Universal Business Language (UBL) Code List Schema White Paper

Version: 1.0 draft 7/27/2005 11:07:00 AM

Document identifier:

wd-ublclsc-codelistSchemaWP-20050727.doc

Location:

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

Author:

Marty Burns for National Institute of Standards and Technology, NIST, burnsmarty@aol.com

Contributors:

Anthony Coates abcoates@londonmarketsystems.com
Stephen Green stephen_green@seventhproject.co.uk

Abstract:

This white paper discusses the options on the table for UBL to select a basic model of schema representation for code lists. Several alternatives have been proposed that can be expected to meet the requirements set forth in the code list requirements document.

Note that this edition is a draft in progress and contains many TBDs.

Status:

This document was developed by the OASIS UBL Code List Subcommittee **[CLSC]**. Your comments are invited. Members of this subcommittee should send comments on this specification to the ubl-clsc@lists.oasis-open.org list. Others should subscribe to and send comments to the ubl-comment@lists.oasis-open.org list.

Table of Contents

1 Introduction and Use Case Overview

Code lists are used to enumerate values in documents that are or will be exchanged electronically. Code list values can, for example, specify items such as the units of measurement to be used, a particular day of the week or one element of an address such as the two letter state abbreviation in a US address.

Associated with each code list is a schema that codifies the relationship between code list elements and the value that can be assigned to each element. These schemas are used to validate each code list at the creation source and thereby prevent creation of invalid lists. Such schemas are typically drafted, circulated for review, edited and eventually adopted in some standardized form.

This standardization process ensures consistency and wide availability. It also helps to promote widespread usage. As adoption spreads, however, so does the need to modify a code list. The reasons for such changes are varied and might include new uses for an existing list or the addition of new code list values. The ease with which modifications can be made as well as the ease of code list creation and use are directly tied to way in which a code list is represented with a schema.

This white paper discusses alternatives for the representation of code lists in XML Schema by third party users. It compares the issues raised by each of these different representations and focuses on the impact each such representation has on code list schema design, use, reuse, extensibility and restrictability. Also discussed are code list instance document creation and modification.

1.1 Schema Modification Use Cases

In an effort to illuminate the issues that real users of code lists face the subsections below present two use cases. The first involves code list extension, in which code list changes must be made to permit a new code list value to be used. The second covers code list restriction, which limits the permissible values that can be assigned to a specific code list element in an instance document.

1.1.1 Use Case of Code List Schema Extension

A trading group such as an automobile manufacturer and its suppliers currently use UBL schemas to validate code list instance documents. These documents are exchanged electronically. Assume that a new currency, FQD (Free Iraqi Dollar), comes into being and is immediately used by these trading organizations. However the maintainer of the CurrencyCode list used by this group updates the list on an annual basis, so a new version of the standard code list is not yet available. Yet trade within the group must go on utilizing the new currency.

Assume that:

- CurrencyCode code list "ISO 4217" is defined by UN/CEFACT and is maintained by that organization. Changes to this list are made at regular intervals.
- The trading partners are using the UBL-Order-1.0 schema to define their order process.
- The following two XML fragments are used in a partner exchange and that instances of these same fragments will be used with the new Iraqi currency during transactions:

```
<cbc:LineExtensionAmount
   amountCurrencyID="EUR"
   amountCurrencyCodeListVersionID="0.3">
50.00
</cbc:LineExtensionAmount>
```

and

```
<PricingCurrencyCode>EUR</PricingCurrencyCode>
```

The challenge:

- How this trading group immediately accommodates use of the FQD without modifying the UBL schemas or the CurrencyCode list schema. Note that it is desired, as well, to be able to validate the fact that the creator of an XML instance file is indeed using only valid codes which now include the FQD.

1.1.2 Use Case of Code List Schema Restriction

Now consider a case in which the permissible values for a particular code list element in an instance document need to be more restricted than those in the associated code list schema. This could occur in the example above if the aforesaid trading group needed to use Euros as the only currency for some set of transactions.

Assume as before that:

- CurrencyCode code list "ISO 4217" is defined by UN/CEFACT and is maintained by that organization. Changes to this list are made at regular intervals.
- The trading partners are using the UBL-Order-1.0 schema to define their order process.

Then the following code list schema fragments are used in a partner exchange to restrict the type of the currency to only Euros:

```
<cbc:LineExtensionAmount amountCurrencyID="EUR"
amountCurrencyCodeListVersionID="0.3">
50.00
</cbc:LineExtensionAmount>
```

and

```
<PricingCurrencyCode>EUR</PricingCurrencyCode>
```

The Challenge:

- How can this trading group limit the use of currency codes to only the Euro without modifying the UBL schemas or the CurrencyCode list schema.

2 Code List Schema

2.1 Code List Schema Usage

The code list can be used as an element or as an attribute of a containing element. All mechanisms should support both usage styles since UBL currently uses both.

The following schema fragments allow the definition of the use cases discussed in this white paper:

Schema fragments:

```
<xsd:element name="LineExtensionAmount" type="ExtensionAmountType"/>
   <xsd:complexType name="ExtensionAmountType">
      <xsd:simpleContent>
         <xsd:extension base="sdt:UBLAmountType"/>
      </xsd:simpleContent>
   </xsd:complexType>

   <xsd:element ref="PricingCurrencyCode" minOccurs="0"/>

   <xsd:element name="PricingCurrencyCode" type="cur:CurrencyCodeType">
      <xsd:annotation>
         <xsd:documentation>
            <ccts:Component>
            <ccts:ComponentType>BBIE</ccts:ComponentType>
            <ccts:DictionaryEntryName>Order. Pricing Currency.
                Code</ccts:DictionaryEntryName>
            <ccts:Definition>the currency in which all pricing on
               the transaction will be specified.</ccts:Definition>
            <ccts:Cardinality>0..1</ccts:Cardinality>
            <ccts:ObjectClass>Order</ccts:ObjectClass>
            <ccts:PropertyTerm>Pricing Currency</ccts:PropertyTerm>

            <ccts:RepresentationTerm>Code</ccts:RepresentationTerm>
            <ccts:DataType>Currency_ Code. Type</ccts:DataType>
            </ccts:Component>
         </xsd:documentation>
      </xsd:annotation>
   </xsd:element>
```

Instance document:

```
<cbc:LineExtensionAmount amountCurrencyID="EUR"
   amountCurrencyCodeListVersionID="0.3">

50.00

</cbc:LineExtensionAmount>

and

<PricingCurrencyCode>EUR</PricingCurrencyCode>
```

2.2 Sample Code List Schema

For every code list, there exists a specific code list schema. This code list schema must have a targetNamespace containing the UBL-specific code list namespace and have a prefix that holds the code list identifier itself.

The element construct in the code list schema can be used to represent a global declared element in the document schemas. The name of the element is the UBL tag name of the specific Business Information Entity (BIE) for a code.

The simpleType can be used to represent the possible codes and the characteristics of code content. The name of the simpleType must always end with “Content”. Within the simpleType is a restriction of the XSD built-in data type “xs:normalizedString”. This restriction includes the specific facets “length”, “minLength”, “maxLength” and “pattern” for regular expressions to describe the specific characteristics of each code list.

Each code will be represented using the facet “enumeration” after its characteristics have been defined. The value of each enumeration represents the specific code value and the annotation includes the further definition of each code, for example “Code.Name”, “Language.Identifier”, and the code description.

The schema definitions to support this might appear as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Universal Business Language (UBL) Schema 1.0

  Copyright (C) OASIS Open (2004). All Rights Reserved.

