



Universal Business Language (UBL) Naming and Design Rules

Publication Date

5 November 2004

Document identifier:

cd-UBL-NDR-1.0.1

Location:

<http://docs.oasis-open.org/ubl/cd-UBL-NDR-1.0.1/>

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 for the UBL vocabulary.

Status:

This is a draft document under consideration by the OASIS Universal Business Language Technical Committee for approval as a Committee Draft and OASIS Standard

40 **Table of Contents**

41	1	Introduction	5
42	1.1	Audiences	5
43	1.2	Scope	5
44	1.3	Terminology and Notation	5
45	1.4	Guiding Principles	7
46	1.4.1	Adherence to General UBL Guiding Principles	7
47	1.4.2	Design For Extensibility	8
48	1.4.3	Code Generation	8
49	1.5	Choice of schema language	8
50	2	Relationship to ebXML Core Components	10
51	2.1	Mapping Business Information Entities to XSD	12
52	3	General XML Constructs	14
53	3.1	Overall Schema Structure	14
54	3.1.1	Root Element	15
55	3.2	Constraints	16
56	3.2.1	Naming Constraints	16
57	3.2.2	Modeling Constraints	16
58	3.3	Reusability Scheme	17
59	3.4	Namespace Scheme	18
60	3.4.1	Declaring Namespaces	18
61	3.4.2	Namespace Uniform Resource Identifiers	18
62	3.4.3	Schema Location	19
63	3.4.4	Persistence	19
64	3.5	Versioning Scheme	19
65	3.6	Modularity	21
66	3.6.1	UBL Modularity Model	21
67	3.6.2	Internal and External schema modules	24
68	3.6.3	Internal schema modules	24
69	3.6.4	External schema modules	24
70	3.7	Annotation and Documentation	27
71	3.7.1	Schema Annotation	28
72	3.7.2	Embedded documentation	28
73	4	Naming Rules	31
74	4.1	General Naming Rules	31
75	4.2	Type Naming Rules	32

76	4.2.1	Complex Type Names for CCTS Aggregate Business Information Entities	33
77			
78	4.2.2	Complex Type Names for CCTS Basic Business Information Entity Properties	33
79			
80	4.2.3	Complex Type Names for CCTS Unspecialized Datatypes	33
81	4.2.4	Complex Type Names for CCTS Core Component Types	34
82	4.2.5	Simple Type Names for CCTS Core Component Types	34
83	4.3	Element Naming Rules	34
84	4.3.1	Element Names for CCTS Aggregate Business Information Entities	34
85	4.3.2	Element Names for CCTS Basic Business Information Entity Properties	35
86	4.3.3	Element Names for CCTS Association Business Information Entities	35
87	4.4	Attribute Naming Rules	36
88	5	Declarations and Definitions	37
89	5.1	Type Definitions	37
90	5.1.1	General Type Definitions	37
91	5.1.2	Simple Types	37
92	5.1.3	Complex Types	37
93	5.2	Element Declarations	41
94	5.2.1	Elements Bound to Complex Types	41
95	5.2.2	Elements Representing ASBIEs	41
96	5.2.3	Elements Bound to Core Component Types	41
97	5.2.4	Code List Import	42
98	5.2.5	Empty Elements	42
99	5.2.6	Global Elements	42
100	5.2.7	XSD:Any Element	42
101	5.3	Attribute Declarations	42
102	5.3.1	User Defined Attributes	42
103	5.3.2	Global Attributes	43
104	5.3.3	Supplementary Components	43
105	5.3.4	Schema Location	43
106	5.3.5	XSD:nil	43
107	5.3.6	XSD:AnyAttribute	43
108	6	Code Lists	44
109	7	Miscellaneous XSD Rules	46
110	7.1	XSD Simple Types	
111		Error! Bookmark not defined.	
112	7.2	Namespace Declaration	46
113	7.3	XSD:Substitution Groups	46

114	7.4 XSD:Final	46
115	7.5 XSD: Notation	46
116	7.6 XSD:All	46
117	7.7 XSD:Choice	46
118	7.8 XSD:Include	47
119	7.9 XSD:Union	47
120	7.10 XSD:Appinfo	47
121	7.11 Extension and Restriction	47
122	8 Instance Documents	48
123	8.1 Root Element	48
124	8.2 Validation	48
125	8.3 Character Encoding	48
126	8.4 Schema Instance Namespace Declaration	48
127	8.5 Empty Content.	49
128	Appendix A. UBL NDR Checklist	50
129	A.1 Attribute Declaration Rules	50
130	A.2 Attribute Naming Rules	51
131	A.3 Code List Rules	51
132	A.4 ComplexType Definition Rules	52
133	A.5 ComplexType Naming Rules	53
134	A.6 Documentation Rules	54
135	A.7 Element Declaration Rules	57
136	A.8 Element Naming Rules	57
137	A.9 General Naming Rules	58
138	A.10General Type Definition Rules	58
139	A.11General XML Schema Rules	59
140	A.12Instance Document Rules	60
141	A.13Modeling Constraints Rules	61
142	A.14Naming Constraints Rules	61
143	A.15Namespace Rules	61
144	A.16Root Element Declaration Rules	62
145	A.17Schema Structure Modularity Rules	62
146	A.18Standards Adherence rules	63
147	A.19SimpleType Naming Rules	64
148	A.20SimpleType Definition Rules	64
149	A.21Versioning Rules	64
150	Appendix B. Approved Acronyms and Abbreviations	66

151	Appendix C. Technical Terminology	67
152	Appendix D. References	71
153	Appendix E. Notices	72

154 1 Introduction

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

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

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

182 The UBL Technical Committee has established the UBL Naming and Design Rules
183 Subcommittee with the charter to "Recommend to the TC rules and guidelines for
184 normative-form schema design, instance design, and markup naming, and write and
185 maintain documentation of these rules and guidelines". Accordingly, this specification
186 documents the rules and guidelines for the naming and design of XML components for
187 the UBL library. It contains only rules that have been agreed on by the OASIS UBL
188 Naming and Design Rules Subcommittee (NDR SC). Proposed rules, and rationales for
189 those that have been agreed on, appear in the accompanying NDR SC position papers,
190 which are available at <http://www.oasis-open.org/committees/ubl/ndrsc/>.

191 1.1 Audiences

192 This document has several primary and secondary targets that together constitute its
193 intended audience. Our primary target audience is the members of the UBL Technical
194 Committee. Specifically, the UBL Technical Committee will use the rules in this
195 document to create normative form schema for business transactions. Developers
196 implementing ebXML Core Components may find the rules contained herein sufficiently
197 useful to merit adoption as, or infusion into, their own approaches to ebXML Core
198 Component based XML schema development. All other XML Schema developers may
199 find the rules contained herein sufficiently useful to merit consideration for adoption as,
200 or infusion into, their own approaches to XML schema development.

201 1.2 Scope

202 This specification conveys a normative set of XML schema design rules and naming
203 conventions for the creation of business based XML schema for business documents
204 being exchanged between two parties using XML constructs defined in accordance with
205 the ebXML Core Components Technical Specification.

206 1.3 Terminology and Notation

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

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

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

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

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

224 *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

225 **Bold** – The bolding of words is used to represent example names or parts of names taken
226 from the library.

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

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

230 Keywords – keywords reflect concepts or constructs expressed in the language of their
231 source standard. Keywords have been given an identifying prefix to reflect their source.
232 The following prefixes are used:

233 ▪ ‘xsd:’ – represents W3C XML Schema Definition Language. If a concept, the
234 words will be in upper camel case, and if a construct, they will be in lower camel
235 case.

236 ○ `xsd:complexType` represents an XSD construct

237 ○ `xsd:SchemaExpression` represents a concept

238 ▪ ‘ccts:’ – represents ISO 15000-5 ebXML Core Components Technical
239 Specification

240 ▪ ‘ubl:’ – represents the OASIS Universal Business Language

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

245 1.4 Guiding Principles

246 The UBL guiding principles encompass three areas:

- 247 ◆ General UBL guiding principles
- 248 ◆ Extensibility
- 249 ◆ Code generation

250 1.4.1 Adherence to General UBL Guiding Principles

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

- 254 ◆ Internet Use – UBL shall be straightforwardly usable over the Internet.
- 255 ◆ Interchange and Application Use – UBL is intended for interchange and
256 application use.
- 257 ◆ Tool Use and Support – The design of UBL will not make any assumptions
258 about sophisticated tools for creation, management, storage, or presentation
259 being available. The lowest common denominator for tools is incredibly low
260 (for example, Notepad) and the variety of tools used is staggering. We do not
261 see this situation changing in the near term.
- 262 ◆ Legibility – UBL documents should be human-readable and reasonably clear.
- 263 ◆ Simplicity – The design of UBL must be as simple as possible (but no
264 simpler).
- 265 ◆ 80/20 Rule – The design of UBL should provide the 20% of features that
266 accommodate 80% of the needs.
- 267 ◆ Component Reuse –The design of UBL document types should contain as
268 many common features as possible. The nature of e-commerce transactions is
269 to pass along information that gets incorporated into the next transaction down
270 the line. For example, a purchase order contains information that will be
271 copied into the purchase order response. This forms the basis of our need for a
272 core library of reusable components. Reuse in this context is important, not
273 only for the efficient development of software, but also for keeping audit
274 trails.
- 275 ◆ Standardization – The number of ways to express the same information in a
276 UBL document is to be kept as close to one as possible.

- 277 ◆ Domain Expertise – UBL will leverage expertise in a variety of domains
278 through interaction with appropriate development efforts.
- 279 ◆ Customization and Maintenance – The design of UBL must facilitate
280 customization and maintenance.
- 281 ◆ Context Sensitivity – The design of UBL must ensure that context-sensitive
282 document types aren't precluded.
- 283 ◆ Prescriptiveness – UBL design will balance prescriptiveness in any single
284 usage scenario with prescriptiveness across the breadth of usage scenarios
285 supported. Having precise, tight content models and datatypes is a good thing
286 (and for this reason, we might want to advocate the creation of more
287 document type “flavors” rather than less). However, in an interchange format,
288 it is often difficult to get the prescriptiveness that would be desired in any
289 single usage scenario.
- 290 ◆ Content Orientation – Most UBL document types should be as “content-
291 oriented” (as opposed to merely structural) as possible. Some document types,
292 such as product catalogs, will likely have a place for structural material such
293 as paragraphs, but these will be rare.
- 294 ◆ XML Technology – UBL design will avail itself of standard XML processing
295 technology wherever possible (XML itself, XML Schema, XSLT, XPath, and
296 so on). However, UBL will be cautious about basing decisions on “standards”
297 (foundational or vocabulary) that are works in progress.
- 298 ◆ Relationship to Other Namespaces – UBL design will be cautious about
299 making dependencies on other namespaces. UBL does not need to reuse
300 existing namespaces wherever possible. For example, XHTML might be
301 useful in catalogs and comments, but it brings its own kind of processing
302 overhead, and if its use is not prescribed carefully it could harm our goals for
303 content orientation as opposed to structural markup.
- 304 ◆ Legacy formats – UBL is not responsible for catering to legacy formats;
305 companies (such as ERP vendors) can compete to come up with good

306 solutions to permanent conversion. This is not to say that mappings to and
307 from other XML dialects or non-XML legacy formats wouldn't be very
308 valuable.

309 ◆ Relationship to xCBL – UBL will not be a strict subset of xCBL, nor will it be
310 explicitly compatible with it in any way.¹

311 1.4.2 Design For Extensibility

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

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

323 To avoid the high degree of cross-application coordination required to handle structural
324 variations common to EDI and XML based systems—it is necessary to accommodate the
325 required variations in basic data structures without either overloading the meaning and
326 use of existing data elements, or requiring wholesale addition of new data elements. This
327 can be accomplished by allowing implementers to specify new element types that inherit
328 the properties of existing elements, and to also specify exactly the structural and data
329 content of the modifications.

330 This approach can be expressed by saying that extensions of core elements are driven by
331 context.² Context driven extensions should be renamed to distinguish them from their

¹ *XML Common Business Library (xCBL) is a set of XML business documents and their components.*

332 parents, and designed so that only the new elements require new processing. Similarly,
333 data structures should be designed so that processes can be easily engineered to ignore
334 additions that are not needed. The UBL context methodology is discussed in the
335 *Guidelines for the Customization of UBL Schemas* available as part of UBL 1.0.

336 1.4.3 Code Generation

337 The UBL NDR makes no assumptions on the availability or capabilities of tools to
338 generate UBL conformant XSD Schemas. In conformance with UBL guiding principles,
339 the UBL NDR design process has scrupulously avoided establishing any naming or
340 design rules that sub-optimize the UBL schemas in favor of tool generation. Additionally,
341 in conformance with UBL guiding principles, the NDR is sufficiently rigorous to avoid
342 requiring human judgment at schema generation time.

343 1.5 Choice of schema language

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

349

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

353 A W3C technical specification holding recommended status represents consensus within
354 the W3C and has the W3C Director's stamp of approval. Recommendations are
355 appropriate for widespread deployment and promote W3C's mission. Before the Director
356 approves a recommendation, it must show an alignment with the W3C architecture. By

² ebXML, Core Components Technical Specification – Part 8 of the ebXML Technical Framework, V2.01, 15 November, 2003

357 aligning with W3C specifications holding recommended status, UBL can ensure that its
358 products and deliverables are well suited for use by the widest possible audience with the
359 best availability of common support tools.

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

362 2 Relationship to ebXML Core Components

363 UBL employs the methodology and model described in *Core Components Technical*
364 *Specification, Part 8 of the ebXML Technical Framework, Version 2.01* of 15 November
365 2003 (CCTS) to build the UBL Component Library. The Core Components work is a
366 continuation of work that originated in, and remains a part of, the ebXML initiative. The
367 Core Components concept defines a new paradigm in the design and implementation of
368 reusable syntactically neutral information building blocks. Syntax neutral Core
369 Components are intended to form the basis of business information standardization
370 efforts and to be realized in syntactically specific instantiations such as ANSI ASC X12,
371 UN/EDIFACT, and XML representations such as UBL.

372 The essence of the Core Components specification is captured in context neutral and
373 context specific building blocks. The context neutral components are defined as Core
374 Components (`ccts:CoreComponents`). Context neutral `ccts:CoreComponents` are
375 defined in CCTS as “A building block for the creation of a semantically correct and
376 meaningful information exchange package. It contains only the information pieces
377 necessary to describe a specific concept.”³ Figure 2-1 illustrates the various pieces of the
378 overall `ccts:CoreComponents` metamodel.

379 The context specific components are defined as Business Information Entities
380 (`ccts:BusinessInformationEntities`).⁴ Context specific `ccts:Business`
381 `InformationEntities` are defined in CCTS as “A piece of business data or a group of
382 pieces of business data with a unique *Business Semantic* definition.”⁵ Figure 2-2
383 illustrates the various pieces of the overall `ccts:BusinessInformationEntity`
384 metamodel and their relationship with the `ccts:CoreComponents` metamodel.

³ *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition)*, UN/CEFACT, 15 November 2003

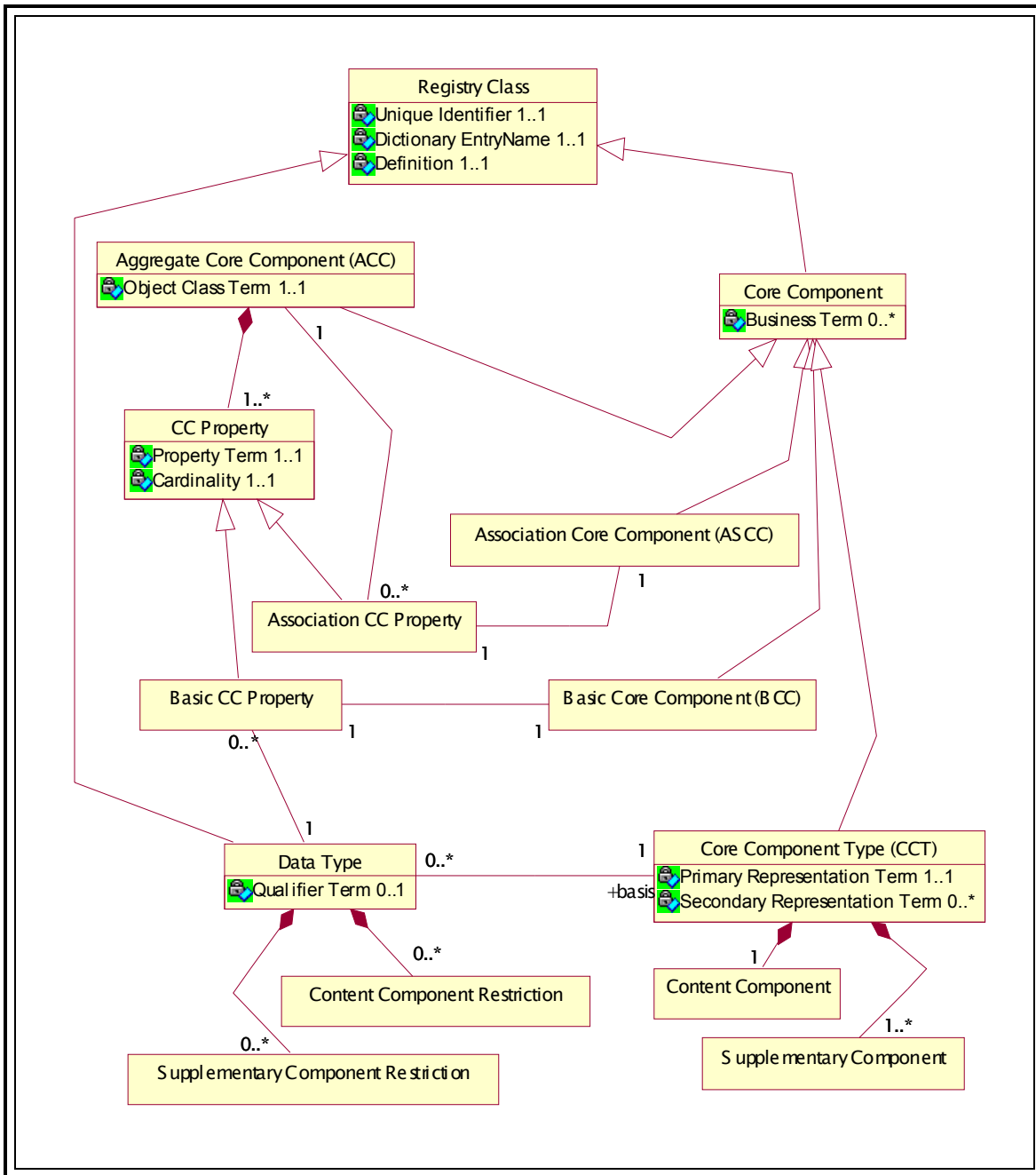
⁴ See CCTS Section 6.2 for a detailed discussion of the ebXML context mechanism.

