Universal Business Language (UBL) Naming and Design Rules

19 November 2003

Document identifier:

wd-ublndrsc-ndrdoc-V1pt0Draftl (Word)

Location:

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

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

Abstract:

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

Status:

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

Table of Contents

Appendix E. Notices

1 Introduction

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

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

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

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

1.1 Audiences

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

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

1.2 Scope

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

1.3 Terminology and Notation

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

[Definition] – A formal definition of a term. Definitions are normative.
[Example] – A representation of a definition or a rule. Examples are informative.
[Note] – Explanatory information. Notes are informative.
[RRR*n*] - Identification of a rule that requires conformance to ensure that an XML Schema is UBL conformant. The value RRR is a prefix to categorize the type of rule where the value of RRR is as defined in Table 1 and n (1..n) indicates the sequential number of the rule within its category. In order to ensure continuity across versions of the specification, rule numbers that are deleted in future versions will not be re-issued, and any new rules will be assigned the next higher number - regardless of location in the text. Future versions will contain an appendix that lists deleted rules and the reason for their deletion. Only rules are normative; all other text is explanatory.

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

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

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

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

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

1.4 Guiding Principles

The UBL guiding principles encompass three areas:

- General UBL guiding principles
- Extensibility
- Code generation

1.4.1 Adherence to General UBL Guiding Principles

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

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

or presentation being available. The lowest common denominator for tools is incredibly low (for example, Notepad) and the variety of tools used is staggering. We do not see this situation changing in the near term.

4. Legibility - UBL documents should be human-readable and reasonably clear.

5. Simplicity - The design of UBL must be as simple as possible (but no simpler).

6. 80/20 Rule - The design of UBL should provide the 20% of features that accommodate 80% of the needs.

7. Component Reuse -The design of UBL document types should contain as many common features as possible. The nature of e-commerce transactions is to pass along information that gets incorporated into the next transaction down the line. For example, a purchase order contains information that will be copied into the purchase order response. This forms the basis of our need for a core library of reusable components. Reuse in this context is important, not only for the efficient development of software, but also for keeping audit trails.

8. Standardization - The number of ways to express the same information in a UBL document is to be kept as close to one as possible.

9. Domain Expertise - UBL will leverage expertise in a variety of domains through interaction with appropriate development efforts.

10. Customization and Maintenance - The design of UBL must facilitate customization and maintenance.

11. Context Sensitivity - The design of UBL must ensure that context-sensitive document types aren't precluded.

12. Prescriptiveness - UBL design will balance prescriptiveness in any single usage scenario with prescriptiveness across the breadth of usage scenarios supported. Having precise, tight content models and datatypes is a good thing (and for this reason, we might want to advocate the creation of more document type "flavors" rather than less; see below). However, in an interchange format, it is often difficult to get the prescriptiveness that would be desired in any single usage scenario.

13. Content Orientation - Most UBL document types should be as "content-oriented" (as opposed to merely structural) as possible. Some document types, such as product catalogs, will likely have a place for structural material such as paragraphs, but these will be rare.

14. XML Technology - UBL design will avail itself of standard XML processing technology wherever possible (XML itself, XML Schema, XSLT, XPath, and so on). However, UBL will be cautious about basing decisions on "standards" (foundational or vocabulary) that are works in progress.

15. Relationship to Other Namespaces - UBL design will be cautious about making dependencies on other namespaces. UBL does not need to reuse existing namespaces wherever possible. For example, XHTML might be useful in catalogs and comments, but it brings its own kind of processing overhead, and if its use is not prescribed carefully it could harm our goals for content orientation as opposed to structural markup.

16. Legacy formats - UBL is not responsible for catering to legacy formats; companies (such as ERP vendors) can compete to come up with good solutions to permanent conversion. This is not to say that mappings to and from other XML dialects or non-XML legacy formats wouldn't be very valuable.

17. Relationship to xCBL - UBL will not be a strict subset of xCBL, nor will it be explicitly compatible with it in any way.

1.4.2 Design For Extensibility

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

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

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

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

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

1.4.3 Code Generation

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

1.5 Choice of schema language

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

[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.

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

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

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

2 Relationship to ebXML Core Components

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

The essence of the Core Components specification is captured in context neutral and context specific building blocks. The context neutral components are defined as Core Components (`ccts:CoreComponents`). Context neutral `ccts:CoreComponents` are defined in CCTS as "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."[2] Figure 2-1 illustrates the various pieces of the overall `ccts:CoreComponents` metamodel.

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

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

---

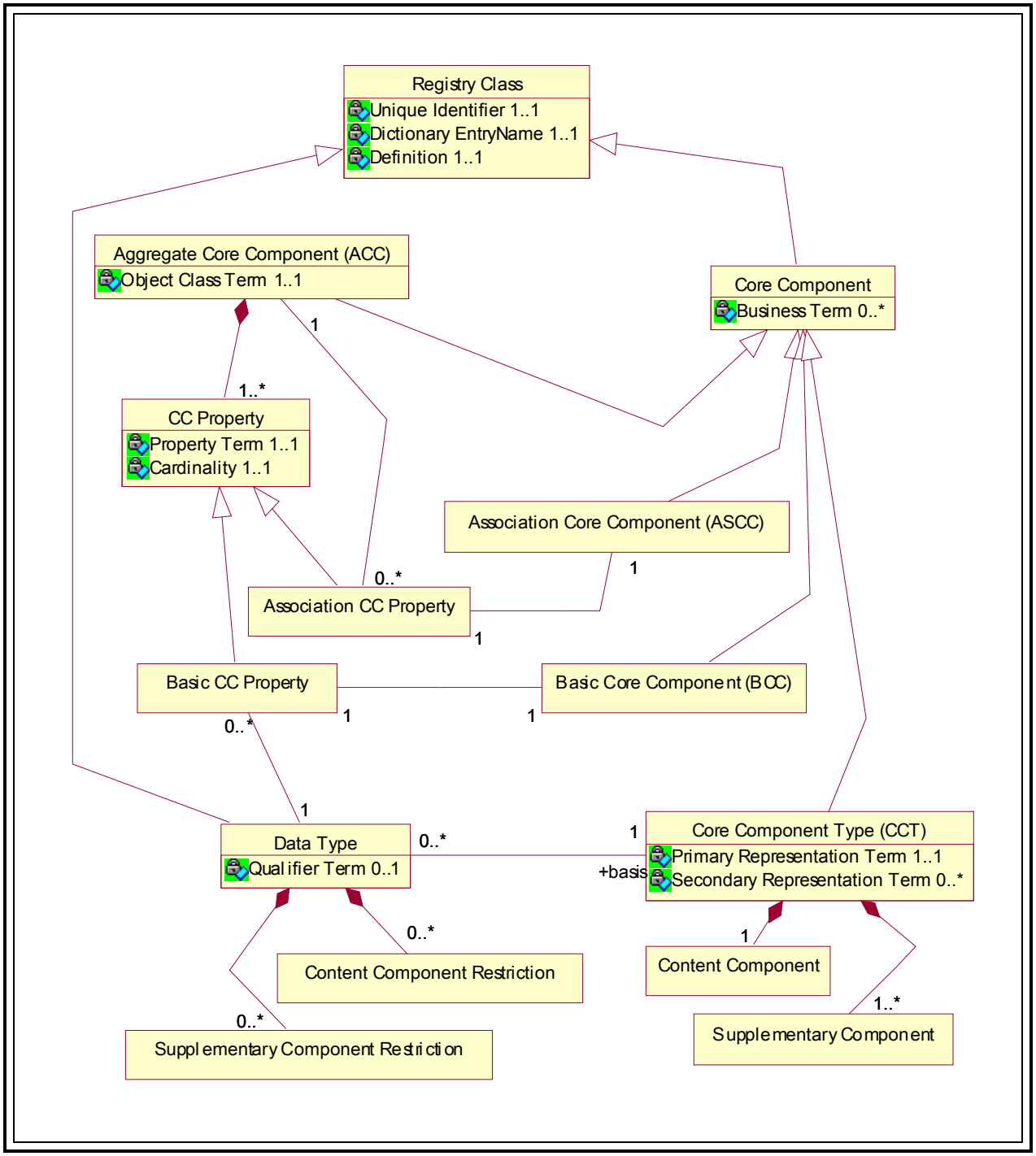
[2] *Core Components Technical Specification, Part 8 of the ebXML Technical Framework Version 2.0 (Second Edition),* UN/CEFACT, 15 November 2003