…

  Universal Business Language Specification
      (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl)
  OASIS Open (http://www.oasis-open.org/)


  Document Type:     CurrencyCode
  Generated On:      Mon Aug 16 14:34:47 2004
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="urn:oasis:names:specification:ubl:schema:xsd:CurrencyCode-1.0"
xmlns:ccts="urn:oasis:names:specification:ubl:schema:xsd:CoreComponentParameters
-1.0"
targetNamespace="urn:oasis:names:specification:ubl:schema:xsd:CurrencyCode-1.0"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">
   <xsd:import
namespace="urn:oasis:names:specification:ubl:schema:xsd:CoreComponentParameters-
1.0" schemaLocation="../common/UBL-CoreComponentParameters-1.0.xsd"/>
   <xsd:simpleType name="CurrencyCodeContentType">
       <xsd:restriction base="xsd:normalizedString">
           <xsd:enumeration value="AED">
               <xsd:annotation>
                   <xsd:documentation>
                       <CodeName>Dirham</CodeName>
                   </xsd:documentation>
               </xsd:annotation>
           </xsd:enumeration>
           <xsd:enumeration value="AFN">
               <xsd:annotation>
                   <xsd:documentation>
                       <CodeName>Afghani</CodeName>
                   </xsd:documentation>
               </xsd:annotation>
           </xsd:enumeration>
           <xsd:enumeration value="ALL">
               <xsd:annotation>
                   <xsd:documentation>
```

```
                    <CodeName>Lek</CodeName>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
        …

        <xsd:enumeration value="ZMK">
            <xsd:annotation>
                <xsd:documentation>
                    <CodeName>Kwacha</CodeName>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
        <xsd:enumeration value="ZWD">
            <xsd:annotation>
                <xsd:documentation>
                    <CodeName>Zimbabwe Dollar</CodeName>
                </xsd:documentation>
            </xsd:annotation>
        </xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CurrencyCodeType">
    <xsd:annotation>
        <xsd:documentation>
            <ccts:Component>
                <ccts:ComponentType>DT</ccts:ComponentType>
                <ccts:DictionaryEntryName>Currency_ Code.
Type</ccts:DictionaryEntryName>
                <ccts:RepresentationTerm>Code</ccts:RepresentationTerm>
                <ccts:DataTypeQualifier>Currency</ccts:DataTypeQualifier>
                <ccts:DataType>Code. Type</ccts:DataType>
            </ccts:Component>
            <ccts:Instance>
                <ccts:CodeListID>ISO 4217 Alpha</ccts:CodeListID>
                <ccts:CodeListAgencyID>6</ccts:CodeListAgencyID>
                <ccts:CodeListAgencyName>United Nations Economic Commission
for Europe</ccts:CodeListAgencyName>
                <ccts:CodeListName>Currency</ccts:CodeListName>
                <ccts:CodeListVersionID>0.3</ccts:CodeListVersionID>
                <ccts:CodeListURI>http://www.bsi-
global.com/Technical%2BInformation/Publications/_Publications/tig90x.doc</ccts:C
odeListURI>

   <ccts:CodeListSchemeURI>urn:oasis:names:specification:ubl:schema:xsd:Currency
Code-1.0</ccts:CodeListSchemeURI>
                <ccts:LanguageID>en</ccts:LanguageID>
            </ccts:Instance>
        </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:extension base="CurrencyCodeContentType">
            <xsd:attribute name="codeListID" type="xsd:normalizedString"
                 use="optional" fixed="ISO 4217 Alpha"/>
            <xsd:attribute name="codeListAgencyID" type="xsd:normalizedString"
                 use="optional" fixed="6"/>
            <xsd:attribute name="codeListAgencyName" type="xsd:string"
                 use="optional"
                    fixed="United Nations Economic Commission for Europe"/>
            <xsd:attribute name="codeListName" type="xsd:string"
                     use="optional" fixed="Currency"/>
            <xsd:attribute name="codeListVersionID"
                     type="xsd:normalizedString"
                     use="optional" fixed="0.3"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
            <xsd:attribute name="languageID" type="xsd:language"
                     use="optional" fixed="en"/>
            <xsd:attribute name="codeListURI" type="xsd:anyURI"
                     use="optional"
```

```
                       fixed="http://www.bsi-global.com/Technical%2
                        BInformation/Publications/_Publications/tig90x.doc"/>
              <xsd:attribute name="codeListSchemeURI" type="xsd:anyURI"
                       use="optional"
                        fixed="urn:oasis:names:specification:ubl:
                          schema:xsd:CurrencyCode-1.0"/>
          </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>

   <xsd:attribute name="CurrencyCode" type="CurrencyCodeContentType"/>

   <xsd:element name="CurrencyCode" type="CurrencyCodeType"/>

</xsd:schema>
```

2.3 Code List Schema Components

This section and its associated subsections:

- Describes the various xml schema components that must (schema filename, xml header, xml schema header, end of schema) be used in the creation of a currency code list schema.
- Presents several possible approaches for currency code list attribute definition in an xml schema. Any particular instance of a currency code list schema can be assembled using one or more of these approaches.
- Provides schema code fragment examples.

Section 3 will discuss applying each of several possible code list schema extensibility and restriction mechanisms to the schema components in this section. Note that the following are the components individually described. For each actual code list implementation, a subset of these components are required. For example in the previous section, a sample possible code list schema is presented. It is composed of the following components described in this section:

- ❑ Schema File Name
- ❑ XML Header
- ❑ XML Schema Header
- ❑ Simple Type to Contain Enumerated Values
- ❑ Complex Type to Hold Enumerated Values and Supplemental Components
- ❑ Global Attribute to Allow Usage of Code Lists as an Attribute
- ❑ Global Element to Allow Usage of Code List as an Element
- ❑ End of Schema

The code list schema components summarized in the following subsections are:

- ❑ Schema File Name
- ❑ XML Header
- ❑ XML Schema Header
- ❑ Unrestricted Element
- ❑ Simple Type to Contain Enumerated Values

- ❑ Complex Type to Hold Enumerated Values and Supplemental Components
- ❑ Global Attributes to Allow Usage of Code Lists as an Attribute
- ❑ Global Element to Allow Usage of Code List as an Element
- ❑ End of Schema

2.3.1 Schema File Name

The name of this schema file should be:

```
UBL-CodeList-{CodeListName}-{CodeListVersionID}.xsd
```

For example:

```
UBL-CodeList-CurrencyCode-1.0.xsd
```

2.3.2 XML Header

The xml header specifies the xml version number and, optionally, the character encoding used by an xml document. It must be the first line of such an xml document. An example is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Universal Business Language (UBL) Schema 1.0-draft-10.1

  Copyright (C) OASIS Open (2004). All Rights Reserved.
…
 …
-->
```

2.3.3 XML Schema Header

The xml schema header declares the namespaces to be used in a schema document. See below for an example.

```
<xs:schema
      targetNamespace="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0-draft-7.1"
                  xmlns="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0-draft-7.1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified" attributeFormDefault="unqualified" version="1:0-draft-7.1">
```

2.3.4 Unrestricted Element

An unrestricted element of type normalizedString is a placeholder that is not constrained in an instance document by a specific list of enumerated codes. A concrete member of element's Substitution Group is used in its place within instance documents validated with a schema containing it. The schema syntax used for an unrestricted currency code element could be:

```
<xs:element name="CurrencyCodeAbstract" type="xs:normalizedString" />
```

2.3.5 Simple Type to Contain the Enumerated Values

An xml simple type can be used to enumerate a list of currency codes permitted to appear in an instance document. The following schema fragment contains such a list:

```
<xs:simpleType name=”CurrencyCodeContentType”>
    <xs:restriction base=”xs:string”>
        <xs:enumeration value=”AED”>
            <xs:annotation>
                <xs:documentation>
                    <CodeName>UAE Dirham</CodeName>
                </xs:documentation>
            </xs:annotation>
        </ xs:enumeration>
        <xs:enumeration value=”ALL”>
            <xs:annotation>
                <xs:documentation>
                    <CodeName>Albanian Lek</CodeName>
                </xs:documentation>
            </xs:annotation>
        </xs:xs:enumeration>
        <xs:enumeration value=”AMD”
            <xs:annotation>
                <xs:documentation>
                    <CodeName>Armenian Dram</CodeName>
                </xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value=”ANG”/>
        <xs:enumeration value=”AOA”/>
        <xs:enumeration value=”XDR”/>
              …
        <xs:enumeration value=”ZAR”/>
        <xs:enumeration value=”ZMK”/>
        <xs:enumeration value=”ZWD”/>
    </xs:restriction>
</xs:simpleType>
```