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

385 As shown in Figure 2-2, there are different types of `ccts:CoreComponents` and
386 `ccts:BusinessInformationEntities`. Each type of `ccts:CoreComponent` and
387 `ccts:BusinessInformationEntity` has specific relationships between and



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

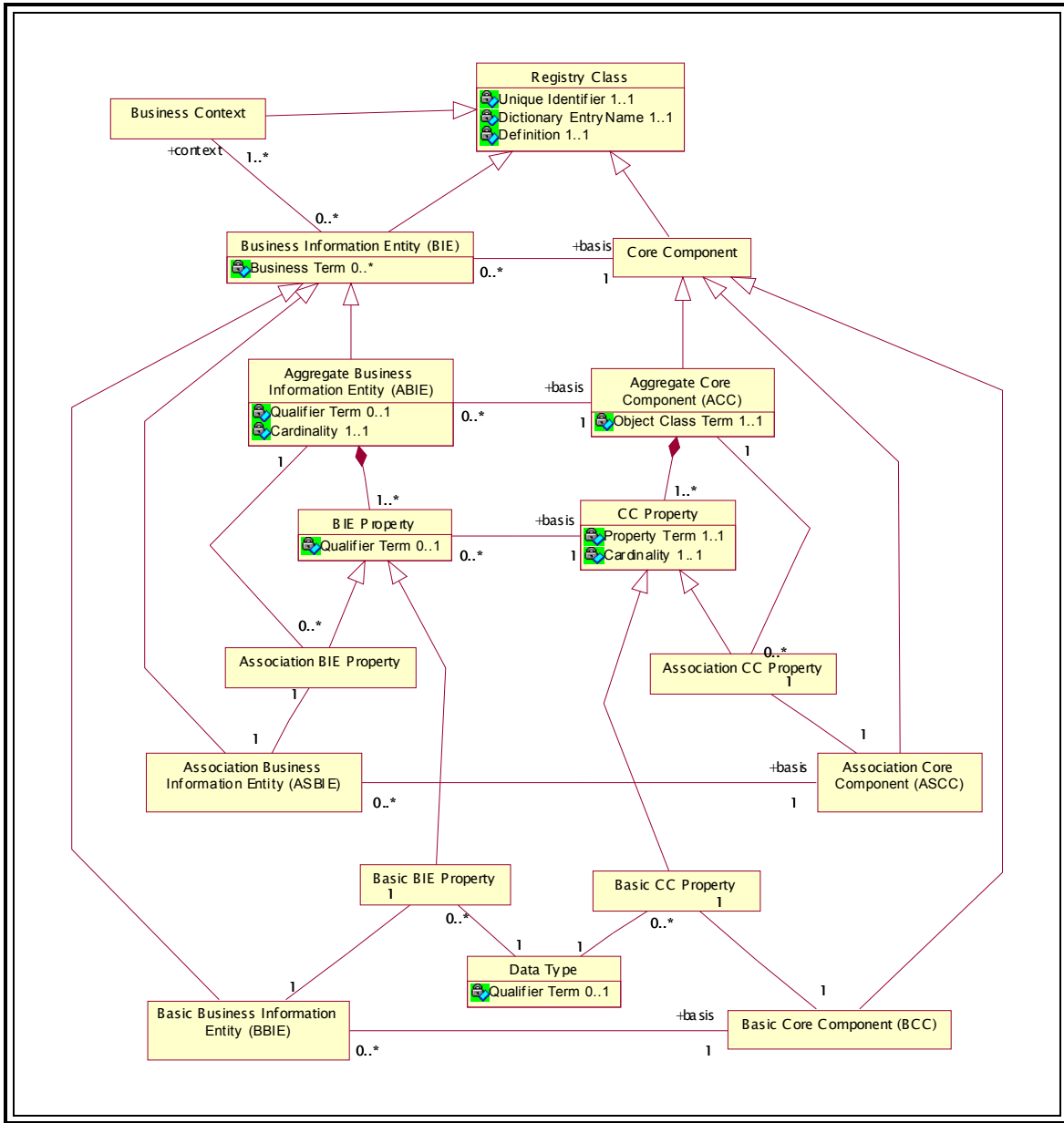


389

390 amongst the other components and entities. The context neutral `cts:Core`
 391 `Components` are the linchpin that establishes the formal relationship between the various
 392 context-specific `cts:BusinessInformationEntities`.

393

Figure 2-2. Business Information Entities Basic Definition Model



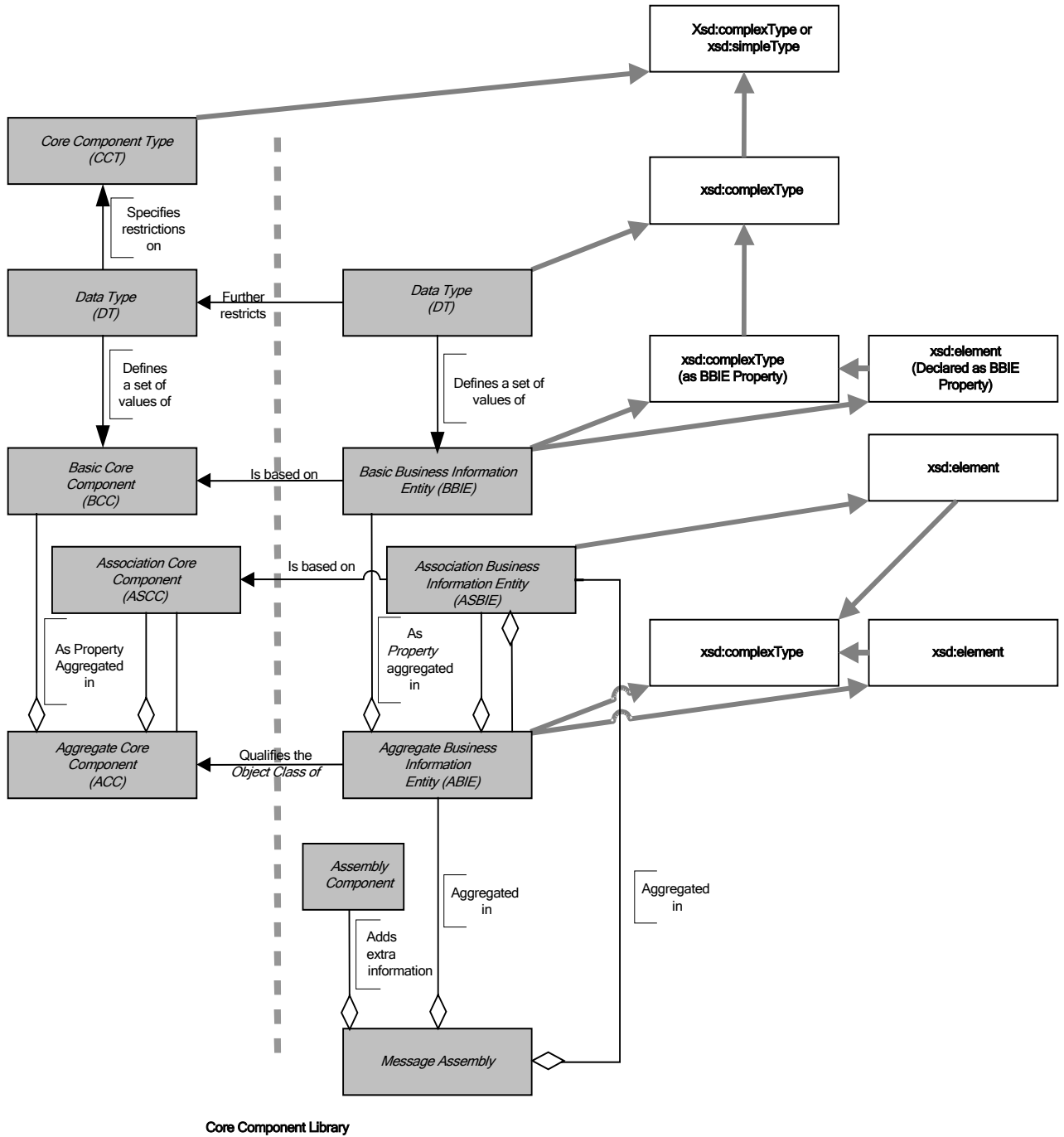
394

395 2.1 Mapping Business Information Entities to XSD

396 UBL consists of a library of `ccts:BusinessInformationEntities`. In creating this
 397 library, UBL has defined how each of the `ccts:BusinessInformationEntity`
 398 components map to an XSD construct (See figure 2-3). In defining this mapping, UBL
 399 has analyzed the CCTS metamodel and determined the optimal usage of XSD to express
 400 the various `ccts:BusinessInformationEntity` components. As stated above, a
 401 `ccts:BusinessInformationEntity` can be a `ccts:AggregateBusiness`

402 InformationEntity, a ccts:BasicBusinessInformationEntity, or a
403 ccts:AssociationBusinessInformationEntity. In understanding the logic of
404 the UBL binding of ccts:BusinessInformationEntities to XSD expressions, it is
405 important to understand the basic constructs of the ccts:AggregateBusiness
406 InformationEntities and their relationships as shown in Figure 2-2.

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



408
409

410 Both Aggregate and Basic Business Information Entities must have a unique name
411 (Dictionary Entry Name). The `ccts:AggregateBusinessInformationEntities`
412 are treated as objects and are defined as `xsd:complexType`s. The `ccts:Basic`

413 BusinessInformationEntities are treated as attributes of the ccts:Aggregate
414 BusinessInformationEntity and are found in the content model of the
415 ccts:AggregateBusinessInformationEntity as a referenced xsd:element.
416 The ccts:BasicBusinessInformationEntities are based on a reusable
417 ccts:BasicBusinessInformationEntityProperty which are defined as
418 xsd:complexTypees.

419 A Basic Business Information Entity Property represents an *intrinsic* property of an
420 Aggregate Business Information Entity. Basic Business Information Entity properties are
421 linked to a Datatype. UBL defines two types of Datatypes – unspecialized and
422 specialized. The ubl:UnspecializedDatatypes correspond to
423 ccts:RepresentationTerms and have no restrictions to the values of the
424 corresponding ccts:ContentComponent or ccts:SupplementaryComponent. The
425 ubl:SpecializedDatatypes are derived from ubl:UnspecializedDatatypes
426 with restrictions to the allowed values or ranges of the corresponding
427 ccts:ContentComponent or ccts:SupplementaryComponent.

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

432 A ccts:Datatype “defines the set of valid values that can be used for a particular
433 *Basic Core Component Property* or *Basic Business Information Entity Property*
434 *Datatype*”⁷ The ccts:Datatypes can be either unspecialized—no restrictions
435 applied—or specialized through the application of restrictions. The sum total of the
436 datatypes is then instantiated as the basis for the various XSD simple and complex types
437 defined in the UBL schemas. CCTS supports datatypes that are specialized, i.e. it enables
438 users to define their own datatypes for their syntax neutral constructs. Thus
439 ccts:Datatypes allow UBL to identify restrictions for elements when restrictions to
440 the corresponding ccts:ContentComponent or ccts:SupplementaryComponent
441 are required.

⁷ *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition), UN/CEFACT, 15 November 2003*

442 There are two kinds of Business Information Entity Properties - Basic and Association. A
443 `ccts:AssociationBusinessInformationEntityProperty` represents an
444 *extrinsic* property – in other words an association from one `ccts:Aggregate`
445 `BusinessInformationEntityProperty` instance to another `ccts:Aggregate`
446 `BusinessInformationEntityProperty` instance. It is the `ccts:Aggregate`
447 `BusinessInformationEntityProperty` that expresses the relationship between
448 `ccts:AggregateBusinessInformationEntities`. Due to their unique extrinsic
449 association role, `ccts:AssociationBusinessInformationEntities` are not
450 defined as `xsd:complexType`, rather they are either declared as elements that are then
451 bound to the `xsd:complexType` of the associated `ccts:AggregateBusiness`
452 `InformationEntity`, or they are reclassified ABIEs.

453 As stated above, `ccts:BasicBusinessInformationEntities` define the intrinsic
454 structure of a `ccts:AggregateBusinessInformationEntity`. These
455 `ccts:BasicBusinessInformationEntities` are the “leaf” types in the system in
456 that they contain no `ccts:AssociationBusinessInformationEntity` properties.

457 A `ccts:BasicBusinessInformationEntity` must have a `ccts:CoreComponent`
458 `Type`. All `ccts:CoreComponentTypes` are low-level types, such as Identifiers and
459 Dates. A `ccts:CoreComponentType` describes these low-level types for use by
460 `ccts:CoreComponents`, and (in parallel) a `ccts:Datatype`, corresponding to that
461 `ccts:CoreComponentType`, describes these low-level types for use by
462 `ccts:BusinessInformationEntities`. Every `ccts:CoreComponentType` has a
463 single `ccts:ContentComponent` and one or more `ccts:Supplementary`
464 `Components`. A `ccts:ContentComponent` is of some `Primitive Type`. All
465 `ccts:CoreComponentTypes` and their corresponding content and supplementary
466 components are pre-defined in the CCTS. UBL has developed an `xsd:SchemaModule`
467 that defines each of the pre-defined `ccts:CoreComponentTypes` as an
468 `xsd:complexType` or `xsd:simpleType` and declares `ccts:Supplementary`
469 `Components` as an `xsd:attribute` or uses the predefined facets of the built-in
470 `xsd:Datatype` for those that are used as the base expression for an
471 `xsd:simpleType`. UBL continues to work with UN/CEFACT and the Open
472 Applications Group to develop a single normative schema for representing
473 `ccts:CoreComponentTypes`.

474 3 General XML Constructs

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

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

483 3.1 Overall Schema Structure

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

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

491 XML Declaration

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

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

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

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

498 `xmlns:xsd`

499 Target namespace

500 Default namespace

501 `CommonAggregateComponents`

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

530 3.1.1 Root Element

531 Per XML 1.0, “There is exactly one element, called the **root**, or document element, no
 532 part of which appears in the content of any other element.” XML 1.0 further states “The

533 **root element** of any document is considered to have signaled no intentions as regards
534 application space handling, unless it provides a value for this attribute or the attribute is
535 declared with a default value.” W3C XSD allows for any globally declared element to be
536 the document root element. To keep consistency in the instance documents and to adhere
537 to the underlying process model that supports each UBL Schema, it is desirable to have
538 one and only one element function as the root element. Since UBL follows a global
539 element declaration scheme (See Rule ELD2), each UBL Schema will identify one
540 element declaration in each schema as the document root element. This will be
541 accomplished through an `xsd:annotation` child element for that element in
542 accordance with the following rule:

543 [ELD1] Each `UBL:DocumentSchema` MUST identify one and only one global
544 element declaration that defines the document `ccts:Aggregate`
545 `BusinessInformationEntity` being conveyed in the Schema expression.
546 That global element MUST include an `xsd:annotation` child element
547 which MUST further contain an `xsd:documentation` child element that
548 declares *“This element MUST be conveyed as the root element*
549 *in any instance document based on this Schema*
550 *expression.”*

551 [Definition] Document schema –

552 The overarching schema within a specific namespace that conveys the business
553 document functionality of that namespace. The document schema declares a target
554 namespace and is likely to pull in by including internal schema modules or importing
555 external schema modules. Each namespace will have one, and only one, document
556 schema.

