Universal Business Language (UBL) 2.1 Naming and Design Rules

Committee Draft in Progress 20090817, 16 August 2010

Document identifier:

prd2-UBL-NDR-20090817

Locations:

Persistent version: http://docs.oasis-open.org/ubl/prd2-UBL-NDR-2.0/
Current version: http://docs.oasis-open.org/ubl/prd2-UBL-NDR-2.0/

Technical committee:

OASIS Universal Business Language (UBL) TC

Chairs:

Jon Bosak, Pinax <bosak@pinax.com>
Tim McGrath, Document Engineering Services <tim.mcgrath@documentengineeringservices.com>

Editors:

Mike Grimley, US Navy <MJGrimley@acm.org>
Mavis Cournane, Cognitran Limited <mavis.Cournane@cognitran.com>

Abstract:

This specification documents the naming and design rules and guidelines for the construction of XML components for the UBL vocabulary.

Status:

This document was last revised or approved by the UBL TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/ubl.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ubl/ipr.php).

The non-normative errata page (if any) for this specification is located at http://www.oasis-open.org/committees/ubl.

Table of Contents

1. Introduction 4
    1.1. Audiences 4
    1.2. Scope 4
    1.3. Terminology and Notation 4
    1.4. Guiding Principles 5
2. Relationship to ebXML Core Components 5
    2.1. Mapping Business Information Entities to XSD 6
3. General XML Constructs 7
    3.1. Overall Schema Structure 7
    3.2. Naming and Modeling Constraints 9
    3.3. Reusability Scheme 10
    3.4. Extension Scheme 11
    3.5. Namespace Scheme 11
    3.6. Versioning Scheme 12
    3.7. Modularity Strategy 14
4. Annotation and Documentation Requirements 18
    4.1. Schema Annotation 18
    4.2. Embedded Documentation 18
5. Naming Rules 21
    5.1. General Naming Rules 21
    5.2. Type Naming Rules 23
    5.3. Element Naming Rules 24
    5.4. Attributes in UBL 26
6. Declarations and Definitions 26
    6.1. Type Definitions 26
    6.2. Element Declarations 28
    6.3. Empty Elements 29
7. Code Lists 29
8. Miscellaneous XSD Rules 29
    8.1. xsd:simpleType 29
    8.2. Namespace Declaration 30
    8.3. xsd:substitutionGroup 30
    8.4. xsd:final 30
    8.5. xsd: notation 30
    8.6. xsd:all 30
    8.7. xsd:choice 30
    8.8. xsd:include 31
    8.9. xsd:union 31
    8.10. xsd:appinfo 31
    8.11. xsd:schemaLocation 31
    8.12. xsd:nillable 31
    8.13. xsd:any 31
    8.14. Extension and Restriction 32

Appendixes

A. Code List Metadata (Informative) 32
B. UBL-approved Acronyms and Abbreviations (Informative) 33
C. Technical Terminology (Informative) 34
D. References 36
E. Notices 37

F. UBL NDR Checklist 37

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. 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.

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 Technical Committee. Consumers of the Naming and Design Rules Specification should consult previous UBL position papers that are available at http://www.oasis-open.org/committees/ubl/ndrsc/. These provide a useful background to the development of the current rule set.

1.1. Audiences

This document has several primary and secondary targets that together constitute its intended audience. Our primary target audience is the members of the UBL Technical Committee. Specifically, the UBL Technical Committee uses the rules in this document to create normative form schemas for business transactions. 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 UBL schemas for business documents being exchanged between two parties using XML constructs defined 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	An example of a definition or a rule. Examples are informative.

Note	Explanatory information. Notes are informative.
RRR*n*	Identifier of a rule to which an XML schema must comply in order to be 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, “Rule Prefix Value”, and n (1..n) is the sequential number of the rule within its category. To ensure continuity across versions of the specification, rule numbers that are deleted in future versions will not be reissued, and any new rules will be assigned the next higher number — regardless of location in the text. Only rules and definitions are normative; all other text is explanatory.

Table 1. Rule Prefix Value

Rule Prefix	Value
CDL	Code List
CTD	ComplexType Definition
CTN	ComplexType Naming Rules
DOC	Documentation
ELD	Element Declaration
ELN	Element Naming
GNR	General Naming
GTD	General Type Definition
GXS	General XML Schema
MDC	Modeling Constraints
NMC	Naming Constraints
NMS	Namespace
RED	Root Element Declaration
SSM	Schema Structure Modularity
VER	Versioning

The term "XSD" is used throughout this document to refer to Parts 1 and 2 of the W3C *XML Schema Definition Language* (XSD) Recommendation.

1.4. Guiding Principles

The UBL NDR primary objectives are to provide the UBL TC with a set of unambiguous, consistent rules for the development of extensible, reusable UBL schemas.

2. Relationship to ebXML Core Components

UBL employs the methodology and model described in *ISO TS 15000-5:2005 -- ebXML Core Components Technical Specification, Version 2.01 [CCTS]* to build the UBL Component Library. CCTS defines a new paradigm in the design and implementation of reusable, syntactically neutral information building blocks. Syntax-neutral Core Components are intended to form the basis of business information standardization efforts and to be realized in syntactically specific instantiations such as ANSI ASC X12, UN/EDIFACT, and various XML representations such as UBL.

Context-neutral and context-specific building blocks are the essence of the Core Components specification. The context-neutral components are called Core Components. A Core Component is 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". Figure 1 illustrates the various pieces of the overall Core Components metamodel.

The context-specific components are called Business Information Entities (BIEs). A BIE is defined in CCTS as "a piece of business data or a group of pieces of business data with a unique Business Semantic definition". Figure 2 illustrates the various pieces of the overall BIE metamodel and its relationship to the Core Components metamodel. As shown here, there are different types of Core Components and BIEs, each of which has specific relationships to the other components and entities. The context-neutral Core Components establish the formal relationship between the various context-specific BIEs.

Figure 1. Core Components and Datatypes Metamodel

Figure 2. Business Information Entities Basic Definition Model

2.1. Mapping Business Information Entities to XSD

UBL consists of a library of CCTS BIEs, each of which is mapped to an XSD construct (See Figure 3).

Figure 3. UBL Document Metamodel

A BIE can be a CCTS Aggregate Business Information Entity (ABIE), a CCTS Basic Business Information Entity (BBIE), or a CCTS Association Business Information Entity (ASBIE). In understanding the logic of the UBL binding of BIEs to XSD expressions, it is important to understand the basic constructs of the BIEs and their relationships as shown in Figure 2. The ABIEs are treated as objects and are defined as xsd:complexTypes. The BBIEs are treated as properties of the ABIE and are found in the content model of the ABIE as a referenced xsd:element. The BBIEs are based on reusable CCTS Basic Business Information Entity Properties (BBIE Properties), which are defined as xsd:complexTypes.

A BBIE Property represents an intrinsic property of an ABIE. BBIE Properties are linked to a data type.

CCTS defines an approved set of primary and secondary representation terms. However, these representation terms are simply naming conventions to identify the data type of an object, not actual constructs.

There are two kinds of BIE Properties — Basic and Association. A CCTS Association BIE Property (ASBIE Property) represents an extrinsic property — in other words, an association from one ABIE instance to another ABIE instance. It is the ASBIE Property that expresses the relationship between ABIEs.

Due to their unique extrinsic association role, ASBIEs are not defined as xsd:complexTypes; rather, they are either declared as elements that are then bound to the xsd:complexType of the associated ABIE, or they are reclassified as ABIEs.

BBIEs define the intrinsic structure of an ABIE. These BBIEs are the "leaf" types in the system in that they contain no other BIEs.

A BBIE must have a CCTS Core Component Type. All CCTS Core Component Types are low-level types such as Identifiers and Dates. A CCTS Core Component Type describes these low-level types for use by CCTS Core Components, and (in parallel) a CCTS data type, corresponding to that CCTS Core Component Type, describes these low-level types for use by BBIEs. Every CCTS Core Component Type has a single CCTS Content Component and one or more CCTS Supplementary Components. A CCTS Content Component is of some Primitive Type. All CCTS Core Component Types and their corresponding content and supplementary components are predefined in CCTS.

UBL has developed an XSD schema module that declares each of the predefined CCTS Core Component Types as an xsd:complexType or xsd:simpleType and declares each CCTS Supplementary Component as an xsd:attribute or uses the predefined facets of the built-in XSD datatypes for those that are used as the base expression for an xsd:simpleType.