2.3.6 Complex Type to Hold Enumerated Values and Supplemental Components

An xml complex type could be used to define a type that would hold currency code values. The xml schema fragment below contains the definition for such a complex type.

```
     <xs:complexType name="CurrencyCodeType">
          <xs:annotation>
               <xsd:documentation>
                    <ccts:Component>
                         <ccts:ComponentType>DT</ccts:ComponentType>
                         <ccts:DictionaryEntryName>Code. Type</ccts:DictionaryEntryName>
                         <ccts:RepresentationTerm>Code</ccts:RepresentationTerm>
                         <ccts:DataTypeQualifier>Currency</ccts:DataTypeQualifier>
                         <ccts:DataType>Code. Type</ccts:DataType>
                    </ccts:Component>
                    <ccts:Instance>
                         <ccts:CodeListID>ISO 4217 Alpha</ccts:CodeListID>
                         <ccts:CodeListAgencyID>6</ccts:CodeListAgencyID>
                         <ccts:CodeListAgencyName>United Nations Economic Commission for
Europe</ccts:CodeListAgencyName>
                         <ccts:CodeListName>Currency</ccts:CodeListName>
                         <ccts:CodeListVersionID>0.3</ccts:CodeListVersionID>
                         <ccts:CodeListUniformResourceID>
                              http://www.bsi-global.com/Technical%2BInformation
                              /Publications/_Publications/tig90x.doc </ccts:CodeListUniformResourceID>
                         <ccts:CodeListSchemeUniformResourceID>
                              urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0-draft-10.1
                              </ccts:CodeListSchemeUniformResourceID>
                         <ccts:LanguageID>en</ccts:LanguageID>
                    </ccts:Instance>
               </xsd:documentation>
          </xs:annotation>
          <xs:simpleContent>
               <xs:extension base="CurrencyCodeContentType">
                    <xsd:attribute name="name" type="xsd:string" use="optional"/>
                    <xsd:attribute name="codeListID" type="xsd:normalizedString" use="optional"
                              fixed="ISO 4217 Alpha"/>
                    <xsd:attribute name="codeListAgencyID" type="xsd:normalizedString" use="optional"
                              fixed="6"/>
                    <xsd:attribute name="codeListAgencyName" type="xsd:string" use="optional"
                              fixed="United Nations Economic Commission for Europe"/>
                    <xsd:attribute name="codeListName" type="xsd:string" use="optional"
                              fixed="Currency"/>
                    <xsd:attribute name="codeListVersionID" type="xsd:normalizedString" use="optional"
                              fixed="0.3"/>
                    <xsd:attribute name="codeListURI" type="xsd:anyURI" use="optional"
                         fixed="http://www.bsi-global.com/
                              Technical%2BInformation/Publications/_Publications/tig90x.doc"/>
                    <xsd:attribute name="codeListSchemeURI" type="xsd:anyURI" use="optional"
                         fixed="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0-draft-10.1"/>
                    <xsd:attribute name="languageID" type="xsd:language" use="optional" fixed="en"/>
               </xs:extension>
          </xs:simpleContent>
     </xs:complexType>
```

2.3.7 Global Attributes to Allow Usage of Code Lists as an Attribute

A currency code list could also be treated as a global attribute in an xml schema. The schema fragment below provides a sample implementation for such an approach.

```
    <xs:attribute name="CurrencyCode" type="CurrencyCodeContentType"/>
    <xs:attribute name="name" type="xs:normalizedString" fixed="cur"/>
    <xs:attribute name="codeListID" type="xs:normalizedString" fixed="ISO 4217 Alpha"/>
    <xs:attribute name="codeListAgencyID" type="xs:normalizedString " fixed="6"/>
    <xs:attribute name="codeListAgencyName" type="xs:string "
          fixed="United Nations Economic Commission for Europe"/>
    <xs:attribute name="codeListVersionID" type="xs:normalizedString " fixed="0.3"/>
    <xs:attribute name="codeListName" type="xs:string" fixed="CurrencyCode"/>
    <xs:attribute name="codeListURI" type="xs:anyURI"
       fixed="http://www.bsi-global.com/Technical%2BInformation/Publications/_Publications/tig90x.doc"/>
    <xs:attribute name="codeListSchemeURI" type="xs:anyURI"
        fixed="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0-draft-8-1"/>
    <xs:attribute name="languageID" type="xs:language" fixed="en"/>
```

2.3.8 Global Element to Allow Usage of Code List as an Element

A currency code list can also be handled using a global element.

```
<xs:element name="CurrencyCode" type="CurrencyCodeType"
            substitutionGroup="CurrencyCodeAbstract"/>
```

2.3.9 End of Schema

All xml schema documents must be terminated with an end of schema statement. An example is provided below.

```
</xs:schema>
```

3 Methods

The methods described below have been proposed by participants to extend or restrict code lists. However each method has strengths and weaknesses with regard to its use for schema creation, use, extension and restriction. This section summarizes the attributes of each such method while section 3.5.3 considers the impact on a schema of these attributes.

3.1 Substitution Groups

In order to promote maximum reusability and ease code list maintenance, code list designers are expected to build new code lists from existing lists. They could for example combine several code lists or restrict an existing code list when creating a new one. These new code lists must be usable in UBL elements in the same manner in which "basic" code lists are used.

Substitution Groups can be used when modifying an existing code list to create a new list. Substitution Groups rely on the fact that a new global element definition used in a code list schema can be substituted for an existing element definition in a schema that is or has been in use. The result of this for instance documents using this schema is that the instance documents will be validated according to the revised schema. Note that this approach is valid only for elements that use any revised global element(s) by reference.

One example of the use of Substitution Groups could be the replacement of a preexisting list of values (i.e. an enumeration) with an extended list. The result in this example would be a change to the permissible values that can be assigned to a line item.

The subsections below consider the use of Substitution Groups for both extension and restriction of an existing code list.

3.1.1 Extending a Code List Using Substitution Groups

The following schema fragment could be used to extend a code list with Substitution Groups so that it contains the new Iraqi currency symbolized by the FRQ code:

```
<xs:schema targetNamespace="cust"
  xmlns:std="std"
  xmlns="cust"
  xmlns:cust="custom"
  xmlns:xs=http://www.w3.org/2001/XMLSchema
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

<xs:import namespace="std"
  schemaLocation="D:\_PROJECT\NIST\XMLSchema\test0513\std.xsd"/>

<xs:element name="LocaleCode" substitutionGroup="std:LocaleCodeA">
  <xs:annotation>
    <xs:documentation>A substitute for the abstract LocaleCodeA
      that extends the enumeration
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:union memberTypes="std:aStdEnum">
      <xs:simpleType>
        <xs:restriction base="xs:token">
         <xs:enumeration value="IL"/>
         <xs:enumeration value="GR"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:union>
  </xs:simpleType>
</xs:element>
</xs:schema>
```

3.1.2 Restricting a Code List Using Substitution Groups

The following schema fragment could use Substitution Groups to restrict a code list so that Euros are the only acceptable currency code:

```
<xs:import namespace="std"
  schemaLocation="D:\_PROJECT\NIST\XMLSchema\test0513\std.xsd"/>
<xs:element name="LocaleCode" substitutionGroup="std:LocaleCodeA">
  <xs:annotation>
    <xs:documentation>
      A substitute for the abstract LocaleCodeA that restricts
        the enumeration
    </xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:token">
    <xs:enumeration value="DE"/>
    <xs:enumeration value="US"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

3.1.3 Issues in applying this method

The principal issue concerning the use of code list is the fact that substitution groups and abstract types are currently prohibited by NDR rules. There is a technical attempt underway to see if these can be eliminated from the code list schema itself but used in extension/restriction schemas.

Below find some email snippets that contain relevant discussion:

```
7/24/05

I'll leave technical details to Marty, but I had a look over the
weekend at the issue of providing "hooks" for substitution groups for
code lists, and I came to the following conclusion:

Supporting substitution groups as a mechanism for customising UBL code
lists means that UBL will have to *passively contain* substitution
group definitions, one for each code list.  However, it is important to
understand that the UBL Schemas will not *actively use* substitution
groups, that they will only passively contain them.  What that means is
that the UBL Schemas will contain substitution group definitions in a
way which *does not change* the way documents are validated by the UBL
Schemas.