557 Example:

```
558 <xsd:element name="Order" type="OrderType">  
559   <xsd:annotation>  
560     <xsd:documentation>This element MUST be conveyed as the root  
561     element in any instance document based on this Schema  
562     expression</xsd:documentation>  
563   </xsd:annotation>  
564 </xsd:element>
```

569 3.2 Constraints

570 A key aspect of UBL is to base its work on process modeling and data analysis as
571 precursors to developing the UBL library. In determining how best to affect this work,

572 several constraints have been identified that directly impact both the process modeling
573 and data analysis, and the resultant UBL Schema.

574 3.2.1 Naming Constraints

575 A primary aspect of the UBL library documentation are its spreadsheet models. The
576 entries in these spreadsheet models fully define the constructs available for use in UBL
577 business documents. These spreadsheet entries contain fully conformant CCTS dictionary
578 entry names as well as truncated UBL XML element names developed in conformance
579 with the rules in section 4. The dictionary entry name ties the information to its
580 standardized semantics, while the name of the corresponding XML element or attribute is
581 only shorthand for this full name. The rules for element and attribute naming and
582 dictionary entry naming are different.

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

585 The fully qualified path anchors the use of that construct to a particular location in a
586 business message. The definition of the construct identifies any semantic dependencies
587 that the FQP has on other elements and attributes within the UBL library that are not
588 otherwise enforced or made explicit in its structural definition.

589 3.2.2 Modeling Constraints

590 In keeping with UBL guiding principles, modeling constraints are limited to those
591 necessary to ensure consistency in development of the UBL library.

592 3.2.2.1 Defining Classes

593 UBL is based on instantiating ebXML `ccts:BusinessInformationEntities`. UBL
594 models and the XML expressions of those models are class driven. Specifically, the UBL
595 library defines classes for each `ccts:AggregateBusinessInformationEntity` and
596 the UBL schemas instantiate those classes. The attributes of those classes consist of
597 `ccts:BasicBusinessInformationEntities`.

598 3.2.2.2 Core Component Types

599 Each `ccts:BasicBusinessInformationEntity` has an associated
600 `ccts:CoreComponentType`. The CCTS specifies an approved set of
601 `ccts:CoreComponentTypes`. To ensure conformance, UBL is limited to using this
602 approved set.

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

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

608 3.2.2.3 Mixed Content

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

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

616 3.3 Reusability Scheme

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

620 3.3.1.4 Reusable Elements

621 UBL elements are global and qualified. Hence in the example below, the `<Address>`
622 element is directly reusable as a modular component and some software can be used
623 without modification.

624 Example

```
625  
626 <xsd:element name="Party" type="PartyType"/>  
627 <xsd:complexType name="PartyType">  
628 <xsd:annotation>  
629 <!--Documentation goes here-->  
630 </xsd:annotation>  
631 <xsd:sequence>  
632 <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0"  
633 maxOccurs="1"/>  
634 ...  
635 </xsd:element>  
636 <xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0"  
637 maxOccurs="1"/>  
638 ...  
639 </xsd:element>  
640 <xsd:element ref="PartyIdentification" minOccurs="0"  
641 maxOccurs="unbounded"/>  
642 ...
```

```

643     </xsd:element>
644     <xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
645         ...
646     </xsd:element>
647     <xsd:element ref="Address" minOccurs="0" maxOccurs="1">
648         ...
649     </xsd:element>
650     ...
651 </xsd:sequence>
652 </xsd:complexType>
653 <xsd:element name="Address" type="AddressType"/>
654 <xsd:complexType name="AddressType">
655     ...
656     <xsd:sequence>
657         <xsd:element ref="cbc:CityName" minOccurs="0" maxOccurs="1">
658             ...
659         </xsd:element>
660         <xsd:element ref="cbc:PostalZone" minOccurs="0" maxOccurs="1">
661             ...
662         </xsd:element>
663         ...
664     </xsd:sequence>
665 </xsd:complexType>

```

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

672 The only cases where locally declared elements are seen to be advantageous are in the
673 case of Identifiers and Code. Code lists and identification schemes are generally specific
674 to trading partner and other user communities. These constructs can require specific
675 validation. Consequently, there is less benefit in declaring them as global elements.

676 Codes are treated as a special case in UBL which is also highly configurable according to
677 trading partner or community preference.

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

680 3.4 Namespace Scheme

681 The concept of XML namespaces is defined in the W3C XML namespaces technical
682 specification.⁸ The use of XML namespace is specified in the W3C XML Schema (XSD)
683 Recommendation. A namespace is declared in the root element of a Schema using a
684 namespace identifier. Namespace declarations can also identify an associated prefix—
685 shorthand identifier—that allows for compression of the namespace name. For each UBL
686 namespace, a normative token is defined as its prefix. These tokens are defined in Section
687 3.6. It is common for an instance document to carry namespace declarations, so that it
688 might be validated.

689 3.4.1 Declaring Namespaces

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

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

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

⁸ *Tim Bray, D Hollander, A Layman, R Tobin; Namespaces in XML 1.1, W3C Recommendation, February 2004.*

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

704 [Definition] Schema Set –

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

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

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

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

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

718 3.4.2 Namespace Uniform Resource Identifiers

719 A UBL namespace name must be a URI reference that conforms to RFC 2396.⁹ UBL has
720 adopted the Uniform Resource Name (URN) scheme as the standard for URIs for UBL
721 namespaces, in conformance with IETF's RFC 3121¹⁰, as defined in this next section.

⁹ T. Berners-Lee, R. Fielding, L. Masinter; *Internet Engineering Task Force (IETF) RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax*, Internet Society, August 1998.

¹⁰ Karl Best, N. Walsh; *Internet Engineering Task Force (IETF) RFC 3121, A URN Namespace for OASIS*, June 2001.

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

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

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

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

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

732 [NMS5] The namespace names for UBL Schemas holding OASIS Standard status
733 MUST be of the form:

734
735 `urn:oasis:names:specification:ubl:schema:<subtype>:<docum`
736 `ent-id>`