3. General XML Constructs

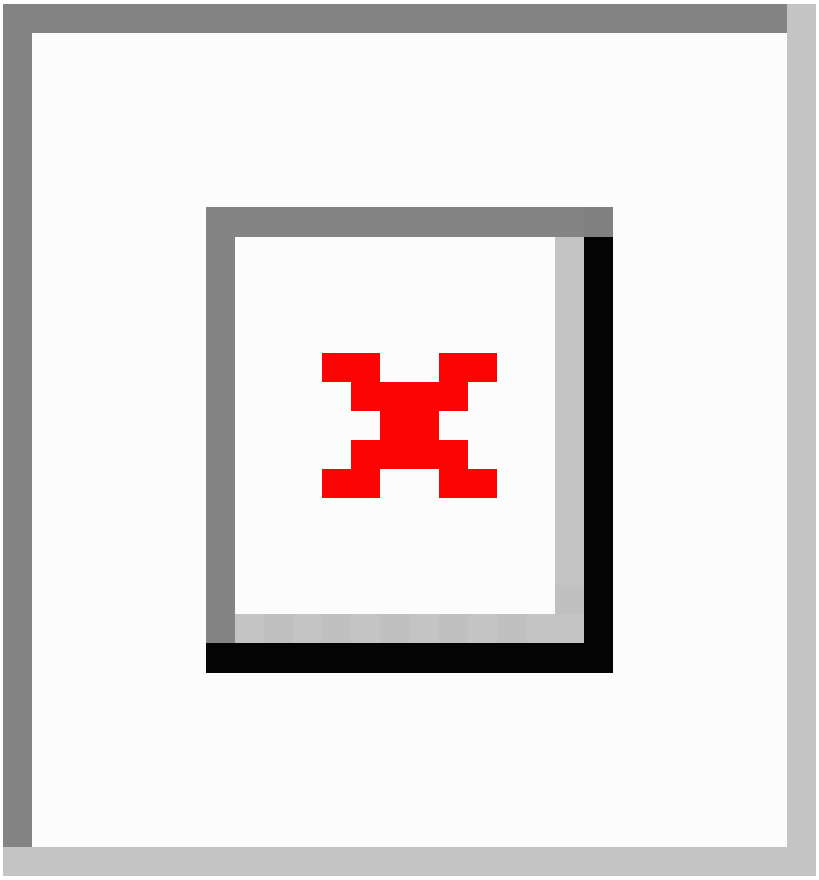
This chapter defines UBL rules related to general XML constructs, including overall schema structure, naming and modeling constraints, reusability, namespaces, versioning, modularity, and documentation.

3.1. Overall Schema Structure

A key aspect of developing standards is to ensure consistency in their implementation. Therefore, 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*] Except in the case of extension, where the "UBL Extensions" element is used, UBL schemas SHOULD conform to the following physical layout as applicable: See Figure 4.

Figure 4. Physical layout



As shown above, a UBL schema should contain a comment block at the top of the schema that functions as a "schema header".

3.1.1. Element Declarations within Document Schemas

A document schema is a schema within a specific namespace that conveys the business document functionality of that namespace. The document schema declares a target namespace and is likely to include (xsd:include) internal schema modules or import (xsd:import) external schema modules. Each namespace will have one, and only one, major version of a document schema as well as any related minor versions.

In order to facilitate the management and reuse of UBL constructs, all global elements, excluding the root element of the document schema, must be declared in either the Common Aggregate Components (CAC) or Common Basic Components (CBC) schema modules and referenced from within the document schema.

3.1.2. Root Element

Only a single global element is declared inside a UBL document schema. The single global element is the root element of every conforming instance.

[*RED2*] The root element MUST be the only global element declared in the document schema.

3.2. Naming and Modeling Constraints

UBL has the following naming and modeling constraints.

3.2.1. Naming Constraints

A primary aspect of the UBL library documentation is its spreadsheet models. The entries in these spreadsheet models fully define the constructs available for use in UBL business documents. The spreadsheet entries contain fully conformant CCTS Dictionary Entry Names (DENs) as well as truncated UBL XML element names developed in conformance with the rules in Section 4. The XML element name is the short form of the DEN. The rules for element naming differ from the rules for DEN naming.

> [*NMC1*] Each Dictionary Entry Name MUST define one and only one fully qualified path (FQP) for an element or attribute.

The FQP anchors the use of the element or attribute to a particular location in a business message. Any semantic dependencies that the element or attribute has on other elements and attributes within the UBL library that are not otherwise enforced or made explicit in its structural definition can be found in its prose definition.

3.2.1.1. Modeling Constraints

Modeling constraints are limited to those necessary to ensure consistency in development of the UBL library.

3.2.1.1.1. Defining Classes

UBL is based on instantiating ebXML CCTS BIEs. UBL models and the XML expressions of those models are class driven. Specifically, the UBL library defines classes for each CCTS ABIE and the UBL schemas instantiate those classes. The properties of those classes consist of CCTS BBIEs and ASBIEs.

3.2.1.1.2. Core Component Types

Each BBIE is associated with one of an approved set of CCTS Core Component Types.

> [*MDC1*] UBL libraries and schemas MUST only use CCTS Core Component Types, except in the case of extension, where the UBLExtensions element is used.

3.2.1.1.3. XML Mixed Content

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

> [*MDC2*] XML mixed content MUST NOT be used except where contained in an xsd:documentation element.

3.2.1.1.4. Sequencing

In the UBL model, the prescribed order for the contents of an ABIE is that ASBIEs follow BBIEs. However, this is, strictly speaking, a rule of the modeling methodology rather than an NDR. The NDR in this case is that the sequential order of entities in the model must be preserved.

> [*MDC0*] The sequence of the business information entities that is expressed in the UBL model MUST be preserved in the schema.

3.3. Reusability Scheme

To promote effective management of the UBL library, all element declarations are unique. Consequently, UBL elements are declared globally.

3.3.1. Reusable Elements

UBL elements are global and qualified. Hence in the example below, the Address element is directly reusable as a modular component.

Example 1.

```
<xsd:element name="Party" type="PartyType"/>
  <xsd:complexType name="PartyType">
    <xsd:annotation>
      <!-- Documentation goes here -->
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="PartyIdentification" minOccurs="0" maxOccurs="unbounded">
          ...
      </xsd:element>
      <xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="Address" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
          ...
    </xsd:sequence>
  </xsd:complexType>

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

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

Software written to work with UBL's standard library should 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 their placement within a particular type.

[*ELD2*] All element declarations MUST be global.

3.4. Extension Scheme

Some organizations are required by law to send additional information not covered by the UBL document structure, thus requiring an extension to the UBL message. The xsd:any construct is seen as the most efficient way to implement this requirement.

In general, UBL restricts the use of xsd:any because this feature permits the introduction of unknown elements into an XML instance. However, limiting its use to a single, predefined element mitigates this risk. For meaningful validation of UBL document instances, the value of the xsd:processContents attribute of the element must be set to "skip", thereby removing the potential for errors in the validation layer. Extension imposes cardinality constraints.

The following rules apply in the order below.

> [*ELD12*] The UBL Extensions element MUST be declared as the first child of the document element with xsd:minOccurs="0".

> [*ELD13*] The UBLProfileID element MUST be declared immediately following the UBL Extensions element with xsd:minOccurs="0".

> [*ELD14*] The UBLSubsetID element MUST be declared immediately following the UBLProfileID element with xsd:minOccurs="0".

3.5. Namespace Scheme

The concept of XML namespaces is defined in the W3C XML namespaces technical specification. The use of XML namespace is specified in the W3C XML Schema (XSD) Recommendation. 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. For each UBL namespace, a normative token is defined as its prefix. These tokens (currently udt, qdt, cac, cbc, ext) are defined in Section 3.7.

3.5.1. Declaring Namespaces

Neither XML 1.0 nor XSD requires the use of namespaces. However, the use of namespaces is essential to managing the complex UBL library. UBL uses UBL-defined schemas (created by the UBL TC) 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, except internal schema modules, MUST declare a namespace using the xsd:targetNamespace attribute.

Each UBL schema module consists of a logical grouping of lower level artefacts that can be used in a variety of UBL schemas. These schema modules are grouped into a schema set. Each schema set is assigned a namespace that identifies that group of schema modules. As constructs are changed, new versions are to be created. The schema set is the versioned entity; all schema modules within that package are of the same version, and each major version has a unique namespace.

Schema set	A collection of schemas that constitute a specific UBL namespace.

Schema validation ensures that an instance conforms to its declared schema. In keeping with Rule NMS1, each UBL schema module is part of a versioned namespace.

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

UBL's extension methodology encourages a wide variety in the number of schema modules that are created as derivations from UBL schema modules. Customized schemas should not be confused with those developed by UBL.

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

3.5.2. Namespace Uniform Resource Identifiers

A UBL namespace name must be a URI that conforms to RFC 2396. UBL has adopted the Uniform Resource Name (URN) scheme as the standard for URIs for UBL namespaces, in conformance with IETF's RFC 3121.

Rule NMS2 requires separate namespaces for each UBL major version schema set. In accordance with OASIS procedures, the UBL namespace 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:<subtype>:<document-id>

The format for document-id is found in Section 3.6.

For each UBL schema holding OASIS Committee Specification or Standard status, a UBL namespace must be declared and named using the same notation, but with the value "specification" replacing the value "tc".

> [*NMS5*] The namespace names for UBL schemas holding OASIS Standard status MUST be of the form
>
> urn:oasis:names:specification:ubl:schema:<subtype>:<document-id>

3.5.3. Schema Location

UBL schemas use a URN namespace scheme. In contrast, schema locations are defined as a Uniform Resource Locator (URL). UBL schemas must be available both at design time and run time. Therefore, the UBL schema locations will differ from the UBL namespace declarations. UBL uses an OASIS URL for hosting retrievable copies of UBL schemas.

3.5.4. Persistence

UBL namespaces use URNs to provide name persistence. UBL namespaces must never change once they have been declared. Conversely, changes to a schema may result in a new namespace declaration. Thus, a published schema version and its namespace association will always be inviolate.

> [*NMS6*] UBL published namespaces MUST never be changed.

3.6. Versioning Scheme

UBL distinguishes between major versions and minor versions. Major versions are not backwards compatible. Minor versions do not break backwards compatibility. In other words, a document instance that validates against version 1 of the schema must also validate against version 1.1 of the schema, where version 1.1 is a minor version change based on version 1. However, the same document instances would not necessarily be valid against version 2 of the schema, where version 2 is a major version change.

Versioning information is indicated both in the namespace URI and in the version attribute of the schema module. However, this information is represented somewhat differently in these two locations.

3.6.1. Versioning Information in the Namespace URI

UBL namespaces conform to the OASIS namespace rules defined in RFC 3121. All UBL namespace URIs have the form:

```
urn:oasis:names:specification:ubl:schema:xsd:<modulename>-<major>
```

where <modulename> is the name of the schema module and <major> is a positive integer representing the major version. The field containing <modulename>-<major> is called the *document-id*.

> [*VER2*] Every UBL schema module major version MUST have an RFC 3121 document-id of the form <*modulename*>-<*major*>

> [*VER6*] Every UBL schema module major version number MUST be a sequentially assigned integer greater than zero.

The value of <major> is "1" for the first release of a namespace. For example, the namespace URI for the first major release of the Invoice domain has the form:

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-1
```

Subsequent major releases increment the value by 1. For example, the second major release of the Invoice domain has the URI

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-2
```

The rule for minor version releases is as follows:

> [*VER4*] Every minor version release of a UBL schema module MUST have a document-id of the form <*modulename*>-<*major*>

For example, the fifth minor version of the release based on the second major release mentioned above will have the URI

```
urn:oasis:names:specification:ubl:schema:xsd:Invoice-2
```

As can be seen, both the rule and the example for the minor version releases is exactly the same as that for the major version. There is even a rule stating this directly.

> [*VER5*] For UBL minor version changes, the namespace name MUST not change.

However, minor versioning is handled differently in the xsd:schema element.

3.6.2. Versioning representation in the xsd:schema element

UBL uses the version attribute in the xsd:schema element to convey minor version releases of the schema module.

> [*VER12*] Every major version release of a UBL schema module MUST capture its version number in the xsd:version attribute of the xsd:schema element in the form <*major*>.*0*

> [*VER14*] Every minor version release of a UBL schema module MUST capture its version information in the xsd:version attribute in the form <*major*>.<*non-zero*>

> [*VER7*] Every UBL schema module minor version number MUST be a sequentially assigned, non-negative integer.

3.6.3. Instance Versioning

UBL version information can also be captured in instances of UBL document schemas via the ubl:UBLVersionID element.

> [*VER15*] Every UBL document schema MUST declare an optional element named UBLVersionID immediately following the optional UBL Extensions element.

3.7. Modularity Strategy

There are many possible mappings of XML schema constructs to namespaces and to files. In addition to the logical taming of complexity that namespaces provide, dividing the physical realization of schemas into multiple schema modules provides a mechanism whereby reusable components can be imported as needed without the need to import complete schemas.

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

Schema module	A schema document containing type definitions and element declarations intended to be reused in multiple schemas.

3.7.1. UBL Modularity Model

UBL relies extensively on modularity in schema design. There is no single UBL root schema. Rather, there are a number of UBL document schemas used to perform different business functions. UBL is structured so that users can reuse individual document schemas without having to import the entire UBL document schema library. A document schema can import individual modules without having to import all UBL schema modules. Each document schema defines its own dependencies. The UBL schema modularity approach reflects logical associations that exist between document and internal schema modules,and it ensures that individual modules can be reused to the maximum extent possible. If the contents of a namespace are small enough then they can be completely specified within a single document. Document and internal schema modules are shown in Figure 5.

Figure 5. UBL Schema Modularity Model

Figure 5 shows the one-to-one correspondence between document schemas and namespaces. It also shows the one-to-one correspondence between files and schema modules. As shown here, there are two types of schemas in the UBL library — document schemas and schema modules. Both types of schemas are conformant with XSD.

Each document schema occupies its own namespace and may include zero or more internal modules. The namespace for a document schema includes any of its internal modules. Schema modules that are not internal to a document occupy a different namespace, as in the qdt, cbc, and cac schema modules.

Figure 6. Schema Modules

Another way to visualize the structure is by example. Figure 6 depicts instances of the various schema modules from the previous diagram.

Figure 7 shows how the Order and Invoice document schemas import the CommonAggregateComponents and CommonBasicComponents external schema modules. It also shows how the Order document schema may include internal

schema modules — modules local to that namespace. The clear boxes show how the various schema modules are grouped into namespaces.

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

Figure 7. Order and Invoice Schema Import of Common Component Schema Modules

If two namespaces are mutually dependent, then importing one will cause the other to be imported as well. For this reason there *must not* exist circular dependencies between UBL schema modules. By extension, there *must not*exist circular dependencies between namespaces. A namespace A dependent upon type definitions or element declarations defined in another namespace B must import B's document schema.

> [*SSM2*] A schema in one UBL namespace that is dependent upon type definitions or element declarations in another schema namespace MUST only import that schema.

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 schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another schema namespace MUST NOT import the internal schema modules of that schema.

3.7.2. Internal and External Schema Modules

As illustrated in figures 5 and 6, UBL schema modules are either internal or external.

3.7.3. 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 are accessed using xsd:include.

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

UBL internal schema modules must have semantically meaningful names. Internal schema module names 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>

```
Example: ExtensionContentDatatype
```

3.7.4. External Schema Modules

External schema modules are used to group complex types and global elements that are used in multiple document schemas.

> [*SSM8*] UBL schema modules MAY be created for reusable components.

UBL external schema modules organize the reusable components into logical groupings. At a minimum, UBL defines the following external schema modules:

1. UBL CommonAggregateComponents

2. UBL CommonBasicComponents

3. UBL Qualified Datatypes

In addition, UBL 2.1 imports the following schema module provided by UN/CEFACT.

1. CCTS Core Component Types

> [*NMS19*] The CCTS Core Component Type schema module must be represented by the namespace prefix "ccts-cct".

Furthermore, where extensions are used, an extension schema module must be provided. This schema module must be named:

```
CommonExtensionComponents
```

> [*SSM21*] The UBL extension schema module MUST be identified as CommonExtensionComponents in the document name within the schema header.

> [*SSM22*] The UBL Qualified Datatypes schema module MUST import the UBL Unqualified Datatypes schema module.

To ensure consistency in expressing the CommonExtensionComponents schema module, a namespace prefix that will be used in all UBL schemas must be defined.

> [*NMS18*] The CommonExtensionComponents schema module namespace MUST be represented by the namespace prefix "ext" when referenced in other schemas.

3.7.4.1. UBL Common Aggregate Components Schema Module

The UBL library contains a wide variety of CCTS ABIEs, each defined as an xsd:complexType. Although some of these complex types may be used in only one UBL schema, many will be reused in multiple UBL schema modules. For ease of reuse, all the ABIE xsd:complexType definitions used in more than one UBL schema module are grouped into a single schema module of their own.

> [*SSM9*] A schema module defining all UBL Common Aggregate Components MUST be created.

> [*SSM10*] The UBL Common Aggregate Components schema module MUST be identified as CommonAggregateComponents in the document name within the schema header.

> [*NMS7*] The UBL Common Aggregate Components schema module MUST reside in its own namespace.

> [*NMS8*] The UBL Common Aggregate Components schema module namespace MUST be represented by the namespace prefix "cac" when referenced in other schemas.

3.7.4.2. UBL CommonBasicComponents Schema Module

The UBL library contains a wide variety of CCTS BBIEs based on CCTS BBIE Properties. BBIE Properties are reusable in multiple BBIEs, and each is defined as an xsd:complexType. Although some of these complex types may be used in only one UBL schema, many will be reused in multiple UBL schema modules. For ease of reuse, all the BBIE Property xsd:complexType definitions used in more than one UBL schema module are grouped into a single schema module of their own.

> [*SSM11*] A schema module defining all UBL Common Basic Components MUST be created.

> [*SSM12*] The UBL Common Basic Components schema module MUST be identified as Common-BasicComponents in the document name within the schema header.

> [*NMS9*] The UBL Common Basic Components schema module MUST reside in its own namespace.

> [*NMS10*] The UBL Common Basic Components schema module namespace MUST be represented by the namespace prefix "cbc" when referenced in other schemas.

3.7.4.3. CCTS CoreComponentType Schema Module

CCTS defines an authorized set of Core Component Types that convey content and supplementary information related to exchanged data. As the basis for all higher level CCTS models, these Core Component Types are reusable in every UBL schema. The complex type definitions for all CCTS Core Component Types are collected in the Core Component Type schema module published by UN/CEFACT.

3.7.4.4. UBL Unqualified Datatypes Schema Module

The UBL Unqualified Datatypes Schema Module imports the CCTS CoreComponentType Schema Module.

> [*NMS20*] The UBL Unqualified Datatypes schema module namespace MUST be represented by the prefix "udt" when referenced in other schemas.

3.7.4.5. UBL Qualified Datatypes Schema Module

UBL Qualified Datatypes are not expressed in the schema. Rather, data type qualifcations are expressed in the cva file.

> [*SSM18*] A schema module without any declarations must exist.

> [*SSM19*] The UBL Qualified Datatypes schema module MUST be identified as QualifiedDatatypes in the document name in the schema header.

> [*NMS15*] The UBL Qualified Datatypes schema module MUST reside in its own namespace.

To ensure consistency in expressing the UBL Qualified Datatypes schema module, a namespace prefix that will be used in all UBL schemas must be defined.

> [*NMS16*] The UBL Qualified Datatypes schema module namespace MUST be represented by the namespace prefix "qdt" when referenced in other schemas.

4. Annotation and Documentation Requirements

Annotation is an essential tool in understanding and reusing a schema. UBL, as an implementation of CCTS, requires an extensive amount of annotation to provide all necessary metadata required by the CCTS specification.

4.1. Schema Annotation

The annotation needed to satisfy CCTS requirements considerably increases the size of the UBL schemas, with undesirable performance impacts. To address this issue, a cut-down alternative has been developed for each UBL schema. A normative, fully annotated schema is provided to facilitate greater understanding of the schema module and its components and to meet the CCTS metadata requirements. A non-normative schema devoid of annotation is provided that can be used at run-time if required to meet processor resource constraints.

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

4.2. Embedded Documentation

UBL spreadsheets contain all necessary information to produce fully annotated schemas, including information about each UBL BBIE. UBL annotations consist of information currently required by Section 7 of the CCTS and supplemented by metadata from the UBL spreadsheet models.

The absence of an optional annotation from the structured set of annotations in a documentation element implies the use of the default value. For example, there are several annotations relating to context, such as CCTS Business Context and CCTS Industry Context; their absence implies that their value is "all contexts".

The following rules describe the documentation requirements for each UBL Qualified Datatype and UBL Unqualified Datatype definition. None of these documentation rules apply in the case of extension where the UBL Extensions element is used.

> [*DOC1*] The xsd:documentation element for every data type MUST contain a set of annotations in the following order (as defined in CCTS Section 7):
>
> - DictionaryEntryName (mandatory)
> - Version (mandatory)
> - Definition (mandatory)
> - RepresentationTerm (mandatory)
> - QualifierTerm(s) (mandatory, where used)
> - UniqueIdentifier (mandatory)
> - Usage Rule(s) (optional)
> - Content Component Restriction (optional)

> [*DOC2*] A datatype definition MAY contain one or more Content Component Restrictions to provide additional information on the relationship between the datatype and its corresponding Core Component

Type. If used, the Content Component Restrictions MUST contain a set of annotations in the following order:

- 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 datatype definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the datatype and its corresponding Core Component Type. If used, the Supplementary Component Restrictions MUST contain a set of annotations in the following order:

- SupplementaryComponentName (mandatory): Identifies the Supplementary Component to 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*] The xsd:documentation element for every BBIE MUST contain a set of annotations in the following order:

- ComponentType (mandatory): The type of component to which the object belongs. For BBIEs this MUST be "BBIE".
- DictionaryEntryName (mandatory): The official name of a BBIE.
- Version (optional): An indication of the evolution over time of the BBIE Entity.
- Definition (mandatory): The meaning of a BBIE.
- Cardinality (mandatory): Indicates whether the BBIE represents a not-applicable, optional, mandatory, or repetitive characteristic of the Aggregate Business Information Entity to which it belongs.
- ObjectClassQualifier (optional): The qualifier for the Object Class.
- ObjectClass (mandatory): The Object Class containing the BBIE.
- PropertyTermQualifier (optional): A word or words which help define and differentiate a BBIE.
- PropertyTerm (mandatory): Conveys the characteristic or Property of the Object Class.
- RepresentationTerm (mandatory): Describes the form in which the BBIE is represented.
- DataTypeQualifier (optional): A meaningful name that differentiates the data type of the BBIE from its underlying Core Component Type.
- DataType (mandatory): Defines the data type used for the BBIE.

- AlternativeBusinessTerms (optional): Any synonymous terms under which the BBIE is commonly known and used in the business.
- Examples (optional): Examples of possible values for the BBIE.

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

[*DOC5*] The xsd:documentation element for every ABIE MUST contain a set of annotations in the following order:

- ComponentType (mandatory): The type of component to which the object belongs. For ABIEs this MUST be "ABIE".
- DictionaryEntryName (mandatory): The official name of the ABIE .
- Version (optional): An indication of the evolution over time of the ABIE.
- Definition (mandatory): The meaning of the ABIE.
- ObjectClassQualifier (optional): The qualifier for the Object Class.
- ObjectClass (mandatory): The Object Class represented by the ABIE.
- AlternativeBusinessTerms (optional): Any synonymous terms under which the ABIE is commonly known and used in the business.

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

[*DOC6*] The xsd:documentation element for every ASBIE element declaration MUST contain a set of annotations in the following order:

- ComponentType (mandatory): The type of component to which the object belongs. For ASBIEs this MUST be "ASBIE".
- DictionaryEntryName (mandatory): The official name of the ASBIE.
- Version (optional): An indication of the evolution over time of the ASBIE.
- Definition (mandatory): The meaning of the ASBIE.
- Cardinality (mandatory): Indicates whether the ASBIE represents an optional, mandatory, or repetitive assocation.
- ObjectClass (mandatory): The Object Class containing the ASBIE.
- PropertyTermQualifier (optional): A word or words which help define and identify the ASBIE.
- PropertyTerm (mandatory): Represents the ASBIE contained by the Association Business Information Entity.
- AssociatedObjectClassQualifier (optional): The Associated Object Class Qualifiers describe the "context" of the relationship with another ABIE. That is, it is the role the contained ABIE plays within its association with the containing ABIE.

- AssociatedObjectClass (mandatory): The Object Class at the other end of the association. It represents the ABIE contained by the ASBIE.

[*DOC8*] The xsd:documentation element for every Supplementary Component attribute declaration MUST contain a set of annotations in the following order:

- Name (mandatory): Name in the Registry of a Supplementary Component of a Core Component Type.
- Definition (mandatory): An explanation of the meaning of a Supplementary Component and its relevance for the related Core Component Type.
- Primitive type (mandatory): The PrimitiveType to be used for the representation of the value of a Supplementary Component.
- Possible Value(s) (optional): Possible values of Supplementary Components.

[*DOC9*] The xsd:documentation element for every Supplementary Component attribute declaration containing restrictions MUST include the following additional information appended to the information required by DOC8:

- Restriction Value(s) (mandatory): The actual value(s) that is (are) valid for the Supplementary Component.

5. Naming Rules

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

1. 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.
2. Lower-level element: An element that appears inside a UBL business message. Lower-level elements consist of intermediate elements and leaf level elements.
3. Intermediate element: An element not at the top level that is of a complex type, containing only other elements and possibly attributes, but no mixed content.
4. 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.

5.1. General Naming Rules

In keeping with CCTS, UBL uses English as its normative language.

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

CCTS adheres to ISO/IEC 11179. The UBL component library is also fully conformant to those rules. The UBL XSD instantiation of the UBL component library in some cases refines 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. Following 11179, CCTS mandates three-part Dictionary Entry Names (DENs) for information items. As an implementation of CCTS, UBL assigns an official DEN to each item and then converts this to the name in UBL schemas using determinate transformation rules.

> [*GNR2*] UBL XML element and type names MUST be consistently derived from CCTS conformant Dictionary Entry Names.

DENs contain spaces and characters not allowed by XML and therefore not appropriate for UBL XML component names.

> [*GNR3*] UBL XML element and type names constructed from CCTS Dictionary Entry Names MUST NOT include periods, spaces, other separators, or characters not allowed by XSD.

Acronyms and abbreviations impair interoperability and therefore are to be avoided to the maximum extent practicable. Since some abbreviations will inevitably be necessary, UBL maintains a normative list of authorized acronyms and abbreviations. Creation and maintainance of this list belongs to content definition rather than Naming and Design, but for convenience, the list used for UBL 2.0 is provided in Appendix B.

> [*GNR4*] UBL XML element names and simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions maintained and published by the UBL TC.

The exception list is maintained and tightly controlled by UBL. Additions are made only when necessary. Once approved, an acronym or abbreviation must always be used to replace the term it stands for.

> [*GNR6*] The acronyms and abbreviations listed in the UBL-approved list MUST always be used in place of the word or phrase they represent.

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

> [*GNR7*] UBL XML element and type names MUST be in singular form unless the concept itself is plural.

Approved acronyms and abbreviations must be used consistently across documents.

> [*GNR10*] Acronyms and abbreviations at the beginning of an attribute name MUST appear in all lower case. Acronyms and abbreviations elsewhere in an attribute name MUST appear in upper case.

> [*GNR11*] Acronyms and abbreviations MUST appear in all upper case for all element and type names.

XML is case sensitive. Consistency in the use of case for a specific XML component (element, type, attribute) is essential to ensure that every occurrence of a component is treated as the same. Capitalization helps readability and consistency. The ebXML architecture document specifies a standard use of upper and lower camel case for expressing XML elements and attributes, respectively. Following this practice, UBL element and type names use UpperCamelCase (UCC), and attribute names use lowerCamelCase (LCC).

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

Example 2.

CurrencyBaseRate

CityNameType

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

Example 3.

currencyID

unitCode

5.2. Type Naming Rules

UBL specifies naming rules for complex types based on CCTS ABIEs, BBIEs, and BBIE Properties. The use of unique CCTS Dictionary Entry Names for these constructs disambiguates their meanings and prevents duplication.

5.2.1. Complex Type Names for CCTS Aggregate Business Information Entities (ABIEs)

UBL xsd:complexType names for ABIEs are derived from their DENs by removing separators to follow general naming rules and appending the suffix "Type" to replace the word "Details".

> [*CTN1*] A UBL xsd:complexType name based on a CCTS ABIE MUST be the CCTS Dictionary Entry Name with the separators removed and with the "Details" suffix replaced with "Type".

Example 4.

CCTS Aggregate Business Information Entity	UBL xsd:complexType
Address. Details	AddressType
Financial Account. Details	FinancialAccountType

5.2.2. Complex Type Names for CCTS Basic Business Information Entity (BBIE) Properties

All BBIE Properties are reusable across multiple BBIEs. The CCTS does not specify, but implies, that BBIE Property names are the reusable property term and representation term of the family of BBIEs that are based on them. The UBL xsd:complexType names for BBIE Properties are derived from the shared Property and Representation terms portion of the DENs in which they appear by removing separators to follow general naming rules and appending the suffix "Type".

> [*CTN2*] A UBL xsd:complexType name based on a CCTS BBIE Property MUST be the CCTS Dictionary Entry Name shared Property Term and its qualifiers and the Representation Term of the BBIE with the separators removed and with the "Type" suffix appended after the Representation Term.

Example 5.

CCTS Business Information Entity Property	UBL xsd:complexType
Declared Customs_ Value. Amount	DeclaredCustomsValueAmountType
Gross_ Weight. Measure	GrossWeightMeasureType

[*CTN6*] A UBL xsd:complexType name based on a CCTS BBIE Property and with a CCTS BBIE Representation Term of "Text" MUST have the word "Text" removed from the end of its name.

Example 6.

CCTS Basic Business Information Entity	UBL xsd:complexType
Agency Name. Text	AgencyNameType
Floor. Text	FloorType

[*CTN7*] A UBL xsd:complexType name based on a CCTS BBIE Property and with a CCTS BBIE Representation Term of "Identifier" MUST replace "Identifier" with "ID" at the end of its name.

Example 7.

CCTS Basic Business Information Entity	UBL xsd:complexType
Agency Identifier. Identifier	AgencyIDType
Vessel Identifier. Identifier	VesselIDType

[*CTN8*] A UBL xsd:complexType name based on a CCTS BBIE Property MUST remove all duplication of words that occurs as a result of duplicate Property Terms and Representation Terms.

Example 8.

CCTS Basic Business Information Entity	UBL xsd:complexType
Issue Date. Date	IssueDateType
Issue Time. Time	IssueTimeType

5.3. Element Naming Rules

As shown in Figure 3, UBL elements are created for each UBL ABIE, BBIE, and ASBIE.

5.3.1. Element Names for CCTS ABIEs (ABIEs)

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

For example, a UBL xsd:complexType name based on the ABIE Party. Details will be PartyType. The global element based on PartyType will be named Party.

Example 9.