I believe this is an acceptable compromise.  The one negative aspect to
this is that the UBL Schemas will require XML Schema validators to
behave consistently when faced with this trivial passive usage of
substitution groups.  Although we haven't tested this yet, I believe we
will have no trouble testing this as part of our normal testing, and I
think the risks are minimal.  That said, we need to finalise that
```

actual proposed code list Schema structure and test it ASAP, to give people confidence.

7/24/05

Yes, I think you are right. It look to me as though (as I'm sure you've said before), if you want to be able to *extend* a code list, the only option is

(i) define an empty abstract type as the head of the substitution group for the code list;
(ii) extend that abstract type for the actual code list type;
(iii) declare the code list element as being part of the substitution group.

I think the thing to be navigated is the question of what it means to "use" substitution groups. With this approach, the UBL Schemas themselves don't make any *real* use of substitution groups, i.e. no substitutions are provided, there is only ever one choice. Formally the Schema validators need to handle the substitution group definitions, so there is a question of validator conformance to overcome. However, there is still, I think, a case for saying that (i)-(iii) constitutes "support" rather than actual "use" of substitution groups in UBL. Thoughts?

Cheers, Tony.

On Sat, 23 Jul 2005 15:04:57 +0100, <Burnsmarty@aol.com> wrote:

```
> Tony,
> Good work. I will try to digest details. I have a nasty suspicion that
> the
> problem is the fact that the derived type is not the same as the
> substitution
> head. This is the reason I orginally had an abstract type as the head
> with the
>  actual code list as a substitution group for the abstract head.
> I still hope with more testing we can resolve this problem.
> Marty
> In a message dated 7/23/2005 9:54:38 A.M. Eastern Daylight Time,
> abcoates@londonmarketsystems.com writes:
>
> Marty, I  had a problem with some back-slashes in the paths in your
> Schema
> (don't work under Linux).  Once I fixed those, the follow problems
> remain
> (both Windows & Linux, using  oXygen/Xerces-J):
>
> Location: 23:82
> Description: E e-props-correct.4:  The {type definition} of element
> 'PricingCurrencyCode' is not  validly derived from the {type definition}
> of
> the substitutionHead  ':PricingCurrencyCode', or the {substitution group
> exclusions}  property of ':PricingCurrencyCode' does not allow this
> derivation.
> URL:  http://www.w3.org/TR/xmlschema-1/#e-props-correct
>
> Location:  42:116
> Description: E derivation-ok-restriction.2.1.2: Error for type
> '#AnonType_LineExtensionAmount'.  The attribute use  'amountCurrencyID'
> in
> this type has type 'null', which is not  validly derived from
> 'CurrencyCodeContentType', the type of the  matching attribute use in the
> base type.
> URL:  http://www.w3.org/TR/xmlschema-1/#derivation-ok-restriction
>
> Location:  38:86
> Description: E e-props-correct.4: The {type definition} of  element
> 'LineExtensionAmount' is not validly derived from the {type  definition}
> of
> the substitutionHead 'cbc:LineExtensionAmount', or  the {substitution
> group
> exclusions} property of  'cbc:LineExtensionAmount' does not allow this
> derivation.
> URL:  http://www.w3.org/TR/xmlschema-1/#e-props-correct
>
> Cheers,  Tony.
>
> On Fri, 22 Jul 2005 20:15:09 +0100, <Burnsmarty@aol.com>  wrote:
>
>> Thanks Tony.  Yeah it worked in XMLSpy for me and not  MSXML.
>> Marty
>> In a message dated 7/22/2005 2:58:03 P.M. Eastern  Daylight Time,
>> abcoates@londonmarketsystems.com  writes:
>>
>> Hi  Marty.  I tested your example with the  latest version of XML Spy,
>> and
>> it's fine.   However, it doesn't work with the latest  version of
>> 'oXygen'
>>  (which uses Xerces-J).  I haven't had a  chance to track down the
>> issue;
>> I'll let you know when I work out  what it is.
```

7/20/05

As seen in the Atlantic call minutes, we have been directed to confirm the extensibility of code lists via substitution groups, without the need for substitution groups or abstract elements within the code list schemas themselves.

At issue is whether the global element and attributes declared in the code list schema itself can have the code values based on the initial enumerated list, and then be substituted for. If we can confirm that this so, we have a clear winner methodology.

We have agreed to use the NIST validation service as a litmus test for the method. This service can be reached at http://www.mel.nist.gov/msid/validation/.

Attached is a draft of UBL 1.0 Schemas with substitution groups used to extend it. Stephen has also circulated some examples based on UBL "2.0" schemas.

The specific issue (if there is one) is whether the code list content must be derived from xsd:normalizedString in order to allow for union based substitutions of an existing list and the extensions. In this case, the standard code list has a substitution group in the code list schema that declares the specific standard list of codes as a substitute for the generic (i.e. normalized string) code.

The more desirable case is that the code list content itself be the definition in the code list document. The key question is whether the test parsers will permit substitution of elements based on this type (the enumerated list) as opposed to the generic type (just normalizedString).

Please comment and test away... See if we can make this work for all the parsers. Note that the attached schema passes XML spy but not msxml4.

7/19/05

Are you arguing with the desirability of:

```
    Extensibility?
    Substitution Groups?
    Both?
```

The use of substitution groups complements the suggested use of unions in extending code lists in wd-ubl-cmsc-cmguidelines-1.0.html. I think that without substitution groups, this mechanism doesn't work.

When a user community needs to extend or restrict UBL for their own reasons, it is desirable to use a mechanism that forces the implementer to declare the extensions

```
explicitly. Substitution groups does this because in order to validate an instance the
definition of the substituting type must be present. Also, the designer of the base
schemas ensures the type consistency of the substitutable information.

I believe what is being proposed is not that substitution groups are necessarily used
extensively within UBL schemas. What is being proposed is that the schemas be designed so
that the substitution group mechanism can be utilized in extending the schemas in a clean
and traceable way (that is the extensions are explicit in the referenced schema in the
instance document). If substitution groups are used in UBL schemas they would only need
to be used in the code list schemas themselves to allow an unconstrained or enumeration
constrained set of values to be used in the code list.

What this requires of UBL primarily is the extensive use of global elements (the ones
that are substitutable), and, code list schema design and usage that facilitates the
extension mechanism.
```

3.2 Redefine

Redefine is used to change the description of an element’s type in an xml schema.

3.2.1 Extending a Code List Using Redefinition

The following schema fragment could be used to extend a code list with redefinition so that it contains the new Iraqi currency symbolized by the FRQ code:

ADD NEW SCHEMA FRAGMENT HERE THAT EXTENDS CODE LIST USING REDEFINITION

3.2.2 Restricting a Code List Using Redefinition

The following schema fragment could be used to restrict with redefinition a code list so that it uses Euros as the only acceptable currency code:

ADD NEW SCHEMA FRAGMENT HERE THAT RESTRICTS CODE LIST USING REDEFINITION

3.2.3 Issues in applying this method

Redefine is a promising technique because it allows redefinition of a type. The challenge to making redefine work involves the order of inclusion of schemas throughout the UBL hierarchy. Somehow it has to be arranged that the redefinition be imported in all schemas that import the base class.

We have not been able to get this to work yet.

Below find some email snippets that contain relevant discussion:

```
7/1/05

So there are interop issues here, but also one trivial point:

Processors are allowed to ignore second and subsequent imports of the
same namespace.  Accordingly, in order to get your example to work
with XSV, I had to _reverse_ the order of the imports in
myUBLExtensions.xsd, so the import of the redefining schema comes
first.  You might try that.