737 3.4.3 Schema Location

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

744 3.4.4 Persistence

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

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

751 3.5 Versioning Scheme

752 UBL namespaces conform to the OASIS namespace rules defined in RFC 3121.¹¹ The
753 last field of the namespace name is called `document-id`. UBL has decided to include
754 versioning information as part of the `document-id` component of the namespace. The version
755 information is divided into `major` and `minor` fields. The `minor` field has an optional
756 `revision` extension. For example, the namespace URI for the draft Invoice domain has
757 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-  
770 number>.0[.<revision>]
```

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

¹¹ Karl Best, N. Walsh,; Internet Engineering Task Force (IETF) RFC 3121, A URN Namespace for OASIS, June 2001.

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

775

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

778 <name>-<major>.0

779 For each document produced by the TC, the TC will determine the value of the <name>
780 variable. In UBL, the major-version field of a namespace URI must be changed in a
781 release that breaks compatibility with the previous release of that namespace. If a change
782 does not break compatibility then only the minor version need change. Subsequent minor
783 releases begin with minor-version 1.

784 Example

785

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

787

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

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-* respectively.
806 If either of the common namespaces change then its namespace URI must change. If its
807 namespace URI changes then any schema that imports the *new version* of the namespace
808 must also change (to update the namespace declaration). And since the importing schema
809 changes, its namespace URI in turn must change. The outcome is twofold:

- 810 ◆ There should never be ambiguity at the point of reference in a namespace
811 declaration or version identification. A dependent schema imports precisely

812 the version of the namespace that is needed. The dependent schema never
813 needs to account for the possibility that the imported namespace can change.

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

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

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

821
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
846 `urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.2`

847 imports
848 urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.1
849 which imports
850 urn:oasis:names:tc:ubl:schema:xsd:Invoice-1.0
851

853 [VER8] A UBL minor version document schema MUST import its immediately
854 preceding version document schema.

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

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

863 In addition to polymorphic processing considerations, semantic compatibility across
864 minor versions (as well as major versions) is essential. Semantic compatibility in this
865 sense pertains to preserving the business function.

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

868 3.6 Modularity

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

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

876 [Definition] schema module –
877 A schema document containing type definitions and element declarations intended to
878 be reused in multiple schemas.

879 3.6.1 UBL Modularity Model

880 UBL relies extensively on modularity in schema design. There is no single UBL root
881 schema. Rather, there are a number of UBL document schemas, each of which expresses
882 a separate business function. The UBL modularity approach is structured so that users

883 can reuse individual document schemas without having to import the entire UBL
884 document schema library. Additionally, a document schema can import individual
885 modules without having to import all UBL schema modules. Each document schema will
886 define its own dependencies. The UBL schema modularity model ensures that logical
887 associations exist between document and internal schema modules and that individual
888 modules can be reused to the maximum extent possible. This is accomplished through the
889 use of document and internal schema modules as shown in Figure 3-1.

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

892 Figure 3-1 shows the one-to-one correspondence between document schemas and
893 namespaces. It also shows the one-to-one correspondence between files and schema
894 modules. As shown in figure 3-1, there are two types of schema in the UBL library –
895 document schema and schema modules. Document schemas are always in their own
896 namespace. Schema modules may be in a document schema namespace as in the case of
897 internal schema modules, or in a separate namespace as in the `ubl:udt`, `ubl:sdt`,
898 `ubl:cbc`, `ubl:cac`, `ubl:cl`, `ubl:cct`, and `ubl:ccts` schema modules. Both types of
899 schema modules are conformant with W3C XSD.

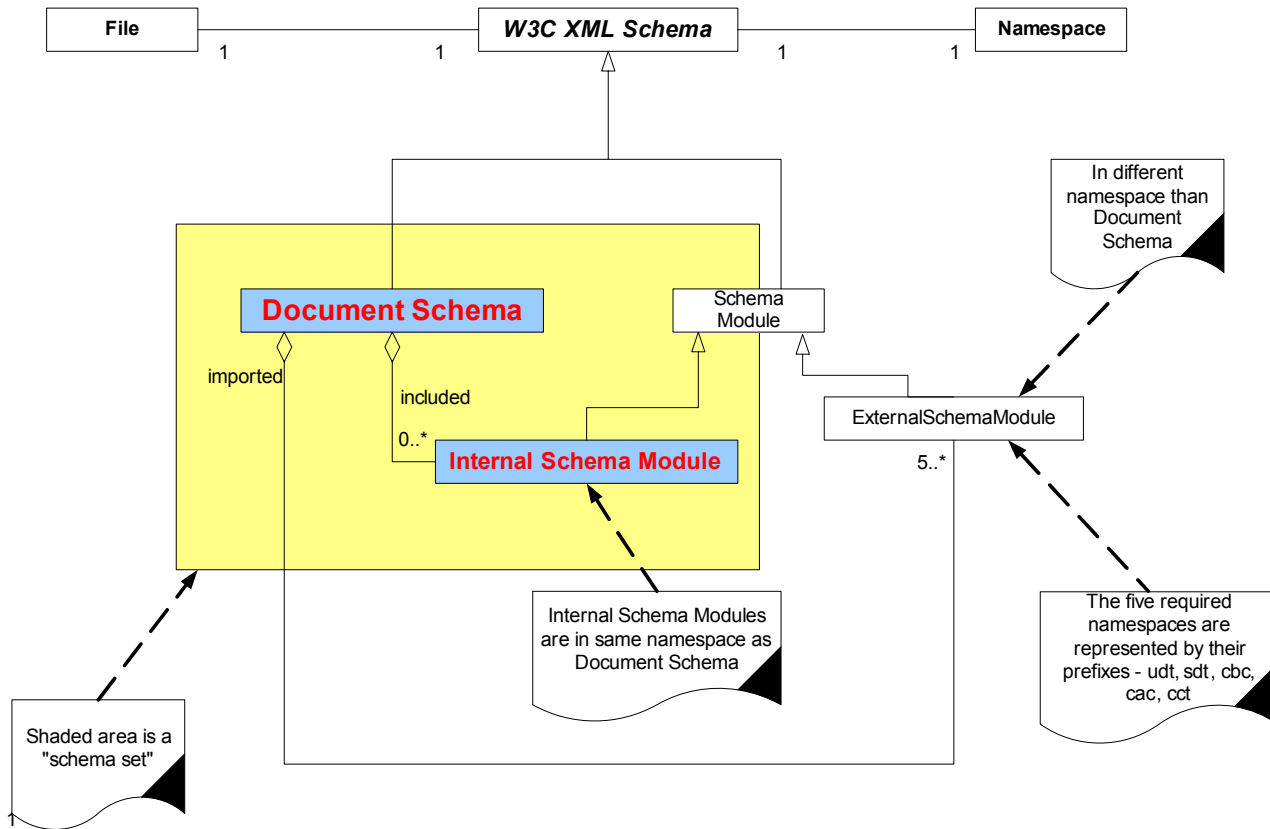
900 A namespace is an indivisible grouping of types. A “piece” of a namespace can never be
901 used without all its pieces. For larger namespaces, schema modules – internal schema
902 modules – may be defined. UBL document schemas may have zero or more internal
903 modules that they include. The document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

905 **[Definition] Internal schema module –**

906 A schema that is part of a schema set within a specific namespace.

907

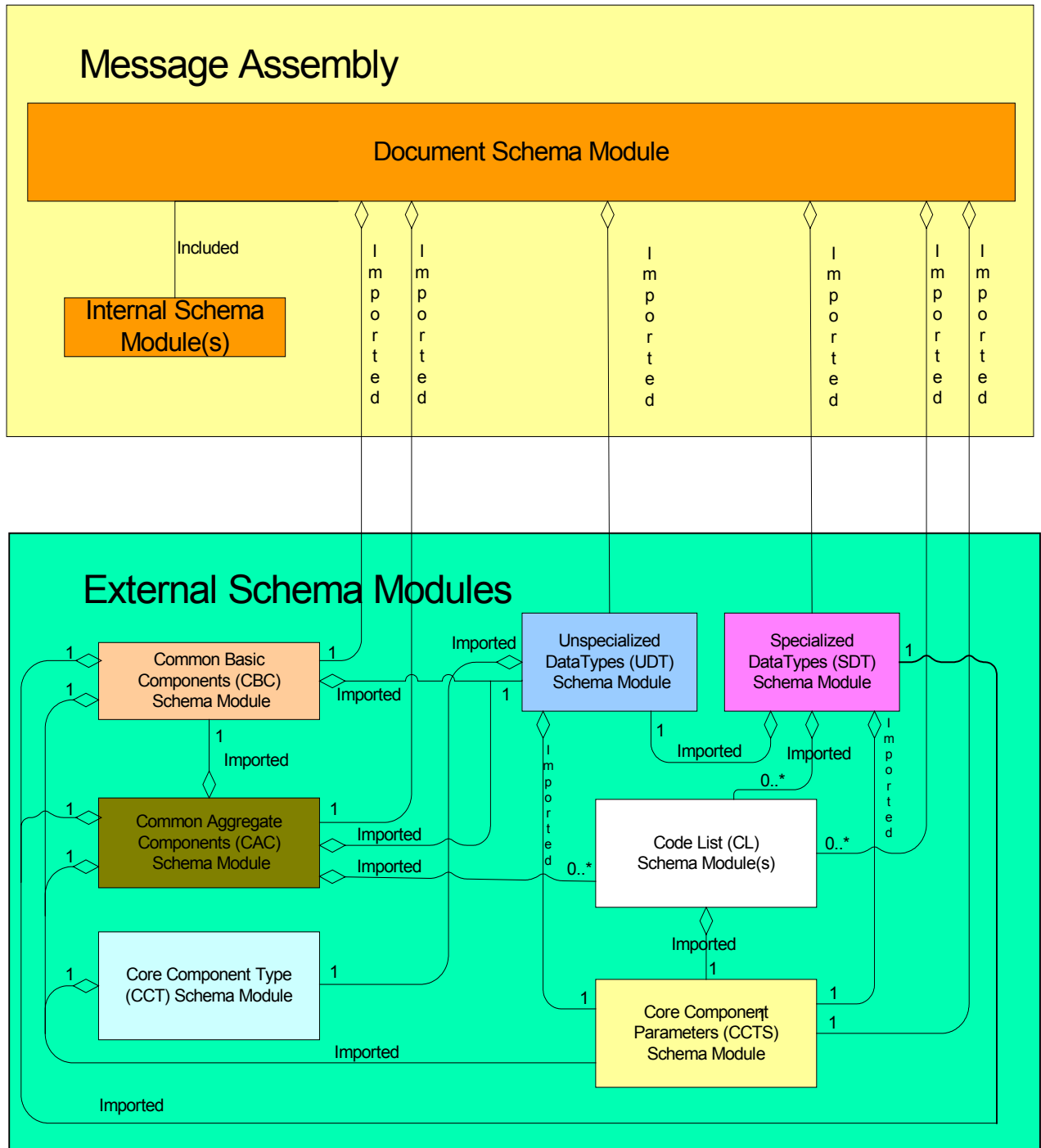
908 **Figure 3-1. UBL Schema Modularity Model**



udt = Unspecialized Datatype, sdt = Specialized Datatype, cbc = Common Basic Components, cac = Common Aggregate Components
cct = Core Component Type

909
910

911 Another way to visualize the structure is by example. Figure 3-2 depicts instances of the
912 various schema modules from the previous diagram.



914
915

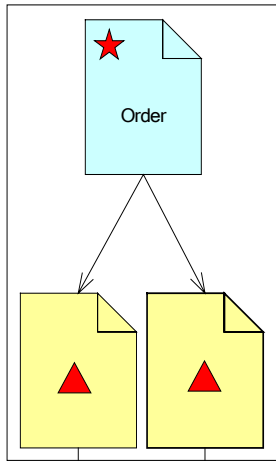
916 Figure 3-3 shows how the order and invoice document schemas import the
917 "CommonAggregateComponents Schema Module" and "CommonBasicComponents
918 Schema Module" external schema modules. It also shows how the order document

919 schema includes various internal modules – modules local to that namespace. The clear
920 boxes show how the various schema modules are grouped into namespaces.

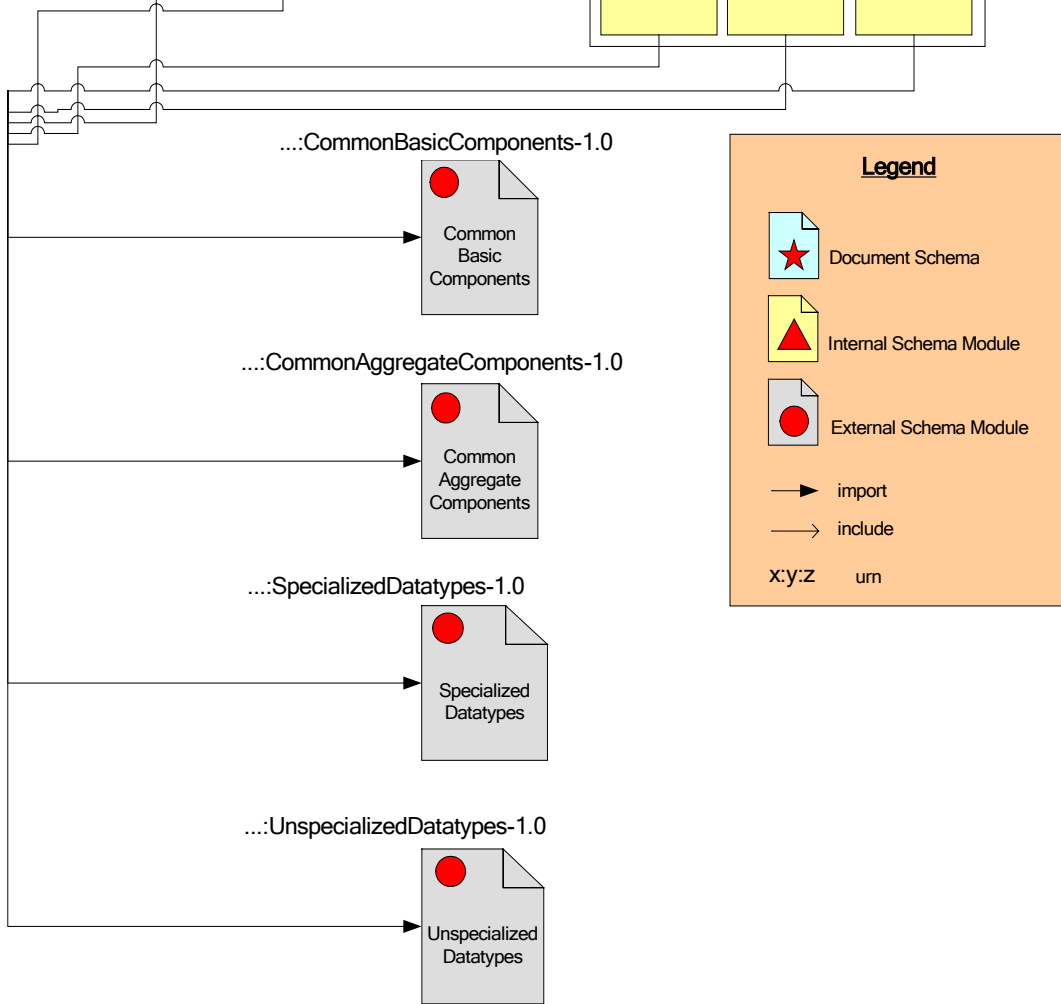
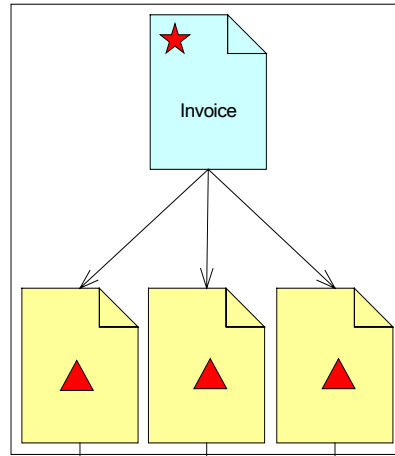
921 Any UBL schema module, be it a document schema or an internal module, may import
922 other document schemas from other namespaces.

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

urn:oasis:names:specification:ubl:schema:Order-1.0



...:Invoice-1.0



924

925 3.6.1.5 Limitations on Import

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

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

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

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

940 3.6.1.6 Module Conformance

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

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

945 3.6.2 Internal and External Schema Modules

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

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

951 3.6.3 Internal Schema Modules

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

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

957 UBL internal schema modules will necessarily have semantically meaningful names.
958 Internal schema module names will identify the parent schema module, the internal
959 schema module function, and the schema module itself.

960 [SSM7] Each UBL internal schema module MUST be named
961 {ParentSchemaModuleName}{InternalSchemaModuleFunction}{sc
962 hema module}

963 3.6.4 External Schema Modules

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

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

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

- 971 ◆ UBL CommonAggregateComponents
- 972 ◆ UBL CommonBasicComponents
- 973 ◆ UBL Code List(s)
- 974 ◆ CCTS Core Component Types
- 975 ◆ CCTS Unspecialized Datatypes
- 976 ◆ UBL Specialized Datatypes
- 977 ◆ CCTS Core Component Parameters

978 3.6.4.7 UBL CommonAggregateComponents Schema Module

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

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

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

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

993 **3.6.4.7.1 UBL CommonAggregateComponents Schema Module Namespace**

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

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

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

1000 [NMS8] The `ubl:CommonAggregateComponents` schema module MUST be
1001 represented by the token “cac”.

1002 **3.6.4.8 UBL CommonBasicComponents Schema Module**

1003 The UBL library will contain a wide variety of `ccts:BasicBusinessInformation`
1004 `Entities`. These `ccts:BasicBusinessInformationEntities` are based on
1005 `ccts:BasicBusinessInformationEntityProperties`. BBIE properties are
1006 reusable in multiple BBIEs. As defined in rule CTD1, each of these `ccts:Basic`
1007 `BusinessInformationEntityProperty` classes is defined as an
1008 `xsd:complexType`. Although some of these complex types may be used in only one
1009 UBL Schema, many will be reused in multiple UBL schema modules. To maximize reuse
1010 and standardization, all of the `ccts:BasicBusinessInformationEntity`
1011 `Property` `xsd:ComplexType` definitions that are used in multiple UBL schema
1012 modules will be aggregated into a 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 CoreComponentType 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 are
1043 not appropriate. Such restrictions will be applied through the application of datatypes.
1044 Accordingly, the `xsd:facet` feature must not be used in the `ccts:CCT` schema module.

1045 [SSM15] The `xsd:facet` feature MUST not be used in the `ccts:CoreComponent`
1046 `Type` schema module.

1047 **3.6.4.9.1 Core Component Type Schema Module Namespace**

1048 In keeping with the overall UBL namespace approach, a single namespace must be
1049 created for storing the `ccts:CoreComponentType` schema module.

1050 [NMS11] The `ccts:CoreComponentType` schema module MUST reside in its own
1051 namespace.

1052 To ensure consistency in expressing the `ccts:CoreComponentType` schema module, a
1053 normative token that will be used consistently in all UBL Schema must be defined.

1054 [NMS12] The `ccts:CoreComponentType` schema module namespace MUST be
1055 represented by the token “cct”.

1056 **3.6.4.10 CCTS Datatypes Schema Modules**

1057 The CCTS defines an authorized set of primary and secondary Representation Terms
1058 (`ccts:RepresentationTerms`) that describes the form of every `ccts:Business`
1059 `InformationEntity`. These `ccts:RepresentationTerms` are instantiated in the
1060 form of datatypes that are reusable in every UBL schema. The `ccts:Datatype` defines
1061 the set of valid values that can be used for its associated `ccts:BasicBusiness`
1062 `InformationEntity` Property. These datatypes may be specialized or unspecialized,
1063 that is to say restricted or unrestricted. We refer to these as `ccts:Unspecialized`
1064 `Datatypes` (even though they are technically `ccts:Datatypes`) or
1065 `ubl:SpecializedDatatypes`.

1066 ***3.6.4.10.1 CCTS UnspecializedDatatypes Schema Module***

1067 An external schema module consisting of a complex type definition for each
1068 `ccts:UnspecializedDatatype` is essential to maximize reusability.

1069 [SSM16] A schema module defining all `ccts:UnspecializedDatatypes` MUST
1070 be created.

1071 The normative name for the `ccts:UnspecializedDatatype` schema module will be
1072 based on its content.

1073 [SSM17] The `ccts:UnspecializedDatatype` schema module MUST be named
1074 “*ccts:UnspecializedDatatype Schema Module*”

1075 In keeping with the overall UBL namespace approach, a singular namespace must be
1076 created for storing the `ccts:UnspecializedDatatype` schema module.

1077 [NMS13] The `ccts:UnspecializedDatatype` schema module MUST reside in its
1078 own namespace.

1079 To ensure consistency in expressing the `ccts:UnspecializedDatatype` schema
1080 module, a normative token that will be used consistently in all UBL Schema must be
1081 defined.

1082 [NMS14] The `ccts:UnspecializedDatatype` schema module namespace MUST
1083 be represented by the token “udt”.

1084 ***3.6.4.10.2 UBL SpecializedDatatypes Schema Module***

1085 The `ubl:SpecializedDatatype` is defined by specifying restrictions on the
1086 `ccts:CoreComponentType` that forms the basis of the `ccts:Unspecialized`

1087 Datatype. To ensure the consistency of UBL specialized Datatypes
1088 (`ubl:SpecializedDatatypes`) with the UBL modularity and reuse goals requires
1089 creating a single schema module that defines all `ubl:SpecializedDatatypes`.

1090 [SSM18] A schema module defining all `ubl:SpecializedDatatypes` MUST be
1091 created.

1092 The `ubl:SpecializedDatatypes` schema module name must follow the UBL module
1093 naming approach.

1094 [SSM19] The `ubl:SpecializedDatatypes` schema module MUST be named
1095 "`ubl:SpecializedDatatypes` schema module"

1096 *3.6.4.10.3 UBL Specialized Datatypes Schema Module Namespace*

1097 In keeping with the overall UBL namespace approach, a singular namespace must be
1098 created for storing the `ubl:SpecializedDatatypes` schema module.

1099 [NMS15] The `ubl:SpecializedDatatypes` schema module MUST reside in its own
1100 namespace.

1101 To ensure consistency in expressing the `ubl:SpecializedDatatypes` schema
1102 module, a normative token that will be used in all UBL schemas must be defined.

1103 [NMS16] The `ubl:SpecializedDatatypes` schema module namespace MUST be
1104 represented by the token "sdt".

1105 *3.7 Annotation and Documentation*

1106 Annotation is an essential tool in understanding and reusing a schema. UBL, as an
1107 implementation of CCTS, requires an extensive amount of annotation to provide all
1108 necessary metadata required by the CCTS specification. Each construct declared or
1109 defined within the UBL library contains the requisite associated metadata to fully
1110 describe its nature and support the CCTS requirement. Accordingly, UBL schema
1111 metadata for each construct will be defined in the UBL core component parameters
1112 schema.

1113 *3.7.1 Schema Annotation*

1114 Although the UBL schema annotation is necessary, its volume results in a considerable
1115 increase in the size of the UBL schemas with undesirable performance impacts. To
1116 address this issue, two normative schema will be developed for each UBL schema. A
1117 fully annotated schema will be provided to facilitate greater understanding of the schema
1118 module and its components, and to meet the CCTS metadata requirements. A schema
1119 devoid of annotation will also be provided that can be used at run-time if required to meet
1120 processor resource constraints.

1121 [GXS2] UBL MUST provide two normative schemas for each transaction. One
1122 schema shall be fully annotated. One schema shall be a run-time schema
1123 devoid of documentation.

1124 3.7.2 Embedded documentation

1125 The information about each UBL `ccts:BusinessInformationEntity` is in the UBL
1126 spreadsheet models. UBL spreadsheets contain all necessary information to produce fully
1127 annotated Schemas. Fully annotated Schemas are valuable tools to implementers to assist
1128 in understanding the nuances of the information contained therein. UBL annotations will
1129 consist of information currently required by Section 7 of the CCTS and supplemented by
1130 metadata from the UBL spreadsheet models.

1131 The absence of an optional annotation inside the structured set of annotations in the
1132 documentation element implies the use of the default value. For example, there are
1133 several annotations relating to context such as `ccts:BusinessContext` or
1134 `ccts:IndustryContext` whose absence implies that their value is "all contexts".

1135 The following rules describe the documentation requirements for each
1136 `ubl:SpecializedDatatype` and `ubl:UnspecializedDatatype` definition.

1137 [DOC1] The `xsd:documentation` element for every Datatype MUST contain a
1138 structured set of annotations in the following sequence and pattern:

- 1139 • `ComponentType` (mandatory): The type of component to which the object
1140 belongs. For Datatypes this must be "DT".
- 1141 • `DictionaryEntryName` (mandatory): The official name of a Datatype.
- 1142 • `Version` (optional): An indication of the evolution over time of the Datatype.
- 1143 • `Definition`(mandatory): The semantic meaning of a Datatype.
- 1144 • `ObjectClassQualifier` (optional): The qualifier for the object class.
- 1145 • `ObjectClass`(optional): The Object Class represented by the Datatype.
- 1146 • `RepresentationTerm` (mandatory): A Representation Term is an element of
1147 the name which describes the form in which the property is represented.
- 1148 • `DataQualifier` (optional): semantically meaningful name that
1149 differentiates the Datatype from its underlying Core Component Type.
