A Layered Approach to XML Canonicalization

(A Position Paper for the W3C Workshop on
Next Steps for XML Signature and XML Encryption)

Ed Simon, XMLsec Inc.

Abstract

XML Canonicalization enables reliable textual and binary comparison of XML documents through the removal of irrelevant differences in structure and content. Though XML Canonicalization is critical for XML Signatures, it also has value in other XML applications such as version control.

Currently, the approach to XML Canonicalization is to write a single specification that details how all parts of XML instances are to be canonicalized. This position paper suggests an alternative approach in which canonicalization is specified separately for the different layers in XML data stack (core, schema, and namespace). In this way, an application can select different canonicalization methods for the different layers; this allows the customization of canonicalization to the application's needs while simplifying the task of writing canonicalization specifications because the specifications need not be all encompassing. As well, it may also be feasible to canonicalize data belonging to different namespaces differently; for example, numbers in an accounting spreadsheet could be canonicalized differently than numbers appearing in descriptive text.

Description

Three layers of canonicalization are proposed: core, schema-aware, and namespace-aware. They are described in the following table.

Layer	Description
Core	Normalizes the elements, attributes, and whitespace of an XML instance. No normalization of aspects outside of the W3C XML 1.1 specification.
Schema-Aware	Normalization of schema-aware aspects including default attributes, schema-defined data types, etc. The OASIS Schema Centric XML Canonicalization [1] specification provides examples pertinent to Schema-Aware canonicalization.
Namespace-Aware	Normalization of XML information set nodes that belong to, or are contained by nodes that belong to, an XML node declared with a particular namespace. Includes the normalization of namespace declarations themselves.

How it Works

Core Canonicalization

All XML instances go through Core Canonicalization. This would ensure, for example, that the XML instances in Listing 1 and Listing 2 were evaluated as identical.

Listing 1: XSL Example 1

```
<xsl:stylesheet version="2.0"

xmlns:ns1="http://www.xmlsec.com/namespaces/a"

                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:template match="/">
      <p>Total Amount: <xsl:value-of select="ns1:expense-report/ns1:total"/></p>
</xsl:template>
</xsl:stylesheet>
```

and

Listing 2: XSL Example 2

```
<xsl:stylesheet version='2.0' xmlns="http://www.w3.org/1999/xhtml"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:ns1="http://www.xmlsec.com/namespaces/a"
                >
<xsl:template match="/">
      <p>Total Amount: <xsl:value-of select="ns1:expense-report/ns1:total"/></p>
</xsl:template>
</xsl:stylesheet>
```

Core Canonicalization would not, however, evaluate those in Listing 3 and Listing 4 as identical because of the different application-specific namespace declarations (xmlns:ns1 and xmlns:ns2).

Listing 3: XSL Example 3

```
<xsl:stylesheet version="2.0"

xmlns:ns1="http://www.xmlsec.com/namespaces/a"

                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">
<xsl:template match="/">
      <p>Total Amount: <xsl:value-of select="ns1:expense-report/ns1:total"/></p>
</xsl:template>
</xsl:stylesheet>
```

and

Listing 4: XSL Example 4

```
<xsl:stylesheet version='2.0' xmlns="http://www.w3.org/1999/xhtml"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:ns2="http://www.xmlsec.com/namespaces/a"
                >
<xsl:template match="/">
      <p>Total Amount: <xsl:value-of select="ns2:expense-report/ns2:total"/></p>
```

```
</xsl:template>
</xsl:stylesheet>
```

Nor would Core Canonicalization evaluate the XML instances in Listing 5 and Listing 6 as the same (each having the schema declaration in Listing 7).

Listing 5: XML instance 1 belonging to schema with default attribute and content type declarations

```
<p language="en-ca"><value>+10</value></p>
```

and

Listing 6: XML instance 2 belonging to schema with default attribute and content type declarations

```
<p><value>10</value></p>
```

Schema-Aware Canonicalization

Schema Canonicalization is Canonicalization that is aware of the impact of an XML instances associated schema (XML Schema, RelaxNG, or any form of XML-applicable schema language). For example, Listing 5 and Listing 6 show two XML instances that, by themselves, have different information sets. Schemas for XML, such as the W3C XML Schema specification bring another level of meaning to an XML instance that can impact the XML information set. For example, this schema

Listing 7: XML Schema with default attribute and content type declarations

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
      <xs:element name="p">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="value" type="xs:integer"/>
                  </xs:sequence>
                  <xs:attribute name="language" type="xs:token" default="en-ca"/>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

indicates that the <p> tag has a default attribute named “language” with a default value of “en-ca”. That means an XML processor will see that attribute with that value in an XML instance associated with the schema even if the instance itself does not actually have the attribute. In addition, the schema declares that the content of the <value> element is an integer which means that a schema-aware XML processor will know that whether a <value> element contains the text “10” or “+10”, those values are equivalent. An XML processor that was not schema aware would have to regard these values as different because it can only do a simple character-by-character comparison.

