

Crane's Schematron implementation of OASIS context/value association files for validation

G. Ken Holman

Crane Softwrights Ltd.

<gkholman@CraneSoftwrights.com>

Copyright © 2013 Crane Softwrights Ltd.

\$Date: 2013/02/07 19:40:50 \$(UTC)

Table of Contents

[1. Introduction](#)

[2. Implementation overview and example](#)

[2.1. Review of context/value association files and genericcode](#)

[2.2. Crane's implementation for validation](#)

[2.3. Implementation restrictions](#)

[3. Value validation](#)

[3.1. Using ISO/IEC 19757-3 Schematron](#)

[3.2. Layering value-validation constraints](#)

[3.3. Implementation XSLT transformation stylesheets](#)

[3.4. Implementation data flow diagram](#)

[3.5. Necessary preconditions for the implementation](#)

[4. A complete running example](#)

[4.1. Example context/value association files](#)

[4.2. Example validation shells](#)

[4.3. Support files required](#)

[4.4. Scenario](#)

[4.5. Running test instances against the scenario](#)

[5. Troubleshooting](#)

[5.1. Specification](#)

[5.2. Runtime artefact generation](#)

[5.3. Runtime artefact execution](#)

[6. Revision history](#)

[References](#)

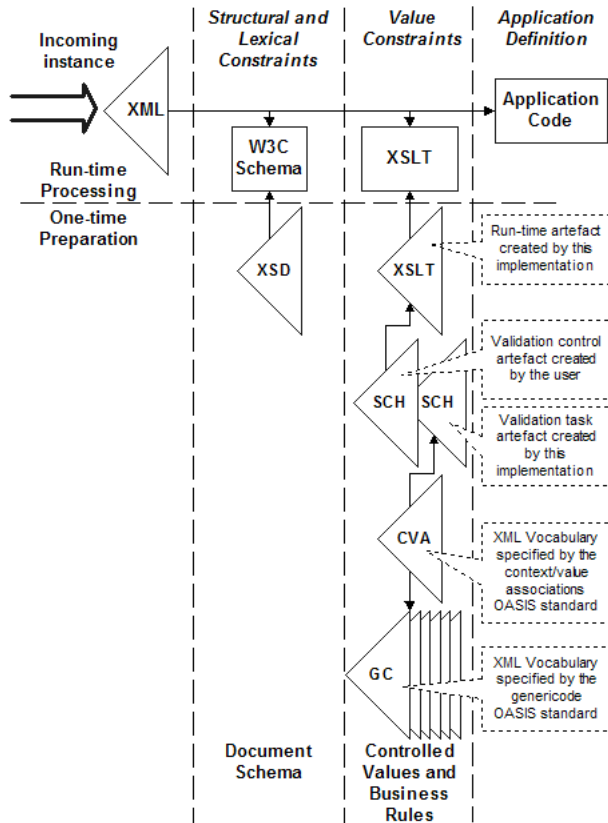
1. Introduction

This describes an implementation of the OASIS context/value association (CVA) file format [[CVA](#)] and the genericcode file format [[genericcode](#)] when used for XML document validation. These specifications were created by the OASIS Code List Representation Technical Committee [[CLRTC](#)]. This implementation generates a Schematron [[Schematron](#)] validation artefact. As there are different ways to deploy Schematron validation, this implementation provides the tools to create an XSLT 1.0 [[XSLT 1.0](#)] run-time artefact for out-of-the-box functionality.

This implementation assumes the addresses in the CVA file are that subset of XPath 1.0 [[XPath 1.0](#)] addresses defined as patterns in XSLT production [1]. There is no check of an appropriate value for the `queryBinding=`.

This implementation is in support of the second pass of a two-pass validation strategy, where the first pass confirms the structural and lexical constraints of a document and the second pass confirms the value constraints of a document. [Figure 1, "Implementation role"](#) illustrates the two-pass validation of an incoming XML instance: the first pass is accomplished with an XML document modeling language such as XML DTD, W3C Schema XSD [[W3C Schema](#)] (illustrated) or ISO/IEC 19757-2 RELAX-NG; the second pass is accomplished with a transformation language such as a W3C XSLT stylesheet (illustrated) or a Python program. This second pass is as an implementation of ISO/IEC 19757-3 Schematron schemas that are created from OASIS context/value association files, independent of the technology used in the first pass.

Figure 1. Implementation role



This implementation documentation focuses only on the second-pass value validation and is independent of the first-pass structural and lexical validation. It assumes that first-pass structural and lexical validation has already been accomplished, so this is not depicted in other diagrams or mentioned in the text. The structural and lexical validation ensures information items are in the correct places and are correctly formed. As a complementary process, value validation addresses those information items governed by controlled vocabularies such as code lists and identifier lists, as well as business rules that might arbitrarily be decided between trading partners.

Note

Of course any information item that is not a code list or an identifier can be expressed as a value that is a member of a predetermined set of values. Trading partners may wish to agree to limit any component of their information interchange based on business requirements for either party.

This implementation package includes the required stylesheets and illustrative test files that are referenced in the prose. While these files as delivered are configured for use with the OASIS Universal Business Language (UBL) 2.0 [[UBL 2.0](#)] and with the genericcode OASIS standard, they can be adapted for instances of any XML vocabulary and any representation of code lists.