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

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

380 ***Figure 2-1 Core Components and Datatypes Metamodel***[5]

381

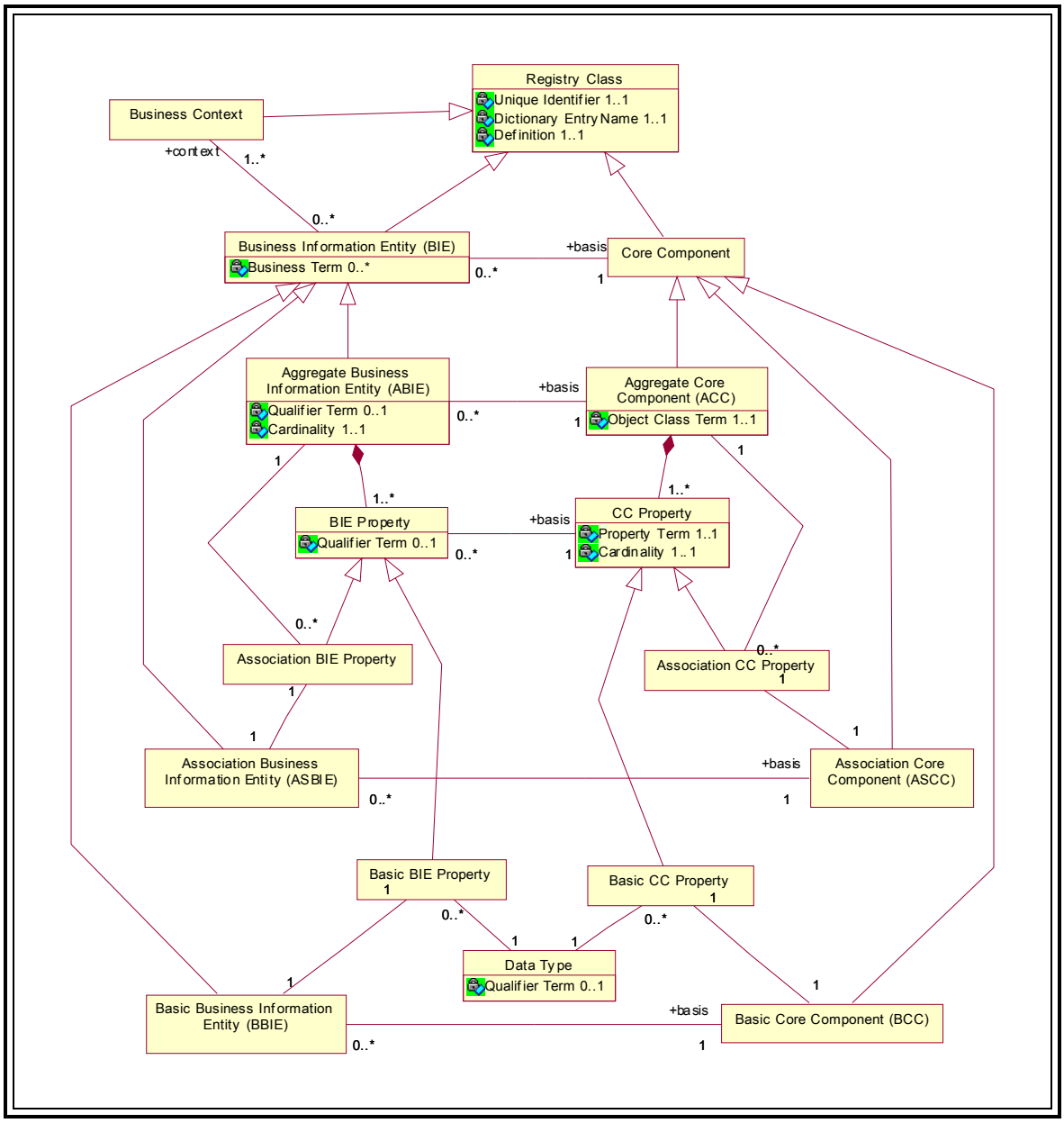


382

383

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

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



384

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

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

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

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

2.1 Mapping Business Information Entities to XSD

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

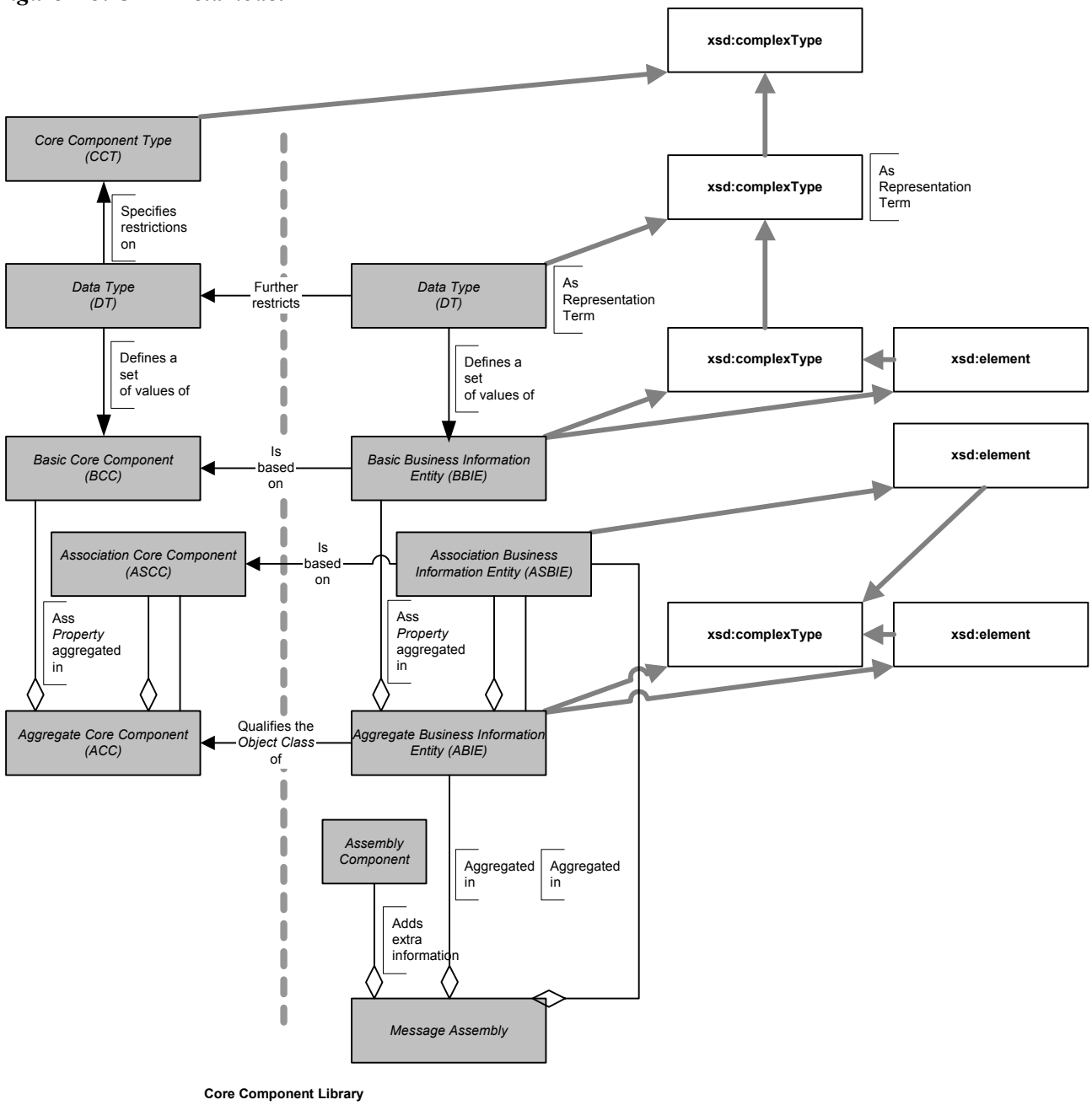
Both Aggregate and Basic Business Information Entities must have a unique name (Object Class Term). Both are treated as objects and both are defined as `xsd:ComplexTypes`.

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

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

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

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



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

derived."[6] Thus ccts:DataTypes allow UBL to identify facets for elements when restrictions to the corresponding ccts:ContentComponent or ccts:SupplementaryComponent is required. These facets will be defined as ubl:DataTypes and will be represented by

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

As stated above, ccts:BasicBusinessInformationEntities define the intrinsic structure of a ccts:AggregateBusinessInformationEntity. These ccts:BasicBusinessInformationEntities are the "leaf" types in the system in that they contain no ccts:AssociationBusinessInformationEntity Properties. A ccts:BasicBusinessInformationEntity must have a ccts:CoreComponentType. Ccts:CoreComponentTypes are low-level types, such as Identifiers and Dates. A Ccts:CoreComponentType describes these low-level types for use by ccts:CoreComponents, and (in parallel) a ccts:DataType – corresponding to that ccts:CoreComponentType, describes these low-level types for use by ccts:BusinessInformationEntities. Every ccts:CoreComponentType have a single ccts:ContentComponent and one or more ccts:Supplementary Components. A ccts:ContentComponent is of some Primitive Type. All ccts:CoreComponentTypes and their corresponding content and supplementary components are pre-defined in the CCTS. UBL, in partnership with the Open Applications Group has developed an xsd:schemaModule that defines each of the pre-defined ccts:CoreComponentTypes as xsd:complexTypes and declares ccts:SupplementaryComponents as xsd:attributes.

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

3 General XML Constructs

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

- Overall Schema Structure
- Naming and Modeling Constraints
- Reusability Scheme
- Namespace Scheme
- Versioning Scheme
- Modularity Strategy
- Schema Documentation Requirements

3.1 Overall Schema Structure

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

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

