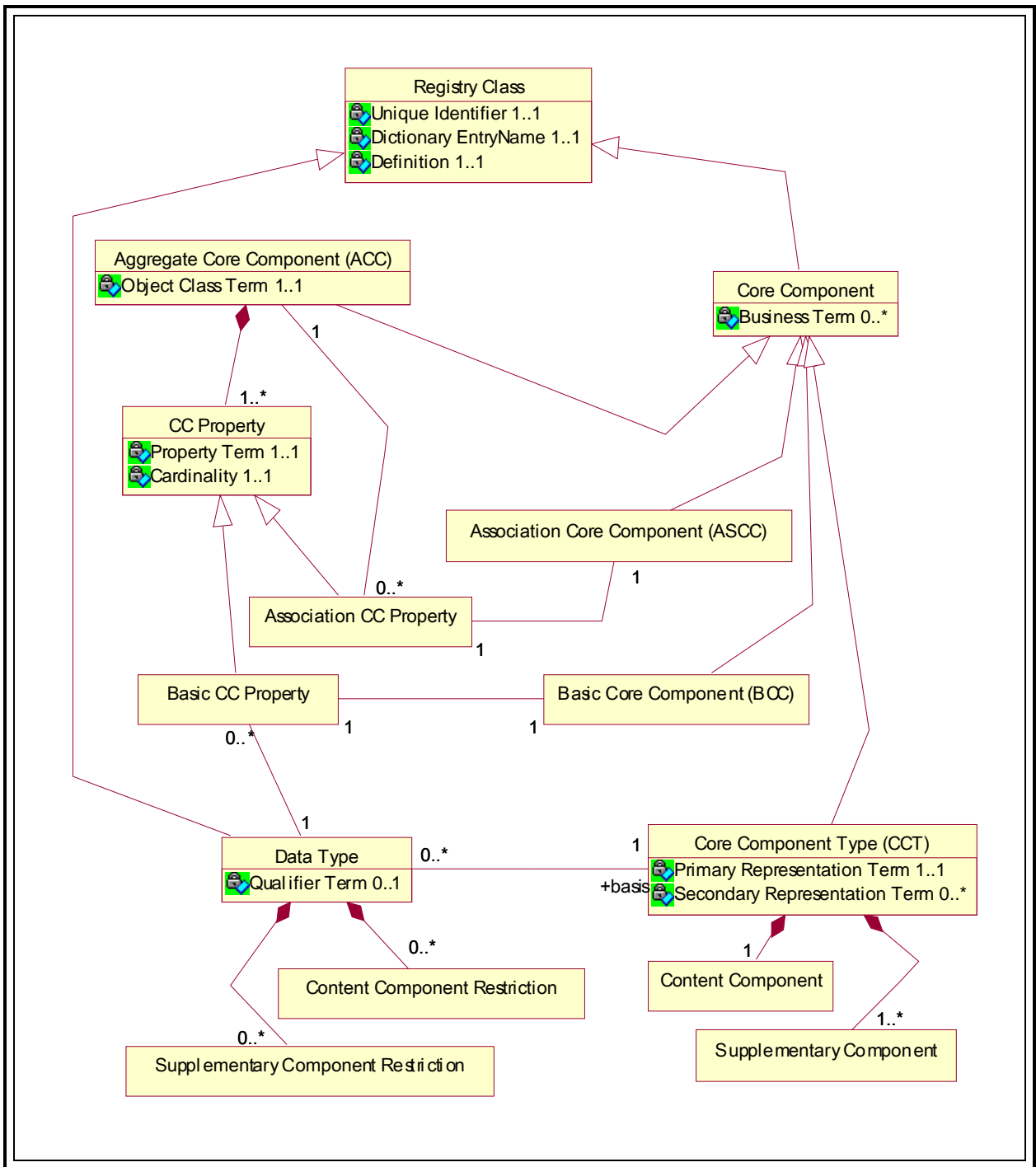
363 The context specific components are defined as Business Information Entities
364 (`ccts:BusinessInformationEntities`).³ Context specific `ccts:Business`
365 `InformationEntities` are defined in CCTS as “A piece of business data or a group of
366 pieces of business data with a unique *Business Semantic*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

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

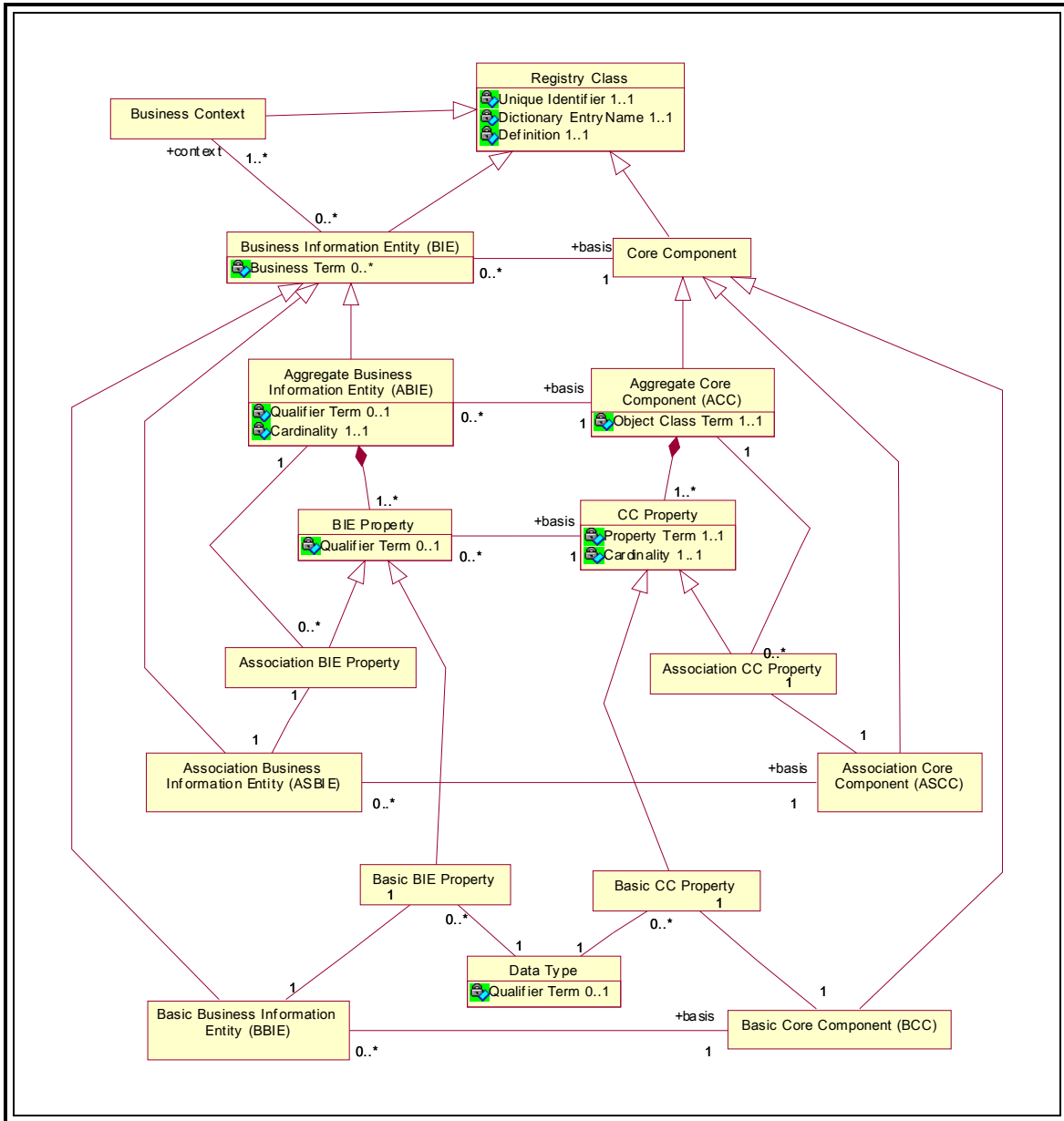
³ See CCTS Section 6.2 for a detailed discussion of the ebXML context mechanism.

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



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

377 **Figure 2-2. Business Information Entities Basic Definition Model**



378

379 2.1 Mapping Business Information Entities to XSD

380 UBL has defined how each of the `ccts:BusinessInformationEntity` components
 381 map to an XSD construct (See figure 2-3). In defining this mapping, UBL has analyzed
 382 the CCTS metamodel and determined the optimal usage of XSD to express the various
 383 `ccts:BusinessInformationEntity` components. As stated above, a
 384 `ccts:BusinessInformationEntity` can be a `ccts:AggregateBusiness`
 385 `InformationEntity`, a `ccts:BasicBusinessInformationEntity`, or a
 386 `ccts:AssociationBusinessInformationEntity`. In understanding the logic of
 387 the UBL binding of `ccts:BusinessInformationEntities` to XSD expressions, it is

388 important to understand the basic constructs of the `ccts:AggregateBusiness`
389 `InformationEntities` and their relationships as shown in Figure 2-2.

390 Both Aggregate and Basic Business Information Entities must have a unique name
391 (Dictionary Entry Name). Both are treated as objects and both are defined as
392 `xsd:ComplexTypes`.

393 There are two kinds of Business Information Entity Properties - Basic and Association. A
394 Basic Business Information Entity Property represents an *intrinsic* property of an
395 Aggregate Business Information Entity. Basic Business Information Entity properties are
396 linked to a Datatype. . UBL defines two types of Datatypes – unspecialised and
397 specialised. The `ubl:UnspecialisedDatatypes` correspond to
398 `ccts:representatioterm`s and have no restrictions to the facets of the
399 corresponding `ccts:ContentComponent` or `ccts:SupplementaryComponent`. The
400 `ubl:SpecialisedDatatypes` are derived from `ubl:UnspecializedDatatypes`
401 with restrictions to the facets of the corresponding `ccts:ContentComponent` or
402 `ccts:SupplementaryComponent.DatatypeDatatype`.

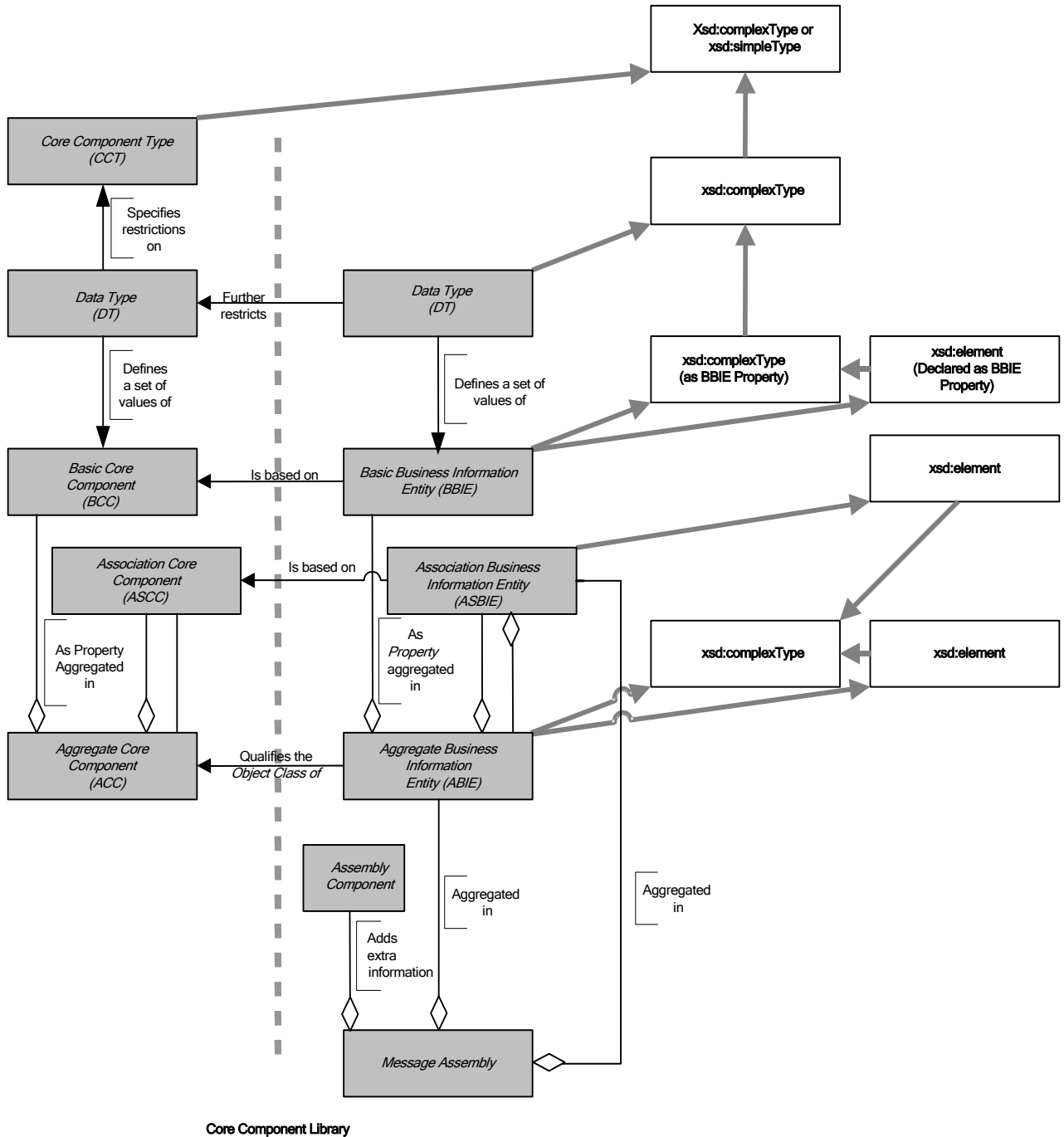
403 CCTS defines an approved set of primary and secondary representation terms. However,
404 these representation terms are simply naming conventions to identify the Datatype of an
405 object, not actual constructs. These representation terms are in fact the basis for
406 Datatypes as defined in the CCTS..

407 A `ccts:Datatype` “defines the set of valid values that can be used for a particular
408 *Basic Core Component Property* or *Basic Business Information Entity Property*
409 *Datatype*”⁶ The `ccts:Datatypes` can be either unspecialized – no restrictions applied –
410 or specialized through the application of restrictions. The sum total of the Datatypes is
411 then instantiated as the basis for the various types defined in the UBL schemas. CCTS
412 supports Datatypes that are unspecialized, i.e. it enables users to define their own
413 Datatypes for their syntax neutral constructs. Thus `ccts:Datatypes` allow UBL to
414 identify facets for elements when restrictions to the corresponding
415 `ccts:ContentComponent` or `ccts:SupplementaryComponent` is required.

416 A `ccts:AssociationBusinessInformationEntityProperty` represents an
417 *extrinsic* property – in other words an association from one `ccts:Aggregate`
418 `BusinessInformationEntityProperty` instance to another `ccts:Aggregate`
419 `BusinessInformationEntityProperty` instance. It is the `ccts:Aggregate`
420 `BusinessInformationEntityProperty` that expresses the relationship between
421 `ccts:AggregateBusinessInformationEntities`. Due to their unique extrinsic

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

422 **Figure 2-3. UBL Document Metamodel**



423
424

425 association role, `cts:AssociationBusinessInformationEntities` are not
 426 defined as `xsd:complexType`s, rather they are either declared as elements that are then
 427 bound to the `xsd:complexType` of the associated `cts:AggregateBusiness`
 428 `InformationEntity`, or they are reclassified ABIEs.

429 As stated above, `ccts:BasicBusinessInformationEntities` define the intrinsic
430 structure of a `ccts:AggregateBusinessInformationEntity`. These
431 `ccts:BasicBusinessInformationEntities` are the “leaf” types in the system in
432 that they contain no `ccts:AssociationBusinessInformationEntity` properties.
433 A `ccts:BasicBusinessInformationEntity` must have a
434 `ccts:CoreComponentType`. `Ccts:CoreComponentTypes` are low-level types, such
435 as Identifiers and Dates. A `Ccts:CoreComponentType` describes these low-level types
436 for use by `ccts:CoreComponents`, and (in parallel) a `ccts:Datatype`, corresponding
437 to that `ccts:CoreComponentType`, describes these low-level types for use by
438 `ccts:BusinessInformationEntities`. Every `ccts:CoreComponentType` has a
439 single `ccts:ContentComponent` and one or more `ccts:Supplementary`
440 `Components`. A `ccts:ContentComponent` is of some `Primitive Type`. All
441 `ccts:CoreComponentTypes` and their corresponding content and supplementary
442 components are pre-defined in the CCTS. UBL, in partnership with the Open
443 Applications Group has developed an `xsd:schemaModule` that defines each of the pre-
444 defined `ccts:CoreComponentTypes` as `xsd:complexType` or `xsd:simpleTypes`
445 and declares `ccts:SupplementaryComponents` as `xsd:attributes` or uses the
446 predefined facets of the built-in `xsd:Datatype` for those that are used as the base
447 expression for an `xsd:simpleType`.

