Universal Business Language (UBL) Naming and Design Rules

Working Draft 12, 31 May 2002

Document identifier:

wd-ublndrsc-ndrdoc-12 (Word, PDF)

Location:

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

Editors:

Bill Burcham, Sterling Commerce <Bill_Burcham@stercomm.com>
Mavis Cournane, Cognitran Ltd <mavis.cournane@cognitran.com> (primary editor)
Mark Crawford, LMI <MCRAWFORD@lmi.org>
Arofan Gregory, CommerceOne <arofan.gregory@commerceone.com>
Eve Maler, Sun Microsystems <eve.maler@sun.com>

Contributors:

Fabrice Desré, France Telecom
Matt Gertner, Schemantix
Jessica Glace, LMI
Phil Griffin, Griffin Consulting
Eduardo Gutentag, Sun Microsystems
Sue Probert, CommerceOne
Gunther Stuhec, SAP
Paul Thorpe, OSS Nokalva

Abstract:

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

Status:

This is a draft document and is likely to change on a weekly basis.

If you are on the ubl-ndrsc@lists.oasis-open.org list for NDR subcommittee members, send comments there. If you are not on that list, subscribe to the ubl-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to ubl-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

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 Security Services TC web page (http://www.oasis-open.org/committees/security/).

Table of Contents

1 Introduction

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

The W3C XML Schema form of the UBL library is currently constructed automatically from the metamodel developed by the OASIS UBL Library Content Subcommittee (LC SC). Thus, most of the rules in this document are used to guide the development of the engine that generates the XSD schema modules; this engine is produced by the OASIS UBL Tools and Techniques Subcommittee (TT SC). Some of the rules address XML instance constructs and other practices that must be undertaken by humans, such as developers who are customizing UBL for their own purposes.

1.1 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 **[RFC2119]**.

The terms “W3C XML Schema” and “XSD” are used throughout this document. They are considered synonymous; both refer to XML Schemas that conform to the W3C Schema Recommendations **[XSD]**. See Section 8 for additional term definitions.

1.2 Guiding Principles

TBS (see draft-ublndrsc-designrules-04 for a draft of this information; will be placed here eventually)

2 The UBL Metamodel

UBL is based **[UBLChart]** on the ebXML Core Components Technical Specification **[CCTS]**, which defines a metamodel and the following concepts, most of which were derived from **[ISONaming]**:

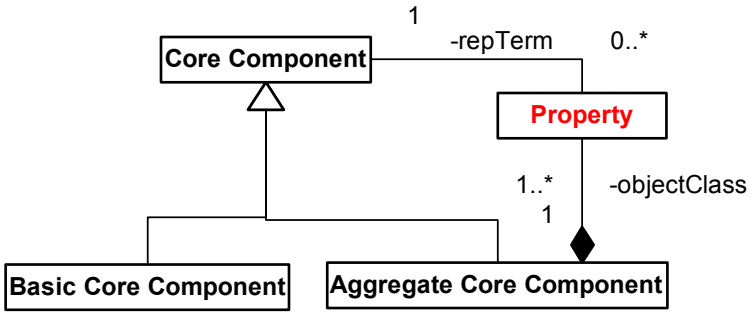
- Object Class
- Property Term
- Qualifier
- Representation Term (RT)
- Core Component Type (CCT)

The UBL metamodel is based on the Core Components metamodel. In certain instances, however, the Core Components metamodel was found to be ambiguous, unwieldy, or insufficient for UBL purposes. Thus, we have provided feedback **[CCFeedback]** on the CCTS.