XML Declaration

<!-- ===== Copyright Notice ===== -->

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

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

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

xmlns:xsd

Target namespace

Default namespace

CommonAggregateComponents

CommonBasicComponents

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

Example:

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--===== Copyright Information ======-->
<!--
  UBL XML Schema of Core Components

  Copyright (C) OASIS-UBL (2003). All Rights Reserved.

  This document and translations of it may be copied and furnished to
  others, and derivative works that comment on or otherwise explain it
  or assist in its implementation may be prepared, copied, published
```

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

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

  Distribution of this document is unlimited.

  ======================================================================

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

  UBL - XML Naming and Design Rules

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

Namespaces

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

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

Imports

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

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

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

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

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

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

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

Global Attributes

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

Type Definitions

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

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

</xs:schema>
```

XML Global Element Declaration

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

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

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

</xsd:schema>
```

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

3.1.1 Root Element

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

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

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

[Definition] Control Schema –

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

Example:

```
    <!--====== Root Element ======-->
    <xsd:element name="Order" type="OrderType">
        <xsd:annotation>
                <xsd:documentation>
                        <ccts:CategoryCode>ABIE</ccts:CategoryCode>
                        <ccts:DictionaryEntryName>
                                Order. Details</ccts:DictionaryEntryName>
                        <ccts:Definition>A document that contains
information directly relating to the economic event of ordering products.
                        </ccts:Definition>
                        <ccts:ObjectClass>Order</ccts:ObjectClass>
                        <ccts:PropertyTerm>Details</ccts:PropertyTerm>