448 3 General XML Constructs

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

- 450 ◆ Overall Schema Structure
- 451 ◆ Naming and Modeling Constraints
- 452 ◆ Reusability Scheme
- 453 ◆ Namespace Scheme
- 454 ◆ Versioning Scheme
- 455 ◆ Modularity Strategy
- 456 ◆ Schema Documentation Requirements

457 3.1 Overall Schema Structure

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

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

465 XML Declaration

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

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

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

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

472 `xmlns:xsd`

473 Target namespace

474 Default namespace

475 `CommonAggregateComponents`

476 `CommonBasicComponents`

```

477         CoreComponentTypes
478                                     Unspecialised Datatypes
479         Specialised Datatypes
480         Identifier Schemes
481         Code Lists
482     Attribute Declarations – elementFormDefault=”qualified”
483         attributeFormDefault=”unqualified”
484     <!-- ===== Imports ===== -->
485     CommonAggregateComponents schema module
486     CommonBasicComponents schema module
487     Unspecialized Types schema module
488     Specialized Types schema module
489     <!-- ===== Global Attributes ===== -->
490     Global Attributes and Attribute Groups
491     <!-- ===== Root Element ===== -->
492     Root Element Declaration
493     Root Element Type Definition
494     <!-- ===== Element Declarations ===== -->
495     alphabetized order
496     <!-- ===== Type Definitions ===== -->
497     All type definitions segregated by basic and aggregates as follows
498     <!-- ===== Aggregate Business Information Entity Type Definitions ===== -->
499     alphabetized order of ccts:AggregateBusinessInformationEntity xsd:TypeDefinitions
500     <!-- =====Basic Business Information Entity Type Definitions ===== -->
501     alphabetized order of ccts:BasicBusinessInformationEntities
502     <!-- ===== Copyright Notice ===== -->
503     Required OASIS full copyright notice.

```

504 3.1.1 Root Element

505 Per XML 1.0, “There is exactly one element, called the **root**, or document element, no
506 part of which appears in the content of any other element.” XML 1.0 further states “The
507 [root element](#) of any document is considered to have signaled no intentions as regards
508 application space handling, unless it provides a value for this attribute or the attribute is
509 declared with a default value.” W3C XSD allows for any globally declared element to be
510 the document root element. To keep consistency in the instance documents and to adhere

511 to the underlying process model that supports each UBL Schema, it is desirable to have
512 one and only one element function as the root element. Since UBL follows a global
513 element declaration scheme (See Rule ELD2), each UBL Schema will identify one
514 element declaration in each schema as the document root element. This will be
515 accomplished through an `xsd:annotation` child element for that element in
516 accordance with the following rule:

517 [ELD1] Each `UBL:DocumentSchema` MUST identify one and only one global
518 element declaration that defines the document
519 `ccts:AggregateBusinessInformationEntity` being conveyed in the
520 Schema expression. That global element MUST include an
521 `xsd:annotation` child element which MUST further contain an
522 `xsd:documentation` child element that declares "*This element MUST*
523 *be conveyed as the root element in any instance document*
524 *based on this Schema expression.*"

525 [Definition] Document schema –
526 The overarching schema within a specific namespace that conveys the business
527 document functionality of that namespace. The document schema declares a target
528 namespace and is likely to pull in by including internal schema modules or importing
529 external schema modules. Each namespace will have one, and only one, document
530 schema.

531 Example:

```
532 <xsd:element name="Order" type="OrderType">  
533   <xsd:annotation>  
534     <xsd:documentation>This element MUST be conveyed as the root  
535     element in any instance document based on this Schema  
536     expression</xsd:documentation>  
537   </xsd:annotation>  
538 </xsd:element>
```

543 3.2 Constraints

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

548 3.2.1 Naming Constraints

549 A primary component of the UBL library documentation is its dictionary. The entries in
550 the dictionary fully define the pieces of information available for use in UBL business

551 messages. These entries contain fully conformant CCTS dictionary entry names as well
552 as truncated UBL XML element names developed in conformance with the rules in
553 section 4. The dictionary entry name ties the information to its standardized semantics,
554 while the name of the corresponding XML element or attribute is only shorthand for this
555 full name. The rules for element and attribute naming and dictionary entry naming are
556 different.

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

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

565 3.2.2 Modeling Constraints

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

568 3.2.2.1 Defining Classes

569 UBL is based on instantiating ebXML `ccts:CoreComponents`. UBL models and the
570 XML expressions of those models are class driven. Specifically, classes are defined for
571 each `ccts:BasicBusinessInformationEntity` and `ccts:AggregateBusiness`
572 `InformationEntity` defined. UBL schemas define classes based on ebXML
573 `ccts:BasicBusinessInformationEntities` and `ccts:AggregateBusiness`
574 `InformationEntities`.

575 3.2.2.2 Core Component Types

576 Each `ccts:BasicBusinessInformationEntity` has an associated
577 `ccts:CoreComponentType`. The CCTS specifies an approved set of
578 `ccts:CoreComponentTypes`. To ensure conformance, UBL is limited to using this
579 approved set.

580 [MDC1] UBL Libraries and Schemas MUST only use ebXML Core Component
581 approved `ccts:CoreComponentTypes`.

582 Customization is a key aspect of UBL's reusability across business verticals. The UBL
583 rules have been developed in recognition of the need to support customizations. Specific
584 UBL customization rules are detailed in the UBL customization guidelines.

585 3.2.2.3 Mixed Content

586 UBL documents are designed to effect data-centric electronic commerce. Including
587 mixed content in business documents is undesirable because business transactions are
588 based on exchange of discrete pieces of data that must be clearly unambiguous. The
589 white space aspects of mixed content make processing unnecessarily difficult and add a
590 layer of complexity not desirable in business exchanges.

591 [MDC2] Mixed content MUST NOT be used except where contained in an
592 `xsd:documentation` element.

593 3.3 Reusability Scheme

594 The effective management of the UBL library requires that all element declarations are
595 unique across the breadth of the UBL library. Consequently, UBL elements are declared
596 globally, with the exception of Code and ID.

597 3.3.1.4 Reusable Elements

598 UBL elements are global and qualified. Hence the `<Address>` element is directly
599 reusable as a modular component and some software can be used without modification.
600 The UBL schema looks like this:

```
601 <xsd:element name="Party" type="PartyType"/>
602 <xsd:complexType name="PartyType">
603 <xsd:annotation>
604
605 <!--Documentation goes here--> </xsd:annotation>
606
607 <xsd:sequence>
608
609 <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0"
610 maxOccurs="1">
611
612 ...
613
614 </xsd:element>
615
616 <xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0"
617 maxOccurs="1">
618
619 ...
620
621 </xsd:element>
622
623 <xsd:element ref="PartyIdentification" minOccurs="0"
624 maxOccurs="unbounded">
625
```

```

626     ...
627
628     </xsd:element>
629
630     <xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
631
632     ...
633
634     </xsd:element>
635
636     <xsd:element ref="Address" minOccurs="0" maxOccurs="1">
637
638     ...
639     </xsd:element>
640
641     ...
642 </xsd:sequence>
643
644 </xsd:complexType>
645 <xsd:element name="Address" type="AddressType"/>
646
647 <xsd:complexType name="AddressType">
648
649     ...
650
651     <xsd:sequence>
652
653         <xsd:element ref="cbc:CityName" minOccurs="0" maxOccurs="1">
654
655         ...
656
657         </xsd:element>
658
659         <xsd:element ref="cbc:PostalZone" minOccurs="0" maxOccurs="1">
660
661         ...
662         </xsd:element>
663     ...
664
665     </xsd:sequence>
666
667 </xsd:complexType>
668
669

```

670 Software written to work with UBL's standard library will work with new assemblies of
671 the same components since global elements will remain consistent and unchanged. The

672 globally declared <Address> element is fully reusable without regard to the reusability
673 of types and provides a solid mechanism for ensuring that extensions to the UBL core
674 library will provide consistency and semantic clarity regardless of its placement within a
675 particular type.

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

682 [ELD2] All element declarations MUST be global with the exception of ID and Code
683 which MUST be local.

684 3.4 Namespace Scheme

685 The concept of XML namespaces is defined in the W3C XML namespaces technical
686 specification.⁷ The use of XML namespace is specified in the W3C XML Schema (XSD)
687 Recommendation. A namespace is declared in the root element of a Schema using a
688 namespace identifier. Namespace declarations can also identify an associated prefix –
689 shorthand identifier – that allows for compression of the namespace name. It is common
690 for an instance document to carry namespace declarations, so that it might be validated.

691 3.4.1 Declaring Namespaces

692 Neither XML 1.0 nor XSD require the use of Namespaces. However the use of
693 namespaces is essential to managing the complex UBL library. UBL will use UBL-
694 defined schemas (created by UBL) and UBL-used schemas (created by external
695 activities) and both require a consistent approach to namespace declarations.

696 [NMS1] Every UBL-defined or -used schema module, except internal schema
697 modules, MUST have a namespace declared using the
698 `xsd:targetNamespace` attribute.

699 Each UBL schema module consists of a logical grouping of lower level artifacts that
700 together comprise an association that will be able to be used in a variety of UBL
701 schemas. These schema modules are grouped into a schema set collection. Each schema
702 set is assigned a namespace that identifies that group of schema modules. As constructs
703 are changed, new versions will be created. The schema set is the versioned entity, all

⁷ Tim Bray, D Hollander, A Layman, R Tobin; *Namespaces in XML 1.1*, W3C Recommendation, February 2004.

704 schema modules within that package are of the same version, and each version has a
705 unique namespace.

706 Definition: Schema Set

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

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

712 [NMS2] Every UBL-defined or -used schema set version MUST have its own unique
713 namespace.

714 UBL's extension methodology encourages a wide variety in the number of schema
715 modules that are created as derivations from UBL schema modules. Clarity and
716 consistency requires that customized schema not be confused with those developed by
717 UBL.

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

719 3.4.2 Namespace Uniform Resource Identifiers

720 A UBL namespace name must be a Uniform Resource Identifier (URI) reference that
721 conforms to RFC 2396.⁸ UBL has adopted the URN scheme as the standard for URIs for
722 UBL namespaces, in conformance with IETF's RFC 3121⁹, as defined in this next
723 section

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

727 [NMS4] The namespace names for UBL Schemas holding committee draft status
728 MUST be of the form:

729 `urn:oasis:names:tc:ubl:schema:<subtype>:<document-id>`

730 The format for `document-id` is found in the next section.

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.*

⁹ Karl Best, N. Walsh; *Internet Engineering Task Force (IETF) RFC 3121, A URN Namespace for OASIS, June 2001.*

731 For each UBL schema holding OASIS Standard status, a UBL namespace must be
732 declared and named using the same notation, but with the value ‘specification’
733 replacing the value ‘tc’ .

734 [NMS5] The namespace names for UBL Schemas holding OASIS Standard status
735 MUST be of the form:
736
737 urn:oasis:names:specification:ubl:schema:<subtype>:<docum
738 ent-id>

739 3.4.3 Schema Location

740 UBL schemas use a URN namespace scheme. In contrast, schema locations are typically
741 defined as a URL. UBL schemas must be available both at design time and run time. As
742 such, the UBL schema locations will differ from the UBL namespace declarations. UBL,
743 as an OASIS TC, will utilize an OASIS URL for hosting UBL schemas. UBL will use
744 the committee directory <http://www.oasis-open.org/committees/ubl/schema/>.

745 3.4.4 Persistence

746 A key differentiator in selecting URNs to define UBL namespaces is URN persistence.
747 UBL namespaces must never violate this functionality by subsequently changing a
748 namespace once it has been declared. Conversely, any changes to a schema will result in
749 a new namespace declaration. Thus a published schema version and its namespace
750 association will always be inviolate.

751 [NMS6] UBL published namespaces MUST never be changed.

752 3.5 Versioning Scheme

753 UBL namespaces conform to the OASIS namespace rules. The last field of the
754 namespace name is called `document-id`. UBL has decided to include versioning
755 information as part of the `document-id` component of the namespace. The version information
756 is divided into `major` and `minor` fields. The `minor` field has an optional `revision`
757 extension. For example, the namespace URI for the draft Invoice domain has this form:

758 urn:oasis:names:tc:ubl:schema:xsd:Invoice-
759 <major>.<minor>[.<revision>]

760 The *major-version* field is “1” for the first release of a namespace. Subsequent major
761 releases increment the value by 1. For example, the first namespace URI for the first
762 major release of the Invoice document has the form:

763 urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.0

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

765 urn:oasis:names:tc:ubl:schema:xsd:Invoice-2.0

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

769 urn:oasis:names:tc:ubl:schema:xsd:Invoice:-<major-number>.0[.<revision>]
770

771 [VER1] Every UBL Schema and schema module major version committee draft
772 MUST have an RFC 3121 document-id of the form

773 <name>-<major>.0[.<revision>]
774

775 [VER2] Every UBL Schema and schema module major version OASIS Standard
776 MUST have an RFC 3121 document-id of the form

777 <name>-<major>.0

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

782 Example:

783 Example

784
785 The namespace URI for the first minor release of the Invoice domain has this
786 form:

787 urn:oasis:names:tc:ubl:schema:xsd:Invoice-<major>.1
788

789

790 [VER3] Every minor version release of a UBL schema or schema module draft MUST
791 have an RFC 3121 document-id of the form

792 <name>-<major >.<non-zero>[.<revision>]
793

794 [VER4] Every minor version release of a UBL schema or schema module OASIS
795 Standard MUST have an RFC 3121 document-id of the form

796 <name>-<major >.<non-zero>

797 Once a schema version is assigned a namespace, that schema version and that namespace
798 will be associated in perpetuity. Any change to any schema module mandates association
799 with a new namespace.

800 [VER5] For UBL Minor version changes <name> MUST not change,

801 UBL is composed of a number of interdependent namespaces. For instance, namespaces
802 whose URI's start with urn:oasis:names:tc:ubl:schema:xsd:Invoice-* are

803 dependent upon the common basic and aggregate namespaces, whose URI's have the
804 form `urn:oasis:names:tc:ubl:schema:xsd:CommonBasicComponents-*` and
805 `urn:oasis:names:tc:ubl:schema:xsd:CommonAggregateComponents-*`
806 respectively. If either of the common namespaces change then its namespace URI must
807 change. If its namespace URI changes then any schema that imports the *new version* of
808 the namespace must also change (to update the namespace declaration). And since the
809 importing schema changes, its namespace URI in turn must change. The outcome is
810 twofold:

- 811 ◆ There should never be ambiguity at the point of reference in a namespace
812 declaration or version identification. A dependent schema imports precisely
813 the version of the namespace that is needed. The dependent schema never
814 needs to account for the possibility that the imported namespace can change.
- 815 ◆ When a dependent schema is upgraded to import a new version of a schema,
816 the dependent schema's version (in its namespace URI) must change.

817 Version numbers are based on a logical progression. All major and minor version
818 numbers will be based on positive integers. Version numbers always increment positively
819 by one.

820 [VER6] Every UBL Schema and schema module major version number MUST be a
821 sequentially assigned, incremental number greater than zero.

822 [VER7] Every UBL Schema and schema module minor version number MUST be a
823 sequentially assigned, incremental non-negative integer.

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

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

828 `urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.2`

829 *will* import the namespace:

830 `urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.1.`

831 The `version 1.2` revision may define new complex types by extending or restricting
832 `version 1.1` types. It may define brand new complex types and elements by
833 composition. It must not use the XSD `redefine` element to change the definition of a type
834 or element in the `1.1` version.

835 The opportunity exists in the `version 1.2` revision to rename derived types. For
836 instance if `version 1.1` defines `Address` and `version 1.2` specializes `Address` it
837 would be possible to give the derived `Address` a new name, e.g. `NewAddress`. This is
838 not required since namespace qualification suffices to distinguish the two distinct types.

839 The minor revision may give a derived type a new name only if the semantics of the two
840 types are distinct.

841 For a particular namespace, the minor versions of a major version form a linearly-linked
842 family. The first minor version imports its parent major version. Each successive minor
843 version imports the schema module of the preceding minor version.

844 Example

```
845 urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.2 imports  
846 urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.1 which  
847 imports urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.0  
848
```

850 [VER8] A UBL minor version document schema MUST import its immediately
851 preceding version document schema.

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

857 [VER9] UBL Schema and schema module minor version changes MUST be limited to
858 the use of `xsd:extension` or `xsd:restriction` to alter existing types or
859 add new constructs.

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

862 [VER10] UBL Schema and schema module minor version changes MUST not break
863 semantic compatibility with prior versions.