Namespace-Aware Canonicalization

Namespace-Aware Canonicalization refers to more than the simple normalization of namespaces (thought that would often be an essential part). Namespaces define semantics and processing for the XML elements and attributes that they cover. As such, Namespace Canonicalization is the normalization, in view of namespace-specific semantics and processing, of the XML information set.

For example, this <Distinguished_Name> element (Listing 8) belonging to the “http://example.org/dn” namespace

Listing 8: Distinguished Name XML instance

```
<Distinguished_Name xmlns="http://example.org/dn">
<Attribute Name="rfcXXXX:cn"
xmlns:rfcXXX="http://example.org/rfcXXXX">Sam</Attribute>
</Distinguished_Name>
```

could have a canonicalization rules that requires X.509 short names to be replaced with their full names as shown in Listing 9:

Listing 9: Canonicalized Distinguished Name XML instance

```
<Distinguished_Name xmlns="http://example.org/dn">
<Attribute Name="rfcXXXX:commonName"
xmlns:rfcXXX="http://example.org/rfcXXXX">Sam</Attribute>
</Distinguished_Name>
```

As another example, consider the XSL stylesheets in Listing 3 and Listing 4. From an XML infoset perspective, these stylesheets are not equivalent because the text content of their respective <xsl:value> @select attributes are different. Yet those who know XSL would recognize these stylesheets as equivalent because the XSL specification indicates the @seclect attribute contains intra-document references to namespaces and both reference the same namespace. An XSL Namespace-Aware canonicalizer would recognize these instances as equivalent.

Note: As an aside, it might be argued that XML Schema should support a form of regex-type declarations for text content that could include an “XML namespace” data type. The author would agree!

Namespace Normalization

The author agrees that the original idea of trying to normalize namespace declarations by replacing their short names with an enumerated sequence (ns1, ns2, etc.) had its shortcomings. On the other hand, simply not canonicalizing namespace declarations is not satisfactory either. As such, this paper proposes that namespace names be normalized in a way that does not depend on information extraneous to the namespace declaration (as the enumeration approach did). Instead, it should be possible to define a namespace short name based solely on the value of the namespace itself. This paper presents two possibilities for doing so:

1. using an escaped, XML token-compliant form of the actual namespace and;
2. using a quasi-base64-formatted hash of the namespace.

For example, take the following namespace: “http://example.org/dn”.

The escaped, XML token-compliant form would be of the form like the following:
“http__COLON__SLASH__SLASHexample__DOTorg__SLASHdn”

Similarly, using the quasi-base64-formatted hash, the short name could look like this:
“BeP8rW4G5UmklOz6Mi_-”
(using '_' and '-' to replace the base64 '/' and '+' respectively to achieve compliance with XML name rules.)

The central advantage of either of the above two schemes is that one can determine the canonical reference name of the namespace simply given the namespace itself. This ensures unambiguous, effective, and easy namespace normalization across applications.

Declaration of Canonicalization Processing

Layered Canonicalization enables an application to apply different canonicalization methods to the different data layers of an XML instance. Normally, an application will apply one form of Core Canonicalization and one form of Schema Canonicalization (the possibility for a schema to import other schemas needs further thought). However, it is quite common for XML instances to contain nodes from multiple namespaces and so there must be a way to specify multiple Namespace Canonicalization methods. The following, initial proposal shows how this might be done.

Listing 10: Canonicalization Methods Declaration Example

```
<Canonicalization xmlns="http://w3c.org/xml/canonicalization">

      <Target Namespace="http://example.org/dn">
            <CanonicalizationMethod
                  Algorithm="http://example.org/dn/canonicalization"/>
      </Target_Namespace>

      <Target Namespace="http://www.w3.org/1999/XSL/Transform" Version="2.0"
            <CanonicalizationMethod
                  Algorithm="http://example.org/xslt/2_0/canonicalization"/>
      </Target_Namespace>

</Canonicalization>
```

References

[1] “OASIS Schema Centric XML Canonicalization”
http://www.uddi.org/pubs/SchemaCentricCanonicalization-20020710.htm