UBL Code List Value Validation Methodology

Working Draft 0.6, 18 June 2006 17:40z - Draft 1

Document identifier:

UBL-codelist-methodology-0.6

Locations:

Persistent version: TBD
Current version: http://www.oasis-open.org/committees/download.php/18809/UBL-codelist-methodology-0.6-D1-20060618-1740z.zip
Previous version: http://www.oasis-open.org/committees/download.php/16765/UBL-codelist-methodology-0.4.zip

Technical committee:

OASIS Universal Business Language (UBL) TC

Chairs:

Jon Bosak, Sun Microsystems
Tim McGrath <tmcgrath@portcomm.com.au>

Author:

G. Ken Holman, Crane Softwrights Ltd.

Abstract:

This Working Draft describes a methodology and supporting document types with which trading partners can agree unambiguously on the sets of coded values against which exchanged documents must validate. The illustration context of this specification is the Universal Business Language 2.0 however the methodology is presented in such a way as to apply to any context. This illustration uses genericode files for the external representation of coded values, however the methodology is presented in such a way that any representation of coded values can be used. This methodology is packaged with document models, functional stylesheets and demonstration test files for both Windows and Linux environments that can be executed to demonstrate the behaviors.

Status:

This is a work in progress and does not at this time represent the consensus of the UBL Technical Committee.

Please send comments on this specification to the <ubl-dev@lists.oasis-open.org> list. To subscribe, send an email message to <ubl-dev-request@lists.oasis-open.org> with the word "subscribe" as the body of the message.

Notices:

Table of Contents

Appendix

1. Introduction

This Working Draft describes a methodology and supporting document types with which trading partners can agree unambiguously on the sets of coded values in a controlled vocabulary against which exchanged documents must validate.

Schemata describe the structural and lexical constraints on a document. Some information items in a schema are described lexically as a token value whereby the token is a coded value representing an agreed-upon semantic concept. The value used is typically chosen from a set of unique coded values enumerating related concepts. These sets of coded values are sometimes termed *code list*s.

For some commonly-understood concepts, publicly-available enumerations of coded values are published and maintained by authorities regarded as the custodians of the set of defined values and their associated concepts. A schema may constrain a value to be one of the entire set of published values so as to ensure the value used represents the published concept. A common example of a publicly-maintained code list is that which enumerates currency value indications [currency] and is used for illustrative purposes in this document.

The use of coded values may also be specified where the coded values themselves are merely agreed upon amongst users but not formally enumerated as a constraint on the definition of a document type. A schema thus constrains the document instance value to be a coded value, but the coded value itself is unconstrained. An example from the Universal Business Language 2.0 [UBL 2.0] is `cbc:CountrySubentityCode` that constrains the indication of jurisdictional or administrative boundaries below the country level, such as provinces (as in Canada) and states (as in the United States). UBL schemata do not constrain the coded values used for information items representing this concept.

Trading partners may agree to use the published UBL schemata for constraining the documents exchanged for electronic commerce, but may find the constraints of some code lists therein too loose. For two examples, the schema-expressed enumerated list of currency indications may contain many more items that the parties are willing to use, and the lack of an enumeration of country sub-entity codes might allow nonsensical or undesired values to be used. Thus, the UBL schemata successfully validate an exchanged document against the standardized constraints, but allow information to be represented that is not agreed upon by the parties. Trading partners might, then, wish to constrain the currency indications and country sub-entity indications used in the exchanged documents.

Furthermore, trading partners may wish to agree that different sets of values from the same code lists be allowed at multiple locations within a single document (perhaps allowing the state for the buyer in an order be from a different set of states than that allowed for the seller). Large or published schemata might not be able to accommodate such differentiation very elegantly or robustly, or possibly could not be able to express such varied constraints due to limitations of the schema language's modeling semantics. Moreover it is not necessarily the role of the creators of schemata to accommodate such differentiation mandated by the use of their work products.

Having a methodology and supporting document types with which to perform code list value validation enables parties involved in document exchange to formally describe the sets of coded values that are to be used and the document contexts in which those sets are to be used. Such a formal and unambiguous description can then become part of a trading partner contractual agreement, supported by processes to ensure the agreement is not being breached by a given document instance.

Note

This is *not* the standard for code list schema representation (as in UBL Naming and Design Rules (NDR)), nor is it the standard for external code list coded value enumeration representation (as in genericode [genericode] files), but rather it is only the methodology for value validation given that you have some instances being validated by a schema with agreed-upon values represented in supplemental files.

Both the ZIP [UBL-codelist-methodology-0.6.zip] and TAR/GZ [UBL-codelist-methodology-0.6.tar.gz] compressed packages of the documentation for this methodology each include the stylesheet, data and test files that are referenced in the prose.

2. Terminology

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this Working Draft are to be interpreted as described in [RFC Keywords]. Note that for reasons of style, these words are not capitalized in this document.

3. Code list definitions inside schemata

Some of the constraints expressed in a schema are used to control the vocabulary used for an information item by specifying the lexical and value limitations used in an XML instance. Such constraints may enumerate all of the possible coded values in a code list. Alternatively, the constraints may merely limit the value lexically to an expected pattern without limiting the actual values used that match that pattern (for example, a token string of characters without any embedded white space).

How code list constraints are expressed in a schema impacts on the flexibility of trading partners to use subsets, supersets or simultaneously use different sets of coded values for a given code list in a given XML instance that needs to be validated by the schema.

3.1. Code list constraints with enumerations

Schema expressions constraining the values of code list coded values often use `xsd:enumeration` elements restricting a base data type of either a normalized string or a tokenized value. The UBL declaration for currency coded values imported from UN/CEFACT uses such an approach as illustrated by this incomplete fragment:

```
<xsd:simpleType name="CurrencyCodeContentType">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="AED">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:CodeName>Dirham</ccts:CodeName>
          <ccts:CodeDescription></ccts:CodeDescription>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="AFN">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:CodeName>Afghani</ccts:CodeName>
          <ccts:CodeDescription></ccts:CodeDescription>
        </xsd:documentation>
      </xsd:annotation>
    </xsd:enumeration>
    <xsd:enumeration value="ALL">
      <xsd:annotation>
        <xsd:documentation>
          <ccts:CodeName>Lek</ccts:CodeName>
          <ccts:CodeDescription></ccts:CodeDescription>
```

```
          </xsd:documentation>
        </xsd:annotation>
      </xsd:enumeration>
      <xsd:enumeration value="AMD">
      ...
```

Trading partners may wish to contractually constrain the set of coded values for these information items to only those values that are allowed in instances being exchanged. This is easily accommodated in the information exchange since any coded value used from a strict subset is, by definition, a coded value in the full enumeration. Instances with one of the limited values can be validated by the unchanged schema for the document model with all of the values.

Note that while techniques are available in some schema expression languages for restricting a data type definition of an enumeration to a subset of values, such a restriction typically has global scope across the instance. Trading partners may wish to constrain the values to different sets in different contexts of the document, which is not possible by way of available restriction techniques in some schema language expression semantics.