864

865 3.6 Modularity

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

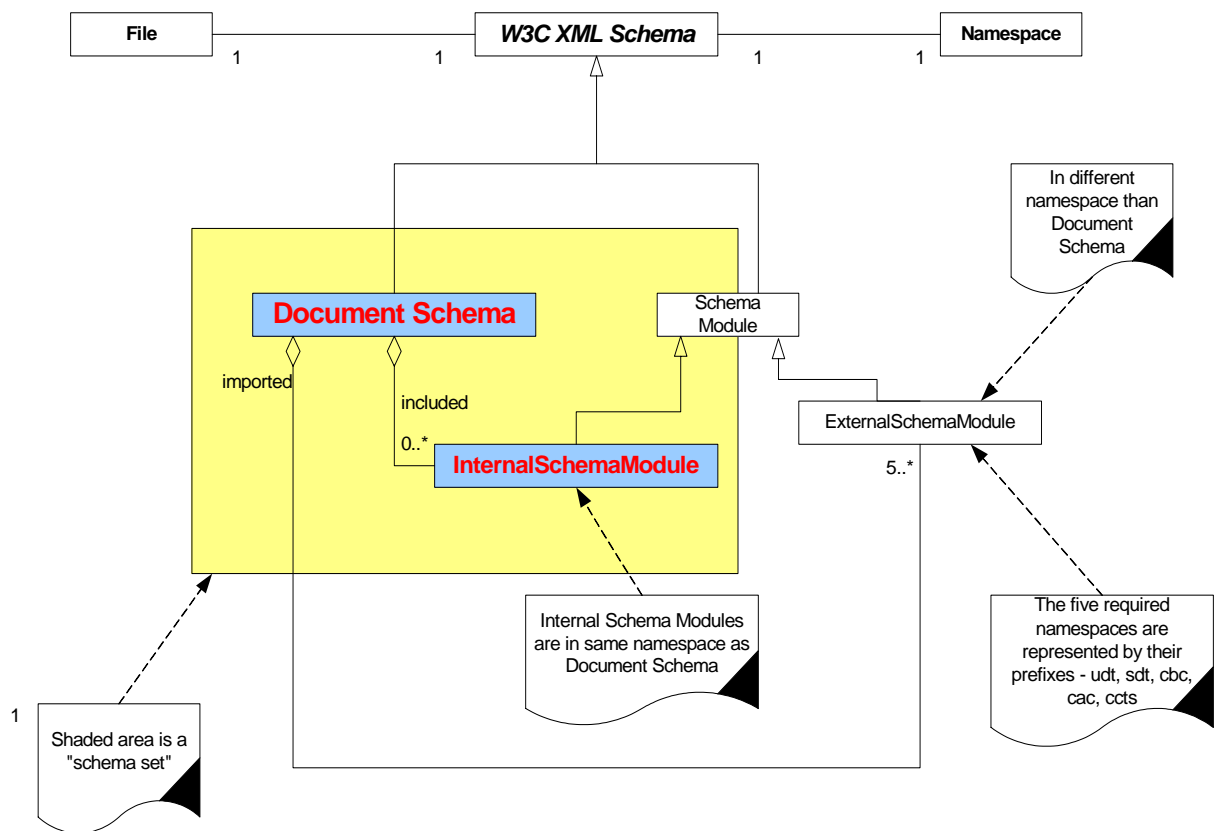
872 [SSM1] UBL Schema expressions MAY be split into multiple schema modules.

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

875 **3.6.1 UBL Modularity Model**

876 UBL relies extensively on modularity in schema design. There is no single UBL root
 877 schema. Rather, there are a number of UBL document schemas, each of which expresses
 878 a separate business function. The UBL modularity approach is structured so that users
 879 can reuse individual document schemas without having to import the entire UBL
 880 document schema library. Additionally, a document schema can import individual
 881 modules without having to import all UBL schema modules. Each document schema will
 882 define its own dependencies. The UBL schema modularity model ensures that logical
 883 associations exist between document and internal schema modules and that individual
 884 modules can be reused to the maximum extent possible. This is accomplished through the
 885 use of document and internal schema modules as shown in Figure 3-1.

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



887
888

889 If the contents of a namespace are small enough then they can be completely specified
890 within the document schema.

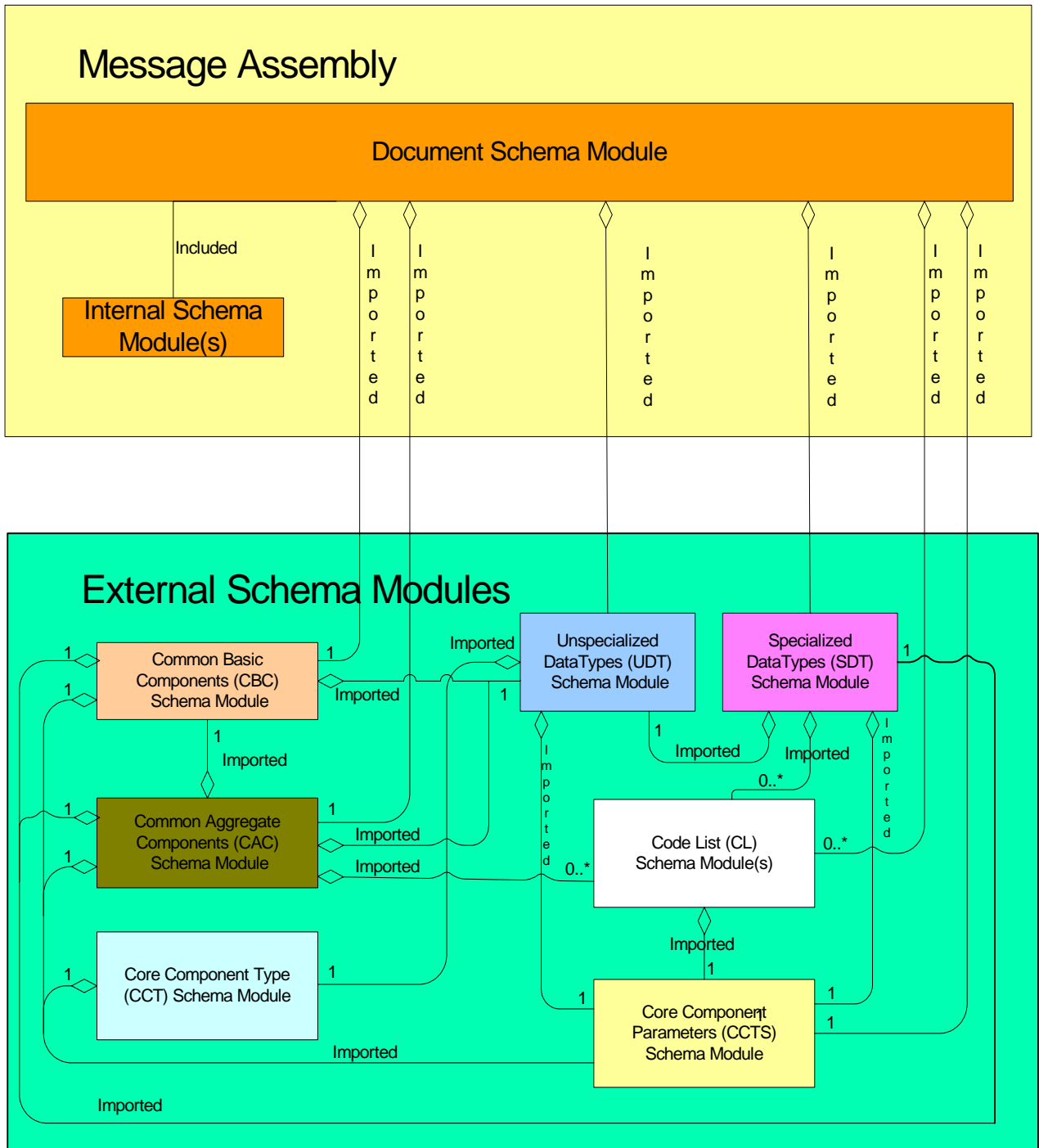
891 Figure 3-1 shows the one-to-one correspondence between document schemas and
892 namespaces. It also shows the one-to-one correspondence between files and schema
893 modules. As shown in figure 3-1, there are two types of schema in the UBL library -
894 DocumentSchema and SchemaModules. Document Schema are always in their own
895 namespace. Schema modules may be in a document schema namespace as in the case of

896 internal schema modules, or in a separate namespace as in the `ubl:udt`, `ubl:sdt`,
897 `ubl:cbc`, `ubl:cac`, `ubl:cl`, `ubl:cct`, and `ubl:ccts` schema modules. Both
898 types of schema modules are conformant with W3C XSD.

899 A namespace is an indivisible grouping of types. A “piece” of a namespace can never be
900 used without all its pieces. For larger namespaces, schema modules – internal schema
901 modules – may be defined. UBL document schemas may have zero or more internal
902 modules that they include. The document schema for a namespace then includes those
903 internal modules.

904 **[Definition] Internal schema module:** A schema that is part of a schema set within a
905 specific namespace.

906 Another way to visualize the structure is by example. Figure 3-2 depicts instances of the
907 various classes from the previous diagram.

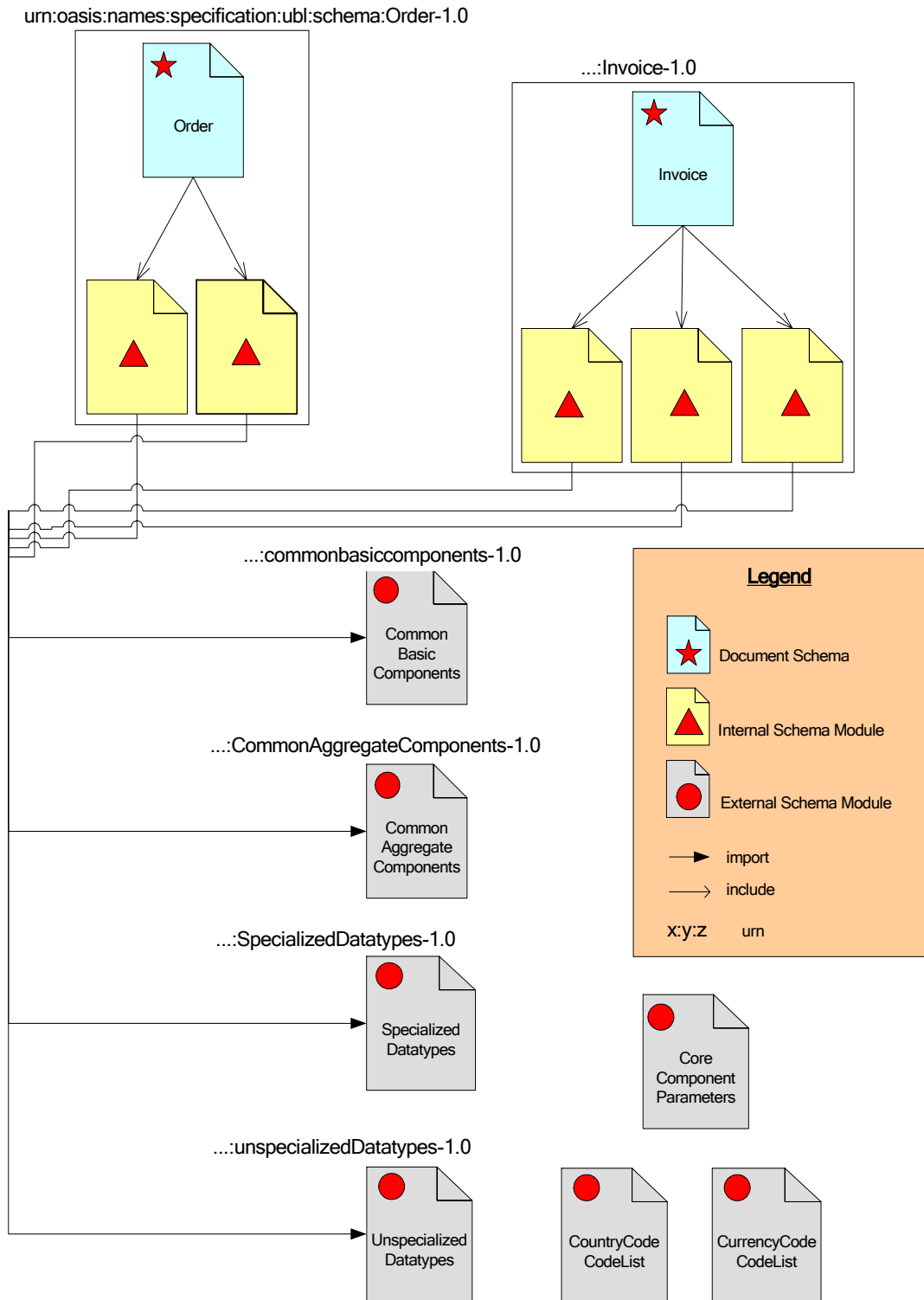


909
910

911 Figure 3-3 shows how the order and invoice document schemas import the
 912 "CommonAggregateComponents" and "CommonBasicComponents" external schema
 913 modules. It also shows how the order document schema includes various internal
 914 modules – modules local to that namespace. The clear boxes show how the various
 915 schema modules are grouped into namespaces.

916 Any UBL schema module, be it a document schema or an internal module may import
 917 other document schemas from other namespaces.

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



919

920 3.6.1.5 Limitations on Import

921 If two namespaces are mutually dependent then clearly, importing one will cause the
922 other to be imported as well. For this reason there must not exist circular dependencies
923 between UBL schema modules. By extension, there must not exist circular dependencies
924 between namespaces. A namespace “A” dependent upon type definitions or element
925 declaration defined in another namespace “B” must import “B’s” document schema.

926 [SSM2] A document schema in one UBL namespace that is dependent upon type
927 definitions or element declarations defined in another namespace MUST only
928 import the document schema from that namespace.

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

932 [SSM3] A UBL document schema in one UBL namespace that is dependant upon type
933 definitions or element declarations defined in another namespace MUST NOT
934 import internal schema modules from that namespace.

935 3.6.1.6 Module Conformance

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

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

940 3.6.2 Internal and External schema modules

941 UBL will create schema modules which, as illustrated in Figure 3-1 and Figure 3-2, will
942 either be located in the same namespace as the corresponding document schema, or in a
943 separate namespace.

944 [SSM5] UBL schema modules MUST either be treated as external schema modules or
945 as internal schema modules of the document schema.

946 3.6.3 Internal schema modules

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

950 [SSM6] All UBL internal schema modules MUST be in the same namespace as their
951 corresponding document schema.

952 UBL internal schema modules will necessarily have semantically meaningful names.
953 Internal schema module names will identify the parent schema module, the internal
954 schema module function, and the schema module itself.

955	[SSM7]	Each UBL internal schema module MUST be named
956		{ParentSchemaModuleName}{InternalSchemaModuleFunction}{sc
957		HEMA module }

958 3.6.4 External schema modules

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

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

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

- 966 ◆ UBL CommonAggregateComponents
- 967 ◆ UBL CommonBasicComponents
- 968 ◆ UBL Code List(s)
- 969 ◆ CCTS Core Component Types
- 970 ◆ CCTS Unspecialized Datatypes
- 971 ◆ UBL Specialized Datatypes
- 972 ◆ CCTS Core Component Parameters - [Ed Note – Lise/Stephen have already
973 written this section get from release and Lisa]

974 3.6.4.7 UBL CommonAggregateComponents schema module

975 The UBL library will also contain a wide variety of
976 `ccts:AggregateBusinessInformationEntities`. As defined in rule CTD1, each
977 of these `ccts:AggregateBusinessInformationEntity` classes will be defined as
978 an `xsd:complexType`. Although some of these `xsd:complexTypes` may be used on
979 only one UBL Schema, many will be reused in multiple UBL schema modules. An
980 aggregation of all of the `ccts:AggregateBusinessInformationEntity`
981 `xsd:ComplexType` definitions that are used in multiple UBL schema modules into a
982 single schema module of common aggregate types will provide for maximum ease of
983 reuse.

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

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

988 [SSM10] The `ubl:CommonAggregateComponents` schema module MUST be named
989 “*ubl:CommonAggregateComponents Schema Module*”

990 **3.6.4.7.1 UBL CommonAggregateComponents schema module Namespace**

991 In keeping with the overall UBL namespace approach, a singular namespace must be
992 created for storing the `ubl:CommonAggregateComponents` schema module.

993 [NMS7] The `ubl:CommonAggregateComponents` schema module MUST reside in
994 its own namespace.

995 To ensure consistency in expressing this module, a normative token that will be used
996 consistently in all UBL Schemas must be defined.

997 [NMS8] The `ubl:CommonAggregateComponents` schema module MUST be
998 represented by the token “*cac*”.

999 **3.6.4.8 UBL CommonBasicComponents schema module**

1000 The UBL library will contain a wide variety of
1001 `ccts:BasicBusinessInformationEntities`. These `ccts:BasicBusiness`
1002 `InformationEntities` are based on `ccts:BasicBusinessInformation`
1003 `EntityProperties`. The BBIE Properties are reusable in multiple BBIEs and per the
1004 CCTS are of type BBIE Property Type which are in turn of type Datatype. The BBIEs are
1005 reusable across multiple schema modules and per the CCTS are of Type BBIE Property
1006 Type. As defined in rule CTD1, each of these `ccts:BasicBusinessInformation`
1007 `EntityProperty` classes will be defined as an `xsd:ComplexType`. Although some of
1008 these `xsd:ComplexTypes` may be used in only one UBL Schema, many will be reused
1009 in multiple UBL schema modules. To maximize reuse and standardization, all of the
1010 `ccts:BasicBusinessInformationEntityProperty` `xsd:ComplexType`
1011 definitions that are used in multiple UBL schema modules will be aggregated into a
1012 single schema module of common basic types.

1013 [SSM11] A schema module defining all `ubl:CommonBasicComponents` MUST be
1014 created.

1015 The normative name for this schema module will be based on its
1016 `ccts:BasicBusinessInformationEntityProperty` `xsd:ComplexType` content.

1017 [SSM12] The `ubl:CommonBasicComponents` schema module MUST be named
1018 “*ubl:CommonBasicComponents Schema Module*”

1019 **3.6.4.8.1 UBL CommonBasicComponents schema module Namespace**

1020 In keeping with the overall UBL namespace approach, a singular namespace must be
1021 created for storing the `ubl:CommonBasicComponents` schema module.

1022 [NMS9] The `ubl:CommonBasicComponents` schema module MUST reside in its
1023 own namespace.

1024 To ensure consistency in expressing the `ubl:CommonBasicComponents` schema
1025 module, a normative token that will be used consistently in all UBL Schema must be
1026 defined.

1027 [NMS10] The `UBL:CommonBasicComponents` schema module MUST be represented
1028 by the token “`cbc`”.