- 1150 • `Data Type` (optional): Defines the underlying Core Component Type.

1151 [DOC2] A Datatype definition MAY contain one or more Content Component
1152 Restrictions to provide additional information on the relationship between the
1153 Datatype and its corresponding Core Component Type. If used the Content
1154 Component Restrictions must contain a structured set of annotations in the
1155 following patterns:
1156

1157 • RestrictionType (mandatory): Defines the type of format restriction that
1158 applies to the Content Component.
1159 • RestrictionValue (mandatory): The actual value of the format restriction that
1160 applies to the Content Component.
1161 • ExpressionType (optional): Defines the type of the regular expression of the
1162 restriction value.
1163

1164 [DOC3] A Datatype definition MAY contain one or more Supplementary Component
1165 Restrictions to provide additional information on the relationship between the
1166 Datatype and its corresponding Core Component Type. If used the
1167 Supplementary Component Restrictions must contain a structured set of
1168 annotations in the following patterns:
1169 • SupplementaryComponentName (mandatory): Identifies the
1170 Supplementary Component on which the restriction applies.
1171 • RestrictionValue (mandatory, repetitive): The actual value(s) that is
1172 (are) valid for the Supplementary Component

1173 The following rule describes the documentation requirements for each `ccts:Basic`
1174 `BusinessInformationEntity` definition.

1175 [DOC4] The `xsd:documentation` element for every Basic Business Information
1176 Entity MUST contain a structured set of annotations in the following patterns:
1177 • ComponentType (mandatory): The type of component to which the object
1178 belongs. For Basic Business Information Entities this must be “BBIE”.
1179 • DictionaryEntryName (mandatory): The official name of a Basic Business
1180 Information Entity.
1181 • Version (optional): An indication of the evolution over time of the Basic
1182 Business Information Entity.
1183 • Definition(mandatory): The semantic meaning of a Basic Business
1184 Information Entity.
1185 • Cardinality(mandatory): Indication whether the Basic Business Information
1186 Entity represents a not-applicable, optional, mandatory and/or repetitive
1187 characteristic of the Aggregate Business Information Entity.
1188 • ObjectClassQualifier (optional): The qualifier for the object class.
1189 • ObjectClass(mandatory): The Object Class containing the Basic Business
1190 Information Entity.
1191 • PropertyTermQualifier (optional): A qualifier is a word or words which help
1192 define and differentiate a Basic Business Information Entity.

- 1193 • PropertyTerm(mandatory): Property Term represents the distinguishing
- 1194 characteristic or Property of the Object Class and shall occur naturally in the
- 1195 definition of the Basic Business Information Entity.
- 1196 • RepresentationTerm (mandatory): A Representation Term describes the
- 1197 form in which the Basic Business Information Entity is represented.
- 1198 • DataTypeQualifier (optional): semantically meaningful name that
- 1199 differentiates the Datatype of the Basic Business Information Entity from its
- 1200 underlying Core Component Type.
- 1201 • DataType (mandatory): Defines the Datatype used for the Basic Business
- 1202 Information Entity.
- 1203 • AlternativeBusinessTerms (optional): Any synonym terms under which the
- 1204 Basic Business Information Entity is commonly known and used in the
- 1205 business.
- 1206 • Examples (optional): Examples of possible values for the Basic Business
- 1207 Information Entity.

1208 The following rule describes the documentation requirements for each
 1209 `ccts:AggregateBusinessInformationEntity` definition.

- 1210 [DOC5] The `xsd:documentation` element for every Aggregate Business
- 1211 Information Entity MUST contain a structured set of annotations in the
- 1212 following sequence and pattern:
- 1213 • ComponentType (mandatory): The type of component to which the object
 - 1214 belongs. For Aggregate Business Information Entities this must be “ABIE”.
 - 1215 • DictionaryEntryName (mandatory): The official name of the Aggregate
 - 1216 Business Information Entity .
 - 1217 • Version (optional): An indication of the evolution over time of the
 - 1218 Aggregate Business Information Entity.
 - 1219 • Definition(mandatory): The semantic meaning of the Aggregate Business
 - 1220 Information Entity.
 - 1221 • ObjectClassQualifier (optional): The qualifier for the object class.
 - 1222 • ObjectClass(mandatory): The Object Class represented by the Aggregate
 - 1223 Business Information Entity.
 - 1224 • AlternativeBusinessTerms (optional): Any synonym terms under which the
 - 1225 Aggregate Business Information Entity is commonly known and used in the
 - 1226 business.

1227 The following rule describes the documentation requirements for each
 1228 `ccts:AssociationBusinessInformationEntity` definition.

1229 [DOC6] The `xsd:documentation` element for every Association Business
 1230 Information Entity element declaration MUST contain a structured set of
 1231 annotations in the following sequence and pattern:

- 1232 • `ComponentType` (mandatory): The type of component to which the object
 1233 belongs. For Association Business Information Entities this must be “ASBIE”.
- 1234 • `DictionaryEntryName` (mandatory): The official name of the Association
 1235 Business Information Entity.
- 1236 • `Version` (optional): An indication of the evolution over time of the
 1237 Association Business Information Entity.
- 1238 • `Definition`(mandatory): The semantic meaning of the Association Business
 1239 Information Entity.
- 1240 • `Cardinality`(mandatory): Indication whether the Association Business
 1241 Information Entity represents an optional, mandatory and/or repetitive
 1242 association.
- 1243 • `ObjectClass`(mandatory): The Object Class containing the Association
 1244 Business Information Entity.
- 1245 • `PropertyTermQualifier` (optional): A qualifier is a word or words which help
 1246 define and differentiate the Association Business Information Entity.
- 1247 • `PropertyTerm`(mandatory): Property Term represents the Aggregate
 1248 Business Information Entity contained by the Association Business
 1249 Information Entity.
- 1250 • `AssociatedObjectClassQualifier` (optional): Associated Object Class
 1251 Qualifiers describe the 'context' of the relationship with another ABIE. That is,
 1252 it is the role the contained Aggregate Business Information Entity plays within
 1253 its association with the containing Aggregate Business Information Entity.
- 1254 • `AssociatedObjectClass` (mandatory); Associated Object Class is the Object
 1255 Class at the other end of this association. It represents the Aggregate Business
 1256 Information Entity contained by the Association Business Information Entity.

1257 The following rule describes the documentation requirements for each
 1258 `ccts:CoreComponentType` definition.

1259 [DOC7] The `xsd:documentation` element for every Core Component Type MUST
 1260 contain a structured set of annotations in the following sequence and pattern:

- 1261 • `ComponentType` (mandatory): The type of component to which the object
 1262 belongs. For Core Component Types this must be “CCT”.
- 1263 • `DictionaryEntryName` (mandatory): The official name of the Core
 1264 Component Type, as defined by [CCTS].

- 1265 • Version (optional): An indication of the evolution over time of the Core
1266 Component Type.
- 1267 • Definition (mandatory): The semantic meaning of the Core Component
1268 Type, as defined by [CCTS].
- 1269 • ObjectClass (mandatory): The Object Class represented by the Core
1270 Component Type, as defined by [CCTS].
- 1271 • PropertyTerm (mandatory): The Property Term represented by the Core
1272 Component Type, as defined by [CCTS].

1273 4 Naming Rules

1274 The rules in this section make use of the following special concepts related to XML
1275 elements and attributes:

- 1276 ◆ Top-level element: An element that encloses a whole UBL business message.
1277 Note that UBL business messages might be carried by messaging transport
1278 protocols that themselves have higher-level XML structure. Thus, a UBL top-
1279 level element is not necessarily the root element of the XML document that
1280 carries it.

- 1281 ◆ Lower-level element: An element that appears inside a UBL business
1282 message. Lower-level elements consist of intermediate and leaf level.
 - 1283 ○ Intermediate element: An element not at the top level that is of a
1284 complex type, only containing other elements and attributes.

 - 1285 ○ Leaf element: An element containing only character data (though it
1286 may also have attributes). Note that, because of the XSD
1287 mechanisms involved, a leaf element that has attributes must be
1288 declared as having a complex type, but a leaf element with no
1289 attributes may be declared with either a simple type or a complex
1290 type.

- 1291 ◆ Common attribute: An attribute that has identical meaning on the multiple
1292 elements on which it appears. A common attribute might or might not
1293 correspond to an XSD global attribute.

1294 4.1 General Naming Rules

1295 The CCTS contains specific Internal Organization for Standardization (ISO)/International
1296 Electrotechnical Commission (IEC) Technical Specification 11179 Information
1297 technology -- Metadata registries (MDR) based naming rules for each CCTS construct.
1298 The UBL component library, as a syntax-neutral representation, is fully conformant to
1299 those rules. The UBL syntax-specific XSD instantiation of the UBL component library—
1300 in some cases—refines the CCTS naming rules to leverage the capabilities of XML and
1301 XSD. Specifically, truncation rules are applied to allow for reuse of element names
1302 across parent element environments and to maintain brevity and clarity.

1303 In keeping with CCTS, UBL will use English as its normative language. If the UBL
1304 Library is translated into other languages for localization purposes, these additional
1305 languages might require additional restrictions. Such restrictions are expected be
1306 formulated as additional rules and published as appropriate.

1307 [GNR1] UBL XML element, attribute and type names MUST be in the English
1308 language, using the primary English spellings provided in the Oxford English
1309 Dictionary.

1310 UBL fully supports the concepts of data standardization contained in ISO 11179. CCTS,
1311 as an implementation of 11179, furthers its basic tenets of data standardization into
1312 higher-level constructs as expressed by the `ccts:DictionaryEntryNames` of those
1313 constructs – such as those for `ccts:BasicBusinessInformationEntities` and
1314 `ccts:AggregateBusinessInformationEntities`. Since UBL is an
1315 implementation of CCTS, UBL uses CCTS dictionary entry names as the basis for UBL
1316 XML schema construct names. UBL converts these `ccts:DictionaryEntryNames`
1317 into UBL XML schema construct names using strict transformation rules.

1318 [GNR2] UBL XML element, attribute and type names MUST be consistently derived
1319 from CCTS conformant dictionary entry names.

1320 The ISO 11179 specifies—and the CCTS uses—periods, spaces, other separators, and
1321 characters not allowed by W3C XML. These separators and characters are not
1322 appropriate for UBL XML component names.

1323 [GNR3] UBL XML element, attribute and type names constructed from
1324 `ccts:DictionaryEntryNames` MUST NOT include periods, spaces,
1325 other separators, or characters not allowed by W3C XML 1.0 for XML names.

1326 Acronyms and abbreviations impact on semantic interoperability, and as such are to be
1327 avoided to the maximum extent practicable. Since some abbreviations will inevitably be
1328 necessary, UBL will maintain a normative list of authorized acronyms and abbreviations.
1329 Appendix B provides the current list of permissible acronyms, abbreviations and word
1330 truncations. The intent of this restriction is to facilitate the use of common semantics and
1331 greater understanding. Appendix B is a living document and will be updated to reflect
1332 growing requirements.

1333 [GNR4] UBL XML element, attribute, and simple and complex type names MUST
1334 NOT use acronyms, abbreviations, or other word truncations, except those in
1335 the list of exceptions published in Appendix B.

1336 UBL does not desire a proliferation of acronyms and abbreviations. Appendix B is an
1337 exception list and will be tightly controlled by UBL. Any additions will only occur after
1338 careful scrutiny to include assurance that any addition is critically necessary, and that any
1339 addition will not in any way create semantic ambiguity.

1340 [GNR5] Acronyms and abbreviations MUST only be added to the UBL approved
1341 acronym and abbreviation list after careful consideration for maximum
1342 understanding and reuse.

1343 Once an acronym or abbreviation has been approved, it is essential to ensuring semantic
1344 clarity and interoperability that the acronym or abbreviation is ***always*** used.

1345 [GNR6] The acronyms and abbreviations listed in Appendix B MUST always be used.

1346 Generally speaking, the names for UBL XML constructs must always be singular. The
1347 only exception permissible is where the concept itself is pluralized.

1348 [GNR7] UBL XML element, attribute and type names MUST be in singular form
1349 unless the concept itself is plural.

1350
1351 Example:

1352
1353 Terms

1354 XML is case sensitive. Consistency in the use of case for a specific XML component
1355 (element, attribute, type) is essential to ensure every occurrence of a component is treated
1356 as the same. This is especially true in a business-based data-centric environment such as
1357 what is being addressed by UBL. Additionally, the use of visualization mechanisms such
1358 as capitalization techniques assist in ease of readability and ensure consistency in
1359 application and semantic clarity. The ebXML architecture document specifies a standard
1360 use of upper and lower camel case for expressing XML elements and attributes
1361 respectively.¹² UBL will adhere to the ebXML standard. Specifically, UBL element and
1362 type names will be in UpperCamelCase (UCC).

1363 [GNR8] The UpperCamelCase (UCC) convention MUST be used for naming elements
1364 and types.

1365 Example:

1366
1367 CurrencyBaseRate
1368 CityNameType
1369

1370

1371 UBL attribute names will be in lowerCamelCase (LCC).

1372 [GNR9] The lowerCamelCase (LCC) convention MUST be used for naming attributes.

1373 Example:
1374

¹² *ebXML, ebXML Technical Architecture Specification v1.0.4, 16 February 2001*

1375
 1376 amountCurrencyCodeListVersionID
 1377 characterSetCode

1378 4.2 Type Naming Rules

1379 UBL identifies several categories of naming rules for types, namely for complex types
 1380 based on Aggregate Business Information Entities, Basic Business Information Entities,
 1381 Primary Representation Terms, Secondary Representation Terms and the Core
 1382 Component Types.

1383 Each of these CCTS constructs have a `ccts:DictionaryEntryName` that is a fully
 1384 qualified construct based on ISO 11179. As such, these names convey explicit semantic
 1385 clarity with respect to the data being described. Accordingly, these `ccts:Dictionary`
 1386 `EntryNames` provide a mechanism for ensuring that UBL `xsd:complexType` names are
 1387 semantically unambiguous, and that there are no duplications of UBL type names for
 1388 different `xsd:type` constructs.

1389 4.2.1 Complex Type Names for CCTS Aggregate Business 1390 Information Entities

1391 UBL `xsd:complexType` names for `ccts:AggregateBusinessInformation`
 1392 `Entities` will be derived from their dictionary entry name by removing separators to
 1393 follow general naming rules, and appending the suffix “Type” to replace the word
 1394 “Details.”

1395 [CTN1] A UBL `xsd:complexType` name based on an `ccts:Aggregate`
 1396 `BusinessInformationEntity` MUST be the `ccts:Dictionary`
 1397 `EntryName` with the separators removed and with the “Details” suffix
 1398 replaced with “Type”.

1399 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>

1401

1402 4.2.2 Complex Type Names for CCTS Basic Business Information 1403 Entity Properties

1404 All `ccts:BasicBusinessInformationEntityProperties` are reusable across
 1405 multiple `ccts:BasicBusinessInformationEntities`. The CCTS does not specify,
 1406 but implies, that `ccts:BasicBusinessInformationEntityProperty` names are
 1407 the reusable property term and representation term of the family of

1408 ccts:BasicBusinessInformationEntities that are based on it. The UBL
1409 xsd:complexType names for ccts:BasicBusinessInformationEntity
1410 properties will be derived from the shared property and representation terms portion
1411 of the dictionary entry names in which they appear by removing separators to follow
1412 general naming rules, and appending the suffix “Type”.

1413 [CTN2] A UBL xsd:complexType name based on a ccts:BasicBusiness
1414 InformationEntityProperty MUST be the ccts:Dictionary
1415 EntryName shared property term and its qualifiers and representation term of
1416 the shared ccts:BasicBusinessInformationEntity, with the
1417 separators removed and with the “Type” suffix appended after the
1418 representation term.

1419 **Example:**