    <ccts:RepresentationTerm>Details</ccts:RepresentationTerm>
                        <ccts:BusinessTerm>PurchaseOrder</ccts:BusinessTerm>
                </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
```

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

Additional root element rules are contained in Section 8.

3.2 Constraints

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

3.2.1 Naming Constraints

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

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

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

3.2.2 Modeling Constraints

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

3.2.2.1 Defining Classes

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

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

Example:

Basic Business Information Entity

Aggregate Business Information Entity

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

3.2.2.2 Core Component Types

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

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

3.2.2.3 Consistent Business Function in Customizations

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

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

3.2.2.4 Mixed Content Models

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

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

3.3 Reusability Scheme

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

3.3.1 Managing by Types

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

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

3.3.1.1 Achieving the Assembly Use Case with Reusable Types

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

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

```
<xsd:complexType name="PartyType">
  <xsd:element name="PartyID" type="IdentifierType"/>
  <xsd:element name="Name" type="NameType"/>
  <xsd:element name="Address" type="AddressType" minOccurs="0"/>
  ...
</xsd:complexType>

<xsd:complexType name="AddressType">
  <xsd:element name="Identifier" type="IdentifierType" minOccurs="0"/>
  <xsd:element name="Street" type="StreetType" minOccurs="0"/>
  ...
```

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

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

```
//Party/Address
//Address
```

There are two choices for reuse:

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

Choice #1 would look like this:

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

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

```
<foo:Foo
 xmlns:foo=”some_namespace_name_for_foo”>
 ...
 <foo:FooAddress>
  <Identifier>...</Identifier>  <!-- real UBL element -->
  <Street>...</Street>          <!-- real UBL element -->
 </foo:FooAddress>
 ...
</foo:Foo>
```

Any `//Address` XPaths in stylesheets would have to be changed to `//foo:FooAddress` XPaths.

Choice #2 would look like this:

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

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

```
<foo:Foo
 xmlns:foo=”some_namespace_name_for_foo”>
 ...
 <foo:FooParty>          <!-- unwanted outer wrapper for Address -->
  <PartyID>...</PartyID> <!-- real UBL element; undesired -->
  <Name>...</Name>       <!-- real UBL element; undesired -->
  <Address>              <!-- real UBL element; desired -->
   <Identifier>...</Identifier>
   <Street>...</Street>
```

```
    ...
   </Address>
  </foo:FooParty>
  ...
</foo:Foo>
```

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

3.3.1.2 Reusable Elements

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

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

<xsd:complexType name="PartyType">
  <xsd:element ref="PartyID"/>
  <xsd:element ref="Name"/>
  <xsd:element ref="Address" minOccurs="0"/>
  ...
</xsd:complexType>

<xsd:element name="Address" type="AddressType">

<xsd:complexType name="AddressType">
  <xsd:element ref="Identifier" minOccurs="0"/>
  <xsd:element ref="Street" minOccurs="0"/>
  ...
</xsd:complexType>
```

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

XPaths will look like this:

```
//ubl:Party/ubl:Address
//ubl:Address
```

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

```
<xsd:element name="Foo" type="ubl:FooType"/>

<xsd:complexType name="FooType">
```

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

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

```
<foo:Foo
 xmlns:foo=”some_namespace_name_for_foo”
 xmlns:ubl=”ubl_namespace_name”>
 ...
 <ubl:Address>
  <ubl:Identifier>...</ubl:Identifier>
  <ubl:Street>...</ubl:Street>
  ...
 </ubl:Address>
 ...
</foo:Foo>
```

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

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

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

3.4 Namespace Scheme

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

[7] *See Note 4.*

3.4.1 Declaring Namespaces

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

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

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

> Definition. Schema Set
>
> A collection of schema instances that together comprise the names in a specific UBL namespace.

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

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

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

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

3.4.2 Namespace Uniform Resource Identifiers

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

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

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

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

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

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

Note:

[ ] = optional.

<> = variable

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

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

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

3.4.3 Schema Location

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

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

[NMS6] UBL Schema modules MUST be hosted under the UBL committee directory:

`http://www.oasis-open.org/committees/ubl/schema/<schema-mod-name>.xsd`

3.4.4 Persistence

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

[NMS7] UBL published namespaces MUST never be changed.

3.5 Versioning Scheme

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

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

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

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

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

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

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

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

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

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

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

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

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

Example:

Example

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

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

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

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

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

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

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

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

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

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

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

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

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

[VER4]	Every UBL Schema and schema module major version number MUST be sequentially assigned, incremental number greater than zero.
[VER5]	Every UBL Schema and schema module minor version number MUST be a sequentially assigned, incremental non-negative integer.

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

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

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

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

Must import the namespace:

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

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

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

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

Example

```
urn:oasis:names:tc:ubl:schema:Invoice:1:2 imports
urn:oasis:names:tc:ubl:schema:Invoice:1:1 which imports
urn:oasis:names:tc:ubl:schema:Invoice:1:0.
```

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

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

[VER8]	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.

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

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

3.6 Modularity

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

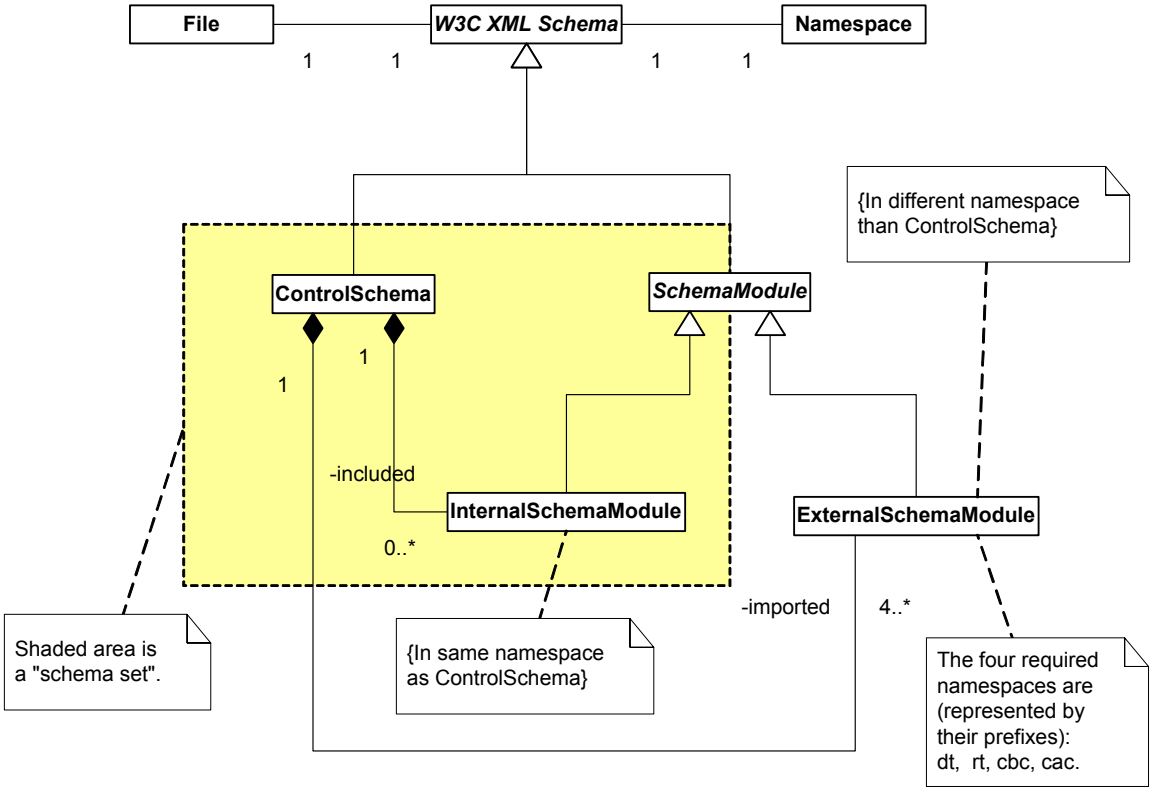
[SSM1]	UBL Schema expressions MAY be split into multiple schema modules.