```
<xsd:element name="Party" type="PartyType"/>
  <xsd:complexType name="PartyType">
    <xsd:annotation>
      <!-- Documentation goes here -->
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element ref="cbc:MarkCareIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="cbc:MarkAttentionIndicator" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="PartyIdentification" minOccurs="0" maxOccurs="unbounded">
          ...
      </xsd:element>
      <xsd:element ref="PartyName" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      <xsd:element ref="Address" minOccurs="0" maxOccurs="1">
          ...
      </xsd:element>
      ...
    </xsd:sequence>
```

5.3.2. Element Names for CCTS BBIE Properties

The same naming concept used for ABIEs applies to BBIE Properties.

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

Example 10.

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

5.3.3. Element Names for CCTS ASBIEs

An ASBIE is not a class like an ABIE or a BBIE Property that is reused as a BBIE. Rather, it is an association between two classes. Therefore, an element representing an ASBIE does not have its own unique xsd:complexType. Instead, when an element representing an ASBIE is declared, the element is bound to the xsd:complexType of its associated ABIE by referencing the ABIE's global element declaration.

> [*ELN3*] A UBL global element name based on a CCTS ASBIE MUST be the CCTS ASBIE Dictionary Entry Name Property Term and its qualifiers and the Object Class Term and qualifiers of its associated CCTS ABIE. All CCTS Dictionary Entry Name separators MUST be removed.

Example 11.

CCTS ASBIE Property Term	Associated ABIE Object Class	Global Element Name
Buyer_Contact	Contact.Details	BuyerContact
Origin_Address	Address.Details	OriginAddress

5.4. Attributes in UBL

As a transaction-based XML exchange format, UBL significantly restricts the use of XML attributes. Attribute usage is relegated to supplementary components only; all "primary" business data appears exclusively in element content. Attributes are defined CCTS CCT schema module.

6. Declarations and Definitions

In XSD, 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 types and their documentation in the UBL Library.

6.1. Type Definitions

6.1.1. 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 12.

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

UBL disallows the use of the type xsd:anyType, because this feature permits the introduction of potentially unknown types into an XML instance.

> [*GTD2*] The predefined XML schema type xsd:anyType MUST NOT be used.

6.1.2. BBIE Data Types

CCTS provides a set of constructs called Core Component Types (CCTs) for the modeling of basic data. These are represented in UBL with a library of complex types. BBIE content is represented as property sets defined according to the CCTs, made up of content components and supplementary components. The supplementary components are expressed as XML attributes, the content component becomes element content, and the CCT is represented with an xsd:complexType.

UBL defines BBIE content types in the UBL Unqualified Datatypes Schema Module, which imports the CCT schema module. Since even BBIE data types are modeled as property sets, the XML expression of these models primarily employs xsd:complexType.

6.1.3. ABIE Data Types

In the UBL model, ABIEs are considered classes (objects). To facilitate reuse, versioning, and customization, all complex types are named.

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

Example 13.

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

Every class identified in the UBL model consists of properties. These properties are either ASBIEs, when the property represents another class, or BBIEs.

> [*CTD25*] For every CCTS BBIE Property identified in the UBL model, a named xsd:complexType MUST be defined.

6.1.3.1. Aggregate Business Information Entities (ABIEs)

An ABIE encapsulates the relationship between a class (the ABIE) and its properties (those data items contained within the ABIE). UBL represents this relationship by defining an xsd:complexType for each ABIE with its properties represented as a sequence of references to global elements.

> [*CTD2*] Every CCTS ABIE xsd:complexType definition content model MUST contain an xsd:sequence element containing the appropriate global element declarations.

Example 14.

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

6.1.3.2. Basic Business Information Entities (BBIEs)

In accordance with CCTS, all BBIEs have a primary or secondary Representation Term. Representation Terms are expressed in the UBL Model as Unqualified Datatypes bound to a Core Component Type that describes their structure. The following set of rules specifies the way these relationships are expressed in the UBL XML library. As discussed above, BBIE Properties are represented with complex types. Within these are xsd:simpleContent elements that extend the Datatypes.

> [*CTD3*] Every CCTS BBIE Property xsd:complexType definition content model MUST contain an xsd:simpleContent element.

> [*CTD4*] Every CCTS BBIE Property xsd:complexType content model xsd:simpleContent element MUST consist of an xsd:extension element.

[*CTD26*] Every CCTS BBIE Property xsd:complexType xsd:base attribute value MUST be the UBL Unqualified Datatype.

Example 15.

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

[*CTD27*] Every BBIE property with the representation term Code MUST be based on the UBL unqualified code data type.

6.1.3.3. Datatypes

UBL Unqualified Datatypes and UBL Qualified Datatypes are identified in the UBL Unqualified Datatype Schema Module and in the UBL Qualified Datatype Schema Module, though it must be noted that the UBL Qualified Datatype Schema Module does not contain any datatype declarations.

6.1.3.4. Core Component Types

UBL uses UN/CEFACT's Core Component Type schema module.

6.2. Element Declarations

6.2.1. 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 BBIEs and ABIEs, the UBL elements are directly associated to their corresponding xsd:complexTypes.

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

Example 16.

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 can be reused in the building of new business messages. The complex type thus created can be used through the declaration of new elements of that type in the building of both new and contextualized business messages.

Example 17.