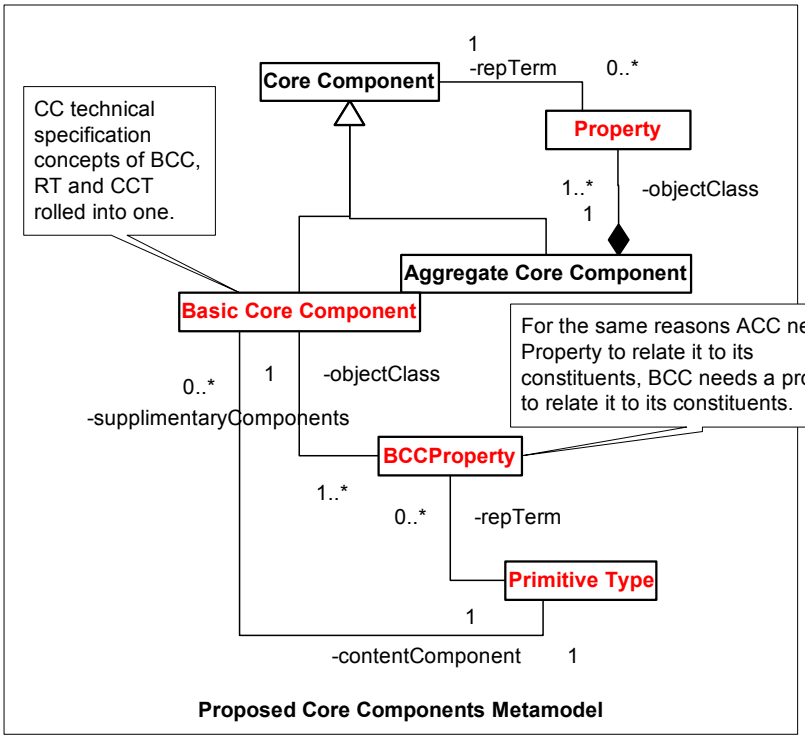
In this section we describe the UBL metamodel in terms of our proposed revisions. To simplify the reading of this section, we will refer only to Core Components (CCs) when in reality these comments apply to both CCs and Basic Information Entities (BIEs). The issue of context (the distinguishing factor between CCs and BIEs) is not covered in this section.

Following is a summary of our proposals:

- Add an explicit *Property* concept, where a Property Term is a name for a Property:



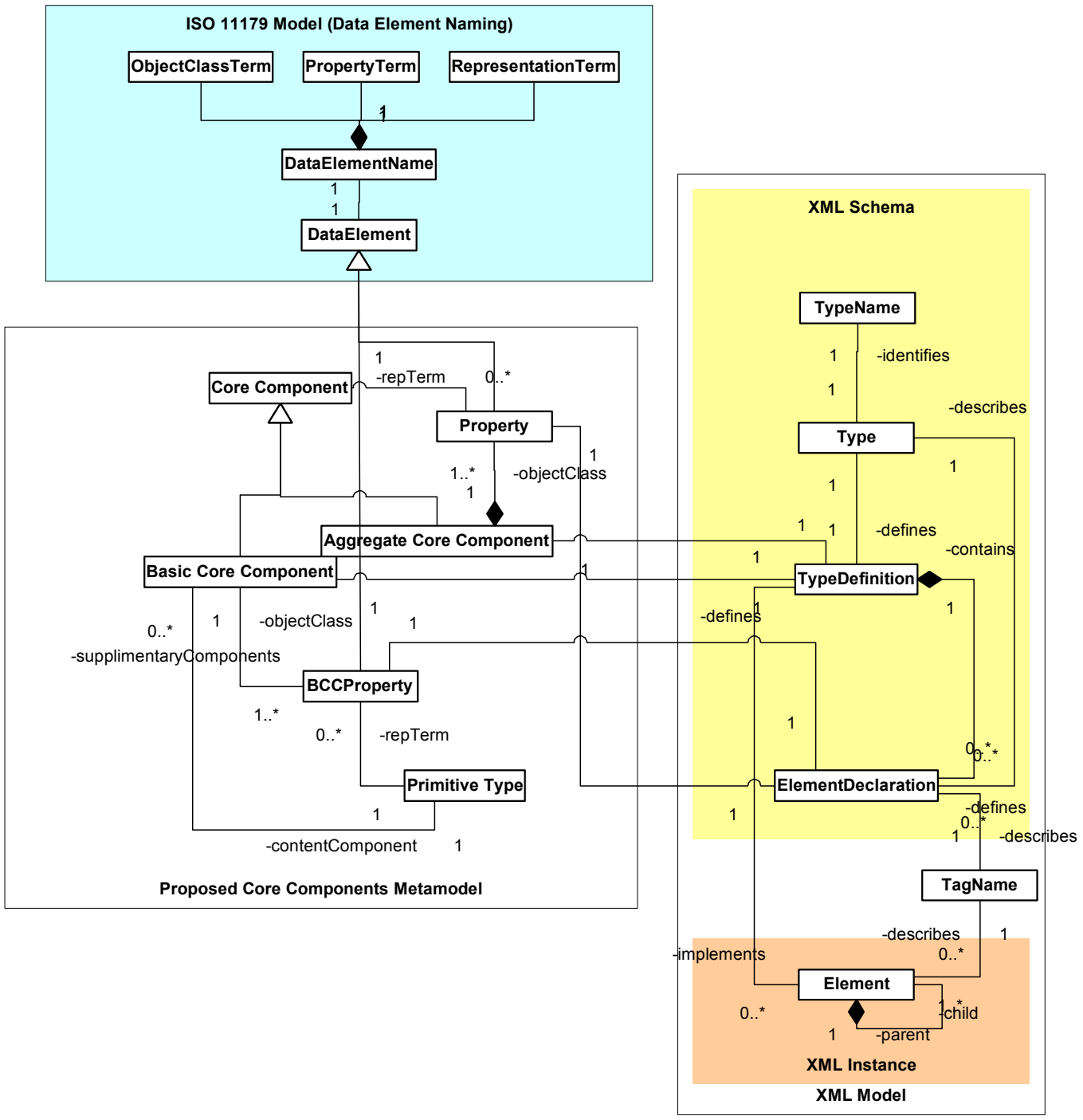
- Make Property Terms reflect the role played by that Property's content relative to its Object Class/Aggregate Core Component.
- Unify the concepts of Data Element (taken from **[ISONaming]**) and Property.
- Construct the Data Element Name (taken from **[ISONaming]**) for a Property by taking the Property's Object Class, its Property Term, and its Representation Term.
- Unify the concepts and definitions of Representation Terms and Core Component Types and consider these to be Basic Core Components.
- Compose Basic Core Components out of a Content Component and zero or more Supplementary Components, and make Content and Supplementary Components be Properties of the Basic Core Component:



Proposed Core Components Metamodel

- Eliminate the “Details” Representation Term and treat Aggregate Core Components as Representation Terms instead.

Following is the entire proposed Core Components metamodel, mapped to XML and XSD concepts.


ISO 11179 Model (Data Element Naming)
ObjectClassTerm
PropertyTerm
RepresentationTerm
1
DataElementName
1
1
DataElement
Core Component
1
-repTerm
0..*
Property
1
1..*
-objectClass
1
Aggregate Core Component
Basic Core Component
1
0..*
-supplimentaryComponents
1
-objectClass
1
BCCProperty
1..*
0..*
-repTerm
Primitive Type
1
-contentComponent
1
Proposed Core Components Metamodel
XML Schema
TypeName
1
-identifies
1
-describes
Type
1
1
1
1
1
-defines
-contains
TypeDefinition
1
-defines
1
1
1
0..*
0..*
ElementDeclaration
1
-defines
0..*
1
-describes
TagName
-describes
1
-implements
0..*
Element
0..*
1
*
-child
1
-parent
XML Instance
XML Model

123

3 XML Constructs

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

3.1 UBL Documentation

3.1.1 The UBL Dictionary

The primary component of the UBL documentation is its dictionary. The entries in the dictionary fully define the pieces of information available to be used in UBL business messages. Each dictionary entry has a **full name** that ties the information to its standardized semantics, while the name of the corresponding XML element or attribute is only a shorthand for this full name. The rules for element and attribute naming and dictionary entry naming are different.