1029 **3.6.4.9 CCTS Core Component Type schema module**

1030 The CCTS defines an authorized set of Core Component Types (`ccts:Core`
1031 `ComponentTypes`) that convey content and supplementary information related to
1032 exchanged data. As the basis for all higher level CCTS models, the `ccts:Core`
1033 `ComponentTypes` are reusable in every UBL schema. An external schema module
1034 consisting of a complex type definition for each `ccts:CoreComponentType` is
1035 essential to maximize reusability.

1036 [SSM13] A schema module defining all `ccts:CoreComponentTypes` MUST be
1037 created.

1038 The normative name for the `ccts:CoreComponentType` schema module will be based
1039 on its content.

1040 [SSM14] The `ccts:CoreComponentType` schema module MUST be named
1041 “`ccts:CoreComponentType Schema Module`”

1042 By design, `ccts:CoreComponentTypes` are generic in nature. As such, restrictions
1043 are not appropriate. Such restrictions will be applied through the application of
1044 Datatypes. Accordingly, the `xsd:facet` feature must not be used in the `ccts:CCT`
1045 schema module.

1046 [SSM15] The `xsd:facet` feature MUST not be used in the
1047 `ccts:CoreComponentType` schema module.

1048 **3.6.4.9.1 Core Component Type schema module Namespace**

1049 In keeping with the overall UBL namespace approach, a singular namespace must be
1050 created for storing the `ccts:CoreComponentType` schema module.

1051 [NMS11] The `ccts:CoreComponentType` schema module MUST reside in its own
1052 namespace.

1053 To ensure consistency in expressing the `ccts:CoreComponentType` schema module, a
1054 normative token that will be used in consistently in all UBL Schema must be defined.

1055 [NMS12] The `ccts:CoreComponentType` schema module namespace MUST be
1056 represented by the token “cct”.

1057 3.6.4.10 CCTS Datatypes schema modules

1058 The CCTS defines an authorized set of primary and secondary Representation Terms
1059 (`ccts:RepresentationTerms`) that describes the form of every
1060 `ccts:BusinessInformationEntity`. These `ccts:RepresentationTerms` are
1061 instantiated in the form of Datatypes that are reusable in every UBL schema. The
1062 `ccts:Datatype` defines the set of valid values that can be used for its associated
1063 `ccts:BasicBusinessInformationEntity` Property. These Datatypes may be
1064 specialized or unspecialized, that is to say restricted or unrestricted. We refer to these as
1065 `ccts:UnspecializedDatatypes` (even though they are technically
1066 `ccts:Datatypes`) or `ubl:SpecialisedDatatypes`.

1067 3.6.4.10.1 CCTS Unspecialised Datatypes Schema Module

1068 An external schema module consisting of a complex type definition for each
1069 `ccts:UnspecialisedDatatype` is essential to maximize reusability. However, since
1070 UBL is also using code list schema modules that themselves import the `ccts:Datatype`
1071 schema module, a separate schema module for `ccts:CodeTypeUnspecialised`
1072 `Datatype` is also required, to avoid circular dependencies.

1073 [SSM16] A schema module defining all `ccts:UnspecialisedDatatypes` MUST
1074 be created.

1075

1076 The normative name for the `ccts:UnspecialisedDatatype` schema module will be
1077 based on its content.

1078 [SSM17] The `ccts:UnspecialisedDatatype` schema module MUST be named
1079 “*ccts:UnspecialisedDatatype Schema Module*”

1080

1081 In keeping with the overall UBL namespace approach, a singular namespace must be
1082 created for storing the `ccts:UnspecialisedDatatype` schema module.

1083 [NMS13] The `ccts:UnspecialisedDatatype` schema module MUST reside in its
1084 own namespace.

1085

1086 To ensure consistency in expressing the `ccts:UnspecialisedDatatype` schema
1087 module, a normative token that will be used consistently in all UBL Schema must be
1088 defined.

1089 [NMS14] The `ccts:UnspecialisedDatatype` schema module namespace MUST
1090 be represented by the token “`udt`”.

1091 ***3.6.4.10.2 UBL Specialised Datatypes***

1092 UBL specialized Datatypes are restrictions on `ccts:UnspecialisedDatatypes`.
1093 These restrictions take the form of restrictions on the underlying `ccts:CoreComponent`
1094 `Type Datatype`. The `ubl:SpecialisedDatatype` is defined by specifying restrictions
1095 on the `ccts:CoreComponentType` that forms the basis of the `ccts:Unspecialised`
1096 `Datatype`. As specialized Datatypes are defined by individual users, they should be
1097 identified by those users. To ensure consistency of UBL specialized Datatypes
1098 (`ubl:SpecialisedDatatypes`) with the UBL modularity and reuse goals requires
1099 creating a single schema module that defines all `ubl:SpecialisedDatatypes`.

1100 [SSM18] A schema module defining all `ubl:SpecialisedDatatypes` MUST be
1101 created.

1102 The `ubl:SpecialisedDatatypes` schema module name must follow the UBL module
1103 naming approach.

1104 [SSM19] The `ubl:SpecialisedDatatypes` schema module MUST be named
1105 “`ubl:SpecialisedDatatypes schema module`”

1106 ***3.6.4.10.3 UBL Specialised Datatype schema module Namespace***

1107 In keeping with the overall UBL namespace approach, a singular namespace must be
1108 created for storing the `ubl:SpecialisedDatatypes` schema module.

1109 [NMS15] The `ubl:SpecialisedDatatypes` schema module MUST reside in its
1110 own namespace.

1111 To ensure consistency in expressing the `ubl:SpecialisedDatatypes` schema
1112 module, a normative token that will be used in all UBL schemas must be defined.

1113 [NMS16] The `ubl:SpecialisedDatatypes` schema module namespace MUST be
1114 represented by the token “`sdt`”.

1115 ***3.7 Annotation and Documentation***

1116 Annotation is an essential tool in understanding and reusing a schema. UBL, as an
1117 implementation of CCTS, requires an extensive amount of annotation to provide all
1118 necessary metadata required by the CCTS specification. Each construct declared or
1119 defined within the UBL library contains the requisite associated metadata to fully

1120 describe its nature and support the CCTS requirement. Accordingly, UBL schema
1121 metadata for each construct will be defined in the core component parameters.

1122 3.7.1 Schema Annotation

1123 Although the UBL schema annotation is necessary, its volume results in a considerable
1124 increase in the size of the UBL schemas with undesirable performance impacts. To
1125 address this issue, two normative schema will be developed for each UBL schema. A
1126 fully annotated schema will be provided to facilitate greater understanding of the schema
1127 module and its components, and to meet the CCTS metadata requirements. A schema
1128 devoid of annotation will also be provided that can be used at run-time if required to meet
1129 processor resource constraints.

1130 [GXS2] UBL MUST provide two normative schemas for each transaction. One
1131 schema shall be fully annotated. One schema shall be a run-time schema
1132 devoid of documentation.

1133 3.7.2 Embedded documentation

1134 The information about each UBL BIE is in the library spreadsheets. UBL spreadsheets
1135 contain all necessary information to produce fully annotated Schemas. Fully annotated
1136 Schemas are valuable tools to implementers to assist in understanding the nuances of the
1137 information contained therein. UBL annotations will consist of information currently
1138 required by Section 7 of the CCTS and supplemented by necessary information identified
1139 by LCSC.

1140 The absence of an optional annotation inside the structured set of annotations in the
1141 documentation element implies the use of the default value. For example, there are
1142 several annotations relating to context such as `BusinessTermContext` or
1143 `IndustryContext` whose absence implies that their value is "all contexts".

1144 The following rules describe the documentation requirements for each Datatype
1145 definition.

1146 [DOC1] The `xsd:documentation` element for every Datatype MUST contain a structured
1147 set of annotations in the following sequence and pattern:

- 1148 • `ComponentType` (mandatory): The type of component to which the object
1149 belongs. For Datatypes this must be "DT".
- 1150 • `DictionaryEntryName` (mandatory): The official name of a Datatype.
- 1151 • `Version` (optional): An indication of the evolution over time of the Datatype.
- 1152 • `Definition`(mandatory): The semantic meaning of a Datatype.
- 1153 • `ObjectClassQualifier` (optional): The qualifier for the object class.
- 1154 • `ObjectClass`(optional): The Object Class represented by the Datatype.

1155 • RepresentationTerm (mandatory): A Representation Term is an element of
1156 the name which describes the form in which the property is represented.
1157 • DataTypeQualifier (optional): semantically meaningful name that
1158 differentiates the Datatype from its underlying Core Component Type.
1159 • DataType (optional): Defines the underlying Core Component Type.
1160

1161 [DOC2] A Datatype definition MAY contain one or more Content Component
1162 Restrictions to provide additional information on the relationship between the
1163 Datatype and its corresponding Core Component Type. If used the Content
1164 Component Restrictions must contain a structured set of annotations in the
1165 following patterns:

1166 • RestrictionType (mandatory): Defines the type of format restriction that
1167 applies to the Content Component.

1168 • RestrictionValue (mandatory): The actual value of the format restriction that
1169 applies to the Content Component.

1170 • ExpressionType (optional): Defines the type of the regular expression of the
1171 restriction value.
1172

1173 [DOC3] A Datatype definition MAY contain one or more Supplementary Component
1174 Restrictions to provide additional information on the relationship between the
1175 Datatype and its corresponding Core Component Type. If used the
1176 Supplementary Component Restrictions must contain a structured set of
1177 annotations in the following patterns:

1178 • SupplementaryComponentName (mandatory): Identifies the
1179 Supplementary Component on which the restriction applies.

1180 • RestrictionValue (mandatory, repetitive): The actual value(s) that is
1181 (are) valid for the Supplementary Component

1182 The following rule describes the documentation requirements for each Basic Business
1183 Information Entity definition.

1184 [DOC4] The `xsd:documentation` element for every Basic Business Information
1185 Entity MUST contain a structured set of annotations in the following patterns:

1186 • ComponentType (mandatory): The type of component to which the object
1187 belongs. For Basic Business Information Entities this must be “BBIE”.

1188 • DictionaryEntryName (mandatory): The official name of a Basic Business
1189 Information Entity.

1190 • Version (optional): An indication of the evolution over time of the Basic
1191 Business Information Entity.

1192 • Definition(mandatory): The semantic meaning of a Basic Business
1193 Information Entity.

- 1194 • Cardinality(mandatory): Indication whether the Basic Business Information
- 1195 Entity represents a not-applicable, optional, mandatory and/or repetitive
- 1196 characteristic of the Aggregate Business Information Entity.
- 1197 • ObjectClassQualifier (optional): The qualifier for the object class.
- 1198 • ObjectClass(mandatory): The Object Class containing the Basic Business
- 1199 Information Entity.
- 1200 • PropertyTermQualifier (optional): A qualifier is a word or words which help
- 1201 define and differentiate a Basic Business Information Entity.
- 1202 • PropertyTerm(mandatory): Property Term represents the distinguishing
- 1203 characteristic or Property of the Object Class and shall occur naturally in the
- 1204 definition of the Basic Business Information Entity.
- 1205 • RepresentationTerm (mandatory): A Representation Term describes the
- 1206 form in which the Basic Business Information Entity is represented.
- 1207 • DataTypeQualifier (optional): semantically meaningful name that
- 1208 differentiates the Datatype of the Basic Business Information Entity from its
- 1209 underlying Core Component Type.
- 1210 • DataType (mandatory): Defines the Datatype used for the Basic Business
- 1211 Information Entity.
- 1212 • AlternativeBusinessTerms (optional): Any synonym terms under which the
- 1213 Basic Business Information Entity is commonly known and used in the
- 1214 business.
- 1215 • Examples (optional): Examples of possible values for the Basic Business
- 1216 Information Entity.

1217 The following rule describes the documentation requirements for each Aggregate
 1218 Business Information Entity definition.

- 1219 [DOC5] The xsd:documentation element for every Aggregate Business Information
 1220 Entity MUST contain a structured set of annotations in the following sequence
 1221 and pattern:
- 1222 • ComponentType (mandatory): The type of component to which the object
 - 1223 belongs. For Aggregate Business Information Entities this must be “ABIE”.
 - 1224 • DictionaryEntryName (mandatory): The official name of the Aggregate
 - 1225 Business Information Entity .
 - 1226 • Version (optional): An indication of the evolution over time of the
 - 1227 Aggregate Business Information Entity.
 - 1228 • Definition(mandatory): The semantic meaning of the Aggregate Business
 - 1229 Information Entity.
 - 1230 • ObjectClassQualifier (optional): The qualifier for the object class.

1231
1232
1233
1234
1235

- **ObjectClass(mandatory):** The Object Class represented by the Aggregate Business Information Entity.
- **AlternativeBusinessTerms (optional):** Any synonym terms under which the Aggregate Business Information Entity is commonly known and used in the business.

1236
1237

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

1238
1239
1240

[DOC6] The xsd:documentation element for every Association Business Information Entity element declaration **MUST** contain a structured set of annotations in the following sequence and pattern:

1241
1242

- **ComponentType (mandatory):** The type of component to which the object belongs. For Association Business Information Entities this must be “ASBIE”.

1243
1244

- **DictionaryEntryName (mandatory):** The official name of the Association Business Information Entity.

1245
1246

- **Version (optional):** An indication of the evolution over time of the Association Business Information Entity.

1247
1248

- **Definition(mandatory):** The semantic meaning of the Association Business Information Entity.

1249
1250
1251

- **Cardinality(mandatory):** Indication whether the Association Business Information Entity represents an optional, mandatory and/or repetitive association.

1252
1253

- **ObjectClass(mandatory):** The Object Class containing the Association Business Information Entity.

1254
1255

- **PropertyTermQualifier (optional):** A qualifier is a word or words which help define and differentiate the Association Business Information Entity.

1256
1257
1258

- **PropertyTerm(mandatory):** Property Term represents the Aggregate Business Information Entity contained by the Association Business Information Entity.

1259
1260
1261
1262

- **AssociatedObjectClassQualifier (optional):** Associated Object Class Qualifiers describe the 'context' of the relationship with another ABIE. That is, it is the role the contained Aggregate Business Information Entity plays within its association with the containing Aggregate Business Information Entity.

1263
1264
1265

- **AssociatedObjectClass (mandatory);** Associated Object Class is the Object Class at the other end of this association. It represents the Aggregate Business Information Entity contained by the Association Business Information Entity.

1266
1267

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

1268 [DOC7] The xsd:documentation element for every Core Component Type MUST contain
1269 a structured set of annotations in the following sequence and pattern:

1270 • ComponentType (mandatory): The type of component to which the object
1271 belongs. For Core Component Types this must be “CCT”.

1272 • DictionaryEntryName (mandatory): The official name of the Core
1273 Component Type, as defined by [CCTS].

1274 • Version (optional): An indication of the evolution over time of the Core
1275 Component Type.

1276 • Definition (mandatory): The semantic meaning of the Core Component
1277 Type, as defined by [CCTS].

1278 • ObjectClass (mandatory): The Object Class represented by the Core
1279 Component Type, as defined by [CCTS].

1280 • PropertyTerm (mandatory): The Property Term represented by the Core
1281 Component Type, as defined by [CCTS].

1282 4 Naming Rules

1283 The rules in this section make use of the following special concepts related to XML
1284 elements and attributes:

- 1285 ◆ Top-level element: An element that encloses a whole UBL business message.
1286 Note that UBL business messages might be carried by messaging transport
1287 protocols that themselves have higher-level XML structure. Thus, a UBL top-
1288 level element is not necessarily the root element of the XML document that
1289 carries it.
- 1290 ◆ Lower-level element: An element that appears inside a UBL business
1291 message.
- 1292 ◆ Intermediate element: An element not at the top level that is of a complex
1293 type, only containing other elements and attributes.
- 1294 ◆ Leaf element: An element containing only character data (though it may also
1295 have attributes). Note that, because of the XSD mechanisms involved, a leaf
1296 element that has attributes must be declared as having a complex type, but a
1297 leaf element with no attributes may be declared with either a simple type or a
1298 complex type.
- 1299 ◆ Common attribute: An attribute that has identical meaning on the multiple
1300 elements on which it appears. A common attribute might or might not
1301 correspond to an XSD global attribute.

1302 4.1 General Naming Rules

1303 The CCTS contains specific ISO/IEC 11179 based naming rules for each CCTS
1304 construct. The UBL component library, as a syntax-neutral representation, is fully
1305 conformant to those rules. The UBL syntax-specific XSD instantiation of the UBL
1306 component library, in some cases refines the CCTS naming rules to leverage the
1307 capabilities of XML and XSD. Specifically, truncation rules are applied to allow for
1308 reuse of element names across parent element environments and to maintain brevity and
1309 clarity.

1310 In keeping with CCTS, UBL will use English as its normative language. If the UBL
1311 Library is translated into other languages for localization purposes, these additional
1312 languages might require additional restrictions. Such restrictions are expected be
1313 formulated as additional rules and published as appropriate.

1314 [GNR1] UBL XML element, attribute and type names MUST be in the English 1315 language, using the primary English spellings provided in the Oxford English 1316 Dictionary.

1317 UBL fully supports the concepts of data standardization contained in ISO 11179. CCTS,
1318 as an implementation of 11179, furthers its basic tenets of data standardization into
1319 higher-level constructs as expressed by the CCTS dictionary entry names of those
1320 constructs – such as those for `ccts:BasicBusinessInformationEntities` and
1321 `ccts:AggregateBusinessInformationEntities`. Since UBL is an
1322 implementation of CCTS, UBL uses CCTS dictionary entry names as the basis for UBL
1323 XML schema construct names. UBL converts these `ccts:DictionaryEntryNames` into
1324 UBL XML schema construct names using strict transformation rules.