```
<xsd:element name="SupplierParty" type="SupplierPartyType"/>
 <xsd:complexType name="SupplierPartyType"/>
    ...
</xsd:complexType>
```

6.2.2. Elements Representing ASBIEs

An ASBIE is not a class like an ABIE. Rather, it is an association between two classes, and therefore the element declaration binds the element to the xsd:complexType of the associated ABIE. There are two types of ASBIEs — those that have qualifiers in the object class, and those that do not.

> [*ELD4*] When a CCTS ASBIE is unqualified, it is bound via reference to the global CCTS ABIE element with which it is associated.

> [*ELD11*] When a CCTS ASBIE is qualified, a new element MUST be declared and bound to the xsd:complexType of its associated CCTS ABIE.

6.3. Empty Elements

> [*ELD7*] Empty elements MUST not be declared, except in the case of extension where the UBL Extensions element is used.

7. Code Lists

UBL uses the Code List Methodology proposed by G. Ken Holman. See the OASIS Code List Representation Technical Committee specifications Code List Representation (Genericode) Version 1.0 and Context/value association using genericode 1.0..

In addition to the methodology, the following rules apply.

> [*CDL1*] All UBL codes MUST be part of a UBL or externally maintained code list.

The majority of code lists are owned and maintained by external agencies. UBL makes 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, UBL may extend an existing code list to meet specific business requirements. In others cases, UBL may create and maintain a code list where a suitable code list does not exist in the public domain. Both of these types of code lists would be considered UBL-internal code lists.

> [*CDL3*] The UBL Library MAY design and use an internally maintained code list where an existing externally maintained code list needs to be extended, or where no suitable externally maintained code list exists.

8. Miscellaneous XSD Rules

As a business standard vocabulary, UBL 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.

8.1. xsd:simpleType

XSD provides for 44 built-in data types expressed as simple types. For maximum reuse, these built-in simple types should be used wherever possible.

[*GXS3*] Built-in xsd:simpleTypes SHOULD be used wherever possible.

8.2. Namespace Declaration

XSD allows any prefixes to be used in referencing its namespaces. To ensure consistency, UBL has adopted the generally accepted convention of using the "xsd" prefix for the XSD namespace.

[*GXS4*] All XSD 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"
```

8.3. xsd:substitutionGroup

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

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

8.4. xsd:final

UBL does not use extensions in its normative schemas. Extensions are allowed by customizers as outlined in the Guidelines for Customization. In cases where type definitions are inappropriate for any customization, the xsd:final attribute is used.

[*GXS6*] The xsd:final attribute MUST be used to control extensions where there is a desire to prohibit further extensions.

8.5. xsd: notation

The UBL schema model does not require or support the use of xsd:notation.

[*GXS7*] xsd:notation MUST NOT be used.

8.6. xsd:all

When xsd:all is used, elements can occur in any order, are always optional, and can never occur more than once. Such restrictions are inconsistent with the applications of UBL.

[*GXS8*] xsd:all MUST NOT be used.

8.7. xsd:choice

xsd:choice allows one of a set of alternatives to appear in a document instance. This is useful in some contexts but xsd:choice cannot be extended and therefore is not recommended.

[*GXS9*] The xsd:choice element SHOULD NOT be used where customization and extensibility are a concern.

8.8. xsd:include

xsd:include may be used in accordance with rule GXS10.

> [*GXS10*] xsd:include can only be used when the including schema is in the same namespace as the included schema.

8.9. xsd:union

The xsd:union feature provides a mechanism whereby a datatype is created as a union of two or more existing datatypes. As UBL strictly adheres to the use of CCTS Datatypes that are explicitly declared in the UBL library, this feature is inappropriate except for code lists.

> [*GXS11*] The xsd:union technique MUST NOT be used except for code lists.

8.10. xsd:appinfo

The xsd:appinfo feature is used by schemas to convey processing instructions to a processing application, stylesheet, or other tool. Some users of UBL believe that this technique poses a security risk and have employed techniques for stripping xsd:appinfo from schemas. As UBL is committed to ensuring the widest possible target audience for its XML library, this feature is used only to convey information.

> [*GXS12*] UBL schemas SHOULD NOT use xsd:appinfo. If used, xsd:appinfo MUST be used only to convey non-normative information.

8.11. xsd:schemaLocation

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

> [*GXS15*] Each xsd:schemaLocation attribute declaration MUST contain a system-resolvable URL, which at the time of release from OASIS shall be a relative URL referencing the location of the schema or schema module in the release package.

8.12. xsd:nillable

> [*GXS16*] The built in xsd:nillable attribute MUST NOT be used for any UBL declared element.

8.13. xsd:any

UBL disallows the use of xsd:any because this feature permits the introduction of unknown attributes into an XML instance. UBL intends that all constructs within an instance be governed by the schemas describing that instance, and therefore xsd:any is not allowed outside of the ExtensionContentType definition.

> [*GXS14*] xsd:any MUST NOT be used except within the ExtensionContentType type definition, and with xsd:processContents= "skip" for non-UBL namespaces.

8.14. Extension and Restriction

UBL recognizes the value of supporting extension and restriction of its core schema library by customizers. The UBL schema extension and restriction recommendations are discussed in the *Guidelines for the Customization of UBL 1.0 Schemas* (SchCust) available as part of the UBL 1.0 Standard.

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

A. Code List Metadata (Informative)

Included here for convenience are some observations regarding instance-level code list metadata defined in UBL 2.0 schemas for the information items governed by code lists. Note that what follows are not UBL Naming and Design Rules but rather implications of UBL's use of the UN/CEFACT Unqualified Data Type Schema Module.

For items based on the unqualified data type Amount, the attribute currencyID has the coded value, and the instance-level metadata is one attribute:

currencyCodeListVersionID

For items based on the unqualified data type MeasureType, the attribute unitCode has the coded value, and the instance-level metadata is one attribute:

unitCodeListVersionID

For items based on the unqualified data type QuantityType, the attribute unitCode has the coded value, and the instance-level metadata consists of three attributes:

unitCodeListID

unitCodeListAgencyID

unitCodeListAgencyName

For an element named <xxxxxCode> based on the unqualified data type CodeType, the element has the coded value, and the instance-level metadata consists of seven attributes:

listName

listID

listVersionID

listSchemeURI

listURI

listAgencyName

listAgencyID

For an element named <yyyyyID> based on the unqualified data type IdentifierType, the element has the coded value, and the instance-level metadata consists of six attributes:

schemeName

```
schemeVersionID

schemeURI

schemeDataURI

schemeAgencyName

schemeAgencyID
```

All instance-level code list metadata attributes are optional and can be specified separately for each coded value used; there are no global document-wide properties representing these attributes.