```
1420 <!--===== Basic Business Information Entity Type Definitions =====>  
1421 ->  
1422 <xsd:complexType name="ChargeIndicatorType">  
1423 ...  
1424 </xsd:complexType>
```

1425

1426 4.2.3 Complex Type Names for CCTS Unspecialized Datatypes

1427 UBL xsd:complexType names for ccts:UnspecializedDatatypes will be
1428 derived from its dictionary entry name by removing separators to follow general naming
1429 rules, and appending the suffix “Type”.

1430 [CTN3] A UBL xsd:complexType for a cct:UnspecializedDatatype used in
1431 the UBL model MUST have the name of the corresponding
1432 ccts:CoreComponentType, with the separators removed and with the
1433 “Type” suffix appended.

1434 **Example:**

```
1435 <!-- ===== Primary Representation Term: AmountType ===== -->  
1436 <xsd:complexType name="AmountType">  
1437 ...  
1438 </xsd:complexType>
```

1439 UBL xsd:complexType names for ccts:UnspecializedDatatypes based on
1440 ccts:SecondaryRepresentationTerms will be derived from the
1441 ccts:SecondaryRepresentationTerm dictionary entry name by removing separators to
1442 follow general naming rules, and appending the suffix “Type”.

1443 [CTN4] A UBL xsd:complexType for a cct:UnspecializedDatatype based on
1444 a ccts:SecondaryRepresentationTerm used in the UBL model MUST
1445 have the name of the corresponding ccts:SecondaryRepresentation
1446 Term, with the separators removed and 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
1461
1462
1463

```
<!-- ===== CCT: QuantityType ===== -->  
<xsd:complexType name="QuantityType">  
  ...  
</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` `xsd: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:BasicBusiness`
1472 `InformationEntities`, and `ccts:AssociationBusinessInformation`
1473 `Entities`. UBL element names will reflect this relationship in full conformance with
1474 ISO11179 element naming rules.

1475 4.3.1 Element Names for CCTS Aggregate Business Information 1476 Entities

1477 [ELN1] A UBL global element name based on a `ccts:ABIE` MUST be the same as
1478 the name of the corresponding `xsd:complexType` to which it is bound,
1479 with the word “Type” removed.

1480 **Example:**

1481 For a `ccts:AggregateBusinessInformationEntity` of `Party`. Details,
1482 Rule CTN1 states that the `Party`. Details object class becomes `PartyType`
1483 `xsd:ComplexType`. Rule ELD3 states that for the `PartyType` `xsd:complexType`,
1484 a corresponding global element must be declared. Rule ELN1 states that the name of
1485 this corresponding global element must be `Party`.

1486
1487
1488
1489
1490
1491
1492
1493
1494
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
1530

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

  <xsd:annotation>

    <!--Documentation goes here-->    </xsd:annotation>

    <xsd:sequence>

      <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0"
maxOccurs="1">

        ...