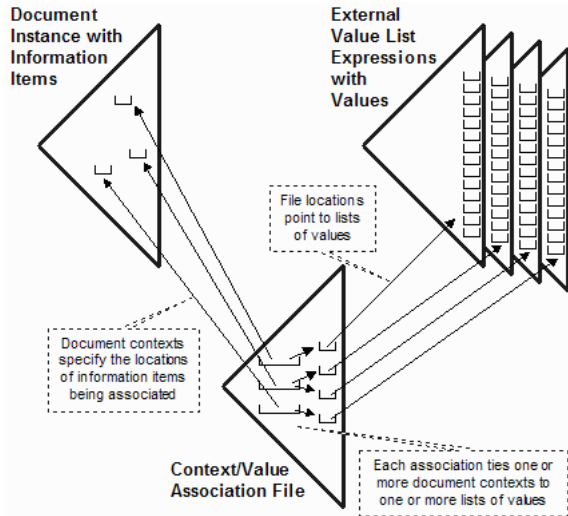
2. Implementation overview and example

2.1. Review of context/value association files and genericcode

The objective of applying this implementation to a set of document instances being validated is to express the lists of values that are allowed in the contexts of information items found in the instances. One asserts in a context/value association file (a "CVA file") that (a) particular values from genericcode files must be used in particular document contexts and (b) particular expressions are evaluated in particular document contexts, and the validation process confirms these assertions do not fail.

[Figure 2. "OASIS context/value association"](#) depicts a file of context/value associations in the lower center, where each association specifies for information items in the document instance being validated which lists of valid values in external value list expressions are to be used.

Figure 2. OASIS context/value association



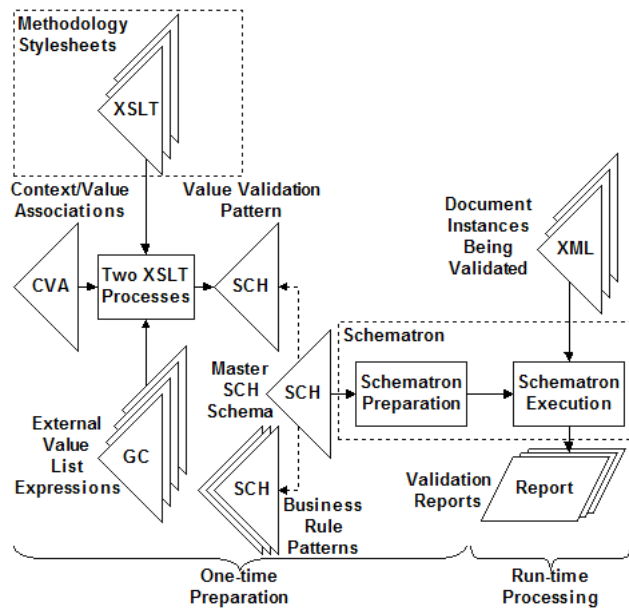
2.2. Crane's implementation for validation

ISO/IEC 19757-3 Schematron [Schematron] is an assertion-based schema language used to confirm the success or failure of a set of assertions made about XML document instances. One can use ISO Schematron to express assertions supporting business rules and other limitations of XML information items so as to aggregate sets of requirements for the value validation of documents.

Crane Softwrights Ltd.'s approach to XML document validation is to translate an OASIS context/value association file into a set of Schematron assertions regarding the information items found in the instance. A feature of this approach is the ability to incorporate the assertions of other business rules expressed in Schematron with the value assertions to produce a single expression of the value constraints of documents.

The synthesis of a pattern of ISO Schematron assertions to validate the values found in document contexts, and the use of ISO Schematron to validate those assertions are illustrated in [Figure 3, "Implementation overview"](#).

Figure 3. Implementation overview



Note

The inputs to the ISO Schematron dotted box in [Figure 3, "Implementation overview"](#) can only be ISO Schematron schemas. How ISO Schematron is implemented is not important to Crane's approach and any conforming implementation may be used. The flow described inside of the ISO Schematron dotted box is the flow implemented by the XSLT stylesheets in the ZIP package supplied with this implementation. The XSLT implementation provides for a runtime component being only the execution of an XSLT stylesheet. Other implementations of ISO Schematron may use other runtime components.

To feed the ISO Schematron process, one needs to express the contexts of information items and the values used in those contexts in a CVA file. The stylesheets provided with this implementation read CVA instances that point to lists of values externally-expressed using genericode. Two XSLT processes are used to produce an ISO Schematron pattern of assertions that can then be combined with other patterns for business rule assertions to aggregate all document value validation requirements into a single ISO Schematron schema.

Provided the first-pass validation process has succeeded, this second-pass value validation process is then used against documents to be validated, producing for each document a report of that document's failures of assertions.

2.3. Implementation restrictions

At this time this implementation is restricted from supporting coded values that include both an apostrophe (single quote) character and a double quote character in the one value string. These stylesheets look for and signal the presence of a coded value that contains such a character combination.

3. Value validation

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

3.1. Using 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. An ISO Schematron validating process tests each information item against the highest-priority assertion (earliest in the list of assertions) to which its XPath address matches.

This implementation implements code list and value validation by expressing the assertions that the information items in the instance being validated, with any related meta data, are valid values from the external controlled-value expressions that are associated to the information items through the context/value association files. By converting the context/value association file into a Schematron pattern, this is accomplished in a modular fashion so as to mesh with other Schematron patterns having other business rules that trading partners may need to express regarding their agreed-upon electronic documents expressed.

The user of this implementation pulls together all of the desired Schematron patterns into a "master" Schematron schema that is converted into the actual validation artefact used at run time to validate XML documents. ISO Schematron offers flexible modes of operation and invocation with multiple phases containing multiple patterns, but the simplest (as illustrated in this implementation) is just to collect all of the patterns into a single unnamed phase.