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

3.1.2 Other UBL Documentation

Additional components of the UBL documentation include definitions of:

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

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

3.2 General Naming Rules for XML Constructs

The following are the naming rules that apply to **all** names of XML constructs in UBL:

1. Names *must* use Oxford English.
2. Names of XML constructs *must not* use non-alphabetic delimiters.
3. Names *must not* use acronyms, abbreviations, or other word truncations, with the exception of **Identifier**. Other exceptions *may* be identified in the future.
4. The Representation Term **Identifier** MUST be represented in XML names as `ID`.
5. Names *must not* contain non-letter characters unless required by language rules.
6. Names *must* be in singular form unless the concept itself is plural (example: **Goods**).

7. Names for XML constructs *must* use “camel-case” capitalization, such that each internal word in the name begins with an initial capital followed by lowercase letters (example: `AmountContentType`). As noted below, all XML constructs other than attributes use “upper camel-case”, with the first word initial-capitalized, while attributes use “lower camel-case”, with the first word all in lowercase. Exceptions are as follows:

```
DUNS for Dun & Bradstreet numbers
```

3.3 General Overview of Types

In XSD, elements are declared to have types, and most types (those complex types that are defined to have “complex contents”) are defined as a pattern of subelements and attributes. Thus, XSD has an indirect nesting structure of elements and types (where, for example, Type 1 below is the **parent type** of Element A and where Type 2 is the parent type of Element B and the type **bound to** Element A):

- Type 1
  - Element A
    - Type 2
      - Element B…

3.4 Elements and Attributes

3.4.1 Rules for UBL Elements

These rules distinguish the following constructs within the structural definitions of messages and their component parts. Note that some of these distinctions are specific to UBL and are not part of the formal definition of XML or XSD.

- Elements:
  - **Top-level elements**: Globally declared root elements, functioning at the level of a whole business message.
  - **Lower-level elements**: Locally declared elements that appear inside a business message.
    - **Intermediate elements**: Elements not at the top level that are of a complex type, only containing other elements and attributes.
    - **Leaf elements**: Elements containing only character data (though they may also have attributes). Note that, because of the XSD mechanisms involved, elements that contain only character data but also have attributes must be declared with complex types, but such elements with no attributes may be declared with simple types or complex types.
    - **Mixed-content elements**: Elements that allow both element content and data in their content models, and which may have attributes.
    - **Empty elements**: Elements that contain nothing (though they may have attributes).

3.4.1.1 Rules for the Naming and Definition of Top-Level Elements

Each UBL business message has a single root element that is a UBL top-level element. This element *must* be globally declared in a UBL root schema (which *may* contain definitions of additional root elements for other related messages in a functional area; see the Modularity, Namespaces, and Versioning paper) with a reference to a named type definition. Only top-level elements are declared globally.

Top-level elements are named according to the portion of the business process that they initiate. Example: <Order>, <AdvanceShipNotice>.

3.4.1.2 Naming and Definition of Lower-Level Elements

3.4.1.2.1 General Rules

Lower-level elements (as well as attributes) are considered Properties of the Object Class represented by their parent type.

Lower-level elements *must* be locally declared (Note: This recommendation is now under discussion and may change) as namespace-unqualified elements by reference to a named type, whether complex or simple, and be accompanied by documentation in the form of an <xsd:annotation> element with an <xsd:documentation> element that has a source attribute value of "Use". The documentation specifies the use of the element within its parent type.

There are several kinds of lower-level elements, each with distinct naming rules discussed in the following sections.