Trading partners wishing to extended the set of coded values for these information items are unable to do so when code list constraints in a schema are expressed with an enumeration. Adding a new value to the list changes the definition of the list such that instances with new values cannot be validated by the unchanged schema for the document model.

3.2. Code list constraints without enumerations

Some code lists are declared in schemata without an enumeration, as there may be far too many possible coded values to be manageable, the sets of coded values may differ in different contexts of a single document, or the coded values are not predefined in any way. Schema expressions constraining such code list coded values often merely constrain the value to a normalized string or a tokenized value. This satisfies the lexical requirements of the coded value without constraining the particular values that meet the requirements.

A UBL example of a code list with an unmanageable number of enumerations is `cbc:CountrySubentityCode` as there are so very many provinces, states, regions, prefectures and other administrative or jurisdictional areas in the world that would be included to be complete. UBL does not supply any sets of values to use.

A UBL example of a code list with predefined values that trading partners may wish to extend or restrict is `DocumentStatusCodeType` where the UBL committee has chosen a set of meaningful values for a certain class of workflow definitions, but trading partners may have a richer set employed in their respective systems. This predefined list, among other predefined lists, is supplied in UBL using an external code list expression conforming to this methodology.

An example of a code list without any predefined values is `cac:AccountTypeCode` as there may be as many account types as there are trading parties, and none are predefined by UBL. Trading partners wishing to validate information items using coded values from this code list are obliged to agree on and express the set of values they expect to use. UBL does not supply any sets of values to use.

UBL has two generic declarations of code lists for these examples based on whether the coded value is an identifier or a code. These declarations are illustrated by this incomplete fragment:

```
<xsd:complexType name="CodeType">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         ...
      </xsd:documentation>
   </xsd:annotation>
   <xsd:simpleContent>
      <xsd:extension base="xsd:normalizedString">
         <xsd:attribute name="listID" type="xsd:normalizedString"
```

```
                        use="optional"/>
            <xsd:attribute name="listAgencyID" type="xsd:normalizedString"
                        use="optional"/>
            <xsd:attribute name="listAgencyName" type="xsd:string"
                        use="optional"/>
            <xsd:attribute name="listName" type="xsd:string" use="optional"/>
            <xsd:attribute name="listVersionID" type="xsd:normalizedString"
                        use="optional"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
            <xsd:attribute name="languageID" type="xsd:language"
                        use="optional"/>
            <xsd:attribute name="listURI" type="xsd:anyURI" use="optional"/>
            <xsd:attribute name="listSchemeURI" type="xsd:anyURI"
                        use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   ...
   <xsd:complexType name="IdentifierType">
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
            ...
         </xsd:documentation>
      </xsd:annotation>
   <xsd:simpleContent>
      <xsd:extension base="xsd:normalizedString">
        <xsd:attribute name="schemeID" type="xsd:normalizedString"
                       use="optional"/>
        <xsd:attribute name="schemeName" type="xsd:string" use="optional"/>
        <xsd:attribute name="schemeAgencyID" type="xsd:normalizedString"
                       use="optional"/>
        <xsd:attribute name="schemeAgencyName" type="xsd:string"
                       use="optional"/>
        <xsd:attribute name="schemeVersionID" type="xsd:normalizedString"
                       use="optional"/>
        <xsd:attribute name="schemeDataURI" type="xsd:anyURI"
                       use="optional"/>
        <xsd:attribute name="schemeURI" type="xsd:anyURI" use="optional"/>
      </xsd:extension>
   </xsd:simpleContent>
</xsd:complexType>
```

Note in the above how the optional metadata attributes do not have fixed attributes, thus allowing the XML instance to indicate not only the coded value but also the metadata related to the code list from which the value is taken. This approach allows XML instances validated with the schema to have any coded value for these information items.

3.3. UBL schemata constraints on code lists

UBL includes both kinds of constraints for coded values in code lists, very few of which are declared with enumerations. The `CurrencyCodeContentType` data type is an example whose coded values are internationally standardized [currency]. The only other two such code lists in UBL enumerate MIME encoding identifiers and unit code values. All three enumerations are imported from UN/CEFACT-standardized sets of coded values.

All other code lists in UBL are not enumerated in the schema expressions. Such code lists can be very large and would be awkward to accommodate in a schema expression. Some code lists have a set of predefined coded values supplied by UBL that trading partners may wish to extend. Other codes lists can only have coded values provided by the users, as standardization either hasn't happened or shouldn't happen because it isn't appropriate to set out what trading partners are allowed to use.

The UBL 2.0 support package [UBL 2.0 Support] includes genericode files for every code list. All the enumerated code lists have complete sets of coded values distilled from the schema expressions, a limited number of code lists have the sets of coded values predefined by the UBL committee, and all other code lists have an empty set of coded values. Trading partners can modify or replace any of the genericode files to meet their business requirements using the code list value validation methodology in this Working Draft.

4. Code list definitions outside schemata

At this time of writing there are no standardized approaches to formally publishing the enumeration of coded value members of a code list that is published by a code list maintainer. Such a formal representation is required for the purposes of machine processing suitable to methodologies supported by automated processes.

The genericode [genericode] approach is a de facto implementation of such a formal expression cataloguing the members of an enumeration with associated member and list documentation and list metadata description. This approach is used in the illustration of this value validation methodology.

It is with genericode files that this methodology's included stylesheets support trading partners expressing the coded values agreed to being used in contexts. Other expressions of coded values in a code list can be accommodated by one of at least the two following methods:

- rewrite this methodology's implementation's stylesheet fragment accessing genericode files with an equivalent fragment that accesses the alternative format;
- transliterating instances of the alternative format to be minimal instances of the genericode format and using this methodology's implementation's existing stylesheet fragment.

4.1. Trading partner genericode definitions

Trading partners will need to replace the genericode definitions for those code lists they wish to restrict or, if allowed, extend. Their flexibility in modifying the code list is based on whether the code list is described in the schema with or without an enumeration.

When limiting the coded values from a list described in the schema with an enumeration, the unedited genericode file is initialized to include all enumerated values. Trading partners can then work from the complete list and prune unwanted values in a copy leaving in the values agreed to be used in XML instances.

An example of this is a genericode file based on the UN/CEFACT currency values that would have over 160 entries. A copy of this file named `scenario/MyCurrencyLimits.gc` is edited where the entire list of coded values has been pruned to only the Canadian dollar and the US dollar. An excerpt from that file reads as:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/" ...
   <Identification>
      <ShortName>CurrencyCode</ShortName>
      <LongName>ISO 4217 Alpha</LongName>
      ...
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
```

```
            <SimpleValue>CAD</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>Canadian Dollar</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>USD</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>US Dollar</SimpleValue>
         </Value>
      </Row>
   </SimpleCodeList>
</gc:CodeList>
```

When specifying the coded values for a list described in the schema without an enumeration, the unedited genericode file may either have some predefined coded values initialized through system design decisions, or may be void of any coded values whatsoever. Trading partners can modify a copy of the genericode file or synthesize a new file from scratch and include the values agreed to be used.

An excerpt from the `scenario/MyCanadianProvinces.gc` file created by hand reads as:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/" ...
   <Identification>
      <ShortName>provinces</ShortName>
      <LongName>Canadian Provinces</LongName>
      <Version>2</Version>
      ...
   </Identification>
   ...
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AB</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>Alberta</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>BC</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>British Columbia</SimpleValue>
         </Value>
      </Row>
      ...
```

An excerpt from the `scenario/MyUSStates.gc` file created by hand reads as:

```
<gc:CodeList xmlns:gc="http://genericode.org/2006/ns/CodeList/0.4/" ...
   <Identification>
```

```
      <ShortName>states</ShortName>
      <LongName>US States</LongName>
      <Version>1</Version>
      ...
   </Identification>
   ...
   <SimpleCodeList>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AL</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>ALABAMA</SimpleValue>
         </Value>
      </Row>
      <Row>
         <Value ColumnRef="code">
            <SimpleValue>AK</SimpleValue>
         </Value>
         <Value ColumnRef="name">
            <SimpleValue>ALASKA</SimpleValue>
         </Value>
      </Row>
      ...
```

Of course genericode files need not be created by hand and could be synthesized as the result of a database query or some other process to create the XML expression of the coded values.