There's certainly no _error_ in your schema docs.
```

ht

5/3/05
Henry,

I am trying to apply your method. I have started with the current UBL schemas. I am having some difficulty making this work.

What I tried was:

1) UBL code list schema contains type xsd:normalizedString code list but no values.
2) Created separate schema that contains a redefine of code list with standard set of enumerated values.
3) Created schema that includes all subsidiary schemas in addition to the one with the desired enumerated values. Some of these included schemas themselves have included the code list schema.

Problem: Parser chokes on the redefine because the base code list was imported or included in the other UBL schema documents -- that is where code lists are used in document designs.

Do I have a cockpit problem here or do you see a workaround?

This is by far the potentially most elegant solution to code list extensibility because the customizer would need to import a list of pointers to his selected code list enumerations to be used in validation of instance documents.

I have attached some working files in case there is a simple error.

3/10/05

I've worked a trivial example to fill out and illustrate the redefine approach I sent earlier.

Here's the base schema, published by the namespace owner:

curEnumBase.xsd:

```
<xs:schema targetNamespace="http://www.example.com/fakeUBL"
 xmlns="http://www.example.com/fakeUBL"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Example base schema for extensible enumerations -->  <xs:element
name="currency" type="currencyCodeType"/>

 <xs:simpleType name="currencyCodeType">
  <xs:restriction base="xs:NMTOKEN"/>
 </xs:simpleType>
</xs:schema>
```

Here's the vanilla driver schema, likewise published by the namespace owner:

curEnumDriver.xsd:

```
<xs:schema targetNamespace="http://www.example.com/fakeUBL"
 xmlns="http://www.example.com/fakeUBL"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Example driver schema for extensible enumerations -->

 <xs:redefine schemaLocation="curEnumBase.xsd">
  <xs:simpleType name="currencyCodeType">
   <xs:restriction base="currencyCodeType">
    <xs:enumeration value="UKL"/>
    <xs:enumeration value="USD"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:redefine>
</xs:schema>
```

And here's a valid instance:

```
<currency xmlns="http://www.example.com/fakeUBL"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.example.com/fakeUBL
 curEnumDriver.xsd">
 UKL
</currency>
```

And an invalid one:

```
<currency xmlns="http://www.example.com/fakeUBL"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.example.com/fakeUBL
 curEnumDriver.xsd">
 CAD
</currency>
```

To make it valid, we make our own extended driver:

curEnumExt.xsd:

```
<xs:schema targetNamespace="http://www.example.com/fakeUBL"
 xmlns="http://www.example.com/fakeUBL"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Example extended schema for extensible enumerations -->

 <xs:redefine schemaLocation="curEnumBase.xsd">
  <xs:simpleType name="currencyCodeType">
   <xs:restriction base="currencyCodeType">
    <xs:enumeration value="UKL"/>
    <xs:enumeration value="USD"/>
    <xs:enumeration value="CAD"/>
   </xs:restriction>
  </xs:simpleType>
 </xs:redefine>
</xs:schema>
```

and if we change the xsi:schemaLocation of our example to point to this driver, it's valid.

As it happens, Dan Vint and colleagues had arrived at this solution long before I proposed it. He described it in much more detail in a recent post to xmlschema-dev [1], including the following, which says it all, IMO:

> 1) Produce a base schema that references a type for each list with no enumerations [e.g. curEnumBase].
> 2) Produce a second schema that redefines those list types to the enumerated values [e.g. curEnumDriver].
>
> Anyone needing to modify those lists modifies the second redefining schema [e.g. curEnumExt]. Now when I release the next version, all that has to happen is the modifier reviews the new redefining schema for new lists. These changes have to be copied into the file they originally modified. They also have to find any changes to the existing lists as well.
>
> They then use the newly produced base schema and point their modified redefine schema to point at the this file instead of the original base schema.
>
> This has the advantage of creating one single file with all the modifications. It still is not a perfect solution, but it is the best compromise that we could come up with.
>
> We also made the type of the lists to be QNAME and we require that anyone adding a value to a list use an appropriate namespace prefix to identify their additions.
>
> ['e.g.'s added]

Some further observations:

1) The publisher could make this approach easier if they used an external general entity to include the enumerations in the redefining schema document:

   curEnumDriver.xsd:

```
<!DOCTYPE xs:schema [
<!ENTITY currencyEnumeration SYSTEM
"http://www.example.com/fakeUBL/currencies.xnt">
]>
<xs:schema targetNamespace="http://www.example.com/fakeUBL"
 xmlns="http://www.example.com/fakeUBL"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <!-- Example driver schema for extensible enumerations -->

 <xs:redefine schemaLocation="curEnumBase.xsd">
  <xs:simpleType name="currencyCodeType">
```

```
       <xs:restriction base="currencyCodeType">
        &currencyEnumeration;
       </xs:restriction>
      </xs:simpleType>
     </xs:redefine>
    </xs:schema>
```

Then the extension can track the official list and changes thereto
more easily:

curEnumExt.xsd:

```
    <!DOCTYPE xs:schema [
    <!ENTITY currencyEnumeration SYSTEM

"http://www.example.com/fakeUBL/currencies.xnt">
    ]>
    <xs:schema targetNamespace="http://www.example.com/fakeUBL"
     xmlns="http://www.example.com/fakeUBL"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
     <!-- Example driver schema for extensible enumerations -->

     <xs:redefine schemaLocation="curEnumBase.xsd">
      <xs:simpleType name="currencyCodeType">
       <xs:restriction base="currencyCodeType">
        &currencyEnumeration;
        <xs:enumeration value="CAD"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:redefine>
    </xs:schema>
```

 2) This approach works directly for types used for attributes _or_
    elements, whereas any approach using substitution groups (which
    are very useful for many things, but not this problem, in my
    opinion) only works directly for elements.

Hope this helps -- if you think it does, please pass it on to the UBL
list. . .

ht

[1] http://lists.w3.org/Archives/Public/xmlschema-dev/2005Mar/0008.html
--
 Henry S. Thompson, HCRC Language Technology Group, University of
Edinburgh
                     Half-time member of W3C Team
    2 Buccleuch Place, Edinburgh EH8 9LW, SCOTLAND -- (44) 131 650-4440
            Fax: (44) 131 650-4587, e-mail: ht@inf.ed.ac.uk
                   URL: http://www.ltg.ed.ac.uk/~ht/ [mail really from
me _always_ has this .sig -- mail without it is forged spam]

The xml-dev list is sponsored by XML.org <http://www.xml.org>, an
initiative of OASIS <http://www.oasis-open.org>

The list archives are at http://lists.xml.org/archives/xml-dev/