3.4.1.2.2 Rules for Intermediate Elements

The names of intermediate elements *must* contain the Property Term describing the element and MAY be preceded by an appropriate Qualifier term as necessary to create semantic clarity at that level. The Object Class *may* be used as a qualifier.

```
[Qualifier] + PropertyTerm
```

3.4.1.2.3 Rules for Leaf Elements

Leaf elements are named as follows:

```
[Qualifier] + PropertyTerm + RepresentationTerm
```

The naming of leaf elements follows these exceptions:

- The Representation Term **Text** is always removed.
- Leaf elements with substantially similar Property Terms and Representation Terms *must* remove the Property Term.

Examples: If the Object Class is **Goods**, the Property Term is **DeliveryDate**, and the Representation Term is **Date**, the element name is truncated to <GoodsDeliveryDate>; the element name for an identifier of a party <PartyIdentificationIdentifier> is truncated to <PartyIdentifier> – and then to <PartyID> because of the truncation rule.

3.4.1.2.4 Rules for Mixed-Content Elements

Mixed content in business documents is undesirable for a variety of reasons:

- White space is difficult to handle and complicates processing.
- Mixed content models allow little useful control over cardinality of elements.

For now mixed-content elements should have a Representation Term of **Prose**. This is currently under discussion with the LC SC.

3.4.1.2.5 Rules for Empty Elements

Empty elements are not permitted in UBL. For further details on the discussion details surrounding this recommendation consult the Elements vs Attributes position paper.

3.4.1.2.6 Rules Governing Elements of the Same Name and Their Respective Types

In those cases where it seems beneficial to have two elements that have the same tag name but are bound to different types, as is currently the case with the BIE Order.Header.Details (tag name Header), it is permissible.

3.4.2 Rules for the Naming and Definition of Attributes General Overview

There are two types of attribute:

- **Global attributes**: Attributes that have common semantics on the multiple elements on which they appear. These might be fixed attributes expressing an XML architectural form, attributes for assigning a unique element identifier, or attributes containing natural-language information (such as `xml:lang`).
- **Local attributes**: Attributes that are specific to the element on which they appear. Most attributes are local.

Attributes, like lower-level elements, are Properties of the Object Class represented by their parent type. They are named identically to leaf elements, except that they use lower camel-case rather than upper camel-case e.g. `amountCurrencyIDCode`.

3.4.2.1 Rules for Global Attributes

A global attribute *should* be used only when its semantics are absolutely unchanged no matter what element it's used on, AND it's made available on every single element. This rule applies to both external and UBL-specific global attributes. This allows common attributes that are everywhere but are *not* global, and that need documentation of their meaning in each XML environment in which they're used.

UBL-specific global attributes should be named just like regular attributes and sub-elements (i.e. as properties of an object class). Hence, by definition, the name of such a property *must* be consistent across all objects.

3.4.2.2 Rules for Local Attributes

All attributes that are not globally declared in UBL are considered to be local attributes.

3.4.2.3 Rules for the Naming and Definition of Types

3.4.2.3.1 General Rules

In UBL all types *must* be named and therefore they are "top-level". Most UBL elements are declared locally inside complex types and are therefore “lower-level”. In terms of ebXML Core Components, UBL complex types are Object Classes, subelements declared within them are Properties of those Object Classes, and the types bound to those subelements are themselves Object Classes which have their own Properties. See below:

```
[Qualifier] + ObjectClass + “Type”
```

Example: `CodeNameType`.

The definition *must* contain a structured set of XSD annotations in an `<xsd:annotation>` element with `<xsd:documentation>` elements that have `source` attribute values indicating the names of the documentation fields below:

- **UBL UID**: The unique identifier assigned to the type in the UBL library.