The compressed package that is part of this implementation includes the following processing XSLT 1.0 stylesheets:

- `iso_schematron_assembly.xsl` assembles a fragmented ISO Schematron schema into a whole ISO Schematron schema through interpretation of the `<sch:include>` directive,
- `iso_schematron_skeleton_for_xslt1.xsl` is a baseline reference XSLT 1.0 implementation of ISO Schematron suitable for being adapted for a particular reporting environment, and
- `{schematron-wrapper}.xsl` is a placeholder for an adaptation of the ISO Schematron skeleton stylesheet for a particular reporting environment:
 - `Message-Schematron-terminator.xsl` is included in the compressed package as an XSLT 1.0 adapted reporting environment for demonstrating this Schematron-based implementation in which all assertions are tested and reported with an XSLT `<xsl:message>` termination signaled in the presence of reports (thus allowing many XSLT processors to signal a non-zero error return code in the presence of reports and a zero error return code in the absence of reports).

These stylesheets are used in tandem to transform a compound set of Schematron sets of assertions into a single XSLT 1.0 stylesheet that in this demonstration environment, 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 most XSLT processors. A zero exit returned from the XSLT processor indicates successful validation.

Note

The supplied ISO Schematron stylesheets are the reference XSLT-based implementations of the specification available from the ISO Schematron web site at this time of writing. Other implementations of ISO Schematron are also publicly available (e.g. Scimitar [[Scimitar](#)]).

3.1.1. ISO Schematron reporting stylesheet

While the supplied demonstrative `Message-Schematron-terminator.xsl` can be used in a production environment if it meets operational requirements, this implementation supports any adaptation of the ISO Schematron stylesheets. The implementation stylesheets synthesize a rudimentary error report for a failed assertion in the absence of a suitably supplied `<Message>` element defining report content.

Users of the implementation may wish to implement more elaborate error reporting through detailed messaging, or possibly implement Schematron Validation Report Language (SVRL)-based errors and reporting through a suitable adaptation of the skeleton ISO stylesheet. Such adaptation should avoid any editing modification of the skeleton ISO stylesheet file and achieve all behaviors through appropriate imported stylesheet customization techniques.

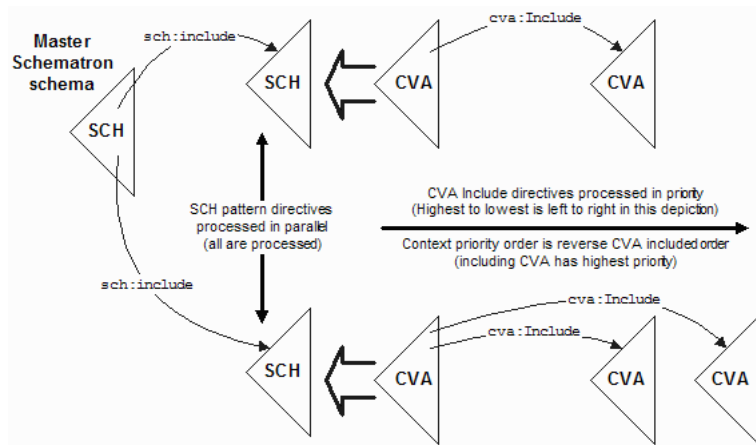
3.2. Layering value-validation constraints

There may be many sources of value-validation constraints and users of the implementation may need to strategize the layering of these constraints to ensure the desired behavior. This will dictate which Schematron patterns need to be created and how to write the "master" Schematron schema that includes them.

Consider that a committee might publish default context/value association files and users may wish to combine these published constraints with their own constraints in one of two manners:

- in parallel: engage both the committee constraints and the user constraints simultaneously without priority override; and
- in priority: engage the user constraints in priority over the committee constraints, such that the committee constraints are only in play for those information items not covered by the user constraints.

[Figure 4. "Implementation layering"](#) illustrates the approaches of engaging constraints in parallel or in priority.

Figure 4. Implementation layering

Observe that all Schematron patterns in a given phase are processed without priority override, so engaging both the committee and user constraints in parallel would entail two `<sch:include>` directives, one for each pattern created by this implementation.

Observe also that all context/value association files included by an including context/value association file are processed in priority order with overriding behaviors. The contexts in the including context/value association file have highest priority, followed in the contexts found in the included context/value association files in reverse document order of the including `<cva:include>` directives. Therefore, to override the committee constraints with the user constraints, the user context/value association file would include the committee context/value association file. The one Schematron pattern would then be created from the including user context/value association file.

3.3. Implementation XSLT transformation stylesheets

The compressed package that is part of this implementation includes an XSLT 1.0 stylesheet (complete with imported fragments) that transforms a context/value association file for a UBL document model and generic code external value 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 implementation with other document models and other external code list expressions.

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

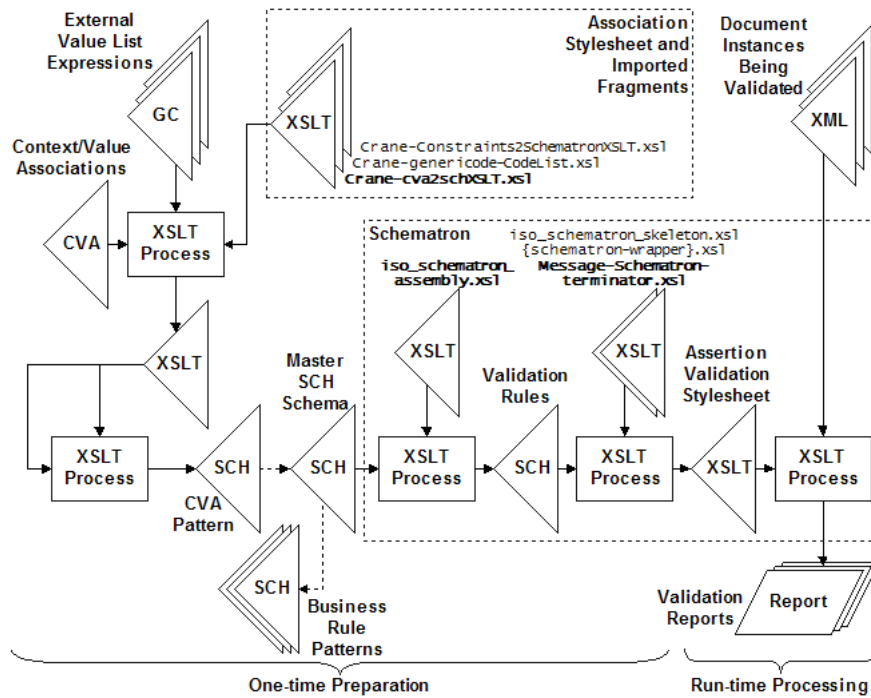
- Crane-cva2schXSLT.xsl is the stylesheet being invoked,
- Crane-{externalFormat}-CodeList.xsl as the module identifying the meta data in the external code list expression, and
- Crane-Constraints2SchematronXSLT.xsl is the module directing the creation of the resulting schematron expression.