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

3.6.1 UBL Modularity Model

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

Figure 3-1. UBL Schema Modularity Model



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

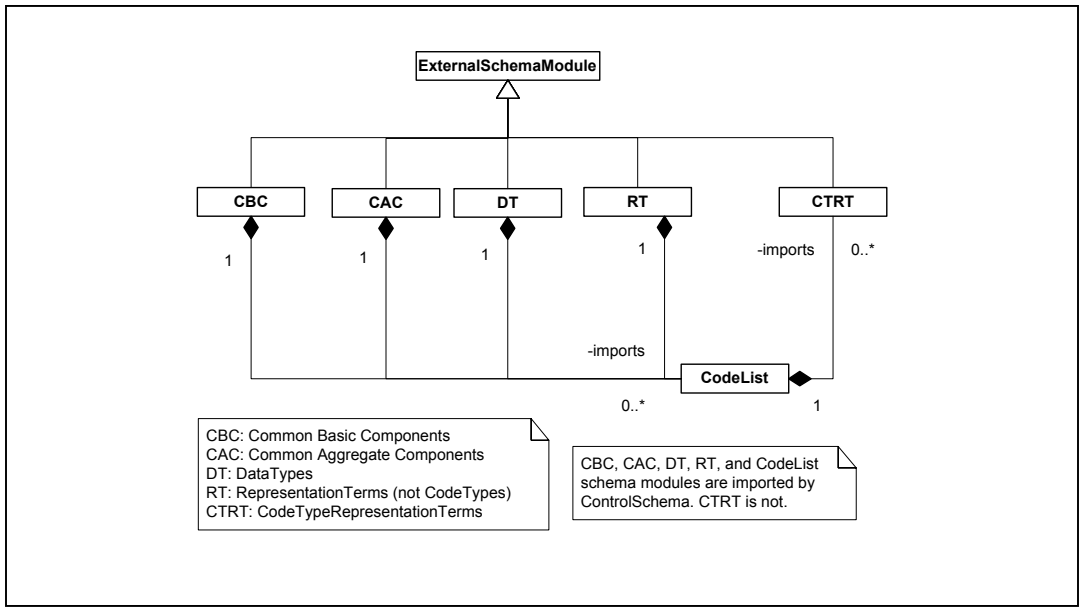
A namespace is an indivisible grouping of types. A “piece” of a namespace can never be used without all its pieces. For larger namespaces, schema modules – internal schema modules – may be defined. UBL control schema may have zero or more internal modules that it includes. The control schema for that namespace then includes those internal modules.

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

[Ed. Note – Is this definition correct?]