4.2. Normative UBL genericode specification and use

To satisfy the normative requirements for references in UBL, the UBL set of deliverables includes a snapshot of the genericode specification.

The UBL support package includes a complete set of genericode files conforming to this snapshot specification, with which trading partners can tailor their needs. Each of the more than 80 code lists in UBL schemata are accommodated by a genericode file structured as one of the following:

- a complete set of predefined coded values specified by UN/CEFACT and matching the enumeration of values in the UBL schemata; these sets of values can be restricted by trading partners to only a subset of the predefined values required for validation;
- a complete set of predefined coded values specified by the UBL Technical Committee and reflecting system design properties of UBL as a whole; these sets of values can be restricted or extended by trading partners to be any set required for validation;
- an empty set with no coded values; these sets can be extended by trading partners to be any set required for validation.

5. Document contexts in XML instances

The different types of information items that are described by code lists are typically declared in few places in document models but, because of document context, the actual instances of these information items are found in possibly very many places in actual instances. Document grammars that validate information items based solely on their declarations cannot distinguish the different uses of the items and any desired differences in value validation required by trading partners exchanging XML instances. Each use of an information item is in a different document context.

Document contexts are expressed structurally as hierarchical tree locations. Without confidence that the document contexts of the information items of an XML instance are sound, no amount of contextual checking of item values is going to be reliable. It is a necessary precondition in advance of using this code list value validation methodology to validate the XML instances against a schema expression of structural constraints. This is a critically important step because the schema constraints will confirm the document contexts of information items are correctly positioned in the XML instance document hierarchy. Only when the information items are known to be in correct contexts will the value checking of the document contexts reflect bona fide results.

In UBL the XML document constraints are expressed using the W3C Schema [W3C Schema] language. This suffices to be a structural schema for all of the information items, and in addition describes the enumerations for the UN/CEFACT-based code lists.

5.1. Using XPath to specify document context

The XML Path Language 1.0 [XPath 1.0] is used to address locations in an XML document according to a data model of processed syntax. This XPath data model differs from other data models such as the Document Object Model [DOM] in that the DOM models more aspects of raw syntax used in the document. Given that syntax is irrelevant (in that it is arbitrary to the creator of XML which syntactic choices are made when marking up documents) the XPath data model is sufficient to talk about the elements and attributes found in documents.

Elements are referred to in an XPath expression by their namespace-qualified names, while the "@" character (an abbreviation for the XPath `attribute::` axis) prefixes attributes referred to by their namespace-qualified names.

Note

XPath 1.0 considers names without prefixes to always be in no namespace, and never uses the default namespace to qualify names without prefixes. For this reason, all namespace-qualified information items in an XML vocabulary being validated must be prefixed when being addressed in XPath 1.0, even if the instances of this vocabulary utilize the default namespace.

The syntax of an XPath expression separates multiple location steps of a single location path using an oblique "`/`" character. Each step to the right names the child element or attached attribute of the immediately preceding step to the left which is always an element. Child elements are one level deeper in the XML hierarchical nesting than their parents. Elements are also parents of their attached attributes.

A fully-qualified absolute XPath location path begins with the oblique indicating the path starts from the root node (the parent of the document element) of the XPath data model document tree. A relative XPath location path starts with the name of an information item without the oblique at the beginning.

Examples of absolute XPath location paths possible for an instance of UBL Order are:

```
/po:Order/cac:TaxTotal/cbc:TaxAmount/@currencyID
/po:Order/cbc:DocumentCurrencyCode
/po:Order/cac:BuyerCustomerParty/cac:Party/cac:Address/cbc:CountrySubentityCode
/po:Order/cac:SellerSupplierParty/cac:Party/cac:Address/cbc:CountrySubentityCode
```

An example of a relative XPath location that matches all currency coded values in attributes in the entire instance of any UBL document model is as follows, as the information item does not include any ancestral distinction to the left:

```
@currencyID
```

An example of a relative XPath location that matches all country sub-entity coded values in elements in the entire instance of any UBL document model is as follows, as the information item does not include any ancestral distinction to the left:

```
cbc:CountrySubentityCode
```

The minimum XPath addresses needed to precisely distinguish the country sub-entity code of the party address of each of the buyer and seller are as follows, as the information item includes explicit ancestry to the left:

```
cac:BuyerCustomerParty/cac:Party/cac:Address/cbc:CountrySubentityCode
cac:SellerSupplierParty/cac:Party/cac:Address/cbc:CountrySubentityCode
```

Note the use of the "//" operator in XPath allows the matching within an entire sub-tree of the hierarchy; the XPath addresses needed to distinguish all (not just in the party address) country sub-entity codes descendent to the buyer and the seller would be as follows indicating only the required (and possibly distant) ancestor:

```
cac:BuyerCustomerParty//cbc:CountrySubentityCode
cac:SellerSupplierParty//cbc:CountrySubentityCode
```

5.2. Trading partner uses of document contexts

When deciding on code list value validation, trading partners must agree in which contexts particular sets of values need to be constrained.

Some business rules may require the same context to be specified across all document types, such as "All currency values must be Canadian or US dollars."

Other business rules may require indistinct document contexts to be specified, such as "all country sub-entity coded values used in the order and in the invoice shall be valid states according to the United States postal service."

Yet other business rules might require more distinct document contexts to be specified, such as "The country sub-entity codes for the seller can only be states of the United States, while country sub-entity codes for the buyer can be both provinces of Canada and states of the United States."

Furthermore, trading partners can choose to employ an agreed-upon controlled vocabulary for document contexts for which code lists are not defined. This UBL value validation methodology is agnostic to the method by which information items are declared in schemata, thus allowing trading partners to specify acceptable values for information items in any context.

Trading partners must, therefore, take the step to agree on which XPath addresses will specify the contexts at which particular values are constrained. Examining the list of contexts in which code-list-typed information items are found, the partners can identify as much specificity as is required to match those contexts in which the values are constrained.

5.3. UBL reporting of document contexts of code list information items

The UBL support package includes context reports for every document type of the UBL suite. Each context report lists all of the minimally-unique document contexts for information items based on code lists in that document type definition expressed by the schema. These reports are algorithmically derived from the UBL W3C Schema expressions.