```
To subscribe or unsubscribe from this list use the subscription
manager: <http://www.oasis-open.org/mlmanage/index.php>
```

3.3 xsi:type

The `xsi:type` construct identifies a derived type in an instance document. This approach can be used to either extend or restrict a code list by replacing content as appropriate. Each of these is discussed in its own subsection below.

3.3.1 Extending a Code List Using xsi:type

The following schema fragment could be used to extend a code list with the `xsi:type` construct so that it contains the new Iraqi currency symbolized by the FRQ code:

ADD NEW SCHEMA FRAGMENT HERE THAT EXTENDS CODE LIST USING xsi:type

3.3.2 Restricting a Code List Using xsi:type

The following schema fragment could be used to restrict with the `xsi:type` construct a code list so that it uses Euros as the only acceptable currency code:

ADD NEW SCHEMA FRAGMENT HERE THAT RESTRICTS CODE LIST USING xsi:type

3.3.3 Issues in applying this method

3.4 Union with Any Type Plus Lax

The xml “union with any type” construct permits an element or value to be any valid type and thus does not constrain content for such elements in instance document. Use of the lax value for an attribute in an xml schema instructs the parser to validate elements and attributes in an instance document if associated schema content is present but to suppress generation of errors if content is not present. This approach can be used to either extend or restrict a code list by replacing content as appropriate. Each of these is discussed in its own subsection below.

3.4.1 Extending a Code List Using Union with Any Type Plus Lax

The following schema fragment could be used to extend a code list using union with any type plus lax so that it contains the new Iraqi currency symbolized by the FRQ code:

ADD NEW SCHEMA FRAGMENT HERE THAT EXTENDS CODE LIST USING UNION WITH ANY TYPE PLUS LAX

3.4.2 Restricting a Code List Using Union with Any Type Plus Lax

The following schema fragment could be used to restrict a code list using union with any type plus lax so that it uses Euros as the only acceptable currency code:

ADD NEW SCHEMA FRAGMENT HERE THAT RESTRICTS CODE LIST USING UNION WITH ANY TYPE PLUS LAX

3.4.3 Issues in applying this method

3.5 AEX 2

The Automating Equipment Information Exchange (AEX) project is sponsored by FIATECH, an industry consortium of facility owners, construction contractors, government agencies and suppliers associated with building construction. The AEX project seeks to develop automated data exchange specifications for use with the design, procurement, installation, maintenance and operation of capital equipment.

3.5.1 Extending a Code List Using AEX 2

The following schema fragment could be used to extend a code list using AEX 2 so that it contains the new Iraqi currency symbolized by the FRQ code:

ADD NEW SCHEMA FRAGMENT HERE THAT EXTENDS CODE LIST USING AEX 2

3.5.2 Restricting a Code List Using AEX 2

The following schema fragment could be used to restrict a code list using AEX 2 so that it uses Euros as the only acceptable currency code:

ADD NEW SCHEMA FRAGMENT HERE THAT RESTRICTS CODE LIST USING AEX 2

3.5.3 Issues in applying this method

4 Impacts

Summary TBD

	substitution groups	redefine	xsi:type	any + lax	AEX 2
Design of code list					
design of schemas using code lists					
NDR					
Instance documents					

4.1 NDR Rules

4.1.1 NDR's that are important to Code List discussion

[ELD9] The `xsd:any` element MUST NOT be used.
[ATD3] If a UBL Schema Expression contains one or more common attributes that apply to all UBL elements contained or included or imported therein, the common attributes MUST be declared as part of a global attribute group.
[ATD6] Each `xsd:schemaLocation` attribute declaration MUST contain a system-resolvable URL, which at the time of release from OASIS shall be a relative URL referencing the location of the schema or schema module in the release package.
[ATD8] The `xsd:anyAttribute` MUST NOT be used.
[CDL1] All UBL Codes MUST be part of a UBL or externally maintained Code List.
[CDL2] The UBL Library SHOULD identify and use external standardized code lists rather than develop its own UBL-native code lists.
[CDL3] The UBL Library MAY design and use an internal code list where an existing external code list needs to be extended, or where no suitable external code list exists.
[CDL4] All UBL maintained or used Code Lists MUST be enumerated using the UBL Code List Schema Module.
[CDL5] The name of each UBL Code List Schema Module MUST be of the form:
`{Owning Organization}{Code List Name}{Code List Schema Module}`
[CDL6] An `xsd:import` element MUST be declared for every code list required in a UBL schema.
[CDL7] Users of the UBL Library MAY identify any subset they wish from an identified code list for their own trading community conformance requirements.
[CDL8] The `xsd:schemaLocation` MUST include the complete URI used to identify the relevant code list schema.
[GXS5] The `xsd:substitutionGroup` feature MUST NOT be used.
[GXS10] The `xsd:include` feature MUST only be used within a document schema.
[GXS11] The `xsd:union` technique MUST NOT be used except for Code Lists. The

`xsd:union` technique MAY be used for Code Lists.
[GXS13] Complex Type extension or restriction MAY be used where appropriate.
[IND1] All UBL instance documents MUST validate to a corresponding schema.

4.1.2 All NDR's

This is a summary of all current NDR rules for reference.

A.1 Attribute Declaration Rules	
[ATD1]	User defined attributes SHOULD NOT be used. When used, user defined attributes MUST only convey `CCT:SupplementaryComponent` information.
[ATD2]	The `CCT:SupplementaryComponents` for the ID `CCT:CoreComponent`MUST be declared in the following order: `Identifier. Content Identification Scheme. Identifier Identification Scheme. Name. Text Identification Scheme. Agency. Identifier Identification Scheme. Agency Name. Text Identification Scheme. Version. Identifier Identification Scheme. Uniform Resource. Identifier Identification Scheme Data. Uniform Resource. Identifier`
[ATD3]	If a UBL Schema Expression contains one or more common attributes that apply to all UBL elements contained or included or imported therein, the common attributes MUST be declared as part of a global attribute group.
[ATD4]	Within the `ccts:CCT xsd:extension` element an `xsd:attribute` MUST be declared for each `ccts:SupplementaryComponent` pertaining to that `ccts:CCT`.
[ATD5]	For each `ccts:CCT simpleType xsd:restriction` element, an `xsd:base` attribute MUST be declared and set to the appropriate `xsd:Datatype`.
[ATD6]	Each `xsd:schemaLocation` attribute declaration MUST contain a system-resolvable URL, which at the time of release from OASIS shall be a relative URL referencing the location of the schema or schema module in the release package.
[ATD7]	The xsd built in nillable attribute MUST NOT be used for any UBL declared element.
[ATD8]	The `xsd:anyAttribute` MUST NOT be used.

A.2 Attribute Naming Rules

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

A.3 Code List Rules

[CDL1]	All UBL Codes MUST be part of a UBL or externally maintained Code List.
[CDL2]	The UBL Library SHOULD identify and use external standardized code lists rather than develop its own UBL-native code lists.
[CDL3]	The UBL Library MAY design and use an internal code list where an existing external code list needs to be extended, or where no suitable external code list exists.
[CDL4]	All UBL maintained or used Code Lists MUST be enumerated using the UBL Code List Schema Module.
[CDL5]	The name of each UBL Code List Schema Module MUST be of the form: `{Owning Organization}{Code List Name}{Code List Schema Module}`
[CDL6]	An `xsd:import` element MUST be declared for every code list required in a UBL schema.

A.3 Code List Rules

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

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

A.4 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 use the `xsd:sequence` element with appropriate global element references, or local element declarations in the case of `ID` and `Code`, to reflect each property of its class as defined in the corresponding UBL model.
[CTD3]	Every `ccts:BBIEProperty xsd:complexType` definition content model MUST use the `xsd:simpleContent` element.
[CTD4]	Every `ccts:BBIEProperty xsd:complexType` content model `xsd:simpleContent` element MUST consist of an `xsd:extension` element.
[CTD5]	Every `ccts:BBIEProperty xsd:complexType` content model `xsd:base` attribute value MUST be the `ccts:CCT` of the unspecialized or specialized UBL datatype as appropriate.
[CTD6]	For every datatype used in the UBL model, a named `xsd:complexType` or `xsd:simpleType` MUST be defined.

A.4 ComplexType Definition Rules

[CTD7]	Every unspecialized Datatype must be based on a `ccts:CCT` represented in the CCT schema module and must represent an approved primary or secondary representation term identified in the CCTS.
[CTD8]	Each unspecialized Datatype `xsd:complexType` must be based on its corresponding `CCT xsd:complexType`.

[CTD9]	Every unspecialized Datatype that represents a primary representation term whose corresponding `ccts:CCT` is defined as an `xsd:simpleType`MUST also be defined as an `xsd:simpleType` and MUST be based on the same `xsd:simpleType`.
[CTD10]	Every unspecialized Datatype that represents a secondary representation term whose corresponding `ccts:CCT` is defined as an `xsd:simpleType` MUST also be defined as an `xsd:simpleType` and MUST be based on the same `xsd:simpleType`.
[CTD11]	Each unspecialized Datatype `xsd:complexType` definition must contain one `xsd:simpleContent` element.
[CTD12]	The unspecialized Primary Representation Term Datatype `xsd:complextType` definition `xsd:simpleContent` element must contain one `xsd:restriction` element with an `xsd:base` attribute whose value is equal to the corresponding `cct:ComplexType`.