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

1134 ***Figure 3-2 Classes***

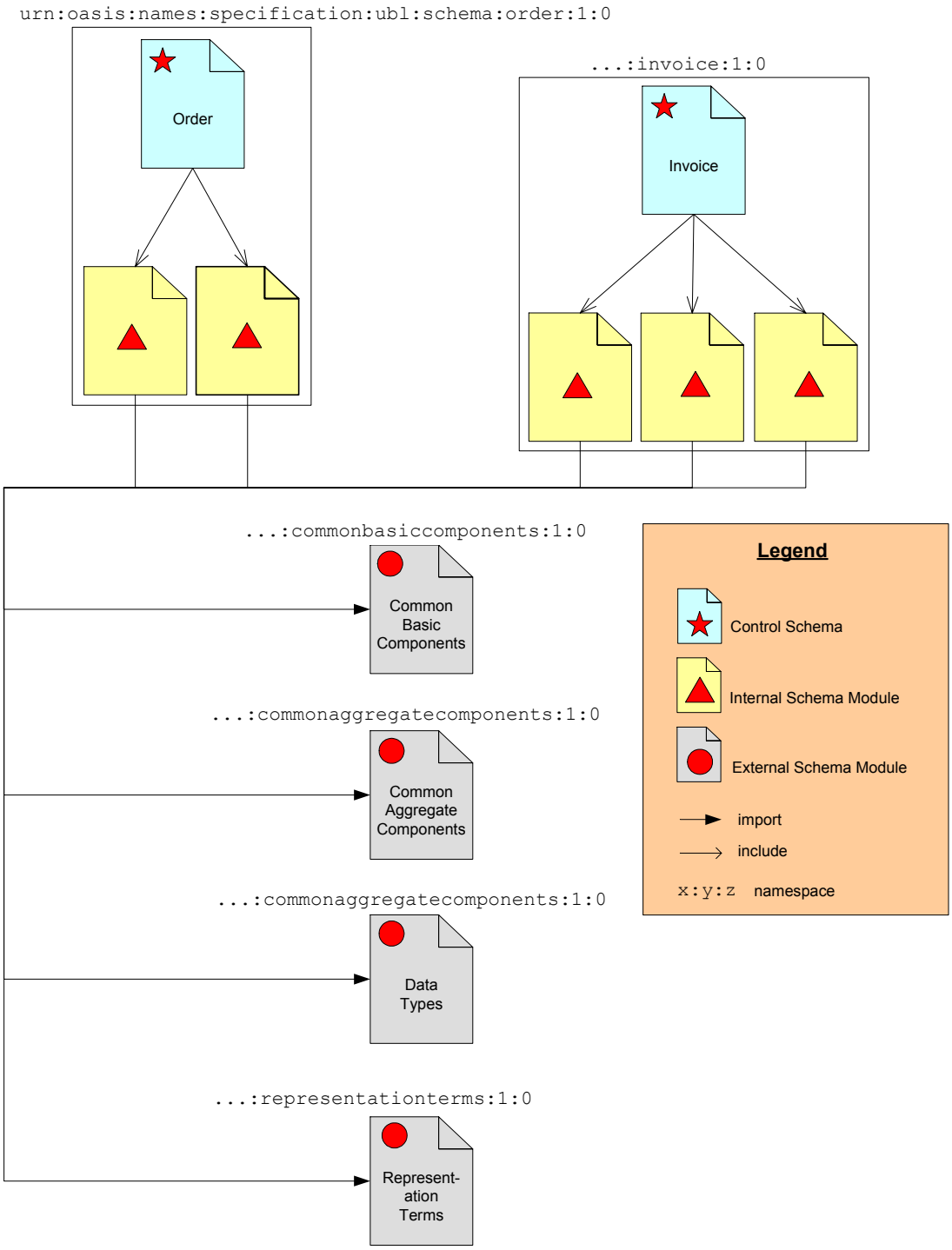


1135

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

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

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



1144

1145 ### 3.6.1.1 Limitations on Import

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

between namespaces. A namespace "A" dependent upon type definitions or element declaration defined in another namespace "B" must import "B's" control schema.

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

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

[SSM3]	A UBL control 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.

3.6.1.2 Module Conformance

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

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

3.6.2 Internal and External schema modules

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

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

3.6.3 Internal schema modules

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

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

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

[SSM7]	Each UBL internal schema module MUST be named `{ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}`

[Ed. Note – need example here].

3.6.4 External schema modules

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

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

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

- UBL CommonAggregateComponents
- UBL CommonBasicComponents
- CCTS Core Component Types
- CCTS Representation Terms
- CCTS Code Type Representation Term
- CCTS Data Types

3.6.4.1 UBL CommonAggregateTypes schema module

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

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

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

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

3.6.4.1.1 UBL CommonAggregateTypes schema module Namespace

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

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

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

[NMS9]	The `ubl:CommonAggregateComponents` schema module MUST be represented by the token "`cac`".

3.6.4.2 UBL CommonBasicTypes schema module

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

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

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

[SSM12]	The `ubl:CommonBasicComponents` schema module MUST be named *"ubl:CommonBasicComponents Schema Module"*

3.6.4.2.1 UBL CommonBasicTypes schema module Namespace

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

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

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

[NMS11]	The `UBL:CommonBasicComponents` schema module MUST be represented by the token "`cbc`".

3.6.4.3 CCTS Core Component Type schema module

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

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

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

[SSM14]	The `ccts:CoreComponentType` schema module MUST be named *"ccts:CoreComponentType Schema Module"*

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

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

3.6.4.3.1 Core Component Type schema module Namespace

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

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

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

[NMS13]	The `ccts:CoreComponentType` schema module namespace MUST be represented by the token "`cct`".

3.6.4.4 CCTS Representation Term schema module

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

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

[SSM16]	A schema module defining all ccts:PrimaryRepresentationTerms and ccts:SecondaryRepresentationTerms with the exception of ccts:CodeType MUST be created
[SSM17]	A schema module defining the ccts:CodeType ccts:RepresentationTerm MUST be created.

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

[SSM17]	The ccts:RepresentationTerm schema module MUST be named *"ccts:RepresentationTerm Schema Module"*
[SSM19]	The ccts:CodeTypeRepresentationTerm schema module MUST be named *"ccts:CodeTypeRepresentationTerm Schema Module"*

3.6.4.4.1 CCTS Representation Term schema module Namespace

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

[NMS14]	The ccts:RepresentationTerm schema module MUST reside in its own namespace.
[NMS15]	The ccts:CodeTypeRepresentationTerm schema module MUST reside in the ccts:RepresentationTerm namespace.

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

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

3.6.4.5 UBL Datatypes

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

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

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

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

[SSM19]	The UBL:Datatypes schema module MUST be named "ubl:Datatypes schema module"

3.6.4.5.1 UBL Datatype schema module Namespace

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

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

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

[NMS17]	The ubl:Datatypes schema module namespace MUST be represented by the token "dt".

3.7 Documentation

The UBL documentation also includes definitions of:

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

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

3.7.1 Embedded documentation

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

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

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

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

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

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

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

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

- 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.7.2 Schema Annotation

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

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

4 Naming Rules

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

- 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.

- Lower-level element: An element that appears inside a UBL business message.

- Intermediate element: An element not at the top level that is of a complex type, only containing other elements and attributes.

- 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.

- 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.

4.1 General Naming Rules

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

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

[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.

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

Example:

TransportEquipmentSeal
ChargeIndicator

UBL attribute names will be in lowerCamelCase (LCC).

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

Example:

AmountCurrencyCodeListVersionIdentifier
binaryObjectFilenameText

4.2 Type Naming Rules

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

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

4.2.1 Complex Type Names for CCTS Aggregate Business Information Entities

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

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

Example:

Ccts:AggregateBusinessInformationEntity	UBL xsd:complexType

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

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

4.2.2 Complex Type Names for CCTS Basic Business Information Entities

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

Example:

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

4.2.3 Complex Type Names for CCTS Representation Terms

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

Example:

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

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

Example:

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

4.2.4 Complex Type Names for CCTS Core Component Types

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

Example:

```
<!-- ===== CCT: QuantityType ===== -->
<xsd:complexType name="QuantityType">
        ...
</xsd:complexType>
```

4.3 Element Naming Rules

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

4.3.1 Element Names for CCTS Aggregate Business Information Entities

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

Example:

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

```
        <!--===== Aggregate Business Information Entity Type Definitions
=====-->
        <xsd:complexType name="TransportEquipmentSealType">
                ...
        </xsd:complexType>
        ...
        <!--===== Aggregate Business Information Entity Element
Declarations
        =====-->
        <xsd:element name="TransportEquipmentSeal"
                type="TransportEquipmentSealType"/>
```

4.3.2 Element Names for CCTS Basic Business Information Entities

The same naming concept applies to `ccts:BasicBusinessInformationEntities`

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

Example:

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

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

4.3.3 Element Names for CCTS Association Business Information Entities

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

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

Example:

[Ed. Note – need to insert example here]

4.4 Attribute Naming Rules

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

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

Example:

ccts:SupplementaryComponent	ubl:attribute

Example:

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

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

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

5 Declarations and Definitions

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

5.1 Type Definitions

5.1.1 General Type Definitions

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

[GTD1]	All types MUST be named.