The number of code-list-based information items ranges from a low of 14 for the Order Response Simple model, to a high of 77 for the Freight Invoice model.

The number of minimally-unique code-list-based information item document contexts ranges from a low of 311 for the Attached Document model, to a high of 153,335 for the Order Response model.

6. Code list values in document contexts

This Working Draft describes a document model with which associations are made between document contexts of information items and genericode files expressing the values allowed for those items.

6.1. Code list context association files

This model is found in the compressed package associated with this specification in `utility/` directory in the `UBL-ContextConstraints-0.6.xsd` file. This model needs an external declaration of the `xml:id=` attribute, for which one is supplied named `xmlid.xsd` (derived from the xml:id Recommendation [xml:id]).

Using this document model, trading partners create an instance of code list context associations. The instance points to genericode files as system resources using URI strings, and names these pointers using XML identifiers unique to the instance. The document context of each information item to be validated using this methodology is then associated with as many pointer identifiers as required to enumerate all of the possible values from all of the possible enumerations.

A pro-forma code list context association instance reads as follows:

```
<?xml version="1.0" encoding="US-ASCII"?>
<CodeListConstraints
  ... namespace declarations as required for XPath addresses ...
  xmlns="urn:oasis:names:tc:ubl:schema:CodeList-Constraints-0.6"
  id="urn:x-optional-unique-identifier-for-external-referencing"
  name="required-unique-name-token-for-internal-referencing">

  <Identification>
    This is the main code list context association file for project X.

    Revision: 27a  2006-06-17 15:00z
  </Identification>

  <Include uri="other-assoc-file-1.xml"/>
  <Include uri="other-assoc-file-2.xml"/>

  <CodeLists>
    <CodeList xml:id="a1" uri="enumeration1.gc"/>
    <CodeList xml:id="a2" uri="enumeration2.gc"/>
    <CodeList xml:id="a3" uri="enumeration3.gc"/>
  </CodeLists>

  <Contexts>
    <Context item="@item-a" codes="a2" value="token"/>
    <Context item="item-b" context="context-b1" codes="a1 a3"/>
    <Context item="item-b" xpath="context-b2/item-b" codes="a3"/>
  </Contexts>
</CodeListConstraints>
```

The required `name=` attribute specifies a name token for internal referencing by downstream processes that take advantage of alternative expressions of this information.

The optional `id=` attribute specifies a public identifier (typically, but not required to be, a URI) for external referencing, such as in formal trading partner agreements.

The optional `<Identification>` element is a text-only string that is made available to be copied into any intermediate results for tracking purposes. This gives an indication in a result file as to where portions of its information originated when there are a number of portions included.

The optional and repeatable `<Include>` element is a directive to incorporate the associations found in other code list context association files into the one generated result. Where two contexts from the suite of association files match the same node, the priority for the single match that is acted upon is highest for the contexts in the invoked association file, then next highest for the last association file included by an `<Include>` directive (e.g. those in `other-assoc-file-2.xml` above), then the next-to-last association file included by an `<Include>` directive (e.g. those in `other-assoc-file-1.xml` above), and so on with the lowest priority being the first association file included by an `<Include>` directive. Any included association files having such directives will treat those in priority before other directives of the including file.

Each `<CodeList>` element declares the unique identifier for the external code list expression and the pointer to the associated system resource itself, in this example a genericode-encoded file.

Each `<Context>` element points to all of the codes for a given information item in the `codes=` attribute as a white-space-separated list of `<CodeList>` identifiers. It also must declare the XPath address of the information item in isolation without context using the `item=` attribute (the first example above is an attribute, the other two are elements). This is sufficient contextual information when testing the item in a document-wide context, however, when the information item needs to be tested in a given sub-document context or a subset of a multi-document context, one of two mutually-exclusive attributes is required.

The `context=` attribute specifies some ancestral element of the information item beneath which all information items addressed are to be considered in context. Alternatively, when more nuanced detail is required, the `xpath=` attribute specifies the precise XPath context of the information item, including in the address the information item itself. These two attributes can be any valid XPath expression, with as much context and as many predicates as is needed to identify the constructs in the instances.

The `value=` attribute is optional and there is no functional difference in the validation result of instances if this attribute is specified or not. Specifying `value="token"` will, however, indicate there are no white-space characters expected in any of the coded values specified in instances, thus giving the opportunity for execution-time optimization of the validation functionality.

Note

The equivalence of contextual XPath addresses documented above can be expressed that for a given `item="a"`, having `context="b"` is equivalent to having `xpath="b//a"` in its place. Validation error reports will report the equivalent `xpath=` value, whether specified or not, to ensure an unambiguous report.

It is possible that regardless of which combination of attributes is used, two XPath expressions will both match the same node in the source document. The `<Context>` elements for the code list context associations must, therefore, be ordered with the more important XPath contexts first and the less important XPath contexts following in order to ensure a predictable result. A given information item from the document being validated can match only a single context declaration in this code list context association file.

Trading partners can choose to have separate expressions of code list context associations for instances of each document type, or using the context of the document element, combine the associations in a single file for a subset of a multi-document context. An example of a code list context association file testing information items in both UBL Order and UBL Invoice instances is as follows, where `@item-a` has the same constraints in both instances, but `@item-b` has different constraints in both instances:

```
<?xml version="1.0" encoding="US-ASCII"?>
<CodeListConstraints
```

```
  xmlns:in="urn:oasis:names:specification:ubl:schema:xsd:Invoice-1.0"
  xmlns:po="urn:oasis:names:specification:ubl:schema:xsd:Order-1.0"
  xmlns="urn:oasis:names:tc:ubl:schema:CodeList-Constraints-1.0"
  id="urn:x-myurn:code-constraints"
  name="code-list-rules">
  <Information>
    This is another example.
  </Information>
  <CodeLists>
    <CodeList xml:id="a1" uri="enumeration1.gc"/>
    <CodeList xml:id="a2" uri="enumeration2.gc"/>
    <CodeList xml:id="a3" uri="enumeration3.gc"/>
  </CodeLists>
  <Contexts>
    <Context item="@item-a" codes="a2" value="token"/>
    <Context item="@item-b" context="/po:Order//context-b1"
             codes="a1 a3"/>
    <Context item="@item-b" xpath="/in:Invoice//context-b1/@item-b"
             codes="a3"/>
  </Contexts>
</CodeListConstraints>
```

7. Code list value validation

The validation process involves checking all of the information items of an XML instance for their being in the code list context associations, and if so, having the instance values and their associated metadata checked against the values and metadata in the corresponding external XML representations of the code lists.

This methodology is supplied to function with information items in a UBL instance and the external XML representations of code lists in genericode instances. The methodology, however, works with information items in any instance and any external XML representation of code lists, requiring only the included modular stylesheets to be modified in a plug-and-play method for their use in other validation scenarios.

7.1. Instance metadata specification

In the absence of metadata properties for coded values in the instance being validated, only the coded values of the associated external code list expressions can be used. There being no qualification of the values, all values are in play as valid codes for validation.