      </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>
```

1531 4.3.2 Element Names for CCTS Basic Business Information Entity 1532 Properties

1533 The same naming concept used for `ccts:AggregateBusinessInformation`
1534 Entities applies to `ccts:BasicBusinessInformationEntityProperty`.

1535 [ELN2] A UBL global element name based on an unqualified `ccts:BBIEProperty`
1536 MUST be the same as the name of the corresponding `xsd:complexType` to
1537 which it is bound, with the word “Type” removed.

1538 **Example:**

```
1539 <!--==== Basic Business Information Entity Type Definitions =====>  
1540 ->  
1541 <xsd:complexType name="ChargeIndicatorType">  
1542 ...  
1543 </xsd:complexType>  
1544 ...  
1545 <!--==== Basic Business Information Entity Property Element  
1546 Declarations =====>  
1547 <xsd:element name="ChargeIndicator" type="ChargeIndicatorType"/>
```

1548 4.3.3 Element Names for CCTS Association Business Information 1549 Entities

1550 A `ccts:AssociationBusinessInformationEntity` is not a class like
1551 `ccts:AggregateBusinessInformationEntities` and like `ccts:Basic`
1552 `BusinessInformationEntityProperties` that are reused as `ccts:Basic`
1553 `BusinessInformationEntities`. Rather, it is an association between two classes.
1554 As such, an element representing the `ccts:AssociationBusinessInformation`
1555 `Entity` does not have its own unique `xsd:ComplexType`. Instead, when an element
1556 representing a `ccts:AssociationBusinessInformationEntity` is declared, the
1557 element is bound to the `xsd:complexType` of its associated `ccts:Aggregate`
1558 `BusinessInformationEntity`.

1559 [ELN3] A UBL global element name based on a qualified `ccts:ASBIE` MUST be the
1560 `ccts:ASBIE` dictionary entry name property term and its qualifiers; and the
1561 object class term and qualifiers of its associated `ccts:ABIE`. All
1562 `ccts:DictionaryEntryName` separators MUST be removed. Redundant
1563 words in the `ccts:ASBIE` property term or its qualifiers and the associated
1564 `ccts:ABIE` object class term or its qualifiers MUST be dropped.

1566 [ELN4] A UBL global element name based on a qualified `ccts:BBIEProperty`
1567 MUST be the same as the name of the corresponding `xsd:complexType` to
1568 which it is bound, with the qualifier prefixed and with the word "Type"
1569 removed.

1570 4.4 Attribute Naming Rules

1571 UBL, as a transactional based XML exchange format, has chosen to significantly restrict
1572 the use of attributes. This restriction is in keeping with the fact that attribute usage is
1573 relegated to supplementary components only; all “primary” business data appears
1574 exclusively in element content.

1575 [ATN1] Each `ccts:SupplementaryComponent` `xsd:attribute` "name" MUST be
 1576 the Dictionary Entry Name object class, property term and representation term
 1577 of the `ccts:SupplementaryComponent` with the separators removed.

1578 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>

1579 UBL currently truncates the `ccts:SupplementaryComponent` `xsd:attribute`
 1580 name. Specifically, if the object class of the `ccts:SupplementaryComponent` is the
 1581 same as the object class of `ccts:CoreComponentType` or `Datatype` to which it relates,
 1582 then the object class term is dropped from the `xsd:attribute` name.

1583 Example:

1584

<code>ccts:SupplementaryComponent</code>	<code>ubl:attribute</code>
<code>Code. Name</code>	<code>name</code>

1585

1586 5 Declarations and Definitions

1587 In W3C XML Schema, elements are defined in terms of complex or simple types and
1588 attributes are defined in terms of simple types. The rules in this section govern the
1589 consistent structuring of these type constructs and the manner for unambiguously and
1590 thoroughly documenting them in the UBL Library.

1591 5.1 Type Definitions

1592 5.1.1 General Type Definitions

1593 Since UBL elements and types are intended to be reusable, all types must be named. This
1594 permits other types to establish elements that reference these types, and also supports the
1595 use of extensions for the purposes of versioning and customization.

1596 [GTD1] All types **MUST** be named.

1597 **Example:**

```
1598 <xsd:complexType name="QuantityType">  
1599   ...  
1600 </xsd:complexType>  
1601
```

1602 UBL disallows the use of `xsd:anyType`, because this feature permits the introduction of
1603 potentially unknown types into an XML instance. UBL intends that all constructs within
1604 the instance be described by the schemas describing that instance - `xsd:anyType` is seen
1605 as working counter to the requirements of interoperability. In consequence, particular
1606 attention is given to the need to enable meaningful validation of the UBL document
1607 instances. Were it not for this, `xsd:anyType` might have been allowed.

1608 [GTD2] The `xsd:anyType` **MUST NOT** be used.

1609 5.1.2 Simple Types

1610 The Core Components Technical Specification provides a set of constructs for the
1611 modeling of basic data, Core Component Types. These are represented in UBL with a
1612 library of complex types, with the effect that most "simple" data is represented as
1613 property sets defined according to the CCTs, made up of content components and
1614 supplementary components. In most cases, the supplementary components are expressed
1615 as XML attributes, the content component becomes element content, and the CCT is
1616 represented with an `xsd:complexType`. There are exceptions to this rule in those cases
1617 where all of a CCTs properties can be expressed without the use of attributes. In these
1618 cases, an `xsd:simpleType` is used.

1619 [STD1] For every `ccts:CCT` whose supplementary components map directly onto the
1620 properties of a built-in `xsd:Datatype`, the `ccts:CCT` MUST be defined as
1621 a named `xsd:simpleType` in the `ccts:CCT` schema module.

1622 **Example:**

```
1623 <!-- ===== CCT: DateTimeType ===== -->  
1624 <xsd:simpleType name="DateTimeType">  
1625   ...  
1626   <xsd:restriction base="cct:DateTimeType"/>  
1627 </xsd:simpleType>
```

1628 5.1.3 Complex Types

1629 Since even simple datatypes are modeled as property sets in most cases, the XML
1630 expression of these models primarily employs `xsd:complexType`. To facilitate reuse,
1631 versioning, and customization, all complex types are named. In the UBL model,
1632 `ccts:AggregateBusinessInformationEntities` are considered classes(objects) .

1633 [CTD1] For every class identified in the UBL model, a named `xsd:complexType`
1634 MUST be defined.

1635 **Example:**

```
1636 <xsd:complexType name="BuildingNameType">  
1637   ...  
1638   ...  
1639   ...  
1640 </xsd:complexType>
```

1641 5.1.3.1 Aggregate Business Information Entities

1642 The relationship expressed by an Aggregate Business Information Entity is not directly
1643 represented with a class. Instead, this relationship is captured in UBL with a containment
1644 relationship, expressed in the content model of the parent object's type with a sequence
1645 of elements. (Sequence facilitates the use of `xsd:extension` for versioning and
1646 customization.) The members of the sequence – elements which are themselves defined
1647 by reference to complex types – are the properties of the containing type.

1648 [CTD2] Every `ccts:ABIE` `xsd:complexType` definition content model MUST
1649 use the `xsd:sequence` element with appropriate global element references,
1650 or local element declarations in the case of ID and Code, to reflect each
1651 property of its class as defined in the corresponding UBL model.

1652 **Example:**

```
1653 <xsd:complexType name="AddressType">  
1654   ...  
1655   ...  
1656   ...  
1657   <xsd:sequence>
```

1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

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

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

1674 5.1.3.2 Basic Business Information Entities

1675 All `ccts:BasicBusinessInformationEntities`, in accordance with the Core
1676 Components Technical Specification, always have a representation term. This may be a
1677 primary or secondary representation term. Representation terms describe the structural
1678 representation of the BBIE. These representation terms are expressed in the UBL Model
1679 as Unspecialized Datatypes bound to a Core Component Type that describes their
1680 structure. In addition to the unspecialized Datatypes defined in CCTS, UBL has defined a
1681 set of Specialized Datatypes that are derived from the CCTS unspecialized
1682 Datatypes. There are a set of rules concerning the way these relationships are expressed in
1683 the UBL XML library. As discussed above, `ccts:BasicBusinessInformation`
1684 `EntityProperties` are represented with complex types. Within these are
1685 simpleContent elements that extend the Datatypes.

1686 [CTD3] Every `ccts:BBIEProperty xsd:complexType` definition content model
1687 MUST use the `xsd:simpleContent` element.

1688
1689 [CTD4] Every `ccts:BBIEProperty xsd:complexType` content model
1690 `xsd:simpleContent` element MUST consist of an `xsd:extension`
1691 element.

1692
1693 [CTD5] Every `ccts:BBIEProperty xsd:complexType` content model `xsd:base`
1694 attribute value MUST be the `ccts:CCT` of the unspecialized or specialized
1695 UBL Datatype as appropriate.

1696 Example:

1697
1698
1699
1700
1701

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


1702 5.1.3.3 Datatypes

1703 There is a direct one-to-one relationship between `ccts:CoreComponentTypes` and
1704 `ccts:PrimaryRepresentationTerms`. Additionally, there are several
1705 `ccts:SecondaryRepresentationTerms` that are subsets of their parent
1706 `ccts:PrimaryRepresentationTerm`. The total set of `ccts:Representation`
1707 `Terms` by their nature represent `ccts:Datatypes`. Specifically, for each
1708 `ccts:PrimaryRepresentationTerm` or `ccts:SecondaryRepresentationTerm`,
1709 a `ccts:UnspecializedDatatype` exists. In the UBL XML Library, these
1710 `ccts:UnspecializedDatatypes` are expressed as complex or simple types that are of
1711 the type of its corresponding `ccts:CoreComponentType`.

1712 [CTD6] For every Datatype used in the UBL model, a named `xsd:complexType` or
1713 `xsd:simpleType` MUST be defined.

1714 5.1.3.3.1 Unspecialized Datatypes

1715 The `ccts:UnspecializedDatatypes` reflect the instantiation of the `ccts:Core`
1716 `ComponentTypes`. Each `ccts:UnspecializedDatatype` declaration is based on its
1717 corresponding qualified `ccts:CoreComponentType` and represents either a primary or
1718 secondary representation term.

1719 [CTD7] Every unspecialized Datatype must be based on a `ccts:CCT` represented in
1720 the CCT schema module, and must represent an approved primary or
1721 secondary representation term identified in the CCTS.

1722 [CTD8] Each unspecialized Datatype `xsd:complexType` must be based on its
1723 corresponding CCT `xsd:complexType`.

1724 [CTD9] Every unspecialized Datatype that represents a primary representation term
1725 whose corresponding `ccts:CCT` is defined as an `xsd:simpleType` MUST
1726 also be defined as an `xsd:simpleType` and MUST be based on the same
1727 `xsd:simpleType`.

1728 [CTD10] Every unspecialized Datatype that represents a secondary representation term
1729 whose corresponding `ccts:CCT` is defined as an `xsd:simpleType` MUST
1730 also be defined as an `xsd:simpleType` and MUST be based on the same
1731 `xsd:simpleType`.

1732 [CTD11] Each unspecialized Datatype `xsd:complexType` definition must contain one
1733 `xsd:simpleContent` element.

1734 [CTD12] The unspecialized Primary Representation Term Datatype
1735 `xsd:complexType` definition `xsd:simpleContent` element must contain
1736 one `xsd:restriction` element with an `xsd:base` attribute whose value is
1737 equal to the corresponding `cct:ComplexType`

1738 5.1.3.4 Core Component Types

1739 A CCT consists of a “content component” which may be supported by a set of properties
1740 referred to as “supplementary components”. CCTs may be expressed as a simple type
1741 (where possible), but may require expression as a complex type. Content components are
1742 expressed as extensions of the set of built-in xsd Datatypes. Supplementary components
1743 are expressed either as extensions of built-in Datatypes, or user-defined simple types.

1744 [CTD13] For every `ccts:CCT` whose supplementary components are not equivalent to
1745 the properties of a built-in `xsd:Datatype`, the `ccts:CCT` MUST be defined
1746 as a named `xsd:complexType` in the `ccts:CCT` schema module.

1747 Each CCT based `xsd:complexType` always has `xsd:simpleContent`, which is an
1748 extension of a built-in `xsd:Datatype`.

1749 [CTD14] Each `ccts:CCT` `xsd:complexType` definition MUST contain one
1750 `xsd:simpleContent` element

1751
1752 [CTD15] The `ccts:CCT` `xsd:complexType` definition `xsd:simpleContent`
1753 element MUST contain one `xsd:extension` element. This
1754 `xsd:extension` element MUST include an `xsd:base` attribute that
1755 defines the specific `xsd:Built-in Datatype` required for the
1756 `ccts:ContentComponent` of the `ccts:CCT`.

1757 Example:

```
1758 <xsd:complexType name="QuantityType">  
1759     ...  
1760     <xsd:simpleContent>  
1761     <xsd:extension base="xsd:decimal">  
1762     <xsd:attribute name="quantityUnitCode" type="xsd:normalizedString"  
1763     use="optional"/>  
1764     <xsd:attribute name="quantityUnitCodeListID"  
1765     type="xsd:normalizedString" use="optional"/>  
1766     <xsd:attribute name="quantityUnitCodeListAgencyID"  
1767     type="xsd:normalizedString" use="optional"/>  
1768     <xsd:attribute name="quantityUnitCodeListAgencyName"  
1769     type="xsd:string" use="optional"/>  
1770     </xsd:extension>  
1771     </xsd:simpleContent>  
1772 </xsd:complexType>
```

1784 5.1.3.5 Supplementary Components

1785 Supplementary components are expressed with references to either built-in
1786 `xsd:Datatypes`, or to user-defined simple types.

1787 [CTD16] Each `CCT:SupplementaryComponent` `xsd:attribute` “type” MUST
1788 define the specific `xsd:Built-inDatatype` or the user defined
1789 `xsd:simpleType` for the `ccts:SupplementaryComponent` of the
1790 `ccts:CCT`.

1791 Example:

```
1792 <xsd:attribute name="measureUnitCode" type="xsd:normalizedString"  
1793 use="required"/>
```

1795 [CTD17] Each `ccts:SupplementaryComponent` `xsd:attribute` user-defined
1796 `xsd:simpleType` MUST only be used when the `ccts:Supplementary`
1797 `Component` is based on a standardized code list for which a UBL conformant
1798 code list schema module has been created.

1799 [CTD18] Each `ccts:SupplementaryComponent` `xsd:attribute` user defined
1800 `xsd:simpleType` MUST be the same `xsd:simpleType` from the
1801 appropriate UBL conformant code list schema module for that type.

1802 Supplementary components are either required or optional, based on the description of
1803 the parent `CCT` in the Core Components Technical Specification.

1804 [CTD19] Each `ccts:SupplementaryComponent` `xsd:attribute` “use” MUST
1805 define the occurrence of that `ccts:SupplementaryComponent` as either
1806 “required”, or “optional”.

1807 Example:

```
1808 <xsd:attribute name="amountCurrencyID" type="xsd:normalizedString"  
1809 use="required"/>  
1810  
1811 <xsd:attribute name="amountCurrencyCodeListVersionID"  
1812 type="xsd:normalizedString" use="optional"/>
```

1813 5.2 Element Declarations

1814 5.2.1 Elements Bound to Complex Types

1815 The binding of UBL elements to their `xsd:complexType` is based on the associations
1816 identified in the UBL model. For the `ccts:BasicBusinessInformationEntities`
1817 and `ccts:AggregateInformationEntities`, the UBL elements will be directly
1818 associated to its corresponding `xsd:complexType`.

1819 [ELD3] For every class identified in the UBL model, a global element bound to the
1820 corresponding `xsd:complexType` MUST be declared.

1821 **Example:**

1822 For the `Party`. `Details` object class, a complex type/global element declaration
1823 pair is created through the declaration of a `Party` element that is of type `PartyType`.

1824 The element thus created is useful for reuse in the building of new business messages.
1825 The complex type thus created is useful for both reuse and customization, in the building
1826 of both new and contextualized business messages.

1827 **Example:**