Example:

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

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

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

5.1.2 Simple Types

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

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

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

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

Example:

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

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

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

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

5.1.3 Complex Types

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

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

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

Example:

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

5.1.3.1 Aggregate Business Information Entities

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

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

Example:

```
<xsd:complexType name="ContactType">
        ...
        <xsd:sequence>
                ...
                <xsd:element ref="Name" minOccurs="0">
                        ...
                </xsd:element>
                <xsd:element ref="Phone" minOccurs="0">
                        ...
                </xsd:element>
                ...
        </xsd:sequence>
</xsd:complexType>
```

5.1.3.2 Basic Business Information Entities

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

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

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

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

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

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

Example:

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

5.1.3.3 Representation Terms

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

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

Example:

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

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

Example:

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

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

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

5.1.3.4 Core Component Types

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

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

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

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

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

Example:

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

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

5.1.3.5 Supplementary Components

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

[CTD12]	Each `CCT:SupplementaryComponent xsd:attribute` "type" MUST define the specific `xsd:built-in Datatype` or the user defined `xsd:simpleType` for the `ccts:SupplementaryComponent` of the `ccts:CCT`.

Example:

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

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

Example:

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

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

[ED NOTE: Add text here to explain].

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

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

Example:

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

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

```
        <xsd:simpleType name="CurrencyCodeContent">
                <xsd:restriction base="xsd:token">
                        <xsd:maxLength value="3"/>
                        ...
                </xsd:restriction>
        </xsd:simpleType>
```

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

[CTD15] Each `ccts:Supplementary Component xsd:attribute` "use" MUST define the occureance of that `ccts:SupplementaryComponent` as either "required", or "optional.

Example:

```
<!-- ===== CCT: AmountType ===== -->
        <xsd:complexType name="AmountType">
                ...
                <xsd:simpleContent>
                        <xsd:extension base="xsd:decimal">
                                <xsd:attribute name="currencyID"
type="xsd:token"
                                        use="required"/>
                                <xsd:attribute name="codeListVersionID"
type="xsd:token"
                                        use="optional"/>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
```

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

5.2 Element Declarations

5.2.1 General Element Declarations

5.2.2 Elements Bound to Complex Types

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

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

Example:

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

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

Example:

```
<xsd:element name="BuyerParty" type="BuyerPartyType"/>
<xsd:complexType name="BuyerPartyType">
        ...
</xsd:complexType>
```

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

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

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

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

5.2.2.1 Elements Representing AS BIEs

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

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

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

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

5.2.2.2 Elements Bound to Representation Terms

5.2.2.3 Elements Bound to Core Component Types

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

5.2.3 Code List Import

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

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

5.2.4 Empty Elements

[ELD7]	Empty elements MUST not be declared.

5.2.5 XSD:Any

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

5.3 Attribute Declarations

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

5.3.1 User Defined Attributes

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

5.3.2 Global Attributes

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

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

5.3.3 Supplementary Components

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

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

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

5.3.4 Schema Location

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

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

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

5.3.5 XSD:Nil

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

5.3.6 XSD:Any

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

6 Code Lists

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

An example URN is provided below.

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

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

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

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

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

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

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

7 Miscellaneous XSD Rules

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

7.1 XSD Simple Types

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

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

7.2 Namespace Declaration

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

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

7.3 XSD:Substitution Groups

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

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

7.4 XSD:Final

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

7.5 XSD: Notations

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

[Ed. Note – do we meen xsd:notation datatype?]

7.6 XSD:All

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

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

7.7 XSD:Choice

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

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

7.8 XSD:Include

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

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

7.9 XSD:Union

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

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

7.10 XSD:Appinfo

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

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

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

7.11 Extension and Restriction

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

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

8 Instance Documents

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

8.1 Root Element

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

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

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

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

Examples:

Order, OrderResponse, ChangeOrder, AdvanceShipNotice, AdvanceShipNoticeResponse.

8.2 Validation

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

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

8.3 Character Encoding

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

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

Example:

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

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

[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.

Example:

<?xml version=”1.0” encoding=”UTF-8” ?>

8.4 Schema Instance Namespace Declaration

The W3C XSD specification defines

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

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

8.5 Empty Content.

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

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

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

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

Appendix A. UBL NDR Checklist

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

Table A1 — Code List Rules (CDL)

Table A2 — Constraint Rules

- Modeling Constraints (MDC}
- Naming Constraints {NMC)

Table A3 — Declaration Rules

- Element Declarations (ELD)
- Attribute Declarations (ATD)

Table A4 — Documentation Rules (DOC)

Table A5 — General XSD Rules (GXS)

Table A6 — Instance Document Rules (IND)

Table A7 — Naming Rules

General Naming Rules (GNR)

Specific Naming Rules

- Element Naming Rules (ELN)
- Attribute Naming Rules (ATN)
- Type Naming Rules (CTN)

Table A8 — Namespace Rules (NMS)

Table A9 — Root Element Declaration Rules (RED)

Table A10 —

2442 ## Table A1 — Code List Rules

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

2443

2444

2445

Table A2. Constraint Rules

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

2447 ## Table A3 — Declarations Rules

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

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

2448

2448 ## Table A4. Documentation Rules

Rule Number	Rule
[DOC1]	Every Data Type definition MUST contain a structured set of annotations in the following patterns: ▪ UniqueIdentifier (mandatory): The identifier that references a Data Type instance in a unique and unambiguous way. ▪ CategoryCode (mandatory): The category to which the object belongs. For example, BBIE, ABIE, ASBIE, RT (Representation Term). ▪ DictionaryEntryName (mandatory): The official name of a Data Type. ▪ Definition (mandatory): The semantic meaning of a Data Type. ▪ Version (mandatory): An indication of the evolution over time of a Data Type instance. ▪ QualifierObjectClass (optional): The qualifier for the object class. ▪ ObjectClass: The Object Class represented by the Data Type. ▪ Qualifier Term (mandatory): A semantically meaningful name that differentiates the Data Type from its underlying Core Component Type. ▪ Usage Rule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Data Type.