However, if the instance being validated does have metadata properties specified for a given coded value, then that coded value is asserted to be a value from a particular version or identified code list. Therefore, when the external XML representation of a set of coded values is qualified by metadata properties indicating information about the set of values, the validation process can accurately correlate the instance's value with a set's values.

7.1.1. UBL metadata

The UBL naming conventions for the metadata properties differ slightly based on the name used for the information item. The following rules determine the associated name stems for the metadata information based on the name of the information item, where the embedded string "`xxxxx`" identifies the information item and "`yyyyy`" identifies the metadata item:

- for an attribute named `@xxxxxID` there is no metadata

- for an attribute named `@xxxxxCode` there is no metadata
- for an element named `xxxxxID` use "`@schemeyyyyy`"
- for an element named `<xxxxxCode>` use "`@listyyyyy`"

The "`yyyyy`" metadata suffixes are "`AgencyID`", "`AgencyName`", "`ID`", "`Name`", "`URI`", "`VersionID`", and one of either "`DataURI`" (for elements suffixed with "`ID`") and "`SchemeURI`" (for elements suffixed with "`Code`").

7.2. External code list metadata validation

In the absence of metadata properties for coded values in the external code list expression, all coded values of the associated external code list expression are used. There being no qualification of the values, all values are in play as valid codes for validation.

In the presence of metadata in the external code list expression, the validation can only check an information item's coded value as being correct when the information item's metadata, when present, matches as well.

7.2.1. genericode metadata

genericode has a number of child elements of `<Identification>`, each corresponding to the UBL 2 code list metadata for, respectively, element information items with names ending in "ID" and element information items with names ending in "Code":

- genericode `ShortName` == UBL `@schemeID` or `@listID`
- genericode `LongName` == UBL `@schemeName` or `@listName`
- genericode `Version` == UBL `@schemeVersionID` or `@listVersionID`
- genericode `CanonicalVersionUri` == UBL `@schemeURI` or `@listSchemeURI`
- genericode `AlternateFormatLocationUri` == UBL `@schemeDataURI` or `@listURI`
- genericode `Agency/Identifier` == UBL `@schemeAgencyID` or `@listAgencyID`
- genericode `Agency/LongName` == UBL `@schemeAgencyName` or `@listAgencyName`

7.3. ISO/IEC 19757-3 Schematron

The ISO assertion-based schema language Schematron [Schematron] works by validating the information items in an instance against a set of assertions. A Schematron validating process tests each information item against the highest-priority assertion (earliest in the list of assertions) to which its XPath address matches.

The compressed package that is part of this methodology includes the `schematron-ISO-assembly.xsl` and `schematron-ISO-incomplete-text.xsl` XSLT 1.0 stylesheets (in the `utility/` subdirectory) that are used in tandem to transform a compound set of Schematron sets of assertions into a single XSLT 1.0 stylesheet that, when applied against a document to be validated, reports any failed assertions that are detected. All validation failures are sent to the operating system standard error port, and a non-zero exit is returned from the XSLT processor. A zero exit returned from the XSLT processor indicates successful validation.

Note

The supplied `schematron-ISO-assembly.xsl` and `schematron-ISO-incomplete-text.xsl` XSLT 1.0 stylesheets are incomplete implementations of ISO Schematron in anticipation of later

receipt of complete and conforming implementations. The versions included in the test scenario are modified Schematron 1.5 stylesheets, changed to recognize the ISO namespace in order that Schematron expressions written by early adopters of this methodology will be portable to complete implementations of ISO Schematron when available.

This methodology implements coded value validation by expressing the assertions that the information items in the instance being validated, with any related metadata, are valid values from the external code list expressions that are associated to the information items through the code list context association files. This is accomplished in a modular fashion so as to mesh with other business rules that trading partners may need to express regarding their agreed-upon electronic documents.

7.4. Methodology XSLT transformation stylesheets

The compressed package that is part of this methodology includes an XSLT 1.0 stylesheet (complete with imported fragments) that transforms a code list context association file for a UBL document model and genericode external code list expressions into a corresponding Schematron set of assertions in a single named Schematron pattern.

These stylesheets are modular in a fashion that allows new stylesheets to take advantage of existing modules to support the methodology with other document models and other external code list expressions.

When adapting this methodology to document models and external code list expressions, the basic stylesheet filename patterns follow these conventions:

- `{documentModel}-{externalFormat}2Schematron.xsl` is the stylesheet being invoked,
- `{documentModel}-Metadata.xsl` is the module identifying the metadata in the instance being validated,
- `{externalFormat}-CodeList.xsl` as the module identifying the metadata in the external code list expression, and
- `Constraints2Schematron.xsl` is the module directing the creation of the resulting `Schematron` expression.

Following this convention, then, the demonstrative example included with this methodology invokes the following stylesheet and fragments:

- `UBL-genericode2Schematron-0.6.xsl` which imports the other fragments,
- `UBL-metadata-0.6.xsl`,
- `genericode-CodeList-0.6.xsl` and
- `Constraints2Schematron-0.6.xsl`.

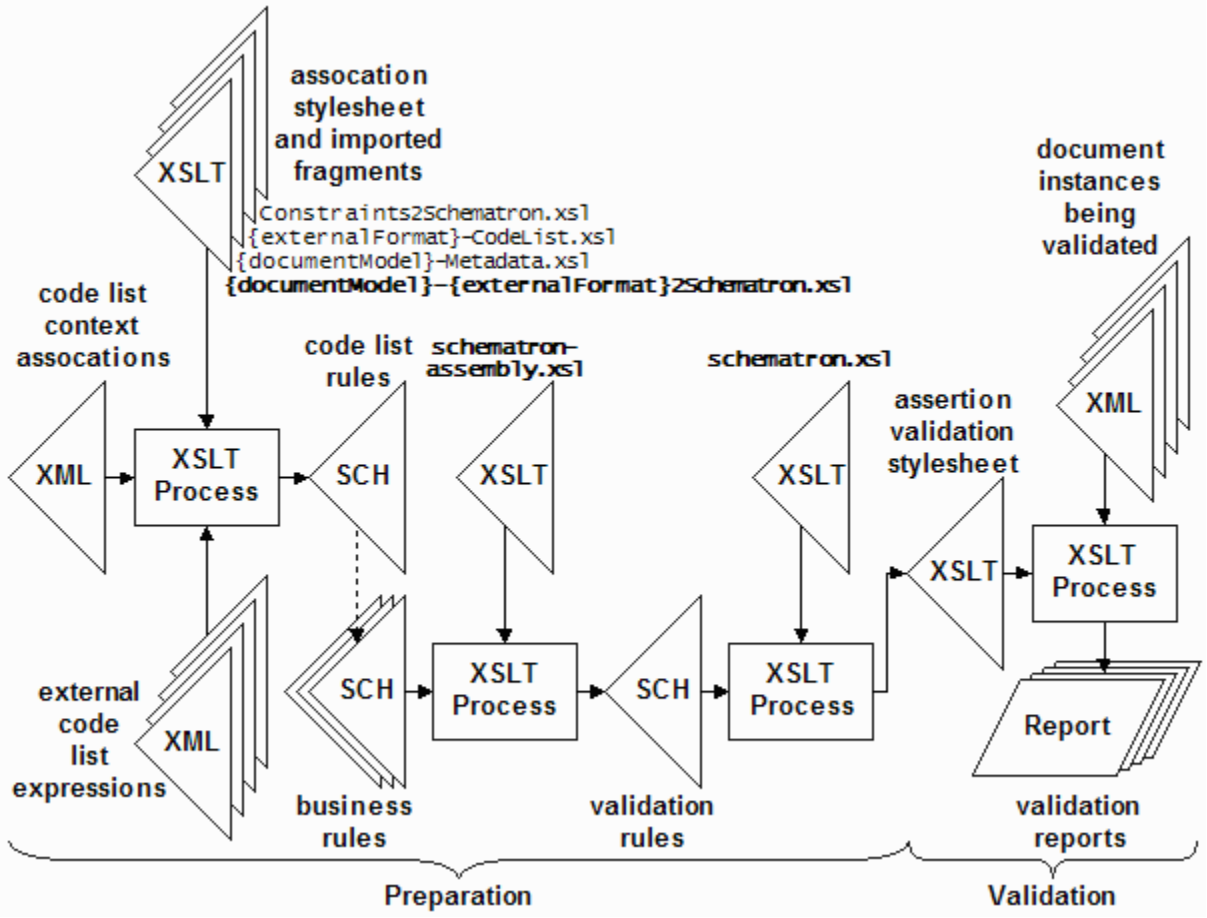
Using the methodology stylesheet for the target document models and external code list expressions, each code list context association file `<Context>` element becomes a `Schematron` assertion (thus requiring the document order of `<Context>` elements to be the required priority order).