- **UBL Name**: The complete name (not the tag name) of the type per the UBL library.
- **Object Class**: The Object Class represented by the type.
- **UBL Definition**: Documentation of how the type is to be used, written such that it addresses the type's function as a reusable component.
- **Code Lists/Standards**: A list of potential standard code lists or other relevant standards that could provide definition of possible values not formally expressed in the UBL structural definitions.
- **Core Component UID**: The UID of the Core Component on which the Type is based.
- **Business Process Context**: A valid value describing the Business Process contexts for which this construct has been designed. Default is "In All Contexts".
- **Geopolitical/Region Context**: A valid value describing the Geopolitical/Region contexts for which this construct has been designed. Default is "In All Contexts".
- **Official Constraints Context**: A valid value describing the Official Constraints contexts for which this construct has been designed. Default is "None".
- **Product Context**: A valid value describing the Product contexts for which this construct has been designed. Default is "In All Contexts".
- **Industry Context**: A valid value describing the Industry contexts for which this construct has been designed. Default is "In All Contexts".
- **Role Context**: A valid value describing the Role contexts for which this construct has been designed. Default is "In All Contexts".
- **Supporting Role Context**: A valid value describing the Supporting Role contexts for which this construct has been designed. Default is "In All Contexts".
- **System Capabilities Context**: A valid value describing the Systems Capabilities contexts for which this construct has been designed. Default is "In All Contexts".

The following is an extended example of the documentation fields for the type:

```
<xsd:complexType name="PartyType">
  <xsd:annotation>
    <xsd:documentation source="UBL UID" xml:lang="en">PS1
    </xsd:documentation>
    <xsd:documentation source="xCBL Name" xml:lang="en">Party
    </xsd:documentation>
    <xsd:documentation source="Object Class" xml:lang="en">Party
    </xsd:documentation>
        <xsd:documentation source="UBL Definition"
      xml:lang="en">
    </xsd:documentation>
    <xsd:documentation source="Code Lists/Standards"
      xml:lang="en">NA
    </xsd:documentation>
    <xsd:documentation source="Core Component UID"
      xml:lang="en">[None]
    </xsd:documentation>
    <xsd:documentation source="Business Process Context"
      xml:lang="en">NA
    </xsd:documentation>
    <xsd:documentation source="Geopolitical/Region Context"
      xml:lang="en">NA
    </xsd:documentation>
    <xsd:documentation source="Official Constraints Context"
      xml:lang="en">NA
    </xsd:documentation>
    <xsd:documentation source="Product Context"
      xml:lang="en">NA
    </xsd:documentation>
```

```
    <xsd:documentation source=”Industry Context”
      xml:lang=”en”>NA
    </xsd:documentation>
    <xsd:documentation source=”Supporting Role Context”
      xml:lang=”en”>NA
    </xsd:documentation>
    <xsd:documentation source=”System Capabilities Context”
      xml:lang=”en”>NA
    </xsd:documentation>
  </xsd:annotation>
  …
</xsd:complexType>
```

4 Modularity, Namespaces, and Versioning

For an overview of current thinking on issues of modularity, namespace and versioning, consult the Modnamver position paper.

4.1.1 Rules for Namespace structure

4.1.2 Rules for Module structure

4.1.3 Rules for Versioning

Each namespace should have a version.

5 Rules for Context

For an overview of current thinking on Context Rules, consult the Specialization Architecture position paper from the Context Methodology Subcommittee.

6 Code Lists

This section recommends how to handle code lists in the UBL library. See the position paper on code lists for the rationale behind these recommendations.

6.1 Guidance to the UBL Modeling Process

Where possible, the UBL design should identify external code lists rather than develop its own internal code lists. Potential reasons for designing an internal code list include the need to combine multiple existing external code lists, or the lack of any suitable external code list. The lack of "easy-to-read" or "easy-to-understand" codes in an otherwise suitable code list is not sufficient reason to define an internal code list.

The UBL documentation must identify, for each UBL construct containing a code, the one or more code lists that must be *minimally* supported when the construct is used. Our recommendations for how to represent code lists in UBL schema modules have the effect of encapsulating this information in schema form as well. It is assumed that whole code lists, and not subsets of those code lists, are to be identified; however, users of the UBL library may customize these code lists by subsetting them.