The Crane-cva2schXSLT.xsl stylesheet creates an XSLT stylesheet. The stylesheet is labeled XSLT 2.0 but is limited in function to XSLT 1.0 features, thus it can be run with either an XSLT 1.0 or 2.0 processor. This synthesized stylesheet is run standalone or with any document input (for example, itself) to create a Schematron pattern, named using the `name=` attribute in the context association file, in the standalone output file. Each file's `<Context>` element becomes a Schematron assertion (thus requiring the document order of `<Context>` elements to be the required priority order). 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

3.4. Implementation data flow diagram

The demonstrative example implementation of this implementation has two distinct phases, one for the preparation of the validation process and one for the validation activity itself, as shown in [Figure 5. "Implementation data flow"](#). The placebo names are used for those stylesheet fragments that are replaced in this demonstrative environment with the stylesheet fragments indicated in the lists above. The information expressed in the context/value 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 5. Implementation data flow



The following MSDOS invocations are an example of the five steps needed (shown as the five boxes in [Figure 5, "Implementation data flow"](#) to convert the context/value association file `myNeeds.cva` into the runtime artefact `myNeedsOnly.xsl` and invoke it, assuming that the master Schematron schema is named `myNeedsOnly.sch`:

```
Step 1: Create intermediate CVA XSLT:
xslt myNeeds.cva Crane-cva2schXSLT.xsl myNeeds.cva.xsl
Step 2: Create Schematron pattern:
xslt myNeeds.cva.xsl myNeeds.cva.xsl myNeeds.sch
Step 3: Assemble master Schematron schema into an assembly:
xslt myNeedsOnly.sch iso_schematron_assembly.xsl myNeedsOnlyAssembled.sch
Step 4: Convert the assembled Schematron schema into XSLT:
xslt myNeedsOnlyAssembled.sch Message-Schematron-terminator.xsl myNeedsOnly.xsl
Step 5: Invoke the generated XSLT at runtime with your data:
xslt myData.xml myNeedsOnly.xsl nul 2>&1 >myDataResult.txt
echo Return code: %errorlevel%
```

Using `Message-Schematron-terminator.xsl` the return code will be zero for success and non-zero for failure. Using error port redirection the failures are reported in `myDataResult.txt` in this example.

3.4.1. Validation preparation

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

The context/value 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 external code list expression structures transforms the input information into an XSLT stylesheet. This stylesheet is then run with itself as input (the input is ignored but must be present for most XSLT processors to function) and produces 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.

A master Schematron schema validation shell `.sch` written by the user controls which Schematron rules are included in the validation artefact. This shell might itself contain business rules but would typically be a skeletal shell that uses `<sch:include>` to incorporate all of the Schematron fragments needed for a particular validation process. Some of these fragments will be those `.sch` artefacts created for the code list rules using the context/value association files. Other fragments would be `.sch` artefacts possibly containing sets of business rules particular to validation scenarios.

The Schematron assembly step starts with a particular validation shell `.sch` and assimilates all included `.sch` constructs into a complete Schematron schema `.sch` of all validation rules dictated by the shell.

The resulting Schematron schema `.sch` is then interpreted into an assertion validation XSLT stylesheet `.xsl` as the run-time artefact that implements the checking of the assertions against an instance for validity.

Note that the validation shell also satisfies a Schematron management issue of declaring the namespaces used by the other Schematron fragments.

3.4.2. Validation processing

The XSLT expression of ISO 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 context/value associations and external code list expressions, which must then be reprocessed to create a replacement validation artefact.

The functionality and results of the ISO Schematron processing are particular to a user's given environment. While the stylesheet fragments included for demonstrative purposes may meet many users' requirements, any implementation of ISO Schematron or of assertion error reporting may be used in this implementation.

3.5. Necessary preconditions for the implementation

Not illustrated in [Figure 5. "Implementation data flow"](#) 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 context/value associations file must have been validated against the `ContextValueAssociation.xsd` constraints.

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

The documents being validated with this implementation 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 context/value association will be properly applied. Without having confirmed the structural integrity of the XML instance, the assertion validation report is meaningless.

4. A complete running example

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

Not included in this demonstration is the necessary step run in advance of the code list value validation implementation 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.