The `Constraints2Schematron-0.6.xsl` stylesheet creates a named Schematron pattern, named using the `name=` attribute in the context association file, in the standalone output file. . This end result can then be incorporated into complete Schematron schemata by reference using the `<sch:include>` directive that is resolved in a strict assembly stage that takes place in advance of semantic interpretation of the other Schematron constructs

7.5. Methodology data flow diagram

The demonstrative example implementation of this methodology has two distinct phases, one for the preparation of the validation process and one for the validation activity itself, as shown in Figure 1, “Methodology data flow”. The information expressed in the code list context association file is thus manipulated, but only when the inputs change, into a form with which the actual validation of the document instances is accomplished as many times as required.

Figure 1. Methodology data flow



There is no obligation that an implementation of this methodology follow this particular data flow, only that the end result include the same reported validation violations of contextual use of coded values in the document instances to be validated.

7.5.1. Preparation

The preparation phase incorporates a number of steps to interpret the requirements for validation in order to prepare the validation artefact.

The code list context association file points each of the document instance contexts to the associated external code list expressions. The XSLT stylesheet that imports fragments with knowledge of the document model's metadata and the external code list expression structures transforms the input information into a set of Schematron assertions that need to be true about the document being validated. These assertions are exported in a Schematron file with only a single named pattern of code list rules.

An including Schematron schema of business rules uses `<sch:include>` to incorporate the code list rules with possibly other files of business rules. The Schematron assembly step assimilates all included constructs into a complete Schematron schema of all validation rules.

The resulting Schematron schema is then interpreted into an assertion validation XSLT stylesheet, which is that artefact that implements the checking of the assertions against an instance for validity.

7.5.2. Validation

The XSLT expression of Schematron assertions created in the preparation phase is run against all of the document instances being validated to produce the corresponding validation reports.

There is no need to recreate the assertion validation stylesheet unless anything changes in the code list context associations and external code list expressions, which must then be reprocessed to create a replacement validation artefact.

7.6. Necessary preconditions for the methodology

Not illustrated in the data flow diagram are three necessary preconditions for the data flow to produce bona fide validation reports. The three inputs to the data flow must each be validated against their respective document models in advance to provide properly structured information to the internal processes. None of the XSLT processes illustrated perform any validation of the inputs.

The code list context associations file must have been validated against the `UBL-ContextConstraints-0.6.xsd` constraints.

The external code list expressions must have been validated against their respective model, in this example the `genericode-code-list-0.4.xsd` constraints.

The documents being validated with this methodology must have been validated against their respective structural schema model, which is not included in this example. This is a critically important precondition because the schema constraints will confirm the information items are correctly positioned in the XML instance document hierarchy. This ensures the XPath instructions in the code list context association will be properly applied. Without having confirmed the structural integrity of the XML instance, the assertion validation report is meaningless.

8. A complete running example

The compressed package that is part of this methodology includes the `scenario/` subdirectory in which the following complete scenario can be run to demonstrate how a file of code list context associations can be used to validate sample UBL instances using this methodology and the documented data flow.

Not included in this demonstration is the necessary step run in advance of the code list value validation methodology of validating the UBL document instances against the UBL W3C Schema expression of structural constraints. As noted above, this precondition ensures the information items of the instances are correctly placed in the document hierarchy to be tested for their validity, thus producing bona fide validation reports.

8.1. Support files required

In both the Linux shell environment and the Windows command-line environment, two shell scripts or batch files are invoked in the test scenario and must be available on the path. The following illustrates how each of these are invoked, the first line for a Windows environment and the second line for a Linux environment:

- ```
  call w3cschema schema-file instance-file
  sh w3cschema.sh schema-file instance-file
  ```
  ```

This invokes a W3C Schema validating processor applying the given W3C Schema set of constraints against the given instance file. A non-zero error is returned if there are any validation errors.

- ```
  call xslt input-file stylesheet-file output-file
  sh xslt.sh input-file stylesheet-file output-file
  ```

  This invokes an XSLT stylesheet processor applying the given stylesheet file against the given input file to produce the named output file. Optionally, any number of name/value pairs can be supplied to bind values to top-level parameters in the stylesheet.

8.1.1. Engaging the illustrative example

The invocation files in the ZIP package are preconfigured ready to use once a number of required JAR files are copied into the scenario directory.

The supplied Java-based command-line invocation of the Xerces [Xerces] W3C Schema processor is xjparse [xjparse], invoked as follows (with the appropriate classpath set):

- `java com.nwalsh.parsers.xjparse -S schema-file instance-file`

The supplied Java-based XSLT processor is Saxon [Saxon], invoked as follows (with the appropriate classpath set):

- `java -jar saxon.jar o output-file input-file stylesheet-file param=value`

The classpaths supplied assume the following JAR files are copied into the `utility/` directory before use:

- from xerces: `resolver.jar` and `xercesImpl.jar`
- from xjparse: `xjparse.jar` (copied from a versioned filename)
- from Saxon 6.5.5: `saxon.jar`

Of course the supplied invocation files can be replaced with invocation files of your choice.

8.2. Scenario

In this scenario two trading partners are going to interchange UBL documents between buyer and seller parties. At a technical level, they are using unmodified UBL W3C Schema expressions publicly available for the structural integrity of their XML instances.

At a business level, the trading partners have agreed that all currencies used in an instance can be only Canadian or US dollars. The `MyCurrencyLimits.gc` file documented in Section 4.1, “Trading partner genericode definitions” expresses this limited number of coded values.

As well, the partners have agreed that the buyer's country sub-entity codes may be either a US state or a Canadian province, but that the seller's country sub-entity codes may only be a US state. The two genericode files `MyUSStates.gc` and `MyCanadianProvinces.gc`, also documented in Section 4.1, “Trading partner genericode definitions”, express these limitations.

The following code list context association file `order-constraints.xml` points each of the UBL instance contexts to the required genericode files in order to satisfy the trading partner agreement:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!--
Example constraints on UBL instances exchanged between trading partners.
-->
```
```