6.2 Handling Code Lists in UBL Schema Modules

We recommend handling codes in UBL by defining a unique XSD complex type/simple type pair for each code list, so that the complex type (a **code list type**) can be bound to a UBL element (a **code element**) and the simple type (its corresponding **code content type**) can be bound to the element's contents. The UBL library will have occasion to define a few such type pairs for **UBL-native code lists**; mostly we recommend that UBL identify external code lists – *and bind its own code-related elements to types defined schema modules owned by external agencies*, where such schema modules (**code list modules**) exist.

In some cases, while an external code list may have been defined, an XSD schema module may not yet (or may not ever) be created and maintained by the code list's owning agency. In these cases, UBL will have to define a schema module on behalf of the agency. It is expected that these **orphaned code list modules** will not have the same validating power, nor be maintained with as much alacrity, as other code list modules with proper owners.

The recommendations here are designed to encourage the creation and maintenance of code list modules by their proper owners as much as possible.

Since the UBL library is based on the ebXML Core Components, the supplementary components identified for the Code. Type core component type are assumed to be sufficient for fully identifying a code list and any code used from it. Following are the components associated with Code.Type (as defined in **[CCTS 1.8]**) and the recommended representation in UBL form. Note that, because of the NDR recommendation on when to use elements vs. attributes, the supplementary components are all recommended to be attributes.

Component Name	Component Definition	Recommended UBL Form
Code. Content	A character string (letters, figures or symbols) that for brevity and/or language independence may be used to represent or replace a definitive value or text of an attribute	The content of the code element. The element is bound to the code list type and the element's content is bound to the code content type.
Code List. Identifier	The name of a list of codes	An attribute on the code element, defined as part of the code list type.
Code List. Agency. Identifier	An agency that maintains one or more code lists	An attribute on the code element, defined as part of the code list type.
Code List. Version. Identifier	The version of the code list	An attribute on the code element, defined as part of the code list type.
Code. Name	The textual equivalent of the code content	An optional attribute on the code element, defined as part of the code list type.
Language. Code	The identifier of the language used in the corresponding text string (in ISO 639 form)	An optional attribute on the code element, applying to the value of the attribute containing the Code.Name.

There are two parts to the handling of code lists in UBL: the creation of code list modules and the binding of code list types and code content types to UBL elements.

6.2.1 Creating Code List Modules

Following are strong recommendations for defining code list types and their corresponding code content types:

- Name the types and define a named namespace in which they are defined. If possible, define the types in their own schema module (XSD file).
- The attributes that you define in the Code List Type *may* be bound to any appropriate simple types but *must* have the following names: For Code List.Identifier use ID, For Code List Agency.Identifier use agencyID, For Code List version.identifier use versionID. Use codeName for Code.Name and languageCode for Language.Code.
- Define the Code. Content component as the element content. Define attributes for the Code List. Identifier, Code List. Agency. Identifier, and Code List. Version. Identifier components. Name component. The definition of the Language. Code component and the Code.Name component is optional.
- Make the XSD definitions as "tight" as you can, defining value defaults or fixed values for supplementary components and circumscribing the valid values of the code content as much as possible without compromising your own maintainability goals.
- **ISSUE:** Do we want to define canonical XSD documentation elements for code list modules? Even if we don't recommend such for external code list modules, should we have them in UBL-native modules or in orphan code list modules?

Following is a minimal template to follow. This hypothetical ISO 3166 code list for locale codes is used merely as an example. For different code lists, it might make sense not to use enumeration but rather to use pattern-matching regular expressions or to avoid strict code validation entirely.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="{namespace for ISO 3166 code list module}"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:iso3166="{namespace for ISO 3166 code list module}"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <xs:simpleType name="iso3166:CodeContentType">
    <xs:extension base="xs:token">
      <xs:enumeration value="DE"/>
      <xs:enumeration value="FR"/>
      <xs:enumeration value="US"/>
      . . .
    </xs:extension>
  </xs:simpleType>