4.1. Example context/value association files

The following is an example of a short context/value association file `order-combined-doc.cva` that defines some constraints but includes another CVA file with other constraints:

```
<?xml-stylesheet type="text/xsl" href="Crane-cva2html.xsl"?>
<cva:ContextValueAssociation
  xmlns:cva=
    "http://docs.oasis-open.org/codelist/ns/ContextValueAssociation/1.0/"
  xmlns:cbc="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"
  xmlns:x="http://www.w3.org/TR/REC-html40"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  id="urn:x-include-example"
  version="$ Id: order-combined-doc.cva,v 1.5 2009/09/13 04:10:33 gkholman Exp $"
  name="extended-rules">

  <Annotation>
    <Description>
    <x:p>This illustrates inclusion and business rules as test constraints.</x:p>
    </Description>
  </Annotation>

  <Title>
    Illustration of value tests
  </Title>

  <Include uri="order-constraints-doc.cva">
    <Annotation>
      <Description>
        <x:p>Pull in other tests.</x:p>
      </Description>
    </Annotation>
  </Include>

  <ValueTests>
    <ValueTest xml:id="max-amount" test=" . &lt;= 10000">
      <Annotation>
        <Description>
          <x:p>A limiting value for an amount.</x:p>
        </Description>
      </Annotation>
    </ValueTest>
  </ValueTests>

  <Contexts>
    <Context address="cbc:ToBePaidAmount" values="max-amount">
      <Annotation>
        <Description>
          <x:p>Limit the amount to be paid.</x:p>
        </Description>
      </Annotation>
      <Message>
        >Total amount 'sch:value-of select="."/>' cannot be $10,000 or more</Message>
      </Context>
    </Contexts>
  </cva:ContextValueAssociation>
```

The following is a complete example of a context/value association file constrained by this specification. This incorporates all the features described in the OASIS specification (these features are specified in detail in the specification and not here in this implementation documentation):

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="Crane-cva2html.xsl"?>
<cva:ContextValueAssociation
  xmlns:cva=
    "http://docs.oasis-open.org/codelist/ns/ContextValueAssociation/1.0/"
  xmlns:cbc="urn:oasis:names:draft:ubl:schema:xsd:CommonBasicComponents-2"
  xmlns:cac="urn:oasis:names:draft:ubl:schema:xsd:CommonAggregateComponents-2"
  xmlns:x="http://www.w3.org/TR/REC-html40"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  id="urn:x-illustration"
  name="code-list-rules"
  version=
"$ Id: order-constraints-doc.cva,v 1.21 2012/04/18 21:53:16 admin Exp $">
  <Annotation>
    <Description>
      <x:p>
        This is an illustrative example of all of the features of specifying
        the context/value constraints that one can express for XML documents.
      </x:p>
      <x:p>
        The validation requirements for this contrived scenario are as follows:
        <x:ul>
          <x:li>the UN/CEFACT currency code list is restricted to be
            only Canadian and US dollars:</x:li>
          <x:li>the seller must be in the US</x:li>
          <x:li>the buyer may be in either Canada or the US</x:li>
          <x:li>the definition for Payment Means is extended to include
            both UBL definitions and additional definitions</x:li>
        </x:ul>
      </x:p>
    </Description>
  </Annotation>

  <Title>
    Illustration of code list constraints -
    <x:samp>order-constraints.cva</x:samp>
  </Title>

  <!--list all test expressions-->
  <ValueTests>
    <ValueTest xml:id="length-35" test="string-length(.)&lt;=35">
      <Annotation>
        <Description>
          <x:p>Certain fields are restricted to 35 characters in length.</x:p>
        </Description>
      </Annotation>
    </ValueTest>
  </ValueTests>

  <!--list all of the genericcode expressions of agreed-upon code list
  value enumerations-->
  <ValueLists>
    <ValueList xml:id="currency" uri="CAUS_CurrencyCode.gc"
      masqueradeUri="UBL_CurrencyCode-2.0.gc">
      <Annotation>
        <Description>
          <x:p>Restricted to only Canadian and US dollars.</x:p>
        </Description>
      </Annotation>
      <Identification>
        <LongName>ISO Currency List</LongName>
      </Identification>
    </ValueList>
    <ValueList xml:id="states" uri="US_CountrySubentityCode.gc">
      <Annotation>
        <Description>
          <x:p>List of US states.</x:p>
        </Description>
      </Annotation>
    </ValueList>
    <ValueList xml:id="provinces" uri="CA_CountrySubentityCode.gc">
      <Annotation>
        <Description>
          <x:p>List of Canadian provinces</x:p>
        </Description>
      </Annotation>
    </ValueList>
    <ValueList xml:id="tax-ids" uri="TaxIdentifier.gc" key="codeKey">
      <Annotation>
        <Description>
          <x:p>List of tax type identifiers</x:p>
        </Description>
      </Annotation>
    </ValueList>
    <ValueList xml:id="payments" uri="UBL_PaymentMeansCode-2.0.gc">
      <Annotation>
        <Description>
          <x:p>
```