```
<CodeListConstraints
  xmlns="urn:oasis:names:tc:ubl:schema:CodeList-Constraints-0.5"
xmlns:cbc="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"
xmlns:cac="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"
  id="urn:x-illustration"
  name="code-list-rules">

  <Identification>
    Illustration of code list constraints.
  </Identification>

  <!--list all of the genericode expressions of agreed-upon code list
      value enumerations-->
  <CodeLists>
    <CodeList xml:id="currency" uri="MyCurrencyLimits.gc"/>
    <CodeList xml:id="states" uri="MyUSStates.gc"/>
    <CodeList xml:id="provinces" uri="MyCanadianProvinces.gc"/>
  </CodeLists>
  <!--list all of the contexts in which the value enumerations are used;
      where two or more contexts might match a given node in the input,
      list them here in order of most-important to least important match-->
  <Contexts>
    <!--all currencies are restricted-->
    <Context item="@currencyID" codes="currency" value="token"/>
    <!--buyer can be in Canada or US-->
    <Context item="cbc:CountrySubentityCode"
             context="cac:BuyerCustomerParty"
             codes="provinces states" value="token"/>
    <!--seller can be only in the US-->
    <Context item="cbc:CountrySubentityCode"
             xpath="cac:SellerSupplierParty//cbc:CountrySubentityCode"
             codes="states" value="token"/>
  </Contexts>
</CodeListConstraints>
```

Note

For illustrative purposes three different methods of expressing the contextual XPath addresses are used in `order-constraints.xml`, though there is no obligation to have to use different ones if this is not necessary. The first `<Context>` element has document wide context by only identifying the item, the second implies descendent context by using `context=`, and the third is explicit about the descendent context by using `xpath=`. Indeed all of these contexts are multi-document contexts as there is no qualification of the document element of the document being validated.

These genericode files and code list context association file together form a formal and unambiguous expression of the contextual coded value constraints that go beyond the constraints of the standardized UBL schema expressions. The package of these files can, therefore, be included in a contractual agreement between the trading partners.

The artefact in this scenario produced by the stylesheet is the file `order-constraints.sch` which includes only a single named Schematron pattern, suitable for inclusion in any ISO Schematron schema. The following Schematron schema `codes-only-constraints.sch` is a minimum expression suitable for including the generated named pattern, as it is acceptable that an assembled Schematron schema contain only a single pattern:

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Code list value assertions</title>
  <ns prefix="cbc"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"/>
  <ns prefix="cac"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"/>
  <include href="order-constraints.sch"/>
</schema>
```

Moreover, trading partners may have additional business rules that apply to instances. Consider there might also be a Schematron expression limiting the total amount of the invoice to less than $10,000. This business rule would be in its own pattern, thus the Schematron file including the `order-constraints.sch` schema would have two patterns. This could be simply expressed merely as two patterns in the schema without any phases, thus implying that all patterns are in play at all times. Alternatively, for more validation flexibility if desired, this can be expressed in Schematron semantics as a single phase that would indicate both patterns as active in the validation process as in the example `total-constraints.sch` schema:

```
<?xml version="1.0" encoding="US-ASCII"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
        defaultPhase="only-phase">
  <title>Business rules for maximum total value</title>
  <ns prefix="cbc"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"/>
  <ns prefix="cac"
      uri="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"/>
  <phase id="only-phase">
    <active pattern="code-list-rules"/>
    <active pattern="business-rules"/>
  </phase>
  <pattern id="business-rules">
    <rule context="cbc:ToBePaidAmount">
     <assert test=". &lt; 10000">Total amount '<value-of select="."/>' cannot be $10,000 or
    </rule>
  </pattern>
  <include href="order-constraints.sch"/>
</schema>
```

Such business rules could also form part of the contract between trading partners, and for one of the tests in the scenario the generated Schematron rules augment authored Schematron rules to make a single assertion validation expression for use against the instances.

8.3. Running test instances against the scenario

A number of test instances are included to demonstrate the code list value validation methodology in this test scenario:

- `order-test-good1.xml` and `order-test-good2.xml` are two instances with valid coded values in context for currency and country sub-entity codes; the buyer in the first instance is in the US and the buyer in the second instance is in Canada; note that the Canadian address uses metadata to indicate version 2 of the associated code list (it happens that a new Canadian territory named Nunavut was recently created by the splitting of the Northwest Territories, thus increasing the number of country sub-entity codes after many decades of not having changed);

- `order-test-bad1.xml` uses the schema-valid currency coded value "UYU" for Peso Uruguayo (which happens to be next in the code list to the US dollar, and might have been inadvertently selected in a data entry user

interface), but this violates the trading partner agreement to use only US or Canadian dollars, while it would have not violated the UBL W3C Schema expression;

- `order-test-bad2.xml` uses the coded value for Canadian country sub-entity code with the metadata for the list indicating the coded value is from the old version "1" of the code list, but this violates the trading partner agreement to use only codes from version "2" of the code list; note that while the same coded value happens to be used as in version "1", it cannot be assumed to be valid because the coded value might represent different semantics in version "2" and the trading partners have agreed to use the updated version;

- `order-test-bad3.xml` uses a Canadian province for the country sub-entity code of the seller's address, but this violates the trading partner agreement that the seller must have a US address. This instance also has a typographical error in the amount to be paid, having omitted the decimal separator, thus indicating a total of $11,500 and not $115.00.

First the scenario establishes the preconditions by validating the code list context association expression of constraints in `order-constraints.xml` and each of the genericode code list expressions (the necessary W3C Schema validation of the test input files is not included in the demonstration).

Next, the code list rules expressed in the code list context association file are translated into a Schematron pattern in `order-constraints.sch`. This can now be reused wherever code list constraints need to be checked. Note that the comments created in this pattern file include the markup for namespace declarations that are necessary in the including Schematron schemas. It is the user's responsibility to ensure that all of the namespaces required by all of the included Schematron pattern files are appropriately declared in the including schema.

The first test scenario, named "Test 1", translates these inputs without business rules into Schematron expression in `codes-only-constraints.sch` without any supplemental business rules, and then translates that first into a complete Schematron instance order-`codes-only.sch` and from there into an XSLT 1.0 expression in `order-codes-only.xsl`.

Preparation is complete for the first test and now the validation stage runs this resulting XSLT against all the test files, producing no errors and a zero return code when there are no problems, and producing a list of errors and a non-zero return code when there are problems. The error report is output to the standard error port, and the return code is testable by the script running the process.

The second test scenario, named "Test 2", augments the Schematron expression of business rules in `total-constraints.sch` with the code list rules into `order-codes-total.sch` and then translates that into an XSLT 1.0 expression in `order-codes-total.xsl`.