1325 [GNR2] UBL XML element, attribute and type names MUST be consistently derived
1326 from CCTS conformant dictionary entry names.

1327 The ISO 11179 specifies, and the CCTS uses, periods, spaces, other separators, and other
1328 characters not allowed by W3C XML. As such, these separators and characters are not
1329 appropriate for UBL XML component names.

1330 [GNR3] UBL XML element, attribute and type names constructed from
1331 `ccts:DictionaryEntryNames` MUST NOT include periods, spaces,
1332 other separators, or characters not allowed by W3C XML 1.0 for XML names.

1333 Acronyms and abbreviations impact on semantic interoperability and as such are to be
1334 avoided to the maximum extent practicable. Since some abbreviations will inevitably be
1335 necessary, UBL will maintain a normative list of authorized acronyms and abbreviations.
1336 Appendix B provides the current list of permissible acronyms, abbreviations and word
1337 truncations. The intent of this restriction is to facilitate the use of common semantics and
1338 greater understanding. Appendix B is a living document and will be updated to reflect
1339 growing requirements.

1340 [GNR4] UBL XML element, attribute, and simple and complex type names MUST
1341 NOT use acronyms, abbreviations, or other word truncations, except those in
1342 the list of exceptions published in Appendix B.

1343 UBL does not desire a proliferation of acronyms and abbreviations. Appendix B is an
1344 exception list and will be tightly controlled by UBL. Any additions will only occur after
1345 careful scrutiny to include assurance that any addition is critically necessary, and that any
1346 addition will not in any way create semantic ambiguity.

1347 [GNR5] Acronyms and abbreviations MUST only be added to the UBL approved
1348 acronym and abbreviation list after careful consideration for maximum
1349 understanding and reuse.

1350 Once an acronym or abbreviation has been approved, it is essential to ensuring semantic
1351 clarity and interoperability that the acronym or abbreviation is ***always*** used.

1352 [GNR6] The acronyms and abbreviations listed in Appendix B MUST always be used.

1353 Generally speaking the names for UBL XML constructs must always be singular, the
1354 only exception permissible is where the concept itself is pluralized.

1355 [GNR7] UBL XML element, attribute and type names MUST be in singular form
1356 unless the concept itself is plural.

1357 Example:
1358 Terms

1359 XML is case sensitive. Consistency in the use of case for a specific XML component
1360 (element, attribute, type) is essential to ensure every occurrence of a component is treated
1361 as the same. This is especially true in a business-based data-centric environment as is
1362 being addressed by UBL. Additionally, the use of visualization mechanisms such as
1363 capitalization techniques assist in ease of readability and ensure consistency in
1364 application and semantic clarity. The ebXML architecture document specifies a standard
1365 use of camel case for expressing XML elements and attributes.¹⁰ UBL will adhere to the
1366 ebXML standard. Specifically, UBL element and type names will be in UpperCamelCase
1367 (UCC).

1368 [GNR8] The UpperCamelCase (UCC) convention MUST be used for naming elements
1369 and types.

1370 Example:
1371
1372 CurrencyBaseRate
1373 CityNameType
1374

1375 UBL attribute names will be in lowerCamelCase (LCC).

1376 [GNR9] The lowerCamelCase (LCC) convention MUST be used for naming attributes.

1377 Example:
1378
1379 amountCurrencyCodeListVersionID
1380 characterSetCode

1381 4.2 Type Naming Rules

1382 UBL identifies several categories of naming rules for types, namely for complex types
1383 based on Aggregate Business Information Entities, Basic Business Information Entities,
1384 Primary Representation Terms, Secondary Representation Terms and the Core
1385 Component Type.

1386 Each of these ccts constructs have a `ccts:DictionaryEntryName` that is a fully
1387 qualified construct based on ISO 11179. As such, these names convey explicit semantic

¹⁰ *ebXML, ebXML Technical Architecture Specification v1.0.4, 16 February 2001*

1388 clarity with respect to the data being described. Accordingly, these `ccts:Dictionary`
 1389 `EntryNames` provide a mechanism for ensuring that UBL `xsd:complexType` names are
 1390 semantically unambiguous, and that there are no duplications of UBL type names for
 1391 different `xsd:type` constructs.

1392 4.2.1 Complex Type Names for CCTS Aggregate Business 1393 Information Entities

1394 UBL `xsd:complexType` names for `ccts:AggregateBusinessInformation`
 1395 `Entities` will be derived from their dictionary entry name by removing the object class
 1396 to follow truncation rules, removing separators to follow general naming rules, and
 1397 appending the suffix “Type”.

1398 [CTN1] A UBL `xsd:complexType` name based on an
 1399 `ccts:AggregateBusinessInformationEntity` MUST be the
 1400 `ccts:DictionaryEntryName` with the separators removed and with the
 1401 “Details” suffix replaced with “Type”.

1402 **Example:**

<code>ccts:AggregateBusiness InformationEntity</code>	<code>UBL xsd:complexType</code>
<code>Address. Details</code>	<code>AddressType</code>
<code>Financial Account. Details</code>	<code>FinancialAccountType</code>

1403

1404 4.2.2 Complex Type Names for CCTS Basic Business Information 1405 Entity Properties

1406 BBIE Properties are reusable across multiple BBIEs. CCTS does not specify, but implies,
 1407 that BBIE property names are the reusable property term and representation term of the
 1408 family of BBIEs that are based on it. The UBL `xsd:complexType` names for
 1409 `ccts:BasicBusinessInformationEntity` properties will be derived from the shared property
 1410 and representation terms portion of the dictionary entry names in which they appear by
 1411 removing separators to follow general naming rules, and appending the suffix “Type”.

1412 [CTN2] A UBL `xsd:complexType` name based on a
 1413 `ccts:BasicBusinessInformationEntityProperty` MUST be the
 1414 `ccts:DictionaryEntryName` shared property term and its qualifiers and
 1415 representation term of the shared `ccts:BasicBusinessInformation-`
 1416 `Entity`, with the separators removed and with the “Type” suffix appended
 1417 after the representation term.

1418 **Example:**

1419
1420
1421
1422
1423

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

1424

1425 4.2.3 Complex Type Names for CCTS Unspecialised Datatypes

1426 UBL `xsd:complexType` names for `ccts:UnspecialisedDatatypes` will be
1427 derived from its dictionary entry name by removing separators to follow general naming
1428 rules, and appending the suffix “Type”.

1429 [CTN3] A UBL `xsd:complexType` for a `cct:UnspecialisedDatatype` used in the
1430 UBL model MUST have the name of the corresponding
1431 `ccts:CoreComponentType`, with the separators removed and with the
1432 “Type” suffix appended.

1433 **Example:**

1434
1435
1436
1437

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

1438 UBL `xsd:complexType` names for `ccts:UnspecialisedDatatypes` based on
1439 `ccts:SecondaryRepresentationTerms` will be derived from the
1440 `ccts:SecondaryRepresentationTerm` dictionary entry name by removing separators to
1441 follow general naming rules, and appending the suffix “Type”.

1442 [CTN4] A UBL `xsd:complexType` for a `cct:UnspecialisedDatatype` based on
1443 a `ccts:SecondaryRepresentationTerm` used in the UBL model MUST
1444 have the name of the corresponding
1445 `ccts:SecondaryRepresentationTerm`, with the separators removed and
1446 with the “Type” suffix appended.

1447 **Example:**

1448
1449
1450
1451

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

1452 4.2.4 Complex Type Names for CCTS Core Component Types

1453 UBL `xsd:complexType` names for `ccts:CoreComponentTypes` will be derived
1454 from the dictionary entry name by removing separators to follow general naming rules,
1455 and appending the suffix “Type”.

1456 [CTN5] A UBL `xsd:complexType` name based on a `ccts:CoreComponentType`
1457 MUST be the Dictionary entry name of the `ccts:CoreComponentType`,
1458 with the separators removed.

1459 **Example:**

```
1460 <!-- ===== CCT: QuantityType ===== -->  
1461 <xsd:complexType name="QuantityType">  
1462     ...  
1463 </xsd:complexType>
```

1464 4.2.5 Simple Type Names for CCTS Core Component Types

1465 UBL `xsd:simpleType` names for `ccts:CoreComponentTypes` will be derived from
1466 the dictionary entry name by removing separators to follow general naming rules.

1467 [STN1] Each `ccts:CCT` `simpleType` definition name MUST be the `ccts:CCT`
1468 dictionary entry name with the separators removed

1469 4.3 Element Naming Rules

1470 As defined in the UBL Model (See Figure 2-3), UBL elements will be created for
1471 `ccts:AggregateBusinessInformationEntities`, `ccts:BasicBusinessInformationEntities`, and
1472 `ccts:AssociationBusinessInformationEntities`. UBL element names will reflect this
1473 relationship in full conformance with ISO11179 element naming rules.

1474 4.3.1 Element Names for CCTS Aggregate Business Information 1475 Entities

1476 [ELN1] A UBL global element name based on a `ccts:ABIE` MUST be the same as
1477 the name of the corresponding `xsd:complexType` to which it is bound,
1478 with the word "Type" removed.

1479 **Example:**

1480 For a `ccts:AggregateBusinessInformationEntity` of `Party`. Details,
1481 Rule CTN1 states that the `Party`. Details object class becomes `PartyType`
1482 `xsd:ComplexType`. Rule ELD3 states that for the `PartyType`
1483 `xsd:ComplexType`, a corresponding global element must be declared. Rule
1484 ELN1 states that the name of this corresponding global element must be `Party`.

```
1486 <xsd:element name="Party" type="PartyType"/>  
1487 <xsd:complexType name="PartyType">  
1488     <xsd:annotation>  
1489         <!--Documentation goes here--> </xsd:annotation>  
1490     <xsd:sequence>  
1491         <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0"  
1492             maxOccurs="1"/>  
1493     </xsd:sequence>  
1494 </xsd:complexType>  
1495  
1496
```

1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529

```
...
</xsd:element>
<xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0"
maxOccurs="1">
...
</xsd:element>
<xsd:element ref="PartyIdentification" minOccurs="0"
maxOccurs="unbounded">
...
</xsd:element>
<xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
...
</xsd:element>
<xsd:element ref="Address" minOccurs="0" maxOccurs="1">
...
</xsd:element>
...
</xsd:sequence>
```

1530 4.3.2 Element Names for CCTS Basic Business Information Entity 1531 Properties

1532 The same naming concept used for `ccts:AggregateBuinssInformationEntities`
1533 applies to `ccts:BasicBusinessInformationEntityProperty`

1534 [ELN2] A UBL global element name based on an unqualified `ccts:BBIEProperty`
1535 MUST be the same as the name of the corresponding `xsd:complexType` to
1536 which it is bound, with the word “Type” removed.

1537 Example:

1538
1539
1540
1541
1542
1543
1544
1545
1546

```
<!--==== Basic Business Information Entity Type Definitions =====>
->
<xsd:complexType name="ChargeIndicatorType">
...
</xsd:comlextType>
...
<!--==== Basic Business Information Entity Property Element
Declarations =====>
<xsd:element name="ChargeIndicator" type="ChargeIndicatorType"/>
```

1547 **4.3.3 Element Names for CCTS Association Business Information**
 1548 **Entities**

1549 A `ccts:AssociationBusinessInformationEntity` is not a class like
 1550 `ccts:AggregateBusinessInformationEntities` and like `ccts:Basic`
 1551 `BusinessInformationEntity` Properties that are reused as `ccts:Basic`
 1552 `BusinessInformationEntities`. Rather, it is an association between two classes.
 1553 As such, an element representing the `ccts:AssociationBusinessInformation`
 1554 `Entity` does not have its own unique `xsd:ComplexType`. Instead, when an element
 1555 representing a `ccts:AssociationBusinessInformationEntity` is declared, the
 1556 element is bound to the `xsd:complexType` of its associated `ccts:Aggregate`
 1557 `BusinessInformationEntity`.

1558 [ELN3] A UBL global element name based on a qualified `ccts:ASBIE` MUST be the
 1559 `ccts:ASBIE` dictionary entry name property term and its qualifiers; and the
 1560 object class term and qualifiers of its associated `ccts:ABIE`. All
 1561 `ccts:DictionaryEntryName` separators MUST be removed. Redundant
 1562 words in the `ccts:ASBIE` property term or its qualifiers and the associated
 1563 `ccts:ABIE` object class term or its qualifiers MUST be dropped.

1565 [ELN4] A UBL global element name based on a qualified `ccts:BBIEProperty` MUST
 1566 be the same as the name of the corresponding `xsd:complexType` to which it is
 1567 bound, with the qualifier prefixed and with the word "Type" removed.

1568 **4.4 Attribute Naming Rules**

1569 UBL, as a transactional based XML exchange format, has chosen to significantly restrict
 1570 the use of attributes. This restriction is in keeping with the fact that attribute usage is
 1571 relegated to supplementary components only; all "primary" business data appears
 1572 exclusively in element content.

1573 [ATN1] Each `CCT:SupplementaryComponent` `xsd:attribute` "name" MUST be the
 1574 Dictionary Entry Name object class, property term and representation term of
 1575 the `ccts:SupplementaryComponent` with the separators removed.

1576 Example:

<code>ccts:SupplementaryComponent</code>	<code>ubl:attribute</code>
<code>Amount Currency.Identifier</code>	<code>amountCurrencyID</code>
<code>Amount Currency. Code List Version.Identifier</code>	<code>amountCurrencyCodeListVersionID</code>
<code>Measure Unit.Code</code>	<code>measureUnitCode</code>

1577

1578 5 Declarations and Definitions

1579 In W3C XML Schema, elements are defined in terms of complex or simple types and
1580 attributes are defined in terms of simple types. The rules in this section govern the
1581 consistent structuring of these type constructs and the manner for unambiguously and
1582 thoroughly documenting them in the UBL Library.

1583 5.1 Type Definitions

1584 5.1.1 General Type Definitions

1585 Since UBL elements and types are intended to be reusable, all types must be named. This
1586 permits other types to establish elements that reference these types, and also supports the
1587 use of extensions for the purposes of versioning and customization.

1588 [GTD1] All types **MUST** be named.

1589 **Example:**

```
1590 <xsd:complexType name="QuantityType">  
1591     ...  
1592 </xsd:complexType>
```

1593 UBL disallows the use of `xsd:any`, because this feature permits the introduction of
1594 potentially unknown elements into an XML instance. UBL intends that all constructs
1595 within the instance be described by the schemas describing that instance - `xsd:any` is seen
1596 as working counter to the requirements of interoperability.

1597 [GTD2] The `xsd:any` Type **MUST NOT** be used.

1598 5.1.2 Simple Types

1599 The Core Components Specification provides a set of constructs for the modeling of
1600 basic data, Core Component Types. These are represented in UBL with a library of
1601 complex types, with the effect that most "simple" data is represented as property sets
1602 defined according to the CCTs, made up of content components and supplementary
1603 components. In most cases, the supplementary components are expressed as XML
1604 attributes, the content component becomes element content, and the CCT is represented
1605 with an `xsd:complexType`. There are exceptions to this rule in those cases where all of a
1606 CCTs properties can be expressed without the use of attributes. In these cases, an
1607 `xsd:simpleType` is used.

1608 [STD1] For every `ccts:CCT` whose supplementary components map directly onto the
1609 properties of a built-in `xsd:Datatype`, the `ccts:CCT` **MUST** be defined as
1610 a named `xsd:simpleType` in the `ccts:CCT` schema module.

1611 **Example:**

1612
1613
1614
1615
1616

```
<!-- ===== CCT: DateTimeType ===== -->  
<xsd:simpleType name="DateTimeType">  
  ...  
  <xsd:restriction base="cct:DateTimeType"/>  
</xsd:simpleType>
```

1617 5.1.3 Complex Types

1618 Since even simple Datatypes are modeled as property sets in most cases, the XML
1619 expression of these models primarily employs `xsd:complexType`. To facilitate reuse,
1620 versioning, and customization, all complex types are named. The main exception to this
1621 form of representation concerns Aggregate Business Information Entities, which
1622 represent the relationship between an aggregate “parent” object and its aggregate
1623 properties, or children. Given the object based concepts defined in `ccts:corecomponents`,
1624 `ccts:AggregateBusinessInformationEntities` and `cct:Basic`
1625 `BusinessInformationEntityProperties` are considered classes(objects) in the
1626 UBL model.

1627 [CTD1] For every class identified in the UBL model, a named `xsd:complexType`
1628 MUST be defined.

1629 Example:

1630
1631
1632
1633
1634

```
<xsd:complexType name="BuildingNameType">  
  
</xsd:complexType>
```

1635 5.1.3.1 Aggregate Business Information Entities

1636 The relationship expressed by an Aggregate Business Information Entity is not directly
1637 represented with a class. Instead, this relationship is captured in UBL with a containment
1638 relationship, expressed in the content model of the parent object’s type with a sequence
1639 of elements. (Sequence facilitates the use of `xsd:extension` for versioning and
1640 customization.) The members of the sequence – elements which are themselves defined
1641 by reference to complex types – are the properties of the containing type.

1642 [CTD2] Every `ccts:ABIE` `xsd:complexType` definition content model MUST
1643 use the `xsd:sequence` element with appropriate global element references,
1644 or local element declarations in the case of ID and Code, to reflect each
1645 property of its class as defined in the corresponding UBL model.