[DOC2]	A Data Type definition MAY contain one or more Content Component Restrictions to provide additional information on the relationship between the Data Type and its corresponding Core Component Type. If used the Content Component Restrictions must contain a structured set of annotations in the following patterns: ▪ 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]	A Data Type definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the Data Type and its corresponding Core Component Type. If used the Supplementary Component Restrictions must contain a structured set of annotations in the following patterns: ▪ SupplementaryComponentName (mandatory): Identifies the Supplementary Component on which the restriction applies. ▪ RestrictionValue (mandatory, repetitive): The actual value(s) that is (are) valid for the Supplementary Component

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

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

DOC5

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

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

DOC6	Every Association Business Information Entity definition MUST contain a structured set of annotations in the following patterns: ▪ UniqueIdentifier (mandatory): The identifier that references an Association Business Information Entity instance in a unique and unambiguous way. ▪ CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be ASBIE. ▪ DictionaryEntryName (mandatory): The official name of an Association Business Information Entity. ▪ Definition (mandatory): The semantic meaning of an Association Business Information Entity. ▪ Version (mandatory): An indication of the evolution over time of an Association Business Information Entity instance. ▪ Cardinality (mandatory): Indication whether the Association Business Information Entity Property represents a not-applicable, optional, mandatory and/or repetitive characteristic of the Aggregate Business Information Entity. ▪ QualifierTerm (optional): Qualifies the Property Term of the associated Core Component Property in the associated Aggregate Core Component. ▪ UsageRule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Association Business Information Entity. ▪ ConstraintLanguage (optional, repetitive): A formal description of a way the Association Business Information Entity is derived from the corresponding stored Core Component and stored Business Context. ▪ BusinessTerm (optional, repetitive): A synonym term under which the Association Business Information Entity is commonly known and used in the business. ▪ Example (optional, repetitive): Example of a possible value of an Association Business Information Entity.

DOC7	Every Core Component definition MUST contain a structured set of annotations in the following patterns: ▪ UniqueIdentifier (mandatory): The identifier that references a Core Component instance in a unique and unambiguous way. ▪ CategoryCode (mandatory): The category to which the object belongs. In this case the value will always be CCT. ▪ DictionaryEntryName (mandatory): The official name of a Core Component. ▪ Definition (mandatory): The semantic meaning of a Core Component. ▪ ObjectClass: The Object Class represented by the type. ▪ PropertyTerm: The Property Term represented by the type. ▪ Version (mandatory): An indication of the evolution over time of a Core Component instance. ▪ Usage Rule (optional, repetitive): A constraint that describes specific conditions that are applicable to the Basic Business Information Entity. ▪ Business Term (optional, repetitive): A synonym term under which the Basic Business Information Entity is commonly known and used in the business.
[DOC8]	Every element declaration MUST contain an annotation as follows: <Documentation>Dictionary Entry Name</Documentation> where Dictionary Entry Name is the complete name (not the tag name) that is the unique official name of the element in the UBL library.
[DOC9]	For each UBL construct containing a code, the UBL documentation MUST identify the zero or more code lists that MUST be minimally supported when the construct is used.

2449

2450

2450 ## Table A5. General XSD Rules

Rule Number	Rule
[GXS1]	UBL Schema MUST conform to the following physical layout as applicable: XML Declaration <!-- ===== Copyright Notice ===== --> “Copyright © 2001-2004 The Organization for the Advancement of Structured Information Standards (OASIS). All rights reserved. <!-- ===== xsd:schema Element With Namespaces Declarations ===== -->> xsd:schema element to include version attribute and namespace declarations in the following order: xmlns:xsd Target namespace Default namespace CommonAggregateComponents CommonBasicComponents CoreComponentTypes Datatypes Identifier Schemes Code Lists Attribute Declarations – elementFormDefault=”qualified” attributeFormDefault=”unqualified” <!-- ===== Imports ===== -->CommonAggregateComponents schema module CommonBasicComponents schema module Representation Term schema module (to include CCT module) Common Basic Types schema module Common Aggregate Types schema module <!-- ===== Global Attributes ===== --> Global Attributes and Attribute Groups <!-- ===== Root Element ===== -->

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

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

2451

2452

2452 ## Table A6 —Instance Documents

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

2453

2453 ## Table A7 — Naming Rules

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

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

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

2454

2455

Table A8 — Namespace Rules

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

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

2456

2457

2458

2458 ## Table A9 — Root Element Declaration Rules

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

2459

2460

Table A10 — Schema Structure Modularity Rules

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

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

2461

2462

2462 ## Table A11 — Standards Adherence Rules

Rule Number	Rule
[STA1]	All UBL schema design rules MUST be based on the W3C XML Schema Recommendations: *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.

2463

2464

2464 ## Table A12 — Type Definition Rules

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

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

[CTD15]	Each `ccts:Supplementary Component xsd:attribute` "use" MUST define the occureance of that `ccts:SupplementaryComponent` as either "required", or "optional.
[CTD16]	Each `ccts:CCT simpleType` definition name MUST be the `ccts:CCT` dictionary entry name with the separators removed.

2465

2465 ## Table A13 — Versioning Rules

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

[VER10]	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.
[VER11]	UBL Schema and schema module minor version changes MUST not break semantic compatibility with prior versions.

2466

Appendix B. Approved Acronyms and Abbreviations

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

- A Dun & Bradstreet number *must* appear as "DUNS". [TBD: need example.]
- "Identifier" *must* appear as "ID".
- "Uniform Resource Identifier" *must* appear as "URI"
- [Example] the "Uniform Resource. Identifier" portion of the **Binary Object. Uniform Resource. Identifier** supplementary component becomes "URI" in the resulting XML name). The use of URI for Uniform Resource Identifier takes precedence over the use of "ID" for "Identifier".

Appendix C. Technical Terminology

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

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

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

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

	may also have attributes). Note that, because of the XSD mechanisms involved, a leaf element that has attributes must be declared as having a complex type, but a leaf element with no attributes may be declared with either a simple type or a complex type.
Lower-level element	An element that appears inside a business message.
Object Class	The logical data grouping (in a logical data model) to which a data element belongs (ISO11179). The *Object Class* is the part of a *Core Component*'s *Dictionary Entry Name* that represents an activity or object in a specific *Context*.
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 *Core Components* and *Business Information Entities* are constructed.
Schema	Never use this term unqualified!
schema module	A "schema document" (as defined by the XSD spec) that is intended to be taken in combination with other such schema documents to be used.
Schema module:	A schema document containing type definitions and element declarations.
Schema Processing	Schema validation checking plus provision of default values and provision of new 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.
Syntax Neutral Model	TBD Need definition.
Aggregate Business Information Entity (ABIE)	A collection of related pieces of business information that together convey a distinct business meaning in a specific Business Context. Expressed in modelling terms, it is the representation of an Object Class, in a specific Business Context.
Well-Formedness Checking	Basic XML 1.0 adherence.

2479

Appendix D. References

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

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

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

[ISONaming] *ISO/IEC 11179,* Final committee draft, Parts 1-6.

(RFC) 2119 S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

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

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

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

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

Appendix E. Notices

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

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