Any combination of allowable metadata attributes can be specified by the author of the UBL instance to identify the semantics associated with the coded value in the information item. Absent any of these attributes, an implementation must make its own judgements about the implied semantics of the code based on the information available.

In some cases, an incomplete set of metadata attributes may be enough to uniquely identify an associated code list. For example, a listSchemeURI or schemeURI value is probably sufficient to uniquely identify, respectively, a code or identifier. A combination of listName or listID with listVersionID for a code, or schemeName and schemeVersionID for an identifier, would probably also be sufficient.

In the extreme case, all code list information associated with a coded value may be missing; for example:

```
<cbc:DocumentCurrencyCode>USD</cbc:DocumentCurrencyCode>
```

There is no harm in omitting code list identification for this code value if the application can safely assume that a value of "USD" for DocumentCurrencyCode means U.S. Dollar, which is usually a safe assumption if the instance comes from a known trading partner.

Omission of code list metadata can be useful when it is desired to leave the exact version unspecified, as for example when making updates to a particular code list within a particular trading community. Omitting the metadata attributes associating instance data with a particular release of a code list makes it unnecessary to change instance generation at the moment the update is deployed. This assumes, of course, that such changes are being managed out-of-band by protocols within the community.

Identifying metadata should be included in the instance if the sender thinks the receiver might misinterpret the code. And if an information item allows the union of two lists, and there happens to be an overlap between the two lists such that one or more codes appear on both lists, then identifying metadata must be used to unambiguously specify which code is intended.

B. UBL-approved Acronyms and Abbreviations (Informative)

The information included in this appendix is historical and has been included for informational purposes only.

Table B.1. Abbreviation and Acronym Table for UBL 2.0

Credit Card Verification Numbering System	CV2
Identifier	ID
Uniform Resource Identifier	URI
United Nations Dangerous Goods	UNDG
Universal Business Language	UBL
Universally Unique Identifier	UUID

C. Technical Terminology (Informative)

Aggregate Business Information Entity (ABIE)	A collection of related pieces of business information that together convey a distinct business meaning in a specific Business Context. Expressed in modelling terms, it is the representation of an Object Class, in a specific Business Context.
Application-level validation	Adherence to business requirements, such as valid account numbers.
Assembly	Using parts of the library of reusable UBL components to create a new kind of business document type.
Business Context	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 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 reflect how business concepts are represented in software. The term "system business objects" is used to designate this usage.
Business semantic(s)	The precise meaning of words from a business perspective.
Business Term	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 be known by 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.
Class diagram	(OMG Distilled) 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. (Rational Unified Process) A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.
Classification scheme	Officially supported scheme to describe a given Context Category.
Document schema	A schema document corresponding to a single namespace, which is likely to include or import schema modules.
Core Component	A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.
Core Component Type	A Core Component which consists of one and only one Content Component that carries the actual content plus one or more Supplementary Components giving an essential extra definition to the Content Component. Core Component Types do not have business semantics.
Data type	(XSD) A descriptor of a set of values that lack identity and whose operations do not have side effects. XSD data types include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string, and time. User-definable types include enumerations. (CCTS) 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.
Instance	An individual entity satisfying the description of a class or type. In XML, an individual document of a certain type (a specific purchase order, invoice, etc.).
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 described using Schematron, for example.
Intermediate element	An element not at the top level that is of a complex type, only containing other elements and attributes.

Internal schema module	A schema module that does not declare a target namespace.
Leaf element	An element containing only character data (though it may also have attributes). Note that, because of the XSD mechanisms involved, a leaf element that has attributes must be declared as having a complex type, but a leaf element with no attributes may be declared with either a simple type or a complex type.
Lower-level element	An element that appears inside a business message. Lower-level elements consist of intermediate and leaf level.
Object Class	The logical data grouping (in a logical data model) to which a data element belongs (ISO11179). The 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 include or import 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.
(XML) Schema	An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items (as defined in [XSD]), and furthermore may specify augmentations to those items and their descendants.
Schema module	A schema that can be included or imported by other schemas.
Schema processing	Schema validation checking plus provision of default values and provision of new infoset properties.
Schema validation	The process of programmatically checking a document instance for 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.

D. References

[CCTS] ISO 15000-5 ebXML Core Components Technical Specification.

[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.

[SchCust] Guidelines for the Customization of UBL v1.0 Schemas, http://docs.oasis-open.org/ubl/cd-UBL-1.0/doc/cm/wd-ubl-cmsc-cmguidelines-1.0.html, an informative annex to the UBL 1.0 Standard.

[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.

E. Notices

F. UBL NDR Checklist

UBL NDR 2.0 Checklist The following checklist constitutes all UBL XML naming and design rules as defined in UBL Naming and Design Rules version 2.0, 19 July 2006. The checklist is in alphabetical sequence as follows:

- Attribute Declaration Rules (ATD)
- Code List Rules (CDL)
- ComplexType Definition Rules (CTD)

- ComplexType Naming Rules (CTN)
- Documentation Rules (DOC)
- Element Declaration Rules (ELD)
- Element Naming Rules (ELN)
- General Naming Rules (GNR)
- General Type Definition Rules (GTD)
- General XML Schema Rules (GXS)
- Instance Document Rules (IND)
- Modeling Constraints Rules (MDC)
- Naming Constraints Rules (NMC)
- Namespace Rules (NMS)
- Root Element Declaration Rules (RED)
- Schema Structure Modularity Rules (SSM)
- Standards Adherence Rules (STA)
- Versioning Rules (VER

Code List Rules	
CDL1	All UBL codes MUST be part of a UBL or externally maintained code list.
CDL2	The UBL Library SHOULD identify and use external standardized code lists rather than develop its own UBL-native code lists.
CDL3	The UBL Library MAY design and use an internally maintained code list where an existing externally maintained code list needs to be extended, or where no suitable externally maintained code list exists.

ComplexType Definition rules	
CTD1	For every class identified in the UBL model, a named xsd:complexType MUST be defined.
CTD2	Every CCTS ABIE xsd:complexType definition content model MUST contain an xsd:sequence element containing the appropriate global element declarations.
CTD3	Every CCTS BBIE Property xsd:complexType definition content model MUST contain an xsd:simple-Content element.
CTD4	Every CCTS BBIE Property xsd:complexType content model xsd:simpleContent element MUST consist of an xsd:extension element.
CTD25	For every CCTS BBIE Property identified in the UBL model, a named xsd:complexType MUST be defined.
CTD26	Every CCTS BBIE Property xsd:complexType xsd:base attribute value MUST be the UBL Unqualified Datatype.
CTD27	Every BBIE property with the representation term Code MUST be based on the UBL unqualified code data type.

Complex Type Naming rules	
CTN1	A UBL xsd:complexType name based on a CCTS ABIE MUST be the CCTS Dictionary Entry Name with the separators removed and with the "Details" suffix replaced with "Type".
CTN2	A UBL xsd:complexType name based on a CCTS BBIE Property MUST be the CCTS Dictionary Entry Name shared Property Term and its qualifiers and the Representation Term of the BBIE with the separators removed and with the "Type" suffix appended after the Representation Term.
CTN6	A UBL xsd:complexType name based on a CCTS BBIE Property and with a CCTS BBIE Representation Term of "Text" MUST have the word "Text" removed from the end of its name.
CTN7	A UBL xsd:complexType name based on a CCTS BBIE Property and with a CCTS BBIE Representation Term of "Identifier" MUST replace "Identifier" with "ID" at the end of its name.
CTN8	A UBL xsd:complexType name based on a CCTS BBIE Property MUST remove all duplication of words that occurs as a result of duplicate Property Terms and Representation Terms.