[CTD13]	For every `ccts:CCT` whose supplementary components are not equivalent to the properties of a built-in `xsd:Datatype`, the `ccts:CCT` MUST be defined as a named `xsd:complexType` in the `ccts:CCT` schema module.
[CTD14]	Each `ccts:CCT xsd:complexType` definition MUST contain one `xsd:simpleContent` element
[CTD15]	The `ccts:CCT xsd:complexType` definition `xsd:simpleContent` element MUST contain one `xsd:extension` element. This `xsd:extension` element MUST include an `xsd:base` attribute that defines the specific `xsd:Built-inDatatype` required for the `ccts:ContentComponent` of the `ccts:CCT`.

1020

A.4 ComplexType Definition Rules

[CTD16]	Each `CCT:SupplementaryComponent xsd:attribute` "type" MUST define the specific `xsd:Built-inDatatype` or the user defined `xsd:simpleType` for the `ccts:SupplementaryComponent` of the `ccts:CCT`.

[CTD17]	Each `ccts:SupplementaryComponent xsd:attribute` user-defined `xsd:simpleType` MUST only be used when the `ccts:SupplementaryComponent` is based on a standardized code list for which a UBL conformant code list schema module has been created.
[CTD18]	Each `ccts:SupplementaryComponent xsd:attribute` user defined `xsd:simpleType` MUST be the same `xsd:simpleType` from the appropriate UBL conformant code list schema module for that type.
[CTD19]	Each `ccts:Supplementary Component xsd:attribute` "`use`" MUST define the occurrence of that `ccts:SupplementaryComponent`as either "`required`", or "`optional`".

1021

A.5 ComplexType Naming Rules

[CTN1]	A UBL `xsd:complexType` name based on an `ccts:AggregateBusinessInformationEntity` MUST be the `ccts:DictionaryEntryName` with the separators removed and with the "`Details`" suffix replaced with "`Type`".
[CTN2]	A UBL `xsd:complexType` name based on a ccts:BasicBusinessInformationEntityProperty MUST be the `ccts:DictionaryEntryName` shared property term and its qualifiers and the representation term of the shared `ccts:BasicBusinessInformationEntity`, with the separators removed and with the "`Type`" suffix appended after the representation term.

1022

A.5 ComplexType Naming Rules

[CTN3]	A UBL `xsd:complexType` for a `cct:UnspecializedDatatype` used in the UBL model MUST have the name of the corresponding `ccts:CoreComponentType`, with the separators removed and with the "`Type`" suffix appended.
[CTN4]	A UBL `xsd:complexType` for a `cct:UnspecializedDatatype` based on a `ccts:SecondaryRepresentationTerm` used in the UBL model MUST have the name of the corresponding `ccts:SecondaryRepresentationTerm`, with the separators removed and with the "`Type`" suffix appended.

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

.

A.6 Documentation Rules

[DOC1]	The `xsd:documentation` element for every Datatype MUST contain a structured set of annotations in the following sequence and pattern: . • ComponentType (mandatory): The type of component to which the object belongs. For Datatypes this must be “DT”. . • DictionaryEntryName (mandatory): The official name of a Datatype. . • Version (optional): An indication of the evolution over time of the Datatype. . • Definition(mandatory): The semantic meaning of a Datatype. . • ObjectClassQualifier (optional): The qualifier for the object class. . • ObjectClass(optional): The Object Class represented by the Datatype. . • RepresentationTerm (mandatory): A Representation Term is an element of the name which describes the form in which the property is represented. . • DataTypeQualifier (optional): semantically meaningful name that differentiates the Datatype from its underlying Core Component Type. • DataType (optional): Defines the underlying Core Component Type.
[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 structured set of annotations in the following patterns: • RestrictionType (mandatory): Defines the type of format restriction that applies to the Content Component. • RestrictionValue (mandatory): The actual value of the format restriction that applies to the Content Component. • ExpressionType (optional): Defines the type of the regular expression of the restriction value.
[DOC3]	A Datatype definition MAY contain one or more Supplementary Component Restrictions to provide additional information on the relationship between the Datatype and its corresponding Core Component Type. If used the Supplementary Component Restrictions must contain a structured set of annotations in the following patterns: • SupplementaryComponentName (mandatory): Identifies the Supplementary Component on which the restriction applies. • RestrictionValue (mandatory, repetitive): The actual value(s) that is (are) valid for the Supplementary Component
[DOC4]	The xsd:documentation element for every Basic Business Information Entity

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

A.6 Documentation Rules

.

.

A.6 Documentation Rules

[DOC5]	The `xsd:documentation` element for every Aggregate Business Information Entity MUST contain a structured set of annotations in the following sequence and pattern: . • ComponentType (mandatory): The type of component to which the object belongs. For Aggregate Business Information Entities this must be “ABIE”. . • DictionaryEntryName (mandatory): The official name of the Aggregate Business Information Entity . . • Version (optional): An indication of the evolution over time of the Aggregate Business Information Entity. . • Definition(mandatory): The semantic meaning of the Aggregate Business Information Entity. . • ObjectClassQualifier (optional): The qualifier for the object class. . • ObjectClass(mandatory): The Object Class represented by the Aggregate Business Information Entity. • AlternativeBusinessTerms (optional): Any synonym terms under which the Aggregate Business Information Entity is commonly known and used in the business.
[DOC6]	The `xsd:documentation` element for every Association Business Information Entity element declaration MUST contain a structured set of annotations in the following sequence and pattern: . • ComponentType (mandatory): The type of component to which the object belongs. For Association Business Information Entities this must be “ASBIE”. . • DictionaryEntryName (mandatory): The official name of the Association Business Information Entity. . • Version (optional): An indication of the evolution over time of the Association Business Information Entity. . • Definition(mandatory): The semantic meaning of the Association Business Information Entity. . • Cardinality(mandatory): Indication whether the Association Business Information Entity represents an optional, mandatory and/or repetitive assocation. . • ObjectClass(mandatory): The Object Class containing the Association Business Information Entity. . • PropertyTermQualifier (optional): A qualifier is a word or words which help define and differentiate the Association Business Information Entity. . • PropertyTerm(mandatory): Property Term represents the Aggregate Business Information Entity contained by the Association Business Information Entity. . • AssociatedObjectClassQualifier (optional): Associated Object Class Qualifiers describe the 'context' of the relationship with another ABIE. That is, it is the role the contained Aggregate Business Information Entity plays within its association with the containing Aggregate Business Information Entity. • AssociatedObjectClass (mandatory); Associated Object Class is the

	Object Class at the other end of this association. It represents the Aggregate Business Information Entity contained by the Association Business Information Entity.
[DOC7]	The `xsd:documentation` element for every Core Component Type MUST contain a structured set of annotations in the following sequence and pattern: . • ComponentType (mandatory): The type of component to which the object belongs. For Core Component Types this must be “CCT”. . • DictionaryEntryName (mandatory): The official name of the Core Component Type, as defined by [CCTS]. . • Version (optional): An indication of the evolution over time of the Core Component Type. . • Definition(mandatory): The semantic meaning of the Core Component Type, as defined by [CCTS]. . • ObjectClass(mandatory): The Object Class represented by the Core Component Type, as defined by [CCTS]. • PropertyTerm(mandatory): The Property Term represented by the Core Component Type, as defined by [CCTS].

.

A.7 Element Declaration Rules

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

A.7 Element Declaration Rules

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

1036

A.8 Element Naming Rules

[ELN1]	A UBL global element name based on a `ccts:ABIE` MUST be the same as the name of the corresponding `xsd:complexType` to which it is bound, with the word "`Type`" removed.
[ELN2]	A UBL global element name based on an unqualified `ccts:BBIEProperty`MUST be the same as the name of the corresponding `xsd:complexType` to which it is bound, with the word "`Type`" removed.

1037

A.8 Element Naming Rules

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

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

A.9 General Naming Rules

[GNR1]	UBL XML element, attribute and type names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
[GNR2]	UBL XML element, attribute and type names MUST be consistently derived from CCTS conformant dictionary entry names.
[GNR3]	UBL XML element, attribute and type names constructed from `ccts:DictionaryEntryNames` MUST NOT include periods, spaces, other separators, or characters not allowed by W3C XML 1.0 for XML names.
[GNR4]	UBL XML element, attribute, and simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions published in Appendix B.
[GNR5]	Acronyms and abbreviations MUST only be added to the UBL approved acronym and abbreviation list after careful consideration for maximum understanding and reuse.
[GNR6]	The acronyms and abbreviations listed in Appendix B MUST always be used.

[GNR7]	UBL XML element, attribute and type names MUST be in singular form unless the concept itself is plural.