1646 Example:

1647
1648
1649
1650
1651
1652
1653
1654
1655

```
<xsd:complexType name="AddressType">  
  ...  
  <xsd:sequence>  
    <xsd:element ref="cbc:CityName" minOccurs="0" maxOccurs="1">  
  ...
```

1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667

```
</xsd:element>
<xsd:element ref="cbc:PostalZone" minOccurs="0" maxOccurs="1">
  ...
</xsd:element>
...
</xsd:sequence>
</xsd:complexType>
```

1668 5.1.3.2 Basic Business Information Entities

1669 Basic Business Information Entities (BBIEs), in accordance with the Core Components
1670 Technical Specification, always have a primary representation term, and may have
1671 secondary representation terms, which describes their structural representation. These
1672 representation terms are expressed in the UBL Model as Unspecialised Datatypes bound
1673 to a Core Component Type that describes their structure. In addition to the unspecialised
1674 Datatypes defined in CCTS, UBL has defined a set of specialised Datatypes that are
1675 derived from the CCTS unqualified Datatypes. There are a set of rules concerning the way
1676 these relationships are expressed in the UBL XML library. As discussed above, BBIE
1677 properties are represented with complex types. Within these are simpleContent elements
1678 that extend the Datatypes.

1679 [CTD3] Every `ccts:BBIEProperty` `xsd:complexType` definition content
1680 model MUST use the `xsd:simpleContent` element.

1681

1682 [CTD4] Every `ccts:BBIEProperty` `ComplexType` content model
1683 `xsd:simpleContent` element MUST consist of an `xsd:extension`
1684 element.

1685

1686 [CTD5] Every `ccts:BBIEProperty` `xsd:complexType` content model `xsd:base`
1687 attribute value MUST be the `ccts:CCT` of the unspecialised or specialised
1688 UBL Datatype as appropriate.

1689 **Example:**

1690
1691
1692
1693
1694

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

1695 5.1.3.3 Datatypes

1696 There is a direct one-to-one relationship between `ccts:CoreComponentTypes` and
1697 `ccts:PrimaryRepresentationTerms`. Additionally, there are several

1698 `ccts:SecondaryRepresentationTerms` that are subsets of their parent
1699 `ccts:PrimaryRepresentationTerm`. The total set of
1700 `ccts:RepresentationTerms` by their nature represent `ccts:Datatypes`.
1701 Specifically, for each `ccts:PrimaryRepresentationTerm` or
1702 `ccts:SecondaryRepresentationTerm`, a `ccts:UnspecialisedDatatype` exists.
1703 In the UBL XML Library, these `ccts:UnspecialisedDatatypes` are expressed as
1704 complex or simple types that are of the type of its corresponding
1705 `ccts:CoreComponentType`.

1706 [CTD6] For every Datatype used in the UBL model, a named `xsd:complexType` or
1707 `xsd:simpleType` MUST be defined.

1708 *5.1.3.3.1 Unspecialised Datatypes*

1709 The `ccts:UnspecialisedDatatypes` reflect the instantiation of the `ccts:Core`
1710 `ComponentTypes`. Each `ccts:UnspecialisedDatatype` declaration is
1711 based on its corresponding qualified `ccts:CoreComponentType` and
1712 represents either a primary or secondary representation term.

1713 [CTD7] Every unspecialised Datatype must be based on a `ccts:CCT` represented in the
1714 CCT schema module, and must represent an approved primary or secondary
1715 representation term identified in the CCTS.

1716 [CTD8] Each unspecialised Datatype `xsd:complexType` must be based on its
1717 corresponding CCT `xsd:complexType`.

1718 [CTD9] Every unspecialised Datatype that represents a primary representation term
1719 whose corresponding `ccts:CCT` is defined as an `xsd:simpleType` MUST also
1720 be defined as an `xsd:simpleType` and MUST be based on the same
1721 `xsd:simpleType`.

1722 [CTD10] Every unspecialised Datatype that represents a secondary representation term
1723 whose corresponding `ccts:CCT` is defined as an `xsd:simpleType` MUST also
1724 be defined as an `xsd:simpleType` and MUST be based on the same
1725 `xsd:simpleType`.

1726 [CTD11] Each unspecialised Datatype `xsd:complexType` definition must contain one
1727 `xsd:simpleContent` element.

1728 [CTD12] The unspecialised Primary Representation Term Datatype `xsd:complexType`
1729 definition `xsd:simpleContent` element must contain one `xsd:restriction`
1730 element with an `xsd:base` attribute whose value is equal to the corresponding
1731 `cct:complexType`

1732 *5.1.3.4 Core Component Types*

1733 A CCT consists of a “content component” which may be supported by a set of properties
1734 referred to as “supplementary components”. CCTs may be expressed as a simple type
1735 (where possible), but may require expression as a complex type. Content components are

1736 expressed as extensions of the set of built-in xsd Datatypes. Supplementary components
1737 are expressed either as extensions of built-in Datatypes, or user-defined simple types.

1738 [CTD13] For every `ccts:CCT` whose supplementary components are not equivalent to
1739 the properties of a built-in `xsd:Datatype`, the `ccts:CCT` MUST be defined
1740 as a named `xsd:complexType` in the `ccts:CCT` schema module.

1741 CCTs complex types always have `xsd:simpleContent`, which is an extension of a built-in
1742 `xsd Datatype`.

1743 [CTD14] Each `ccts:CCT xsd:complexType` definition MUST contain one
1744 `xsd:simpleContent` element

1745
1746 [CTD15] The `ccts:CCT xsd:complexType` definition `xsd:simpleContent`
1747 element MUST contain one `xsd:extension` element. This
1748 `xsd:extension` element MUST include an `xsd:base` attribute that
1749 defines the specific `xsd:built-in Datatype` required for the
1750 `ccts:ContentComponent` of the `ccts:CCT`.

1751 **Example:**

```
1752 <xsd:complexType name="QuantityType">
1753     ...
1754     <xsd:simpleContent>
1755         <xsd:extension base="xsd:decimal">
1756             <xsd:attribute name="quantityUnitCode" type="xsd:normalizedString"
1757             use="optional"/>
1758             <xsd:attribute name="quantityUnitCodeListID"
1759             type="xsd:normalizedString" use="optional"/>
1760             <xsd:attribute name="quantityUnitCodeListAgencyID"
1761             type="xsd:normalizedString" use="optional"/>
1762             <xsd:attribute name="quantityUnitCodeListAgencyName"
1763             type="xsd:string" use="optional"/>
1764         </xsd:extension>
1765     </xsd:simpleContent>
1766 </xsd:complexType>
```

1778 5.1.3.5 Supplementary Components

1779 Supplementary components are expressed with references to either built-in xsd
1780 Datatypes, or to user-defined simple types.

1781 [CTD16] Each `CCT:SupplementaryComponent xsd:attribute` "type" MUST
1782 define the specific `xsd:built-in Datatype` or the user defined

1783
1784

`xsd:simpleType` for the `ccts:SupplementaryComponent` of the `ccts:CCT`.

1785 **Example:**

1786
1787
1788

```
<xsd:attribute name="measureUnitCode" type="xsd:normalizedString"
use="required"/>
```

1789
1790
1791
1792

[CTD17] Each `ccts:SupplementaryComponent` `xsd:attribute` user-defined `xsd:simpleType` MUST only be used when the `ccts:SupplementaryComponent` is based on a standardized code list for which a UBL conformant code list schema module has been created.

1793
1794
1795

[CTD18] Each `ccts:SupplementaryComponent` `xsd:attribute` user defined `xsd:simpleType` MUST be the same `xsd:simpleType` from the appropriate UBL conformant code list schema module for that type.

1796
1797

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

1798
1799
1800

[CTD19] Each `ccts:SupplementaryComponent` `xsd:attribute` “use” MUST define the occurrence of that `ccts:SupplementaryComponent` as either “required”, or “optional”.

1801 **Example:**

1802
1803
1804
1805
1806

```
<xsd:attribute name="amountCurrencyID" type="xsd:normalizedString"
use="required"/>

<xsd:attribute name="amountCurrencyCodeListVersionID"
type="xsd:normalizedString" use="optional"/>
```

1807 5.2 Element Declarations

1808 5.2.1 General Element Declarations

1809 5.2.2 Elements Bound to Complex Types

1810 The binding of UBL elements to their `xsd:complexType`s is based on the associations
1811 identified in the UBL model. For the `ccts:BasicBusinessInformationEntities`
1812 and `ccts:AggregateInformationEntities`, the UBL elements will be directly
1813 associated to its corresponding `xsd:complexType`.

1814
1815

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

1816 **Example:**

1817 For the Party. Details object class, a complex type/global element declaration
1818 pair is created through the declaration of a Party element that is of type
1819 PartyType.

1820 The element thus created is useful for reuse in the building of new business messages.
1821 The complex type thus created is useful for both reuse and customization, in the building
1822 of both new and contextualized business messages.

1823 **Example:**

```
1824 <xsd:element name="BuyerParty" type="BuyerPartyType" />  
1825 <xsd:complexType name="BuyerPartyType">  
1826     ...  
1827 </xsd:complexType>
```

1828 **5.2.2.6 Elements Representing ASBIEs**

1829 A `ccts:AssociationBusinessInformationEntity` is not a class like
1830 `ccts:AggregateBusinessInformationEntities` and `ccts:BasicBusiness`
1831 `InformationEntities` are. Rather, it is an association between two classes. As such,
1832 the element declaration will reference the `xsd:complexType` of the associated
1833 `ccts:AggregateBusinessInformationEntity`. There are two types of ASBIEs – those that
1834 have qualifiers in the object class, and those that do not.

1835 [ELD4] When a `ccts:ASBIE` is unqualified, it is bound via reference to the global
1836 `ccts:ABIE` element to which it is associated. When an `ccts:ABIE` is
1837 qualified, a new element **MUST** be declared and bound to the
1838 `xsd:complexType` of its associated
1839 `ccts:AggregateBusinessInformationEntity`.

1840 **5.2.2.7 Elements Bound to Core Component Types**

1841 [ELD5] For each `ccts:CCT simpleType`, an `xsd:restriction` element
1842 **MUST** be declared.

1843 **5.2.3 Code List Import**

1844 [ELD6] The code list `xsd:import` element **MUST** contain the namespace and
1845 schema location attributes.

1846 **5.2.4 Empty Elements**

1847 [ELD7] Empty elements **MUST** not be declared.

1848 **5.2.5 Global Elements**

1849 [ELD8] Global elements declared for Qualified BBIE Properties must be of the same
1850 type as its corresponding Unqualified BBIE Property. (i.e. Property Term +
1851 Representation Term.)

1852 **Example:**

1853 `<xsd:element name="AdditionalStreetName" type="cbc:StreetNameType"/>`

1854 5.2.6 XSD:Any

1855 [ELD9] The `xsd:any` element MUST NOT be used.

1856 5.3 Attribute Declarations

1857 Attributes are W3C Schema constructs associated with elements that provide further
1858 information regarding elements. While elements can be thought of as containing data,
1859 attributes can be thought of as containing metadata. Unlike elements, attributes cannot be
1860 nested within each other—there are no “subattributes.” Therefore, attributes cannot be
1861 extended as elements can. Attribute order is not enforced by XML processors—that is, if
1862 the attribute order in an XML instance document is different than the order in which the
1863 attributes are declared in the schema to which the XML instance document conforms, no
1864 error will result. UBL has determined that these limitations dictate that UBL restrict the
1865 use of attributes to either XSD built-in attributes, or to Supplementary Components
1866 which by their nature within the CCTS metamodel only carry metadata.

1867 5.3.1 User Defined Attributes

1868 [ATD1] User defined attributes SHOULD NOT be used. When used, user defined
1869 attributes MUST only convey `CCT:SupplementaryComponent`
1870 information.

1871

1872 [ATD2] The `CCT:SupplementaryComponents` for the ID `CCT:CoreComponent` MUST
1873 be declared in the following order:

1874 Identifier. Content

1875 Identification Scheme. Identifier

1876 Identification Scheme. Name. Text

1877 Identification Scheme. Agency. Identifier

1878 Identification Scheme. Agency Name. Text

1879 Identification Scheme. Version. Identifier

1880 Identification Scheme. Uniform Resource. Identifier

1881 Identification Scheme Data. Uniform Resource. Identifier

1882 5.3.2 Global Attributes

1883 Rule ATD1 limits the use of attributes to `cct:SupplementaryComponents`. The current
1884 UBL library does not contain any attributes that are common to all UBL elements,

1885 however such a situation may arise in the future. If such common attributes are defined,
1886 then they will be declared using the `xsd:globalattribute` element using the
1887 following rules.

1888 [ATD3] If a UBL `xsd:SchemaExpression` contains one or more common
1889 attributes that apply to all UBL elements contained or included or imported
1890 therein, the common attributes MUST be declared as part of a global attribute
1891 group.

1892

1893 5.3.3 Supplementary Components

1894 [ATD4] Within the `ccts:CCT` `xsd:extension` element an `xsd:attribute`
1895 MUST be declared for each `ccts:SupplementaryComponent` pertaining
1896 to that `ccts:CCT`.

1897

1898 [ATD5] For each `ccts:CCT` `simpleType` `xsd:Restriction` element, an
1899 `xsd:base` attribute MUST be declared and set to the appropriate
1900 `xsd:Datatype`.

1901 5.3.4 DatatypeSchema Location

1902 UBL is an international standard that will be used in perpetuity by companies around the
1903 globe. It is important that these users have unfettered access to all UBL schema.

1904 [ATD6] Each `xsd:schemaLocation` attribute declaration MUST contain a system-
1905 resolvable URL, which at the time of release from OASIS shall be a relative
1906 URL referencing the location of the schema or schema module in the release
1907 package.

1908 5.3.5 XSD:nil

1909 [ATD7] The `xsd` built in nillable attribute MUST NOT be used for any UBL declared
1910 element.

1911 5.3.6 XSD:Any

1912 [ATD8] The `xsd:any` attribute MUST NOT be used.

1913

6 Code Lists

1914 UBL has determined that the best approach for code lists is to handle them as schema
1915 modules. In recognition of the fact that most code lists are maintained by external
1916 agencies, UBL has determined that if code list owners all used the same normative form
1917 schema module, all users of those code lists could avoid a significant level of code list
1918 maintenance. By having each code list owner develop, maintain, and make available via
1919 the internet their code lists using the same normative form schema, code list users would
1920 be spared the unnecessary and duplicative efforts required for incorporation in the form
1921 of enumeration of such code lists into Schema, and would subsequently avoid the
1922 maintenance of such enumerations since code lists are handled as imported schema
1923 modules rather than cumbersome enumerations. To make this mechanism operational,
1924 UBL has defined a number of rules. To avoid enumeration of codes in the document or
1925 reusable schemas, UBL has determined that:

1926 [CDL1] All UBL Codes MUST be part of a UBL or externally maintained Code List.

1927 Because the majority of code lists are owned and maintained by external agencies, UBL
1928 will make maximum use of such external code lists where they exist.

1929 [CDL2] The UBL Library SHOULD identify and use external standardized code lists
1930 rather than develop its own UBL-native code lists.

1931 In some cases the UBL Library may extend an existing code list to meet specific business
1932 requirements. In others cases the UBL Library may have to create and maintain a code
1933 list where a suitable code list does not exist in the public domain. Both of these type of
1934 code lists would be considered UBL-internal code lists.

1935 [CDL3] The UBL Library MAY design and use an internal code list where an existing
1936 external code list needs to be extended, or where no suitable external code list
1937 exists.

1938 UBL-internal code lists will be designed with maximum re-use in mind to facilitate
1939 maximum use by others.

1940 If a UBL code list is created, the lists should be globally scoped (designed for reuse and
1941 sharing, using named types and namespaced Schema Modules) rather than locally scoped
1942 (not designed for others to use and therefore hidden from their use).

1943 To guarantee consistency within all code list schema modules all ubl-internal code lists
1944 and externally used code lists will use the UBL Code List Schema Module. This schema
1945 module will contain an enumeration of code list values.

1946 [CDL4] All UBL maintained or used Code Lists MUST be enumerated using the UBL
1947 Code List Schema Module.

1948 To guarantee consistency of code list schema module naming, the name of each UBL
1949 Code List Schema Module will adhere to a prescribed form.

1950 [CDL5] The name of each UBL Code List Schema Module MUST be of the form:
1951 {Owning Organization}{Code List Name}{Code List Schema Module}

1952 Each code list used in the UBL schema MUST be imported individually.

1953 [CDL6] An `xsd:Import` element MUST be declared for every code list required in a
1954 UBL schema.

1955 The UBL library allows partial implementations of code lists which may required by
1956 customizers.

1957 [CDL7] Users of the UBL Library MAY identify any subset they wish from an
1958 identified code list for their own trading community conformance
1959 requirements.

1960 The following rule describes the requirements for the `xsd:schemaLocation` for the
1961 importation of the code lists into a UBL business document.

1962 [CDL8] The `xsd:schemaLocation` MUST include the complete URI used to identify
1963 the relevant code list schema.

1964

1965 7 Miscellaneous XSD Rules

1966 UBL, as a business standard vocabulary, requires consistency in its development. The
1967 number of UBL Schema developers will expand over time. To ensure consistency, it is
1968 necessary to address the optional features in XSD that are not addressed elsewhere.

1969 7.1 XSD Simple Types

1970 UBL guiding principles require maximum reuse. XSD provides for forty four built-in
1971 Datatypes expressed as simple types. In keeping with the maximize re-use guiding
1972 principle, these built-in `xsd:SimpleTypes` should be used wherever possible.

1973 [GXS3] Built-in XSD Simple Types SHOULD be used wherever possible.