```

Copied from the UBL 2.0 suite:
<x:a href="http://docs.oasis-open.org/ubl/cs-UBL-2.0/">
  <x:samp>http://docs.oasis-open.org/ubl/cs-UBL-2.0/</x:samp>
</x:a>
</x:p>
</Description>
</Annotation>
</ValueList>
<ValueList xml:id="additional_payments"
  uri="Additional_PaymentMeansCode.gc">
  <Annotation>
    <Description>
      <x:p>An extra set of possible payment means.</x:p>
    </Description>
  </Annotation>
</ValueList>
</ValueLists>
<!--list all of the instance-level metadata components associated with
  genericcode <Identification> components-->
<InstanceMetadataSets>
  <Annotation>
    <Description>
      <x:p>UN/CEFACT CCTS V2.01 supplementary components to genericcode</x:p>
    </Description>
  </Annotation>
  <InstanceMetadataSet xml:id="cctsV2.01-amount">
    <Annotation>
      <Description>
        <x:p>CCTS 2.01 AmountType instance metadata</x:p>
      </Description>
    </Annotation>
    <InstanceMetadata address="../@currencyCodeListVersionID"
      identification="Version"/>
  </InstanceMetadataSet>
  <InstanceMetadataSet xml:id="cctsV2.01-measure">
    <Annotation>
      <Description>
        <x:p>CCTS 2.01 MeasureType instance metadata</x:p>
      </Description>
    </Annotation>
    <InstanceMetadata address="../@unitCodeListVersionID"
      identification="Version"/>
  </InstanceMetadataSet>
  <InstanceMetadataSet xml:id="cctsV2.01-quantity">
    <Annotation>
      <Description>
        <x:p>CCTS 2.01 QuantityType instance metadata</x:p>
      </Description>
    </Annotation>
    <InstanceMetadata address="../@unitCodeListID"
      identification="Version"/>
    <InstanceMetadata address="../@unitCodeListAgencyName"
      identification="Agency/LongName"/>
    <InstanceMetadata address="../@unitCodeListAgencyID"
      identification="Agency/Identifier"/>
  </InstanceMetadataSet>
  <InstanceMetadataSet xml:id="cctsV2.01-code">
    <Annotation>
      <Description>
        <x:p>CCTS 2.01 CodeType instance metadata</x:p>
      </Description>
    </Annotation>
    <InstanceMetadata address="@listName"
      identification="LongName[not(@Identifier='listID')]" />
    <InstanceMetadata address="@listID"
      identification="LongName[@Identifier='listID']" />
    <InstanceMetadata address="@listVersionID"
      identification="Version" />
    <InstanceMetadata address="@listSchemeURI"
      identification="CanonicalUri" />
    <InstanceMetadata address="@listURI"
      identification="LocationUri" />
    <InstanceMetadata address="@listAgencyName"
      identification="Agency/LongName" />
    <InstanceMetadata address="@listAgencyID"
      identification="Agency/Identifier" />
  </InstanceMetadataSet>
  <InstanceMetadataSet xml:id="cctsV2.01-identifier">
    <Annotation>
      <Description>
        <x:p>CCTS 2.01 IdentifierType instance metadata</x:p>
      </Description>
    </Annotation>
    <InstanceMetadata address="@schemeName"
      identification="LongName[not(@Identifier='listID')]" />
    <InstanceMetadata address="@schemeID"
      identification="LongName[@Identifier='listID']" />
    <InstanceMetadata address="@schemeVersionID"
      identification="Version" />
    <InstanceMetadata address="@schemeURI"
      identification="CanonicalUri" />
  </InstanceMetadataSet>

```

```

<InstanceMetadata address="@schemeDataURI"
  identification="LocationUri"/>
<InstanceMetadata address="@schemeAgencyName"
  identification="Agency/LongName"/>
<InstanceMetadata address="@schemeAgencyID"
  identification="Agency/Identifier"/>
</InstanceMetadataSet>
</InstanceMetadataSets>

<!--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>
  <Context address="@currencyID" values="currency" mark="money"
    metadata="cctsV2.01-amount">
    <Annotation>
      <Description>
        <x:p>All currencies are restricted to only Canadian and US dollars.</x:p>
      </Description>
    </Annotation>
  </Context>
  <Context address="cac:BuyerCustomerParty//cbc:CountrySubentityCode"
    values="provinces states"
    metadata="cctsV2.01-code">
    <Annotation>
      <Description>
        <x:p>The buyer can be in either Canada or the US.</x:p>
      </Description>
    </Annotation>
    <Message>Invalid province or state '<sch:value-of select="."/>' for buyer "<sch:value-of select="ancestor::cac:BuyerCustomerParty/cac:Party/c
  </Context>
  <Context address="cac:SellerSupplierParty//cbc:CountrySubentityCode"
    values="states"
    metadata="cctsV2.01-code">
    <Annotation>
      <Description>
        <x:p>The seller can only be in the US.</x:p>
      </Description>
    </Annotation>
    <Message>Invalid state '<sch:value-of select="."/>' for seller "<sch:value-of select="ancestor::cac:SellerSupplierParty/cac:Party/cac:PartyNa
  </Context>
  <Context address="cac:TaxCategory/cbc:ID"
    values="tax-ids" mark="money"
    metadata="cctsV2.01-identifier">
    <Annotation>
      <Description>
        <x:p>Limit the recognized tax identifiers</x:p>
      </Description>
    </Annotation>
  </Context>
  <Context address="cbc:PaymentMeansCode" mark="money"
    values="payments additional_payments"
    metadata="cctsV2.01-code">
    <Annotation>
      <Description>
        <x:p>The payments can be by either standard or supplemental means.</x:p>
      </Description>
    </Annotation>
  </Context>
  <Context address="cbc:Name" values="length-35">
    <Annotation>
      <Description>
        <x:p>Names are not allowed to be too long.</x:p>
      </Description>
    </Annotation>
  </Context>
  <Context address="cbc:StreetName" values="length-35">
    <Annotation>
      <Description>
        <x:p>Addresses are not allowed to be too long.</x:p>
      </Description>
    </Annotation>
  </Context>
</Contexts>
</cva:ContextValueAssociation>

```

4.2. Example validation shells

The following codes-only-constraints.sch is a complete example of a validation shell suitable for including the Schematron fragment created for the order-constraints-doc.cva CVA file in [Section 4.1, "Example context/value association files"](#) as the only value constraints to be engaged:

```

<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>

```

The following `order-combined-only.sch` is a complete example of a validation shell suitable for including the Schematron fragment created for the `order-combined.cva` CVA file in [Section 4.1, “Example context/value association files”](#) as the only value constraints to be engaged:

```
<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-combined.sch"/>
</schema>
```

The following `total-constraints.sch` is a complete example of a validation shell that incorporates both a business rule Schematron schema fragment as well as the Schematron fragment created for the `order-constraints-doc.cva` CVA example:

```
<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="total-limit"/>
  </phase>
  <include href="total-limit-constraint.sch"/>
  <include href="order-constraints.sch"/>
</schema>
```

That validation shell pulls in a Schematron expression limiting the total amount of the invoice to less than \$10,000. This business rule would be in its own pattern, as in the example `total-limits-constraint.sch` schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<pattern xmlns="http://purl.oclc.org/dsdl/schematron" id="total-limit">
  <rule context="cbc:ToBePaidAmount">
    <assert test="." &lt; 10000">Total amount '<value-of select="."/>'
cannot be $10,000 or more</assert>
  </rule>
</pattern>
```

After creating all of the Schematron fragments needed by a validation shell, the run-time artefact is created from an assembly of the fragmented Schematron schema into a monolithic Schematron schema.

Note that the `order-combined-only.sch` and `total-constraints.sch` produce the same result using two different approaches to express the business rule of the total amount limitation. In `order-combined.cva` the business rule is expressed as a value test, while in `total-limits-constraint.sch` the business rule is expressed as a Schematron pattern.

4.3. 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.

4.3.1. 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` (version 2 or greater) [\[xjparse\]](#), invoked as follows (with the appropriate path to the jar file set):

- `java -jar xjparse.jar -S schema-file instance-file`

The supplied Java-based XSLT processor is Saxon [\[Saxon\]](#), invoked as follows (with the appropriate path to the jar file 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 Saxon 6.5.5: `saxon.jar`

- from xjparse (version 2 or greater): `xjparse.jar` (copied from a versioned filename)
- from xjparse (version 2 or greater): the entire `lib/` directory as a new `utility/lib/` directory

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

4.4. 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 `CAUS_CurrencyCode.gc` file expresses this limited number of coded values. The `UBL_CurrencyCode-2.0.gc` file is the genericcode file from which the restricted file is derived, thus the restricted file's members have the same semantics as that from the original file. The CVA file reflects this through the use of the `masqueradeUri=` attribute.

The partners have also 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 genericcode files `US_CountrySubentityCode.gc` and `CA_CountrySubentityCode.gc` express these limitations.

Using the `TaxIdentifier.gc` set of identifiers, the transactions can make reference to appropriate taxes.

Finally, the trading partners have agreed to use both the complete set of payment means used in UBL 2.0 plus an additional payment means not used in UBL 2.0, expressed in `Additional_PaymentMeansCode.gc`.

The context/value association file `order-constraints.cva` is listed in its entirety in [Section 4.1, "Example context/value association files"](#) and points each of the UBL instance contexts to the required genericcode files in order to satisfy the trading partner agreement for this scenario.

These genericcode files and context/value 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 `order-constraints.cva` file is used in three ways in this test scenario. First it is used standalone by the `order-codes-only.sch` master Schematron schema. Then it is included by the `order-combined.cva` CVA file illustrating how CVA files are combined in the `order-combined-only.sch` master Schematron schema. Finally the standalone version is included in the `total-limit-constraint.sch` master Schematron schema that includes the `total-constraints.sch` Schematron schema. There are no business rule violations in the first set of tests. The same business rules are violated in the second and third sets of tests, with the same violation messages, but these are produced using the two different methods of a value test and a Schematron test.

4.5. Running test instances against the scenario

A number of test instances are included to demonstrate the code list value validation implementation 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 meta data 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 meta data 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. There is a name in this file that is longer than the limit imposed by a value test. 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 context/value association expression of constraints in `order-constraints.cva`, `order-combined.cva` and each of the genericcode 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 context/value association files are translated into a Schematron pattern in, respectively, `order-constraints.sch` and `order-combined.sch`. These can now be reused wherever 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 master schema.

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

The third test scenario, named "Test 3", 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 third 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 context/value 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 ContextValueAssociation.xsd order-constraints.cva
Attempting validating, namespace-aware parse
Parse succeeded (0.311) with no errors and no warnings.
w3cschema ContextValueAssociation.xsd order-combined.cva
Attempting validating, namespace-aware parse
Parse succeeded (0.300) with no errors and no warnings.

Validating code lists...
w3cschema genericcode.xsd CA_CountrySubentityCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.320) with no errors and no warnings.
w3cschema genericcode.xsd US_CountrySubentityCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.351) with no errors and no warnings.
w3cschema genericcode.xsd CAUS_CurrencyCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.331) with no errors and no warnings.
w3cschema genericcode.xsd TaxIdentifier.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.320) with no errors and no warnings.
w3cschema genericcode.xsd Additional_PaymentMeansCode.gc
Attempting validating, namespace-aware parse
Parse succeeded (0.321) with no errors and no warnings.

Preparing code list rules...