[GNR8]	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.
[GNR9]	The lowerCamelCase (LCC) convention MUST be used for naming attributes.

A.10 General Type Definition Rules

[GTD1]	All types MUST be named.
[GTD2]	The `xsd:anyType` MUST NOT be used.

A.11 General XML Schema Rules

[GXS1]

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

. • XML Declaration
. • <!-- ===== Copyright Notice ===== -->
. • “Copyright © 2001-2004 The Organization for the Advancement of Structured Information Standards (OASIS). All rights reserved.
. • <!-- ===== xsd:schema Element With Namespaces Declarations ===== -->
. • xsd:schema element to include version attribute and namespace declarations in the following order:
. • xmlns:xsd
. • Target namespace
. • Default namespace

A.11 General XML Schema Rules

. • CommonAggregateComponents
. • CommonBasicComponents
. • CoreComponentTypes
. • Datatypes
. • Identifier Schemes
. • Code Lists
. • Attribute Declarations – elementFormDefault=”qualified” attributeFormDefault=”unqualified”
. • <!-- ===== Imports ===== -->CommonAggregateComponents schema module
. • CommonBasicComponents schema module
. • Representation Term schema module (to include CCT module)
. • Unspecialized Types schema module
. • Specialized Types schema module
. • <!-- ===== Global Attributes ===== -->
. • Global Attributes and Attribute Groups
. • <!-- ===== Root Element ===== -->
. • Root Element Declaration
. • Root Element Type Definition
. • <!-- ===== Element Declarations ===== -->
. • alphabetized order

. • <!-- ===== Type Definitions ===== -->

A.11 General XML Schema Rules

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

A.11 General XML Schema Rules

[GXS10]	The `xsd:include` feature MUST only be used within a document schema.
[GXS11]	The `xsd:union` technique MUST NOT be used except for Code Lists. The `xsd:union` technique MAY be used for Code Lists.

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

A.12 Instance Document Rules

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

A.13 Modeling Constraints Rules

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

A.14 Naming Constraints Rules

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

A.15 Namespace Rules

[NMS1]	Every UBL-defined or -used schema module MUST have a namespace declared using the `xsd:targetNamespace` attribute.
[NMS2]	Every UBL defined or used schema set version MUST have its own unique namespace.
[NMS3]	UBL namespaces MUST only contain UBL developed schema modules.
[NMS4]	The namespace names for UBL Schemas holding committee draft status MUST be of the form: `urn:oasis:names:tc:ubl:schema:<subtype>:<document-id>`
[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>`
[NMS6]	UBL published namespaces MUST never be changed.
[NMS7]	The `ubl:CommonAggregateComponents` schema module MUST reside in its own namespace.
[NMS8]	The `ubl:CommonAggregateComponents` schema module MUST be represented by the token "`cac`".
[NMS9]	The `ubl:CommonBasicComponents` schema module MUST reside in its own namespace.

[NMS10]	The `UBL:CommonBasicComponents` schema module MUST be represented by the token "`cbc`".
[NMS11]	The `ccts:CoreComponentType` schema module MUST reside in its own namespace.
[NMS12]	The `ccts:CoreComponentType` schema module namespace MUST be represented by the token "`cct`".
[NMS13]	The `ccts:UnspecializedDatatype` schema module MUST reside in its own namespace.
[NMS14]	The `ccts:UnspecializedDatatype` schema module namespace MUST be represented by the token "`udt`".
[NMS15]	The `ubl:SpecializedDatatypes` schema module MUST reside in its own namespace.
[NMS16]	The `ubl:SpecializedDatatypes` schema module namespace MUST be represented by the token "`sdt`".
[NMS17]	Each `UBL:CodeList` schema module MUST be maintained in a separate namespace.

A.16 Root Element Declaration Rules

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

A.17 Schema Structure Modularity Rules

[SSM1]	UBL Schema expressions MAY be split into multiple schema modules.
[SSM2]	A document schema in one UBL namespace that is dependent upon type definitions or element declarations defined in another namespace MUST only import the document schema from that namespace.

[SSM3]	A UBL document schema in one UBL namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import internal schema modules from that namespace.
[SSM4]	Imported schema modules MUST be fully conformant with UBL naming and design rules.
[SSM5]	UBL schema modules MUST either be treated as external schema modules or as internal schema modules of the document schema.
[SSM6]	All UBL internal schema modules MUST be in the same namespace as their corresponding document schema.
[SSM7]	Each UBL internal schema module MUST be named `{ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}`
[SSM8]	A UBL schema module MAY be created for reusable components.
[SSM9]	A schema module defining all `ubl:CommonAggregateComponents` MUST be created.

1091

A.17 Schema Structure Modularity Rules

[SSM10]	The `ubl:CommonAggregateComponents` schema module MUST be named `"ubl:CommonAggregateComponents Schema Module"`
[SSM11]	A schema module defining all `ubl:CommonBasicComponents` MUST be created.
[SSM12]	The `ubl:CommonBasicComponents` schema module MUST be named `"ubl:CommonBasicComponents Schema Module"`
[SSM13]	A schema module defining all `ccts:CoreComponentTypes` MUST be created.
[SSM14]	The `ccts:CoreComponentType` schema module MUST be named `"ccts:CoreComponentType Schema Module"`

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

A.18 Standards Adherence rules

[STA1] All UBL schema design rules MUST be based on the W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[STA2] All UBL schema and messages MUST be based on the W3C suite of technical specifications holding recommendation status.

A.19 SimpleType Naming Rules

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

2142

A.20 SimpleType Definition Rules

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

A.21 Versioning Rules

[VER1]	Every UBL Schema and schema module major version committee draft MUST have an RFC 3121 document-id of the form `<name>-<major>.0[.<revision>]`
[VER2]	Every UBL Schema and schema module major version OASIS Standard MUST have an RFC 3121 document-id of the form `<name>-<major>.0`
[VER3]	Every minor version release of a UBL schema or schema module draft MUST have an RFC 3121 document-id of the form `<name>-<major >.<non-zero>[.<revision>]`

A.21 Versioning Rules

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

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

5 Samples

6 References

[3166-XSD] UN/ECE XSD code list module for ISO 3166-1,

[CCTS2.01] *UN/CEFACT Core Components Technical Specification – Part 8 of the ebXML Framework*, 15 November 2003, Version 2.01.

[CLSC] OASIS UBL Code List Subcommittee. Portal: http://www.oasis-open.org/committees/sc_home.php?wg_abbrev=ubl-clsc . Email archive: http://lists.oasis-open.org/archives/ubl-clsc/.

[SPENCER] http://www.oasis-open.org/apps/org/workgroup/ubl-clsc/download.php/5195/Spencer-CodeList-PositionPaper1-0.pdf

[STUHEC] <need reference>

[COATES] **http://www.oasis-open.org/apps/org/workgroup/ubl-clsc/download.php/4522/draft-coates-codeListDataModels-0p2.doc**

[CLTemplate] OASIS UBL Naming and Design Rules code list module template, http://www.oasis-open.org/committees/ubl/ndrsc/archive/.

[eBSC] "eBusiness Standards Convergence Forum", http://www.nist.gov/ebsc.

[eBSCMemo] M. Burns, S. Damodaran, F.Yang, "Draft Code List Implementation description", http://www.oasis-open.org/apps/org/workgroup/ubl-clsc/download.php/4503/nistTOUbl20031119.zip

[NDR] M. Cournane et al., *Universal Business Language (UBL) Naming and Design Rules*, OASIS, 15 November 2004, http://docs.oasis-open.org//ubl/cd-UBL-NDR-1.0.1.

[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.

[CL5] **http://www.oasis-open.org/apps/org/workgroup/ubl-clsc/download.php/4502/wd-ublndrsc-codelist-05_las_20030702.doc**

[ISO 11179] http://www.iso.org/iso/en/StandardsQueryFormHandler.StandardsQueryFormHandler?scope=CATALOGUE&keyword=&isoNumber=11179

[UBL1-SD] http://ibiblio.org/bosak/ubl/UBL-1.0/art/UBL-1.0-SchemaDependency.jpg

[UNTDED 3055] <need reference>

[XSD] *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May 2001. http://www.unece.org/etrades/unedocs/repository/codelist.htm.

[CLR] Editor M. Burns, *Universal Business Language (UBL) Code List Representation*, Version: 1.1 draft 5 April 2005

Appendix A. Revision History

Revision	Editor	Description
2004-01-13	Marty Burns	First complete version converted from NDR revision 05
2004-01-14	Marty Burns	Minor edit of chapter heading 3 & 4
2004-01-20	Marty Burns	Incorporated descriptions from AS and KH
2004-02-06	Marty Burns	Cleaned up requirements and other sections – removed some redundant content from merge of contributions. Explicitly identified Data Model and Metadata models separately from XML representations of the same.
2004-02-11	Marty Burns	Added comments from 2/11 conference call
2004-02-29	Marty Burns	Added resolutions from February Face to Face meeting
2004-03-03	Marty Burns	Incorporated Tim McGrath's corrections of data model
2004-03-09	Marty Burns	Addressed Eve Maler's comments Addressed Tony Coates comments Addressed 2004-03-03 telecon comments Added some elaboration of the model usage in ubl
2004-03-15	Marty Burns	Added example mapping schema paper to section 4.6
2004-03-23	Marty Burns	Added data model for supplementary components, Marked future features for UBL 1.1 as (future) Added comment about UBL1.0 release vs. future.
2004-04-01	Marty Burns	Clean up for UBL version 1.0
2004-04-14	Marty Burns	Incorporated suggested edits from GKH
2005-01-02	Marty Burns	Incorporated elaborations of requirements for better clarity to kick off the UBL 1.1 revisions. Incorporated comments from Tony Coates.

Appendix B. Notices