  <xs:complexType name="iso3166:CodeType">
    <simpleContent>
      <xs:extension base="iso3166:CodeContentType">
        <xs:attribute name="ID"
          type="xs:token" fixed="ISO 3166 Locale Code"/>
        <xs:attribute name="agencyID"
          type="xs:token" fixed="ISO"/>
        <xs:attribute name="versionID"
          type="string" fixed="1.0"/>
    </simpleContent>
  </xs:complexType>
</xs:schema>
```

6.3 Binding Code List Types and Code Content Types to UBL Elements

No matter whether type pairs for code lists are defined by UBL or by an external agency, the UBL library must define its own elements for the provision of the actual codes in an instance. Such an element must be bound to the code list type (a complex type), and the element's contents must be bound to the code content type (a simple type). This creates a unique element for each kind of code.

Following is an example of this binding is created. Here, a UBL `LocaleCode` element, of type `LocaleCodeType`, is assumed to require a code from the hypothetical ISO 3166 locale code list defined in the previous section. Thus, it needs to contain an `ISO3166LocaleCode` element bound to the `iso3166:LocaleCodeType` type.

```
<xsd:complexType name="{LocaleCode element's parent}">
  <xsd:sequence>
    . . .
    <xsd:element name="LocaleCode" type="ubl:LocaleCodeType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="LocaleCodeType" id=". . .">
    <xsd:element name="ISO3166Code" type="iso3166:CodeType"/>
</xsd:complexType>
```

If the UBL library allows a choice of codes from different lists in any one location, it will do this by allowing a choice of *elements* in that location. There is no problem with the interpretation of

clashing codes from different lists because the surrounding code element distinguishes them. For example, if locale codes from two different code lists – ISO 3166 and the Codes “R” Us locale code list – are allowed, following is how to allow them in the UBL library.

```
<xsd:complexType name="{LocaleCode element’s parent}">
  <xsd:sequence>
    . . .
    <xsd:element name="LocaleCode" type="ubl:LocaleCodeType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name=”LocaleCodeType” id=”. . .”>
  <xs:choice>
    <xsd:element name=”ISO3166Code” type=”iso3166:CodeType”/>
    <xsd:element name=”CodesRUsCode” type=”codesrus:CodeType”/>
  </xs:choice>
</xsd:complexType>
```

7 References

[CCTS] *UN/CEFACT Draft Core Components Specification*, Part 1, 8 February, 2002, Version 1.8.

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

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

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

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

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

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

[XSD] *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May 2001.

506 # 8 Technical Terminology

507

Application-level validation	Adherence to business requirements, such as valid account numbers.
Ad hoc schema processing	Doing partial schema processing, but not with official schema validator software; e.g., reading through schema to get the default values out of it.
Assembly	Using parts of the library of reusable UBL components to create a new kind of business document type.
Context	A particular set of context driver values.
DTD validation	Adherence to an XML 1.0 DTD.
Instance constraint checking	Additional validation checking of an instance, beyond what XSD makes available, that relies only on constraints describable in terms of the instance and not additional business knowledge; e.g., checking co-occurrence constraints across elements and attributes. Such constraints might be able to be described in terms of Schematron.
Generic BIE	A semantic model that has a “zeroed” context. We are assuming that it covers the requirements of 80% of business uses, and therefore is useful in that state.
Instance root/doctype	This is still mushy. The transitive closure of all the declarations imported from whatever namespaces are necessary. A doctype may have several namespaces used within it.
Root Schema	A schema document corresponding to a single namespace, which is likely to pull in (by including or importing) schema modules. Issue: Should a root schema always pull in the “meat” of the definitions for that namespace, regardless of how small it is?
Schema	Never use this term unqualified!
Schema Module	A “schema document” (as defined by the XSD spec) that is intended to be taken in combination with other such schema documents to be used.
Schema Processing	Schema validation checking plus provision of default values and provision of new infoset properties.
Schema Validation	Adherence to an XSD schema.
Well-Formedness Checking	Basic XML 1.0 adherence.

Appendix A. Notices

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

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