Documentation rules	
DOC1	The xsd:documentation element for every data type MUST contain a set of annotations in the following order (as defined in CCTS Section 7):
DOC2	A datatype definition MAY contain one or more Content Component Restrictions to provide additional information on the relationship between the datatype and its corresponding Core Component Type. If used, the Content Component Restrictions MUST contain a set of annotations in the following order:
DOC3	A datatype definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the datatype and its corresponding Core Component Type. If used, the Supplementary Component Restrictions MUST contain a set of annotations in the following order:
DOC4	The xsd:documentation element for every BBIE MUST contain a set of annotations in the following order:
DOC5	The xsd:documentation element for every ABIE MUST contain a set of annotations in the following order:
DOC6	The xsd:documentation element for every ASBIE element declaration MUST contain a set of annotations in the following order:
DOC8	The xsd:documentation element for every Supplementary Component attribute declaration MUST contain a set of annotations in the following order:
DOC9	The xsd:documentation element for every Supplementary Component attribute declaration containing restrictions MUST include the following additional information appended to the information required by DOC8:

Element Declaration rules	
ELD2	All element declarations MUST be global.
ELD3	For every class and property identified in the UBL model, a global element bound to the corresponding xsd:complexType MUST be declared.
ELD4	When a CCTS ASBIE is unqualified, it is bound via reference to the global CCTS ABIE element with which it is associated.
ELD7	Empty elements MUST not be declared, except in the case of extension where the UBL Extensions element is used.
ELD11	When a CCTS ASBIE is qualified, a new element MUST be declared and bound to the xsd:complexType of its associated CCTS ABIE.

Element Declaration rules	
ELD12	The UBL Extensions element MUST be declared as the first child of the document element with xsd:minOccurs="0".
ELD13	The UBLProfileID element MUST be declared immediately following the UBL Extensions element with xsd:minOccurs="0".
ELD14	The UBLSubsetID element MUST be declared immediately following the UBLProfileID element with xsd:minOccurs="0".

Element Naming rules	
ELN1	A UBL global element name based on a CCTS ABIE MUST be the same as the name of the corresponding xsd:complexType to which it is bound, with the word "Type" removed.
ELN2	A UBL global element name based on a CCTS BBIE Property MUST be the same as the name of the corresponding xsd:complexType to which it is bound, with the word "Type" removed.
ELN3	A UBL global element name based on a CCTS ASBIE MUST be the CCTS ASBIE Dictionary Entry Name Property Term and its qualifiers and the Object Class Term and qualifiers of its associated CCTS ABIE. All CCTS Dictionary Entry Name separators MUST be removed.

General Naming rules	
GNR1	UBL XML element and type names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
GNR2	UBL XML element and type names MUST be consistently derived from CCTS conformant Dictionary Entry Names.
GNR3	UBL XML element and type names constructed from CCTS Dictionary Entry Names MUST NOT include periods, spaces, other separators, or characters not allowed by XSD.
GNR4	UBL XML element names and simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions maintained and published by the UBL TC.
GNR6	The acronyms and abbreviations listed in the UBL-approved list MUST always be used in place of the word or phrase they represent.
GNR7	UBL XML element and type names MUST be in singular form unless the concept itself is plural.
GNR8	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.
GNR9	The lowerCamelCase (LCC) convention MUST be used for naming attributes.
GNR10	Acronyms and abbreviations at the beginning of an attribute name MUST appear in all lower case. Acronyms and abbreviations elsewhere in an attribute name MUST appear in upper case.
GNR11	Acronyms and abbreviations MUST appear in all upper case for all element and type names.

General Type Definition Rules	
GTD1	All types MUST be named.
GTD2	The predefined XML schema type xsd:anyType MUST NOT be used.

General XML Schema Rules	
GXS1	Except in the case of extension, where the "UBL Extensions" element is used, UBL schemas SHOULD conform to the following physical layout as applicable: See .

General XML Schema Rules	
GXS2	UBL MUST provide two schemas for each transaction. One normative schema shall be fully annotated. One non-normative schema shall be a run-time schema devoid of documentation.
GXS3	Built-in xsd:simpleTypes SHOULD be used wherever possible.
GXS4	All XSD constructs in UBL schema and schema modules MUST contain the following namespace declaration on the xsd:schema element:
GXS5	The xsd:substitutionGroup feature MUST NOT be used.
GXS6	The xsd:final attribute MUST be used to control extensions where there is a desire to prohibit further extensions.
GXS7	xsd:notation MUST NOT be used.
GXS8	xsd:all MUST NOT be used.
GXS9	The xsd:choice element SHOULD NOT be used where customization and extensibility are a concern.
GXS10	xsd:include can only be used when the including schema is in the same namespace as the included schema.
GXS11	The xsd:union technique MUST NOT be used except for code lists.
GXS12	UBL schemas SHOULD NOT use xsd:appinfo. If used, xsd:appinfo MUST be used only to convey non-normative information.
GXS15	Each xsd:schemaLocation attribute declaration MUST contain a system-resolvable URL, which at the time of release from OASIS shall be a relative URL referencing the location of the schema or schema module in the release package.
GXS16	The built in xsd:nillable attribute MUST NOT be used for any UBL declared element.
GXS14	xsd:any MUST NOT be used except within the ExtensionContentType type definition, and with xsd:processContents= "skip" for non-UBL namespaces.
GXS13	Complex type extension or restriction MAY be used where appropriate.

Modelling constraint rules	
MDC0	The sequence of the business information entities that is expressed in the UBL model MUST be preserved in the schema.
MDC1	UBL libraries and schemas MUST only use CCTS Core Component Types, except in the case of extension, where the UBLExtensions element is used.
MDC2	XML mixed content MUST NOT be used except where contained in an xsd:documentation element.

Naming constraint rules	
NMC1	Each Dictionary Entry Name MUST define one and only one fully qualified path (FQP) for an element or attribute.

Namespace Rules	
NMS1	Every UBL-defined or -used schema module, except internal schema modules, MUST declare a namespace using the xsd:targetNamespace attribute.
NMS2	Every UBL-defined or -used major version schema set 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
NMS5	The namespace names for UBL schemas holding OASIS Standard status MUST be of the form

Namespace Rules	
NMS6	UBL published namespaces MUST never be changed.
NMS7	The UBL Common Aggregate Components schema module MUST reside in its own namespace.
NMS8	The UBL Common Aggregate Components schema module namespace MUST be represented by the namespace prefix "cac" when referenced in other schemas.
NMS9	The UBL Common Basic Components schema module MUST reside in its own namespace.
NMS10	The UBL Common Basic Components schema module namespace MUST be represented by the namespace prefix "cbc" when referenced in other schemas.
NMS15	The UBL Qualified Datatypes schema module MUST reside in its own namespace.
NMS16	The UBL Qualified Datatypes schema module namespace MUST be represented by the namespace prefix "qdt" when referenced in other schemas.
NMS18	The CommonExtensionComponents schema module namespace MUST be represented by the namespace prefix "ext" when referenced in other schemas.
NMS19	The CCTS Core Component Type schema module must be represented by the namespace prefix "cctscct".
NMS20	The UBL Unqualified Datatypes schema module namespace MUST be represented by the prefix "udt" when referenced in other schemas.

Root element declaration rules	
RED2	The root element MUST be the only global element declared in the document schema.

Schema structure modularity rules	
SSM1	UBL schema expressions MAY be split into multiple schema modules.
SSM2	A schema in one UBL namespace that is dependent upon type definitions or element declarations in another schema namespace MUST only import that schema.
SSM3	A schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another schema namespace MUST NOT import the internal schema modules of that schema.
SSM6	All UBL internal schema modules MUST be in the same namespace as their corresponding document schema.
SSM7	Each UBL internal schema module MUST be named <ParentSchemaModuleName><InternalSchemaModuleFunction>
SSM8	UBL schema modules MAY be created for reusable components.
SSM9	A schema module defining all UBL Common Aggregate Components MUST be created.
SSM10	The UBL Common Aggregate Components schema module MUST be identified as CommonAggregateComponents in the document name within the schema header.
SSM11	A schema module defining all UBL Common Basic Components MUST be created.
SSM12	The UBL Common Basic Components schema module MUST be identified as CommonBasicComponents in the document name within the schema header.
SSM18	A schema module without any declarations must exist.
SSM19	The UBL Qualified Datatypes schema module MUST be identified as QualifiedDatatypes in the document name in the schema header.
SSM21	The UBL extension schema module MUST be identified as CommonExtensionComponents in the document name within the schema header.

Schema structure modularity rules	
SSM22	The UBL Qualified Datatypes schema module MUST import the UBL Unqualified Datatypes schema module.

Versioning rules	
VER2	Every UBL schema module major version MUST have an RFC 3121 document-id of the form
VER4	Every minor version release of a UBL schema module MUST have a document-id of the form
VER5	For UBL minor version changes, the namespace name MUST not change.
VER6	Every UBL schema module major version number MUST be a sequentially assigned integer greater than zero.
VER7	Every UBL schema module minor version number MUST be a sequentially assigned, non-negative integer.
VER12	Every major version release of a UBL schema module MUST capture its version number in the xsd:version attribute of the xsd:schema element in the form
VER14	Every minor version release of a UBL schema module MUST capture its version information in the xsd:version attribute in the form
VER15	Every UBL document schema MUST declare an optional element named UBLVersionID immediately following the optional UBL Extensions element.