Preparation is complete for the second test and the validation stage runs the resulting XSLT against the `order-test-bad3.xml` file with the corrupted total amount, indicating the violation of both the code list and business constraints.

Note again how the constraints need only be prepared once, translating the requirements into the XSLT in the preparation phase to the artefact which is then reused during the validation phase as often as required. In a production environment the XSLT used for validation need only be recreated whenever the trading partner agreement changes to include a new formal expression of the coded value constraints in either the code list context association file or a code list expression file.

Run "`sh test-all.sh`" in a Linux environment or "`test-all.bat`" in a Windows command-line environment to get the following results of running the test scenario:

```
Precondition validation...

Validating partner-agreed constraints...
  w3cschema UBL-ContextConstraints-0.6.xsd order-constraints.xml
Attempting validating, namespace-aware parse
```

```
Parse succeeded (0.230) with no errors and no warnings.

Validating code lists...
  w3cschema genericode-code-list-0.4.xsd MyCanadianProvinces.xml
Attempting validating, namespace-aware parse
Parse succeeded (0.291) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd MyUSStates.xml
Attempting validating, namespace-aware parse
Parse succeeded (0.311) with no errors and no warnings.
  w3cschema genericode-code-list-0.4.xsd MyCurrency.xml
Attempting validating, namespace-aware parse
Parse succeeded (0.281) with no errors and no warnings.

Preparing code list rules...

Translating partner-agreed constraints into Schematron rules...
  xslt order-constraints.xml ../utility/UBL-genericode2Schematron-0.6.xsl
                                                    order-constraints.sch

Test 1 - standalone code list rules

Assembling rules into a Schematron schema...
  xslt codes-only-constraints.sch ../utility/schematron-ISO-assembly.xsl
                                                    order-codes-only.sch

Translating Schematron into validation stylesheet...
  xslt order-codes-only.sch ../utility/schematron-ISO-incomplete-text.xsl
                                                    order-codes-only.xsl

Document validation...

Testing order-test-good1.xml...
  xslt order-test-good1.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 0


Testing order-test-good2.xml...
  xslt order-test-good2.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 0


Testing order-test-bad1.xml...
  xslt order-test-bad1.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1

Value supplied ' UYU ' is unacceptable for codes identified by 'currency'
in the context: @currencyID
Processing terminated by xsl:message at line 18


Testing order-test-bad2.xml...
  xslt order-test-bad2.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1
```

```
Value supplied ' ON ' is unacceptable for codes identified by 'provinces
states' in the context: cac:BuyerCustomerParty//cbc:CountrySubentityCode
Processing terminated by xsl:message at line 18


Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1

Value supplied ' ON ' is unacceptable for codes identified by 'states' in
the context: cac:SellerSupplierParty//cbc:CountrySubentityCode
Processing terminated by xsl:message at line 18


Test 2 - with business rules

Assembling rules into a Schematron schema...
  xslt total-constraints.sch ../utility/schematron-ISO-assembly.xsl
                                                    order-codes-total.xsl

Translating Schematron into validation stylesheet...
  xslt order-codes-total.sch ../utility/schematron-ISO-incomplete-text.xsl
                                                    order-codes-total.xsl

Document validation...

Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-total.xsl nul "2>test-constraints.txt"
Result: 1

Total amount ' 11500 ' cannot be $10,000 or more.
Value supplied ' ON ' is unacceptable for codes identified by 'states' in
the context: cac:SellerSupplierParty//cbc:CountrySubentityCode
Processing terminated by xsl:message at line 19


Done.
```

9. Future work

The compressed package that is part of this methodology includes a modified version of Schematron 1.5, changed to recognize the ISO Schematron namespace but not implementing the ISO Schematron functionality. ISO Schematron has been standardized and when a proper implementation of it is created the test scenario will be changed to employ the latest freely-available version. Nevertheless, the provided incomplete stylesheet is sufficiently functional to implement the checking of a wide range of business rules.

References

[currency] UN/ECE Working Party on Facilitation of International Trade Procedures *Alphabetical Code for the Representation of Currencies [http://www.unece.org/cefact/recommendations/rec09/rec09_ecetrd203.pdf]* Recommendation 9 (Second Edition) January 1996, ECE/TRADE/203 [Edition 96.1]

[DOM] World Wide Web Consortium *Document Object Model [http://www.w3.org/DOM/DOMTR]*

[genericode] Tony Coates *genericode [http://www.genericode.org/]*

[ISO 3166-2] *International Organization for Standardization ISO 3166-2:1998 Codes for the representation of names of countries and their subdivisions - Part 2: Country subdivision code [http://www.iso.org/iso/en/prods-services/iso3166ma/04background-on-iso-3166/iso3166-2.html]*

[OASIS] *Organization for the Advancement of Structured Information Standards [http://www.oasis-open.org/]*

[RFC Keywords] S. Bradner *Key words for use in RFCs to Indicate Requirement Levels [http://rfc.net/rfc2119.txt]* Internet Engineering Task Force, March 1997

[Saxon] Michael Kay *Saxon [http://saxon.sf.net/]*

[Schematron] Rick Jelliffe *ISO/IEC 19757-3 Schematron [http://www.schematron.com/], Document Schema Definition Languages (DSDL) Part 3 [http://www.DSDL.org], ISO/IEC JTC 1/SC 34/WG 1 [http://www.jtc1sc34.org/]*

[UBL 2.0] Jon Bosak *Universal Business Language (UBL) Version 1.0 [http://docs.oasis-open.org/ubl/cd-UBL-1.0/]* (currently under revision), *OASIS UBL Technical Committee [http://www.oasis-open.org/committees/ubl/]* 2005-09-15

[UBL 2.0 Support] Jon Bosak *Universal Business Language (UBL) Version 1.0 Support Materials [http://docs.oasis-open.org/ubl/cd-UBL-1.0/]* (currently under revision), *OASIS UBL Technical Committee [http://www.oasis-open.org/committees/ubl/]* 2005-09-15

[UBL 1.0 SBS] *Universal Business Language (UBL) Version 1.0 Small Business Subset [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl-sbsc]*

[W3C Schema] *XML Schema Part 0: Primer [http://www.w3.org/TR/xmlschema-0/], XML Schema Part 1: Structures [http://www.w3.org/TR/xmlschema-1/], XML Schema Part 2: Datatypes [http://www.w3.org/TR/xmlschema-2/]* 2004-10-28

[Xerces] The Apache XML Project *Xerces [http://xerces.apache.org/xerces2-j/]*

[xjparse] Norman Walsh *xjparse [http://nwalsh.com/java/xjparse/]*

[xml:id] Jonathan Marsh, Daniel Veillard, Norman Walsh *xml:id Version 1.0 [http://www.w3.org/TR/2005/REC-xml-id-20050909/], Annex D.2 [http://www.w3.org/TR/2005/REC-xml-id-20050909/#with-schema-validation]* 2005-09-09

[XPath 1.0] James Clark, Steve DeRose *XML Path Language (XPath) Version 1.0 [http://www.w3.org/TR/1999/REC-xpath-19991116]* 1999-11-16