1974 7.2 Namespace Declaration

1975 The W3C XSD specification allows for the use of any token to represent its location. To
1976 ensure consistency, UBL has adopted the generally accepted convention of using the
1977 “xsd” token for all UBL schema and schema modules.

1978 [GXS4] All W3C XML Schema constructs in UBL Schema and schema modules
1979 MUST contain the following namespace declaration on the `xsd` schema
1980 element:

1981 `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

1982 7.3 XSD:Substitution Groups

1983 The `xsd:SubstitutionGroups` feature enables a type definition to identify substitution
1984 elements in a group. Although a useful feature in document centric XML applications,
1985 this feature is not used by UBL.

1986 [GXS5] The `xsd:SubstitutionGroups` feature MUST NOT be used.

1987 7.4 XSD:Final

1988 [GXS6] The `xsd:final` attribute MUST be used to control extensions.

1989 7.5 XSD: Notation

1990 The `xsd:notation` attribute identifies a notation. Notation declarations corresponding to all
1991 the `<notation>` element information items in the [children], if any, plus any included or
1992 imported declarations. Per XSD Part 2, “It is an **error** for **NOTATION** to be used
1993 directly in a schema. Only Datatypes that are **derived** from **NOTATION** by specifying

1994 a value for **enumeration** can be used in a schema.” The UBL schema model does not
1995 require or support the use of this feature.

1996 [GXS7] `xsd:notation` MUST NOT be used.

1997 7.6 XSD:All

1998 The `xsd:all` compositor requires occurrence indicators of `minOccurs = 0` and `maxOccurs`
1999 `= 1`. The `xsd:all` compositor allows for elements to occur in any order. The result is that in
2000 an instance document, elements can occur in any order, are always optional, and never
2001 occur more than once. Such restrictions are inconsistent with data-centric scenarios such
2002 as UBL.

2003 [GXS8] The `xsd:all` element MUST NOT be used.

2004 7.7 XSD:Choice

2005 The `xsd:choice` compositor allows for any element declared inside it to occur in the
2006 instance document, but only one. As with the `xsd:all` compositor, this feature is
2007 inconsistent with business transaction exchanges and is not allowed in UBL. While
2008 `xsd:choice` is a very useful construct in situations where customisation and extensibility
2009 are not a concern, UBL does not use it because `xsd:choice` cannot be extended.

2010 [GXS9] The `xsd:choice` element SHOULD NOT be used where customisation and
2011 extensibility are a concern.

2012 7.8 XSD:Include

2013 The `xsd:include` feature provides a mechanism for bringing in schemas that reside in the
2014 same namespace. UBL employs multiple schema modules within a namespace. To avoid
2015 circular references, this feature will not be used except by the document schema.

2016 [GXS10] The `xsd:include` feature MUST only be used within a document schema.

2017 7.9 XSD:Union

2018 The `xsd:union` feature provides a mechanism whereby a Datatype is created as a
2019 union of two or more existing Datatypes. With UBL’s strict adherence to the use of
2020 `ccts:Datatypes` that are explicitly declared in the UBL library, this feature is inappropriate
2021 except for codelists. In some cases external customizers may choose to use this technique
2022 for Codelists and as such the use of the union technique may prove beneficial for
2023 customizers.

2024 [GXS11] The `xsd:union` technique MUST NOT be used except for Code Lists. The
2025 `xsd:union` technique MAY be used for Code Lists.

2026 7.10 XSD:Appinfo

2027 The `xsd:appinfo` feature is used by schema to convey processing instructions to a
2028 processing application, Stylesheet, or other tool. Some users of UBL have determined
2029 that this technique poses a security risk and have employed techniques for stripping
2030 `xsd:appinfo` from schemas. As UBL is committed to ensuring the widest possible
2031 target audience for its XML library, this feature is not used – except to convey non-
2032 normative information.

2033 [GXS12] UBL designed schema SHOULD NOT use `xsd:appinfo`. If used,
2034 `xsd:appinfo` MUST only be used to convey non-normative information.

2035 7.11 Extension and Restriction

2036 UBL fully recognizes the value of supporting extension and restriction of its core library
2037 by customizers.

2038 [GXS13] Complex Type extension or restriction MAY be used where appropriate.

2039 8 Instance Documents

2040 Consistency in UBL instance documents is essential in a trade environment. UBL has
2041 defined several rules to help affect this consistency.

2042 8.1 Root Element

2043 UBL has chosen a global element approach. In XSD, every global element is eligible to
2044 act as a root element in an instance document. Rule ELD1 requires the identification of a
2045 single global element in each UBL schema to be carried as the root element in the
2046 instance document. UBL business documents (UBL instances) must have a single root
2047 element as defined in the corresponding UBL XSD.

2048 [RED1] Every UBL instance document must use the global element defined as the root
2049 element in the schema as its root element.

2050 8.2 Validation

2051 The UBL library and supporting schema are targeted at supporting business information
2052 exchanges. Business information exchanges require a high degree of precision to ensure
2053 that application processing and corresponding business cycle actions are reflective of the
2054 purpose, intent, and information content agreed to by both trading partners. Schemas
2055 provide the necessary mechanism for ensuring that instance documents do in fact support
2056 these requirements.

2057 [IND1] All UBL instance documents MUST validate to a corresponding schema.

2058 8.3 Character Encoding

2059 XML supports a wide variety of character encodings. Processors must understand which
2060 character encoding is employed in each XML document. XML 1.0 supports a default
2061 value of UTF-8 for character encoding, but best practice is to always identify the
2062 character encoding being employed.

2063 [IND2] All UBL instance documents MUST always identify their character encoding
2064 with the XML declaration.

2065 Example:

2066

2067

Xml expression: UTF-8

2068 UBL, as an OASIS TC, is obligated to conform to agreements OASIS has entered into.
2069 OASIS is a liaison member of the ISO/IETF/ITU/UNCEFACT Memorandum of
2070 Understanding Management Group (MOUMG). Resolution 01/08 (MOU/MG01n83)
2071 requires the use of UTF-8.

2072 [IND3] In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of
2073 Understanding Management Group (MOUMG) Resolution 01/08
2074 (MOU/MG01n83) as agreed to by OASIS, all UBL XML SHOULD be
2075 expressed using UTF-8.

2076 Example:
2077
2078 <?xml version="1.0" encoding="UTF-8" ?>
2079

2080 8.4 Schema Instance Namespace Declaration

2081 [IND4] All UBL instance documents MUST contain the following namespace
2082 declaration in the root element:
2083 `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`

2084 8.5 Empty Content.

2085 Usage of empty elements within XML instance documents are a source of controversy
2086 for a variety of reasons. An empty element does not simply represent data that is missing.
2087 It may express data that is not applicable for some reason, trigger the expression of an
2088 attribute, denote all possible values instead of just one, mark the end of a series of data, or
2089 appear as a result of an error in XML file generation. Conversely, missing data elements
2090 can also have meaning - data not provided by a trading partner. In information exchange
2091 environments, different Trading Partners may allow, require or ban empty elements. UBL
2092 has determined that empty elements do not provide the level of assurance necessary for
2093 business information exchanges and as such will not be used.

2094 [IND5] UBL conformant instance documents MUST NOT contain an element devoid
2095 of content or null values.

2096 To ensure that no attempt is made to circumvent rule IND5, UBL also prohibits
2097 attempting to convey meaning by not conveying an element.

2098 [IND6] The absence of a construct or data in a UBL instance document MUST NOT
2099 carry meaning.

2100

2101 **Appendix A. UBL NDR Checklist**

2102 The following checklist constitutes all UBL XML naming and design rules as defined in
2103 *UBL Naming and Design Rules version 1.0*, xx November 2003. The checklist is in
2104 alphabetical sequence as follows:

- 2105 Attribute Declaration Rules (ATD)
- 2106 Attribute Naming Rules (ATN)
- 2107 Code List Rules (CDL)
- 2108 ComplexType Definition Rules (CTD)
- 2109 ComplexType Naming Rules (CTN)
- 2110 Documentation Rules (DOC0)
- 2111 Element Declaration Rules (ELD)
- 2112 General Naming Rules (GNR)
- 2113 General Type Definition Rules (GTD)
- 2114 General XML Schema Rules (GXS)
- 2115 Instance Document Rules (IND)
- 2116 Modeling Constraints Rules (MDC)
- 2117 Naming Constraints Rules (NMC)
- 2118 Namespace Rules (NMS)
- 2119 Root Element Declaration Rules (RED)
- 2120 Schema Structure Modularity Rules (SSM)
- 2121 Standards Adherence Rules (STA)
- 2122 SimpleType Naming Rules (STN)
- 2123 SimpleType Definition Rules (STD)
- 2124 Versioning Rules (VER)
- 2125

A.1 Attribute Declaration Rules

[ATD1]	User defined attributes SHOULD NOT be used. When used, user defined attributes MUST only convey CCT:SupplementaryComponent information.
[ATD2]	The CCT:SupplementaryComponents for the ID CCT:CoreComponent MUST be declared in the following order: Identifier. Content Identification Scheme. Identifier Identification Scheme. Name. Text Identification Scheme. Agency. Identifier Identification Scheme. Agency Name. Text Identification Scheme. Version. Identifier Identification Scheme. Uniform Resource. Identifier Identification Scheme Data. Uniform Resource. Identifier
[ATD3]	If a UBL xsd:SchemaExpression 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.
[ATD4]	Within the ccts:CCT xsd:extension element an xsd:attribute MUST be declared for each ccts:SupplementaryComponent pertaining to that ccts:CCT.
[ATD5]	For each ccts:CCT simpleType xsd:Restriction element, an xsd:base attribute MUST be declared and set to the appropriate xsd:datatype.
[ATD6]	Each xsd:schemaLocation attribute declaration MUST contain a system-resolvable URL, which at the time of release from OASIS shall be a relative URL referencing the location of the schema or schema module in the release package.
[ATD7]	The xsd built in nillable attribute MUST NOT be used for any UBL declared element.

[ATD8]	The xsd:any attribute MUST NOT be used.
--------	---

2126

A.2 Attribute Naming Rules	
[ATN1]	Each CCT:SupplementaryComponent xsd:attribute "name" MUST be the dictionary entry name object class, property term and representation term of the ccts:SupplementaryComponent with the separators removed.

2127

A.3 Code List Rules	
[CDL1]	All UBL Codes MUST be part of a UBL or externally 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]	All UBL maintained or used Code Lists MUST be enumerated using the UBL Code List Schema Module.
[CDL5]	The name of each UBL Code List Schema Module MUST be of the form: {Owning Organization}{Code List Name}{Code List Schema Module}
[CDL6]	An xsd:Import element MUST be declared for every code list required in a UBL schema.
[CDL7]	Users of the UBL Library MAY identify any subset they wish from an identified code list for their own trading community conformance requirements.
[CDL8]	The xsd:schemaLocation MUST include the complete URI used to identify the relevant code list schema.

<h2>A.4 ComplexType Definition Rules</h2>	
[CTD1]	For every class identified in the UBL model, a named xsd:complexType MUST be defined.
[CTD2]	Every ccts:ABIE xsd:complexType definition content model MUST use the xsd:sequence 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.
[CTD3]	Every ccts:BBIEProperty xsd:complexType definition content model MUST use the xsd:simpleContent element.
[CTD4]	Every ccts:BBIEProperty ComplexType content model xsd:simpleContent element MUST consist of an xsd:extension element.
[CTD5]	Every ccts:BBIEProperty xsd:complexType content model xsd:base attribute value MUST be the ccts:CCT of the unspecialised or specialised UBL datatype as appropriate.
[CTD6]	For every datatype used in the UBL model, a named xsd:complexType or xsd:simpleType MUST be defined.
[CTD7]	Every unspecialised Datatype must be based on a ccts:CCT represented in the CCT schema module and must represent an approved primary or secondary representation term identified in the CCTS.
[CTD8]	Each unspecialised Datatype xsd:complexType must be based on its corresponding CCT xsd:complexType.
[CTD9]	Every unspecialised Datatype that represents a primary representation term whose corresponding ccts:CCT is defined as an xsd:simpleType MUST also be defined as an xsd:simpleType and MUST be based on the same xsd:simpleType.
[CTD10]	Every unspecialised Datatype that represents a secondary representation term whose corresponding ccts:CCT is defined as an xsd:simpleType MUST also be defined as an xsd:simpleType and MUST be based on the same xsd:simpleType.

A.4 ComplexType Definition Rules

[CTD11]	Each unspecialised Datatype <code>xsd:complexType</code> definition must contain one <code>xsd:simpleContent</code> element.
[CTD12]	The unspecialised Primary Representation Term Datatype <code>xsd:complexType</code> definition <code>xsd:simpleContent</code> element must contain one <code>xsd:restriction</code> element with an <code>xsd:base</code> attribute whose value is equal to the corresponding <code>cct:complexType</code> .
[CTD13]	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.
[CTD14]	Each <code>ccts:CCT</code> <code>xsd:complexType</code> definition MUST contain one <code>xsd:simpleContent</code> element
[CTD15]	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> .
[CTD16]	Each <code>CCT:SupplementaryComponent</code> <code>xsd:attribute</code> "type" MUST define the specific <code>xsd:built-in</code> Datatype or the user defined <code>xsd:simpleType</code> for the <code>ccts:SupplementaryComponent</code> of the <code>ccts:CCT</code> .
[CTD17]	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.
[CTD18]	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.
[CTD19]	Each <code>ccts:Supplementary Component</code> <code>xsd:attribute</code> "use" MUST define the occurrence of that <code>ccts:SupplementaryComponent</code> as either "required", or "optional".

A.5 ComplexType Naming Rules

[CTN1]	A UBL xsd:complexType name based on an ccts:AggregateBusinessInformationEntity MUST be the ccts:DictionaryEntryName with the separators removed and with the "Details" suffix replaced with "Type".
[CTN2]	A UBL xsd:complexType name based on a ccts:BasicBusinessInformationEntityProperty MUST be the ccts:DictionaryEntryName shared property term and its qualifiers and the representation term of the shared ccts:BasicBusinessInformationEntity, with the separators removed and with the "Type" suffix appended after the representation term.
[CTN3]	A UBL xsd:complexType for a cct:UnspecialisedDatatype used in the UBL model MUST have the name of the corresponding ccts:CoreComponentType, with the separators removed and with the "Type" suffix appended.
[CTN4]	A UBL xsd:complexType for a cct:UnspecialisedDatatype based on a ccts:SecondaryRepresentationTerm used in the UBL model MUST have the name of the corresponding ccts:SecondaryRepresentationTerm, with the separators removed and with the "Type" suffix appended.
[CTN5]	A UBL xsd:complexType name based on a ccts:CoreComponentType MUST be the Dictionary entry name of the ccts:CoreComponentType, with the separators removed.

2130

A.6 Documentation Rules

[DOC1]	<p>The xsd:documentation element for every Datatype MUST contain a structured set of annotations in the following sequence and pattern:</p> <ul style="list-style-type: none">• ComponentType (mandatory): The type of component to which the object belongs. For Datatypes this must be “DT”.• DictionaryEntryName (mandatory): The official name of a Datatype.• Version (optional): An indication of the evolution over time of the Datatype.• Definition(mandatory): The semantic meaning of a Datatype.• ObjectClassQualifier (optional): The qualifier for the object class.• ObjectClass(optional): The Object Class represented by the Datatype.• RepresentationTerm (mandatory): A Representation Term is an element of the name which describes the form in which the property is represented.• DataTypeQualifier (optional): semantically meaningful name that differentiates the Datatype from its underlying Core Component Type.• DataType (optional): Defines the underlying Core Component Type.
[DOC2]	<p>A Datatype definition MAY contain one or more Content Component Restrictions to provide additional information on the relationship between the Datatype 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.

A.6 Documentation Rules

[DOC3]

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

- **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

A.6 Documentation Rules

[DOC4]

The xsd:documentation element for every Basic Business Information Entity MUST contain a structured set of annotations in the following sequence and pattern:

- **ComponentType (mandatory):** The type of component to which the object belongs. For Basic Business Information Entities this must be “BBIE”.
- **DictionaryEntryName (mandatory):** The official name of a Basic Business Information Entity.
- **Version (optional):** An indication of the evolution over time of the Basic Business Information Entity.
- **Definition(mandatory):** The semantic meaning of a Basic Business Information Entity.
- **Cardinality(mandatory):** Indication whether the Basic Business Information Entity represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Aggregate Business Information Entity.
- **ObjectClassQualifier (optional):** The qualifier for the object class.
- **ObjectClass(mandatory):** The Object Class containing the Basic Business Information Entity.
- **PropertyTermQualifier (optional):** A qualifier is a word or words which help define and differentiate a Basic Business Information Entity.
- **PropertyTerm(mandatory):** Property Term represents the distinguishing characteristic or Property of the Object Class and shall occur naturally in the definition of the Basic Business Information Entity.
- **RepresentationTerm (mandatory):** A Representation Term describes the form in which the Basic Business Information Entity is represented.
- **DataTypeQualifier (optional):** semantically meaningful name that differentiates the Datatype of the Basic Business Information Entity from its underlying Core Component Type.
- **DataType (mandatory):** Defines the Datatype used for the Basic Business Information Entity.
- **AlternativeBusinessTerms (optional):** Any synonym terms under which the Basic Business Information Entity is commonly known

A.6 Documentation Rules

[DOC5]

The xsd:documentation element for every Aggregate Business Information Entity MUST contain a structured set of annotations in the following sequence and pattern:

- **ComponentType (mandatory):** The type of component to which the object belongs. For Aggregate Business Information Entities this must be “ABIE”.
- **DictionaryEntryName (mandatory):** The official name of the Aggregate Business Information Entity .
- **Version (optional):** An indication of the evolution over time of the Aggregate Business Information Entity.
- **Definition(mandatory):** The semantic meaning of the Aggregate Business Information Entity.
- **ObjectClassQualifier (optional):** The qualifier for the object class.
- **ObjectClass(mandatory):** The Object Class represented by the Aggregate Business Information Entity.
- **AlternativeBusinessTerms (optional):** Any synonym terms under which the Aggregate Business Information Entity is commonly known and used in the business.