Translating partner-agreed constraints into Schematron rules...
xslt order-constraints.cva
  ..\utility\Crane-cva2schXSLT.xsl
  order-constraints.sch.xsl
xslt order-constraints.sch.xsl order-constraints.sch.xsl order-constraints.sch

xslt order-combined.cva
  ..\utility\Crane-cva2schXSLT.xsl
  order-combined.sch.xsl
xslt order-combined.sch.xsl order-combined.sch.xsl order-combined.sch

Test 1 - standalone code list rules

Assembling rules into a Schematron schema...
xslt codes-only-constraints.sch ..\utility\iso_schematron_assembly.xsl
  order-codes-only.sch

Translating Schematron into validation stylesheet...
xslt order-codes-only.sch ..\utility\Message-Schematron-terminator.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 constraints identified by 'currency' in
the context '@currencyID': /Order/cac:TaxTotal[1]/cbc:TaxAmount[1]/@currencyID

Processing terminated by xsl:message at line 151

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

```
Invalid province or state 'ON' for buyer "Elliot's Electronics":
/Order/cac:BuyerCustomerParty[1]/cac:Party[1]/cac:Address[1]/
cbc:CountrySubentityCode[1]
```

Processing terminated by xsl:message at line 151

```
Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-only.xsl nul "2>test-constraints.txt"
Result: 1
Value supplied 'Joes Office Supply With A Very Long Name' is unacceptable
for constraints identified by 'length-35' in the context 'cbc:Name':
/Order/cac:BuyerCustomerParty[1]/cac:Party[1]/cac:PartyName[1]/cbc:Name[1]
Invalid state 'ON' for seller "Elliot's Electronics":
/Order/cac:SellerSupplierParty[1]/cac:Party[1]/cac:Address[1]/
cbc:CountrySubentityCode[1]
```

Processing terminated by xsl:message at line 151

Test 2 - with business rule expressed as a value test

```
Assembling rules into a Schematron schema...
  xslt order-combined-only.sch ..\utility\iso_schematron_assembly.xsl
                                     order-codes-combined.sch
```

```
Translating Schematron into validation stylesheet...
  xslt order-codes-combined.sch ..\utility\Message-Schematron-terminator.xsl
                                     order-codes-combined.xsl
```

Document validation...

```
Testing order-test-bad3.xml...
  xslt order-test-bad3.xml order-codes-combined.xsl nul "2>test-constraints.txt"
Result: 1
Value supplied 'Joes Office Supply With A Very Long Name' is unacceptable
for constraints identified by 'length-35' in association file with uri
'order-constraints-doc.cva' in the context 'cbc:Name':
/Order/cac:BuyerCustomerParty[1]/cac:Party[1]/cac:PartyName[1]/cbc:Name[1]
Invalid state 'ON' for seller "Elliot's Electronics":
/Order/cac:SellerSupplierParty[1]/cac:Party[1]/cac:Address[1]/
cbc:CountrySubentityCode[1]
Total amount '11500' cannot be $10,000 or more:
/Order/cac:LegalTotal[1]/cbc:ToBePaidAmount[1]
```

Processing terminated by xsl:message at line 151

Test 3 - with business rule expressed as Schematron

```
Assembling rules into a Schematron schema...
  xslt total-constraints.sch ..\utility\iso_schematron_assembly.xsl
                                     order-codes-total.sch
```

```
Translating Schematron into validation stylesheet...
  xslt order-codes-total.sch ..\utility\Message-Schematron-terminator.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:
/Order/cac:LegalTotal[1]/cbc:ToBePaidAmount[1]
Value supplied 'Joes Office Supply With A Very Long Name' is unacceptable
for constraints identified by 'length-35' in the context 'cbc:Name':
/Order/cac:BuyerCustomerParty[1]/cac:Party[1]/cac:PartyName[1]/cbc:Name[1]
Invalid state 'ON' for seller "Elliot's Electronics":
/Order/cac:SellerSupplierParty[1]/cac:Party[1]/cac:Address[1]/
cbc:CountrySubentityCode[1]
```

Processing terminated by xsl:message at line 152

Done.

5. Troubleshooting

This section will collect reports from users of specification, invocation and execution problems encountered and their resolution. Please do not hesitate to contact info@CraneSoftwrights.com with any comments or questions that will embellish this detail for yourself and other users.

5.1. Specification

Remember not to omit the `metadata=` attribute of the `<Context>` element when you need to match up the authored metadata with the values. Without it, the process will appear to work just fine when your metadata is correct, but it will not fail when your metadata is incorrect.

5.2. Runtime artefact generation

Review [Figure 5, “Implementation data flow”](#) for the steps of invocation.

Remember not to run the generated Schematron script as it is, rather to include it into a master Schematron schema and use the master to create the runtime XSLT artefact.

5.3. Runtime artefact execution

XPath syntax errors at runtime may be due to omitting required namespace declarations in the master Schematron schema for namespace prefixes used in the included Schematron fragments (generated or authored). The Schematron processor used for runtime artefact generation does not detect this error at preparation time.

6. Revision history

2013-02-07 - repaired an important bug to support the union operator in the identification XPath address

References

[CLRTC] [OASIS Code List Representation Technical Committee](#)

[CVA] G. Ken Holman [OASIS Context/Value Association Committee Specification 01](#)

[genericode] Tony Coates [OASIS Genericode Committee Specification 01](#)

[Saxon] Michael Kay [Saxon](#)

[Schematron] Rick Jelliffe [ISO/IEC 19757-3 Schematron](#), [Document Schema Definition Languages \(DSDL\) Part 3](#), [ISO/IEC JTC 1/SC 34/WG 1](#)

[Scimitar] Fourthought, Inc. [Scimitar](#)

[UBL 2.0] Jon Bosak, Tim McGrath, G. Ken Holman [Universal Business Language \(UBL\) Version 2.0](#), [OASIS UBL Technical Committee](#) 2006

[W3C Schema] [XML Schema Part 0: Primer](#), [XML Schema Part 1: Structures](#), [XML Schema Part 2: Datatypes](#) 2004-10-28

[Xerces] The Apache XML Project [Xerces](#)

[xjparse] Norman Walsh [xjparse](#)

[XPath 1.0] James Clark, Steve DeRose [XML Path Language \(XPath\) Version 1.0](#) 1999-11-16

[XSLT 1.0] James Clark [XSL Transformations \(XSLT\) Version 1.0](#) 1999-11-16