```
1828 <xsd:element name="BuyerParty" type="BuyerPartyType"/>  
1829 <xsd:complexType name="BuyerPartyType">  
1830 ...  
1831 </xsd:complexType>
```

1832 5.2.2 Elements Representing ASBIEs

1833 A `ccts:AssociationBusinessInformationEntity` is not a class like
1834 `ccts:AggregateBusinessInformationEntities`. Rather, it is an association
1835 between two classes. As such, the element declaration will bind the element to the
1836 `xsd:complexType` of the associated `ccts:AggregateBusinessInformation`
1837 `Entity`. There are two types of ASBIEs – those that have qualifiers in the object class,
1838 and those that do not.

1839 [ELD4] When a `ccts:ASBIE` is unqualified, it is bound via reference to the global
1840 `ccts:ABIE` element to which it is associated. When an `ccts:ABIE` is
1841 qualified, a new element MUST be declared and bound to the
1842 `xsd:complexType` of its associated `ccts:AggregateBusiness`
1843 `InformationEntity`.

1844 5.2.3 Elements Bound to Core Component Types

1845 [ELD5] For each `ccts:CCT simpleType`, an `xsd:restriction` element
1846 MUST be declared.

1847 5.2.4 Code List Import

1848 [ELD6] The code list `xsd:import` element MUST contain the namespace and
1849 schema location attributes.

1850 5.2.5 Empty Elements

1851 [ELD7] Empty elements MUST not be declared.

1852 5.2.6 Global Elements

1853 The `ccts:BasicBusinessInformationEntityProperties` are reused in multiple
1854 contexts. Their reuse in a specific context is typically identified in part through the use of
1855 qualifiers. However, these qualifiers do not change the nature of the underlying concept
1856 of the `ccts:BasicBusinessInformationEntityProperties`. As such, qualified
1857 `ccts:BasicBusinessInformationEntityProperties` are always bound to the
1858 same type as that of its unqualified corresponding `ccts:BasicBusiness`
1859 `InformationEntityProperties`.

1860 [ELD8] Global elements declared for Qualified BBIE Properties must be of the same
1861 type as its corresponding Unqualified BBIE Property. (i.e. Property Term +
1862 Representation Term.)

1863 **Example:**

1864 `<xsd:element name="AdditionalStreetName" type="cbc:StreetNameType"/>`

1865 5.2.7 XSD:Any Element

1866 UBL disallows the use of `xsd:any`, because this feature permits the introduction of
1867 potentially unknown elements into an XML instance. UBL intends that all constructs
1868 within the instance be described by the schemas describing that instance - `xsd:any` is
1869 seen as working counter to the requirements of interoperability. In consequence,
1870 particular attention is given to the need to enable meaningful validation of the UBL
1871 document instances. Were it not for this, `xsd:any` might have been allowed.

1872

1873 [ELD9] The `xsd:any` element MUST NOT be used.

1874 5.3 Attribute Declarations

1875 Attributes are W3C Schema constructs associated with elements that provide further
1876 information regarding elements. While elements can be thought of as containing data,
1877 attributes can be thought of as containing metadata. Unlike elements, attributes cannot be
1878 nested within each other—there are no “subattributes.” Therefore, attributes cannot be
1879 extended as elements can. Attribute order is not enforced by XML processors—that is, if
1880 the attribute order in an XML instance document is different than the order in which the
1881 attributes are declared in the schema to which the XML instance document conforms, no
1882 error will result. UBL has determined that these limitations dictate that UBL restrict the
1883 use of attributes to either XSD built-in attributes, or to Supplementary Components
1884 which by their nature within the CCTS metamodel only carry metadata.

1885 5.3.1 User Defined Attributes

1886 [ATD1] User defined attributes SHOULD NOT be used. When used, user defined
1887 attributes MUST only convey `CCT:SupplementaryComponent`
1888 information.

1889

1890 [ATD2] The `CCT:SupplementaryComponents` for the ID
1891 `CCT:CoreComponent` MUST be declared in the following order:
1892 Identifier. Content
1893 Identification Scheme. Identifier
1894 Identification Scheme. Name. Text
1895 Identification Scheme. Agency. Identifier
1896 Identification Scheme. Agency Name. Text
1897 Identification Scheme. Version. Identifier
1898 Identification Scheme. Uniform Resource. Identifier
1899 Identification Scheme Data. Uniform Resource.
1900 Identifier

1901 [Note:] Rule ATD2, while being part of UBL version 1.0, is deprecated. It will be
1902 deleted in the next version of UBL as its deletion does not affect backwards
1903 compatability.

1904 5.3.2 Global Attributes

1905 Rule ATD1 limits the use of attributes to `cct:SupplementaryComponents`. The
1906 current UBL library does not contain any attributes that are common to all UBL
1907 elements, however such a situation may arise in the future. If such common attributes are
1908 defined, then they will be declared using the `xsd:globalattributegroup` element
1909 using the following rules.

1910 [ATD3] If a UBL Schema Expression contains one or more common attributes that
1911 apply to all UBL elements contained or included or imported therein, the
1912 common attributes MUST be declared as part of a global attribute group.

1913

1914 5.3.3 Supplementary Components

1915 [ATD4] Within the `ccts:CCT` `xsd:extension` element an `xsd:attribute`
1916 MUST be declared for each `ccts:SupplementaryComponent` pertaining
1917 to that `ccts:CCT`.

1918

1919 [ATD5] For each `ccts:CCT simpleType xsd:restriction` element, an
1920 `xsd:base` attribute MUST be declared and set to the appropriate
1921 `xsd:Datatype`.

1922 5.3.4 Schema Location

1923 UBL is an international standard that will be used in perpetuity by companies around the
1924 globe. It is important that these users have unfettered access to all UBL schema.

1925 [ATD6] Each `xsd:schemaLocation` attribute declaration MUST contain a system-
1926 resolvable URL, which at the time of release from OASIS shall be a relative
1927 URL referencing the location of the schema or schema module in the release
1928 package.

1929 5.3.5 XSD:nil

1930 [ATD7] The `xsd` built in nillable attribute MUST NOT be used for any UBL declared
1931 element.

1932 5.3.6 XSD:anyAttribute

1933 UBL disallows the use of `xsd:anyAttribute`, because this feature permits the
1934 introduction of potentially unknown attributes into an XML instance. UBL intends that
1935 all constructs within the instance be described by the schemas describing that instance -
1936 `xsd:anyAttribute` is seen as working counter to the requirements of interoperability.
1937 In consequence, particular attention is given to the need to enable meaningful validation
1938 of the UBL document instances. Were it not for this, `xsd:anyAttribute` might have
1939 been allowed.
1940

1941 [ATD8] The `xsd:anyAttribute` MUST NOT be used.

1942

1943

6 Code Lists

1944 UBL has determined that the best approach for code lists is to handle them as schema
1945 modules. In recognition of the fact that most code lists are maintained by external
1946 agencies, UBL has determined that if code list owners all used the same normative form
1947 schema module, all users of those code lists could avoid a significant level of code list
1948 maintenance. By having each code list owner develop, maintain, and make available via
1949 the internet their code lists using the same normative form schema, code list users would
1950 be spared the unnecessary and duplicative efforts required for incorporation in the form
1951 of enumeration of such code lists into Schema, and would subsequently avoid the
1952 maintenance of such enumerations since code lists are handled as imported schema
1953 modules rather than cumbersome enumerations. To make this mechanism operational,
1954 UBL has defined a number of rules. To avoid enumeration of codes in the document or
1955 reusable schemas, UBL has determined that codes will be handled in their own schema
1956 modules.

1957 [CDL1] All UBL Codes MUST be part of a UBL or externally maintained Code List.

1958 Because the majority of code lists are owned and maintained by external agencies, UBL
1959 will make maximum use of such external code lists where they exist.

1960 [CDL2] The UBL Library SHOULD identify and use external standardized code lists
1961 rather than develop its own UBL-native code lists.

1962 In some cases the UBL Library may extend an existing code list to meet specific business
1963 requirements. In others cases the UBL Library may have to create and maintain a code
1964 list where a suitable code list does not exist in the public domain. Both of these types of
1965 code lists would be considered UBL-internal code lists.

1966 [CDL3] The UBL Library MAY design and use an internal code list where an existing
1967 external code list needs to be extended, or where no suitable external code list
1968 exists.

1969 UBL-internal code lists will be designed with maximum re-use in mind to facilitate
1970 maximum use by others.

1971 If a UBL code list is created, the lists should be globally scoped (designed for reuse and
1972 sharing, using named types and namespaced Schema Modules) rather than locally scoped
1973 (not designed for others to use and therefore hidden from their use).

1974 To guarantee consistency within all code list schema modules all ubl-internal code lists
1975 and externally used code lists will use the UBL Code List Schema Module. This schema
1976 module will contain an enumeration of code list values.

1977 [CDL4] All UBL maintained or used Code Lists MUST be enumerated using the UBL
1978 Code List Schema Module.

1979 To guarantee consistency of code list schema module naming, the name of each UBL
1980 Code List Schema Module will adhere to a prescribed form.

1981 [CDL5] The name of each UBL Code List Schema Module MUST be of the form:
1982 {Owning Organization}{Code List Name}{Code List Schema
1983 Module}

1984 **Example**

1985 ISO 8601 Country Code Code List Schema Module

1986 ISO 3055 Kitchen equipment -- Coordinating sizes Code Code

1987 List Schema Module

1988 Each code list used in the UBL schema MUST be imported individually.

1989 [CDL6] An `xsd:import` element MUST be declared for every code list required in a
1990 UBL schema.

1991 The UBL library allows partial implementations of code lists which may required by
1992 customizers.

1993 [CDL7] Users of the UBL Library MAY identify any subset they wish from an
1994 identified code list for their own trading community conformance
1995 requirements.

1996 The following rule describes the requirements for the `xsd:schemaLocation` for the
1997 importation of the code lists into a UBL business document.

1998 [CDL8] The `xsd:schemaLocation` MUST include the complete URI used to
1999 identify the relevant code list schema.

2000

2001 7 Miscellaneous XSD Rules

2002 UBL, as a business standard vocabulary, requires consistency in its development. The
2003 number of UBL Schema developers will expand over time. To ensure consistency, it is
2004 necessary to address the optional features in XSD that are not addressed elsewhere.

2005 7.1 `xsd:simpleType`

2006 UBL guiding principles require maximum reuse. XSD provides for forty four built-in
2007 Datatypes expressed as simple types. In keeping with the maximize re-use guiding
2008 principle, these built-in simple types should be used wherever possible.

2009 [GXS3] Built-in XSD Simple Types SHOULD be used wherever possible.

2010 7.2 Namespace Declaration

2011 The W3C XSD specification allows for the use of any token to represent its location. To
2012 ensure consistency, UBL has adopted the generally accepted convention of using the
2013 “xsd” token for all UBL schema and schema modules.

2014 [GXS4] All W3C XML Schema constructs in UBL Schema and schema modules
2015 MUST contain the following namespace declaration on the `xsd` schema
2016 element:

2017 `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

2018 7.3 `xsd:substitutionGroup`

2019 The `xsd:substitutionGroup` feature enables a type definition to identify substitution
2020 elements in a group. Although a useful feature in document centric XML applications,
2021 this feature is not used by UBL.

2022 [GXS5] The `xsd:substitutionGroup` feature MUST NOT be used.

2023 7.4 `xsd:final`

2024 [GXS6] The `xsd:final` attribute MUST be used to control extensions.

2025 7.5 `xsd:notation`

2026 The `xsd:notation` attribute identifies a notation. Notation declarations corresponding
2027 to all the `<notation>` element information items in the `[children]`, if any, plus any
2028 included or imported declarations. Per XSD Part 2, “It is an error for NOTATION to be

2029 used directly in a schema. Only Datatypes that are *derived* from NOTATION by
2030 specifying a value for *enumeration* can be used in a schema.” The UBL schema model
2031 does not require or support the use of this feature.

2032 [GXS7] `xsd:notation` MUST NOT be used.

2033 7.6 `xsd:all`

2034 The `xsd:all` compositor requires occurrence indicators of `minOccurs = 0` and
2035 `maxOccurs = 1`. The `xsd:all` compositor allows for elements to occur in any order.
2036 The result is that in an instance document, elements can occur in any order, are always
2037 optional, and never occur more than once. Such restrictions are inconsistent with data-
2038 centric scenarios such as UBL.

2039 [GXS8] The `xsd:all` element MUST NOT be used.

2040 7.7 `xsd:choice`

2041 The `xsd:choice` compositor allows for any element declared inside it to occur in the
2042 instance document, but only one. As with the `xsd:all` compositor, this feature is
2043 inconsistent with business transaction exchanges and is not allowed in UBL. While
2044 `xsd:choice` is a very useful construct in situations where customization and
2045 extensibility are not a concern, UBL does not use it because `xsd:choice` cannot be
2046 extended.

2047 [GXS9] The `xsd:choice` element SHOULD NOT be used where customisation and
2048 extensibility are a concern.

2049 7.8 `xsd:include`

2050 The `xsd:include` feature provides a mechanism for bringing in schemas that reside in
2051 the same namespace. UBL employs multiple schema modules within a namespace. To
2052 avoid circular references, this feature will not be used except by the document schema.

2053 [GXS10] The `xsd:include` feature MUST only be used within a document schema.

2054 7.9 `xsd:union`

2055 The `xsd:union` feature provides a mechanism whereby a datatype is created as a union
2056 of two or more existing datatypes. With UBL’s strict adherence to the use of
2057 `cts:Datatypes` that are explicitly declared in the UBL library, this feature is
2058 inappropriate except for codelists. In some cases external customizers may choose to use
2059 this technique for codelists and as such the use of the union technique may prove
2060 beneficial for customizers.

2061 [GXS11] The `xsd:union` technique MUST NOT be used except for Code Lists. The
2062 `xsd:union` technique MAY be used for Code Lists.

2063 7.10 `xsd:appinfo`

2064 The `xsd:appinfo` feature is used by schema to convey processing instructions to a
2065 processing application, Stylesheet, or other tool. Some users of UBL have determined
2066 that this technique poses a security risk and have employed techniques for stripping
2067 `xsd:appinfo` from schemas. As UBL is committed to ensuring the widest possible
2068 target audience for its XML library, this feature is not used – except to convey non-
2069 normative information.

2070 [GXS12] UBL designed schema SHOULD NOT use `xsd:appinfo`. If used,
2071 `xsd:appinfo` MUST only be used to convey non-normative information.

2072 7.11 Extension and Restriction

2073 UBL fully recognizes the value of supporting extension and restriction of its core library
2074 by customizers. The UBL extension and restriction recommendations are discussed in the
2075 *Guidelines for the Customization of UBL Schemas* available as part of UBL 1.0.

2076 [GXS13] Complex Type extension or restriction MAY be used where appropriate.

2077 8 Instance Documents

2078 Consistency in UBL instance documents is essential in a trade environment. UBL has
2079 defined several rules to help affect this consistency.

2080 8.1 Root Element

2081 UBL has chosen a global element approach. In XSD, every global element is eligible to
2082 act as a root element in an instance document. Rule ELD1 requires the identification of a
2083 single global element in each UBL schema to be carried as the root element in the
2084 instance document. UBL business documents (UBL instances) must have a single root
2085 element as defined in the corresponding UBL XSD.

2086 [RED1] Every UBL instance document must use the global element defined as the root
2087 element in the schema as its root element.

2088 8.2 Validation

2089 The UBL library and supporting schema are targeted at supporting business information
2090 exchanges. Business information exchanges require a high degree of precision to ensure
2091 that application processing and corresponding business cycle actions are reflective of the
2092 purpose, intent, and information content agreed to by both trading partners. Schemas
2093 provide the necessary mechanism for ensuring that instance documents do in fact support
2094 these requirements.

2095 [IND1] All UBL instance documents MUST validate to a corresponding schema.

2096 8.3 Character Encoding

2097 XML supports a wide variety of character encodings. Processors must understand which
2098 character encoding is employed in each XML document. XML 1.0 supports a default
2099 value of UTF-8 for character encoding, but best practice is to always identify the
2100 character encoding being employed.

2101 [IND2] All UBL instance documents MUST always identify their character encoding
2102 with the XML declaration.

2103 Example:

2104
2105 `xml expression: UTF-8`

2106 UBL, as an OASIS TC, is obligated to conform to agreements OASIS has entered into.
2107 OASIS is a liaison member of the ISO/IETF/ITU/UNCEFACT Memorandum of

2108 Understanding Management Group (MOUMG). Resolution 01/08 (MOU/MG01n83)
2109 requires the use of UTF-8.

2110 [IND3] In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of
2111 Understanding Management Group (MOUMG) Resolution 01/08
2112 (MOU/MG01n83) as agreed to by OASIS, all UBL XML SHOULD be
2113 expressed using UTF-8.

2114 Example:

2115
2116 <?xml version="1.0" encoding="UTF-8" ?>

2117 8.4 Schema Instance Namespace Declaration

2118 [IND4] All UBL instance documents MUST contain the following namespace
2119 declaration in the root element:

2120 `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`

2121 8.5 Empty Content.

2122 Usage of empty elements within XML instance documents are a source of controversy
2123 for a variety of reasons. An empty element does not simply represent data that is missing.
2124 It may express data that is not applicable for some reason, trigger the expression of an
2125 attribute, denote all possible values instead of just one, mark the end of a series of data, or
2126 appear as a result of an error in XML file generation. Conversely, missing data elements
2127 can also have meaning - data not provided by a trading partner. In information exchange
2128 environments, different trading partners may allow, require or ban empty elements. UBL
2129 has determined that empty elements do not provide the level of assurance necessary for
2130 business information exchanges and as such will not be used.

2131 [IND5] UBL conformant instance documents MUST NOT contain an element devoid
2132 of content or null values.

2133 To ensure that no attempt is made to circumvent rule IND5, UBL also prohibits
2134 attempting to convey meaning by not conveying an element.

2135 [IND6] The absence of a construct or data in a UBL instance document MUST NOT
2136 carry meaning.

2137

2138 **Appendix A. UBL NDR Checklist**

2139 The following checklist constitutes all UBL XML naming and design rules as defined in
2140 *UBL Naming and Design Rules version 1.0*, xx November 2003. The checklist is in
2141 alphabetical sequence as follows:

2142 Attribute Declaration Rules (ATD)

2143 Attribute Naming Rules (ATN)

2144 Code List Rules (CDL)

2145 ComplexType Definition Rules (CTD)

2146 ComplexType Naming Rules (CTN)

2147 Documentation Rules (DOC)

2148 Element Declaration Rules (ELD)

2149 General Naming Rules (GNR)

2150 General Type Definition Rules (GTD)

2151 General XML Schema Rules (GXS)

2152 Instance Document Rules (IND)

2153 Modeling Constraints Rules (MDC)

2154 Naming Constraints Rules (NMC)

2155 Namespace Rules (NMS)

2156 Root Element Declaration Rules (RED)

2157 Schema Structure Modularity Rules (SSM)

2158 Standards Adherence Rules (STA)

2159 SimpleType Naming Rules (STN)

2160 SimpleType Definition Rules (STD)

2161 Versioning Rules (VER)

A.1 Attribute Declaration Rules	
[ATD1]	User defined attributes SHOULD NOT be used. When used, user defined attributes MUST only convey <code>CCT:SupplementaryComponent</code> information.
[ATD2]	<p>The <code>CCT:SupplementaryComponents</code> for the ID <code>CCT:CoreComponent</code> MUST be declared in the following order:</p> <p>Identifier. Content</p> <p>Identification Scheme. Identifier</p> <p>Identification Scheme. Name. Text</p> <p>Identification Scheme. Agency. Identifier</p> <p>Identification Scheme. Agency Name. Text</p> <p>Identification Scheme. Version. Identifier</p> <p>Identification Scheme. Uniform Resource. Identifier</p> <p>Identification Scheme Data. Uniform Resource. Identifier</p>
[ATD3]	If a UBL Schema Expression 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 <code>ccts:CCT xsd:extension</code> element an <code>xsd:attribute</code> MUST be declared for each <code>ccts:SupplementaryComponent</code> pertaining to that <code>ccts:CCT</code> .
[ATD5]	For each <code>ccts:CCT simpleType xsd:restriction</code> element, an <code>xsd:base</code> attribute MUST be declared and set to the appropriate <code>xsd:Datatype</code> .
[ATD6]	Each <code>xsd:schemaLocation</code> 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 <code>xsd:builtIn</code> attribute MUST NOT be used for any UBL declared element.
[ATD8]	The <code>xsd:anyAttribute</code> MUST NOT be used.

2163

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

2164

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: <code>{Owning Organization}{Code List Name}{Code List Schema Module}</code>
[CDL6]	An <code>xsd:import</code> element MUST be declared for every code list required in a UBL schema.

A.3 Code List Rules

[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 <code>xsd:schemaLocation</code> MUST include the complete URI used to identify the relevant code list schema.

2165

A.4 ComplexType Definition Rules

[CTD1]	For every class identified in the UBL model, a named <code>xsd:complexType</code> MUST be defined.
[CTD2]	Every <code>ccts:ABIE</code> <code>xsd:complexType</code> definition content model MUST use the <code>xsd:sequence</code> element with appropriate global element references, or local element declarations in the case of <code>ID</code> and <code>Code</code> , to reflect each property of its class as defined in the corresponding UBL model.
[CTD3]	Every <code>ccts:BBIEProperty</code> <code>xsd:complexType</code> definition content model MUST use the <code>xsd:simpleContent</code> element.
[CTD4]	Every <code>ccts:BBIEProperty</code> <code>xsd:complexType</code> content model <code>xsd:simpleContent</code> element MUST consist of an <code>xsd:extension</code> element.
[CTD5]	Every <code>ccts:BBIEProperty</code> <code>xsd:complexType</code> content model <code>xsd:base</code> attribute value MUST be the <code>ccts:CCT</code> of the unspecialized or specialized UBL datatype as appropriate.
[CTD6]	For every datatype used in the UBL model, a named <code>xsd:complexType</code> or <code>xsd:simpleType</code> MUST be defined.
[CTD7]	Every unspecialized Datatype must be based on a <code>ccts:CCT</code> represented in the CCT schema module and must represent an approved primary or

A.4 ComplexType Definition Rules

	secondary representation term identified in the CCTS.
[CTD8]	Each unspecialized Datatype <code>xsd:complexType</code> must be based on its corresponding CCT <code>xsd:complexType</code> .
[CTD9]	Every unspecialized Datatype that represents a primary representation term whose corresponding <code>ccts:CCT</code> is defined as an <code>xsd:simpleType</code> MUST also be defined as an <code>xsd:simpleType</code> and MUST be based on the same <code>xsd:simpleType</code> .
[CTD10]	Every unspecialized Datatype that represents a secondary representation term whose corresponding <code>ccts:CCT</code> is defined as an <code>xsd:simpleType</code> MUST also be defined as an <code>xsd:simpleType</code> and MUST be based on the same <code>xsd:simpleType</code> .
[CTD11]	Each unspecialized Datatype <code>xsd:complexType</code> definition must contain one <code>xsd:simpleContent</code> element.
[CTD12]	The unspecialized 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-inDatatype</code> or the user defined

A.4 ComplexType Definition Rules

	<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:SupplementaryComponent</code> <code>xsd:attribute</code> "use" MUST define the occurrence of that <code>ccts:SupplementaryComponent</code> as either "required", or "optional".

2166

A.5 ComplexType Naming Rules

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

A.5 ComplexType Naming Rules

[CTN4]	A UBL <code>xsd:complexType</code> for a <code>cct:UnspecializedDatatype</code> based on a <code>ccts:SecondaryRepresentationTerm</code> used in the UBL model MUST have the name of the corresponding <code>ccts:SecondaryRepresentationTerm</code> , with the separators removed and with the "Type" suffix appended.
[CTN5]	A UBL <code>xsd:complexType</code> name based on a <code>ccts:CoreComponentType</code> MUST be the Dictionary entry name of the <code>ccts:CoreComponentType</code> , with the separators removed.

2167

A.6 Documentation Rules

[DOC1]

The `xsd:documentation` element for every Datatype 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 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.

A.6 Documentation Rules

[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.
[DOC3]	<p>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:</p> <ul style="list-style-type: none">• SupplementaryComponentName (mandatory): Identifies the Supplementary Component on which the restriction applies.• RestrictionValue (mandatory, repetitive): The actual value(s) that is (are) valid for the Supplementary Component

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.

cd-UBL-NDR-1.0.1

88

6 November 2004

- **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 <code>xsd:documentation</code> 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].
--------	--

2168

2169

A.7 Element Declaration Rules

[ELD1]	<p>Each <code>UBL:DocumentSchema</code> MUST identify one and only one global element declaration that defines the document <code>ccts:AggregateBusinessInformationEntity</code> being conveyed in the Schema expression. That global element MUST include an <code>xsd:annotation</code> child element which MUST further contain an <code>xsd:documentation</code> child element that declares "<i>This element MUST be conveyed as the root element in any instance document based on this Schema expression.</i>"</p>
[ELD2]	<p>All element declarations MUST be global with the exception of <code>ID</code> and <code>Code</code> which MUST be local.</p>

A.7 Element Declaration Rules

[ELD3]	For every class identified in the UBL model, a global element bound to the corresponding <code>xsd:complexType</code> MUST be declared.
[ELD4]	When a <code>ccts:ASBIE</code> is unqualified, it is bound via reference to the global <code>ccts:ABIE</code> element to which it is associated. When an <code>ccts:ABIE</code> is qualified, a new element MUST be declared and bound to the <code>xsd:complexType</code> of its associated <code>ccts:AggregateBusinessInformationEntity</code> .
[ELD5]	For each <code>ccts:CCT simpleType</code> , an <code>xsd:restriction</code> element MUST be declared.
[ELD6]	The code list <code>xsd:import</code> element MUST contain the namespace and schema location attributes.
[ELD7]	Empty elements MUST not be declared.
[ELD8]	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 <code>xsd:any</code> element MUST NOT be used.

2170

A.8 Element Naming Rules

[ELN1]	A UBL global element name based on a <code>ccts:ABIE</code> MUST be the same as the name of the corresponding <code>xsd:complexType</code> to which it is bound, with the word "Type" removed.
[ELN2]	A UBL global element name based on an unqualified <code>ccts:BBIEProperty</code> MUST be the same as the name of the corresponding <code>xsd:complexType</code> to which it is bound, with the word "Type" removed.
[ELN3]	A UBL global element name based on a qualified <code>ccts:ASBIE</code> MUST be the <code>ccts:ASBIE</code> dictionary entry name property term and its qualifiers; and the

A.8 Element Naming Rules

	object class term and qualifiers of its associated <code>ccts:ABIE</code> . All <code>ccts:DictionaryEntryName</code> separators MUST be removed. Redundant words in the <code>ccts:ASBIE</code> property term or its qualifiers and the associated <code>ccts:ABIE</code> object class term or its qualifiers MUST be dropped.
[ELN4]	A UBL global element name based on a Qualified <code>ccts:BBIEProperty</code> MUST be the same as the name of the corresponding <code>xsd:complexType</code> to which it is bound, with the qualifier prefixed and with the word "Type" removed.

2171

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.

2172

A.10 General Type Definition Rules	
[GTD1]	All types MUST be named.
[GTD2]	The <code>xsd:anyType</code> MUST NOT be used.

2173

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> • <code>xsd:schema</code> element to include version attribute and namespace declarations in the following order: <ul style="list-style-type: none"> • <code>xmlns:xsd</code> • Target namespace • Default namespace • <code>CommonAggregateComponents</code> • <code>CommonBasicComponents</code> • <code>CoreComponentTypes</code>

A.11 General XML Schema Rules

- Datatypes
- Identifier Schemes
- Code Lists
- Attribute Declarations – elementFormDefault=”qualified”
attributeFormDefault=”unqualified”
- <!-- ===== Imports ===== -->CommonAggregateComponents schema
module
- CommonBasicComponents schema module
- Representation Term schema module (to include CCT module)
- Unspecialized Types schema module
- Specialized 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

A.11 General XML Schema Rules

	<p>xsd:TypeDefinitions</p> <ul style="list-style-type: none"> • <!-- =====Basic Business Information Entity Type Definitions ===== --> • alphabetized order of ccts:BasicBusinessInformationEntities • <!-- ===== Copyright Notice ===== --> • Required OASIS full copyright notice.
[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 <code>xsd:simpleType</code> 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 <code>xsd</code> schema element: <code>xmlns:xsd="http://www.w3.org/2001/XMLSchema"</code>
[GXS5]	The <code>xsd:SubstitutionGroups</code> feature MUST NOT be used.
[GXS6]	The <code>xsd:final</code> attribute MUST be used to control extensions.
[GXS7]	<code>xsd:notations</code> MUST NOT be used.
[GXS8]	The <code>xsd:all</code> element MUST NOT be used.
[GXS9]	The <code>xsd:choice</code> element SHOULD NOT be used where customisation and extensibility are a concern.
[GXS10]	The <code>xsd:include</code> feature MUST only be used within a document schema.
[GXS11]	The <code>xsd:union</code> technique MUST NOT be used except for Code Lists. The <code>xsd:union</code> technique MAY be used for Code Lists.
[GXS12]	UBL designed schema SHOULD NOT use <code>xsd:appinfo</code> . If used, <code>xsd:appinfo</code> MUST only be used to convey non-normative information.

A.11 General XML Schema Rules

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

2174

2175

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.
--------	---

2176

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.
--------	---

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.

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

A.15 Namespace Rules

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

2179

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.
--------	--

2180

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.
[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 <code>{ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}</code>
[SSM8]	A UBL schema module MAY be created for reusable components.
[SSM9]	A schema module defining all <code>ubl:CommonAggregateComponents</code> MUST be created.
[SSM10]	The <code>ubl:CommonAggregateComponents</code> schema module MUST be named <code>"ubl:CommonAggregateComponents Schema Module"</code>
[SSM11]	A schema module defining all <code>ubl:CommonBasicComponents</code> MUST be created.
[SSM12]	The <code>ubl:CommonBasicComponents</code> schema module MUST be named <code>"ubl:CommonBasicComponents Schema Module"</code>

A.17 Schema Structure Modularity Rules

[SSM13]	A schema module defining all <code>ccts:CoreComponentTypes</code> MUST be created.
[SSM14]	The <code>ccts:CoreComponentType</code> schema module MUST be named "ccts:CoreComponentType Schema Module"
[SSM15]	The <code>xsd:facet</code> feature MUST not be used in the <code>ccts:CoreComponentType</code> schema module.
[SSM16]	A schema module defining all <code>ccts:UnspecializedDatatypes</code> MUST be created.
[SSM17]	The <code>ccts:UnspecializedDatatype</code> schema module MUST be named "ccts:UnspecializedDatatype Schema Module"
[SSM18]	A schema module defining all <code>ubl:SpecializedDatatypes</code> MUST be created.
[SSM19]	The <code>ubl:SpecializedDatatypes</code> schema module MUST be named "ubl:SpecializedDatatypes schema module"

2181

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 xsd:simpleType</code> definition name MUST be the <code>ccts:CCT</code> dictionary entry name with the separators removed.

2182

A.19 SimpleType Naming Rules

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

2183

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.
--------	---

2184

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

A.21 Versioning Rules

	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.

2185 **Appendix B. Approved Acronyms and Abbreviations**

2186 The following Acronyms and Abbreviations have been approved by the UBL NDR
2187 Subcommittee for UBL use:

- 2188 ◆ A Dun & Bradstreet Data Universal Numbering System (DUNS) number *must*
2189 appear as "DUNS".
- 2190 ◆ "Identifier" *must* appear as "ID".
- 2191 ◆ "Uniform Resource Identifier" *must* appear as "URI"
- 2192 ◆ [Example] the "Uniform Resource. Identifier" portion of the **Binary Object.**
2193 **Uniform Resource. Identifier** supplementary component becomes "URI" in
2194 the resulting XML name). The use of URI for Uniform Resource Identifier
2195 takes precedence over the use of "ID" for "Identifier".

2196 This list will henceforth be maintained by the UBL TC as a committee of the whole, and
2197 additions included in current and future versions of the UBL standard will be maintained
2198 and published separately.

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.
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.
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</p>

	<p>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	<p>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)</p> <p>A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. (Rational Unified Process)</p>
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	One of the individual entities contributing to a whole.
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.
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 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. Core Component Types 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. (XSD)</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>. (CCTS)</p>
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. Lower-level elements consist of intermediate and leaf level.
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.
(XML) Schema	An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items (as defined in [XML-Infoset]), and furthermore may specify augmentations to those items and their descendants.
Schema module	A collection of XML constructs that together constitute an XSD conformant schema. Schema modules are intended to be used in combination with other XSD conformant schema.
Schema Processing	Schema validation checking plus provision of default values and provision of new infoset 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	Description of a set of entities that share common characteristics, relations, attributes, and semantics. 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.
--	--

2201

2202 **Appendix D. References**

2203 [CCTS] ISO 15000-5 ebXML Core Components Technical Specification
2204 [ISONaming] *ISO/IEC 11179*, Final committee draft, Parts 1-6.
2205 (RFC) 2119 S. Bradner, *Key words for use in RFCs to Indicate Requirement*
2206 *Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March
2207 1997.
2208 [UBLChart] UBL TC Charter, [http://oasis-](http://oasis-open.org/committees/ubl/charter/ubl.htm)
2209 [open.org/committees/ubl/charter/ubl.htm](http://oasis-open.org/committees/ubl/charter/ubl.htm)
2210 [XML] *Extensible Markup Language (XML) 1.0* (Second Edition), W3C
2211 Recommendation, October 6, 2000
2212 (XSD) *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May
2213 2001.
2214
2215 (XHTML) *XHTML™ Basic*, W3C Recommendation 19 December 2000:
2216 <http://www.w3.org/TR/2000/REC-xhtml-basic-20001219>
2217

2218

Appendix E. Notices

2219 OASIS takes no position regarding the validity or scope of any intellectual property or
2220 other rights that might be claimed to pertain to the implementation or use of the
2221 technology described in this document or the extent to which any license under such
2222 rights might or might not be available; neither does it represent that it has made any effort
2223 to identify any such rights. Information on OASIS's procedures with respect to rights in
2224 OASIS specifications can be found at the OASIS website. Copies of claims of rights
2225 made available for publication and any assurances of licenses to be made available, or the
2226 result of an attempt made to obtain a general license or permission for the use of such
2227 proprietary rights by implementors or users of this specification, can be obtained from the
2228 OASIS Executive Director.

2229 OASIS invites any interested party to bring to its attention any copyrights, patents or
2230 patent applications, or other proprietary rights which may cover technology that may be
2231 required to implement this specification. Please address the information to the OASIS
2232 Executive Director.

2233 Copyright © The Organization for the Advancement of Structured Information Standards
2234 [OASIS] 2001, 2002, 2003, 2004. All Rights Reserved.

2235 This document and translations of it may be copied and furnished to others, and
2236 derivative works that comment on or otherwise explain it or assist in its implementation
2237 may be prepared, copied, published and distributed, in whole or in part, without
2238 restriction of any kind, provided that the above copyright notice and this paragraph are
2239 included on all such copies and derivative works. However, this document itself does not
2240 be modified in any way, such as by removing the copyright notice or references to
2241 OASIS, except as needed for the purpose of developing OASIS specifications, in which
2242 case the procedures for copyrights defined in the OASIS Intellectual Property Rights
2243 document must be followed, or as required to translate it into languages other than
2244 English.

2245 The limited permissions granted above are perpetual and will not be revoked by OASIS
2246 or its successors or assigns.

2247 This document and the information contained herein is provided on an "AS IS" basis and
2248 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
2249 BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
2250 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
2251 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
2252 PURPOSE.

2253