A.6 Documentation Rules

[DOC6]

The xsd:documentation element for every Association Business Information Entity element declaration MUST contain a structured set of annotations in the following sequence and pattern:

- **ComponentType (mandatory):** The type of component to which the object belongs. For Association Business Information Entities this must be “ASBIE”.
- **DictionaryEntryName (mandatory):** The official name of the Association Business Information Entity.
- **Version (optional):** An indication of the evolution over time of the Association Business Information Entity.
- **Definition(mandatory):** The semantic meaning of the Association Business Information Entity.
- **Cardinality(mandatory):** Indication whether the Association Business Information Entity represents an optional, mandatory and/or repetitive association.
- **ObjectClass(mandatory):** The Object Class containing the Association Business Information Entity.
- **PropertyTermQualifier (optional):** A qualifier is a word or words which help define and differentiate the Association Business Information Entity.
- **PropertyTerm(mandatory):** Property Term represents the Aggregate Business Information Entity contained by the Association Business Information Entity.
- **AssociatedObjectClassQualifier (optional):** Associated Object Class Qualifiers describe the 'context' of the relationship with another ABIE. That is, it is the role the contained Aggregate Business Information Entity plays within its association with the containing Aggregate Business Information Entity.
- **AssociatedObjectClass (mandatory):** Associated Object Class is the Object Class at the other end of this association. It represents the Aggregate Business Information Entity contained by the Association Business Information Entity.

A.6 Documentation Rules

[DOC7]	<p>The xsd:documentation element for every Core Component Type MUST contain a structured set of annotations in the following sequence and pattern:</p> <ul style="list-style-type: none">• ComponentType (mandatory): The type of component to which the object belongs. For Core Component Types this must be “CCT”.• DictionaryEntryName (mandatory): The official name of the Core Component Type, as defined by [CCTS].• Version (optional): An indication of the evolution over time of the Core Component Type.• Definition(mandatory): The semantic meaning of the Core Component Type, as defined by [CCTS].• ObjectClass(mandatory): The Object Class represented by the Core Component Type, as defined by [CCTS].• PropertyTerm(mandatory): The Property Term represented by the Core Component Type, as defined by [CCTS].
--------	--

2131

2132

A.7 Element Declaration Rules

[ELD1]	<p>Each UBL:ControlSchema MUST identify one and only one global element declaration that defines the document ccts:AggregateBusinessInformationEntity being conveyed in the Schema expression. That global element MUST include an xsd:annotation child element which MUST further contain an xsd:documentation child element that declares "This element MUST be conveyed as the root element in any instance document based on this Schema expression."</p>
[ELD2]	<p>All element declarations MUST be global with the exception of ID and Code which MUST be local.</p>
[ELD3]	<p>For every class identified in the UBL model, a global element bound to the corresponding xsd:complexType MUST be declared.</p>

A.7 Element Declaration Rules

[ELD4]	When a ccts:ASBIE is unqualified, it is bound via reference to the global ccts:ABIE element to which it is associated. When an ccts:ABIE is qualified, a new element MUST be declared and bound to the xsd:complexType of its associated ccts:AggregateBusinessInformationEntity.
[ELD5]	For each ccts:CCT simpleType, an xsd:restriction element MUST be declared.
[ELD6]	The code list xsd:import element MUST contain the namespace and schema location attributes.
[ELD7]	Empty elements MUST not be declared.
[ELD8]	Global elements declared for Qualified BBIE Properties must be of the same type as its corresponding Unqualified BBIE Property. (i.e. Property Term + Representation Term.)
[ELD9]	The xsd:any element MUST NOT be used.

2133

A.8 Element Naming Rules

[ELN1]	A UBL global element name based on a ccts:ABIE MUST be the same as the name of the corresponding xsd:complexType to which it is bound, with the word "Type" removed.
[ELN2]	A UBL global element name based on an unqualified ccts:BBIEProperty MUST be the same as the name of the corresponding xsd:complexType to which it is bound, with the word "Type" removed.
[ELN3]	A UBL global element name based on a qualified ccts:ASBIE MUST be the ccts:ASBIE dictionary entry name property term and its qualifiers; and the object class term and qualifiers of its associated ccts:ABIE. All ccts:DictionaryEntryName separators MUST be removed. Redundant words in the ccts:ASBIE property term or its qualifiers and the associated ccts:ABIE object class term or its qualifiers MUST be dropped.
[ELN4]	A UBL global element name based on a Qualified ccts:BBIEProperty MUST be

A.8 Element Naming Rules

the same as the name of the corresponding `xsd:complexType` to which it is bound, with the Qualifier prepended(?) and with the word "Type" removed.

2134

A.9 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 consistently derived 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]	The acronyms and abbreviations listed in Appendix B MUST always be used.
[GNR7]	UBL XML element, attribute and type names MUST be in singular form unless the concept itself is plural.
[GNR8]	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.
[GNR9]	The lowerCamelCase (LCC) convention MUST be used for naming attributes.

2135

A.10 General Type Definition Rules

[GTD1]	All types MUST be named.
[GTD2]	The xsd:any Type MUST NOT be used.

2136

A.11 General XML Schema Rules

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

A.11 General XML Schema Rules

attributeFormDefault="unqualified"

- <!-- ===== Imports ===== -->CommonAggregateComponents schema module
- CommonBasicComponents schema module
- Representation Term schema module (to include CCT module)
- Unspecialised Types schema module
- Specialised Types schema module
- <!-- ===== Global Attributes ===== -->
- Global Attributes and Attribute Groups
- <!-- ===== Root Element ===== -->
- Root Element Declaration
- Root Element Type Definition
- <!-- ===== Element Declarations ===== -->
- alphabetized order
- <!-- ===== Type Definitions ===== -->
- All type definitions segregated by basic and aggregates as follows
- <!-- ===== Aggregate Business Information Entity Type Definitions ===== -->
- alphabetized order of ccts:AggregateBusinessInformationEntity xsd:TypeDefinitions
- <!-- ===== Basic Business Information Entity Type Definitions ===== -->
- alphabetized order of ccts:BasicBusinessInformationEntities
- <!-- ===== Copyright Notice ===== -->
- Required OASIS full copyright notice.

A.11 General XML Schema Rules

[GXS2]	UBL MUST provide two normative schemas for each transaction. One schema shall be fully annotated. One schema shall be a run-time schema devoid of documentation.
[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 SHOULD NOT be used where customisation and extensibility are a concern.
[GXS10]	The xsd:include feature MUST only be used within a document schema.
[GXS11]	The xsd:union technique MUST NOT be used except for Code Lists. The xsd:union technique MAY be used for Code Lists.
[GXS12]	UBL designed schema SHOULD NOT use xsd:appinfo. If used, xsd:appinfo MUST only be used to convey non-normative information.
[GXS13]	Complex Type extension or restriction MAY be used where appropriate.

2137

2138

A.12 Instance Document Rules

[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: <code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>
[IND5]	UBL conformant instance documents MUST NOT contain an element devoid of content or null values.
[IND6]	The absence of a construct or data in a UBL instance document MUST NOT carry meaning.

2139

A.13 Modeling Constraints Rules

[MDC1]	UBL Libraries and Schemas MUST only use ebXML Core Component approved <code>ccts:CoreComponentTypes</code> .
[MDC2]	Mixed content MUST NOT be used except where contained in an <code>xsd:documentation</code> element.

2140

A.14 Naming Constraints Rules

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

2141

A.15 Namespace Rules

[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:<subtype>:<document-id></code>
[NMS5]	The namespace names for UBL Schemas holding OASIS Standard status MUST be of the form: <code>urn:oasis:names:specification:ubl:schema:<subtype>:<document-id></code>
[NMS6]	UBL published namespaces MUST never be changed.
[NMS7]	The <code>ubl:CommonAggregateComponents</code> schema module MUST reside in its own namespace.
[NMS8]	The <code>ubl:CommonAggregateComponents</code> schema module MUST be represented by the token "cac".
[NMS9]	The <code>ubl:CommonBasicComponents</code> schema module MUST reside in its own namespace.
[NMS10]	The <code>UBL:CommonBasicComponents</code> schema module MUST be represented by the token "cbc".
[NMS11]	The <code>ccts:CoreComponentType</code> schema module MUST reside in its own namespace.
[NMS12]	The <code>ccts:CoreComponentType</code> schema module namespace MUST be represented by the token "cct".

A.15 Namespace Rules

[NMS13]	The ccts:UnspecialisedDatatype schema module MUST reside in its own namespace.
[NMS14]	The ccts:UnspecialisedDatatype schema module namespace MUST be represented by the token "udt".
[NMS15]	The ubl:SpecialisedDatatypes schema module MUST reside in its own namespace.
[NMS16]	The ubl:SpecialisedDatatypes schema module namespace MUST be represented by the token "sdt".
[NMS17]	Each UBL:CodeList schema module MUST be maintained in a separate namespace.

2142

A.16 Root Element Declaration Rules

[RED1]	Every UBL instance document must use the global element defined as the root element in the schema as its root element.
--------	--

2143

A.17 Schema Structure Modularity Rules

[SSM1]	UBL Schema expressions MAY be split into multiple schema modules.
[SSM2]	A document schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another namespace MUST only import the document schema from that namespace.
[SSM3]	A UBL document schema in one UBL namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import internal schema modules from that namespace.

A.17 Schema Structure Modularity Rules

[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 document schema.
[SSM6]	All UBL internal schema modules MUST be in the same namespace as their corresponding document schema.
[SSM7]	Each UBL internal schema module MUST be named {ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}
[SSM8]	A UBL schema module MAY be created for reusable components.
[SSM9]	A schema module defining all ubl:CommonAggregateComponents MUST be created.
[SSM10]	The ubl:CommonAggregateComponents schema module MUST be named "ubl:CommonAggregateComponents Schema Module"
[SSM11]	A schema module defining all ubl:CommonBasicComponents MUST be created.
[SSM12]	The ubl:CommonBasicComponents schema module MUST be named "ubl:CommonBasicComponents Schema Module"
[SSM13]	A schema module defining all ccts:CoreComponentTypes MUST be created.
[SSM14]	The ccts:CoreComponentType schema module MUST be named "ccts:CoreComponentType Schema Module"
[SSM15]	The xsd:facet feature MUST not be used in the ccts:CoreComponentType schema module.
[SSM16]	A schema module defining all ccts:UnspecialisedDatatypes MUST be created.
[SSM17]	The ccts:UnspecialisedDatatype schema module MUST be named "ccts:UnspecialisedDatatype Schema Module"

A.17 Schema Structure Modularity Rules

[SSM18]	A schema module defining all <code>ubl:SpecialisedDatatypes</code> MUST be created.
[SSM19]	The <code>ubl:SpecialisedDatatypes</code> schema module MUST be named " <code>ubl:SpecialisedDatatypes</code> schema module"

2144

A.18 Standards Adherence rules

[STA1]	All UBL schema design rules MUST be based on the W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[STA2]	All UBL schema and messages MUST be based on the W3C suite of technical specifications holding recommendation status.
[STN1]	Each <code>CCTS:CCT</code> <code>simpleType</code> definition name MUST be the <code>ccts:CCT</code> dictionary entry name with the separators removed.

2145

A.19 SimpleType Naming Rules

[STN1]	Each <code>CCTS:CCT</code> <code>simpleType</code> definition name MUST be the <code>ccts:CCT</code> dictionary entry name with the separators removed.
--------	---

2146

A.20 SimpleType Definition Rules

[STD1]	For every <code>ccts:CCT</code> whose supplementary components map directly onto the properties of a built-in <code>xsd:DataType</code> , the <code>ccts:CCT</code> MUST be defined as a named <code>xsd:simpleType</code> in the <code>ccts:CCT</code> schema module.
--------	--

2147

A.21 Versioning Rules

[VER1]	Every UBL Schema and schema module major version committee draft MUST have an RFC 3121 document-id of the form <name>-<major>.0[.<revision>]
[VER2]	Every UBL Schema and schema module major version OASIS Standard MUST have an RFC 3121 document-id of the form <name>-<major>.0
[VER3]	Every minor version release of a UBL schema or schema module draft MUST have an RFC 3121 document-id of the form <name>-<major >.<non-zero>[.<revision>]
[VER4]	Every minor version release of a UBL schema or schema module OASIS Standard MUST have an RFC 3121 document-id of the form <name>-<major >.<non-zero>
[VER5]	For UBL Minor version changes, the name of the version construct MUST NOT change.
[VER6]	Every UBL Schema and schema module major version number MUST be a sequentially assigned, incremental number greater than zero.
[VER7]	Every UBL Schema and schema module minor version number MUST be a sequentially assigned, incremental non-negative integer.
[VER8]	A UBL minor version document schema MUST import its immediately preceding version document schema.
[VER9]	UBL Schema and schema module minor version changes MUST be limited to the use of xsd:extension or xsd:restriction to alter existing types or add new constructs.
[VER10]	UBL Schema and schema module minor version changes MUST not break semantic compatibility with prior versions.

2148 **Appendix B. Approved Acronyms and Abbreviations**

2149

2150 The following Acronyms and Abbreviations have been approved for UBL use:

2151 ◆ A Dun & Bradstreet number *must* appear as "DUNS". [TBD: need example.]

2152 ◆ "Identifier" *must* appear as "ID".

2153 ◆ "Uniform Resource Identifier" *must* appear as "URI"

2154 ◆ [Example] the "Uniform Resource. Identifier" portion of the **Binary Object.**
2155 **Uniform Resource. Identifier** supplementary component becomes "URI" in
2156 the resulting XML name). The use of URI for Uniform Resource Identifier
2157 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.
Document 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>Datatype</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.

2160

2161 **Appendix D. References**

2162 [CCTS] Core Components Technical Specification – Part 8 of the ebXML
2163 Technical Framework, Version 2.0 (Second Edition) 15 November
2164 2003

2165 [CCFeedback] *Feedback from OASIS UBL TC to Draft Core Components*
2166 *Specification 1.8*, version 5.2, May 4, 2002, [http://oasis-
open.org/committees/ubl/lcsc/doc/ubl-cctscomments-5p2.pdf](http://oasis-
2167 open.org/committees/ubl/lcsc/doc/ubl-cctscomments-5p2.pdf).

2168 [GOF] *Design Patterns*, Gamma, et al. ISBN 0201633612

2169 [ISONaming] *ISO/IEC 11179*, Final committee draft, Parts 1-6.
2170 (RFC) 2119 S. Bradner, *Key words for use in RFCs to Indicate Requirement*
2171 *Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March
2172 1997.

2173 [UBLChart] UBL TC Charter, [http://oasis-
open.org/committees/ubl/charter/ubl.htm](http://oasis-
2174 open.org/committees/ubl/charter/ubl.htm)

2175 [XML] *Extensible Markup Language (XML) 1.0* (Second Edition), W3C
2176 Recommendation, October 6, 2000

2177 (XSD) *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May
2178 2001.

2179

2180 (XHTML) *XHTML™ Basic*, W3C Recommendation 19 December 2000:
2181 <http://www.w3.org/TR/2000/REC-xhtml-basic-20001219>
2182

2183

Appendix E. Notices

2184 OASIS takes no position regarding the validity or scope of any intellectual property or
2185 other rights that might be claimed to pertain to the implementation or use of the
2186 technology described in this document or the extent to which any license under such
2187 rights might or might not be available; neither does it represent that it has made any effort
2188 to identify any such rights. Information on OASIS's procedures with respect to rights in
2189 OASIS specifications can be found at the OASIS website. Copies of claims of rights
2190 made available for publication and any assurances of licenses to be made available, or the
2191 result of an attempt made to obtain a general license or permission for the use of such
2192 proprietary rights by implementors or users of this specification, can be obtained from the
2193 OASIS Executive Director.

2194 OASIS invites any interested party to bring to its attention any copyrights, patents or
2195 patent applications, or other proprietary rights which may cover technology that may be
2196 required to implement this specification. Please address the information to the OASIS
2197 Executive Director.

2198 Copyright © The Organization for the Advancement of Structured Information Standards
2199 [OASIS] 2001, 2002, 2003, 2004. All Rights Reserved.

2200 This document and translations of it may be copied and furnished to others, and
2201 derivative works that comment on or otherwise explain it or assist in its implementation
2202 may be prepared, copied, published and distributed, in whole or in part, without
2203 restriction of any kind, provided that the above copyright notice and this paragraph are
2204 included on all such copies and derivative works. However, this document itself does not
2205 be modified in any way, such as by removing the copyright notice or references to
2206 OASIS, except as needed for the purpose of developing OASIS specifications, in which
2207 case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2208 document must be followed, or as required to translate it into languages other than
2209 English.

2210 The limited permissions granted above are perpetual and will not be revoked by OASIS
2211 or its successors or assigns.

2212 This document and the information contained herein is provided on an "AS IS" basis and
2213 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
2214 BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
2215 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2216 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
2217 PURPOSE.

2218