XML Canonicalization, Part 2

October 9, 2002

Bilal Siddiqui (/pub/au/140)

In the previous installment (http://www.xml.com/pub/a/ws/2002/09/18/c14n.html) of this article, I introduced Canonical XML, and I discussed when and why you need to canonicalize an XML file. I also demonstrated a step-by-step process that results in the canonical form of an XML document.

In this second and final installment, I'll take the concept further and explain the canonicalization requirements of CDATA sections, processing instructions, comments, external entity references and XML document subsets.

Let's start with an example. Listing 1 (/2002/10/09/examples/Listing1.txt) is an XML file that contains, among other things, a CDATA section, comments, a processing instruction, and an external entity references. The thirteen steps of part 1 are not sufficient to canonicalize it. We need to perform a few additional steps.

14. CDATA Sections

The canonical form requires all CDATA sections to be replaced with their equivalent PCDATA XML content. This is what we have done in Listing 2 (/2002/10/09/examples/Listing2.txt). If you compare the two listings, you will find that the markup for CDATA section ("<![CDATA[" in the beginning and "]]>" at the end) has been deleted and "<" character in the CDATA section of Listing 1 (/2002/10/09/examples/Listing1.txt) has been replaced with its equivalent escape sequence (<) in Listing 2 (/2002/10/09/examples/Listing2.txt).

15. Processing Instructions

We need to normalize whitespace inside processing instructions. This means that the whitespace between the target and its data will be reduced to a single space (the #x20 character).

There is only one processing instruction in the XML file of Listing 2 (/2002/10/09/examples/Listing2.txt). The target in this processing instruction is `xml-stylesheet`, which is followed by the data string. Listing 3 (/2002/10/09/examples/Listing3.txt) is the same as Listing 2 (/2002/10/09/examples/Listing2.txt), except that all whitespace between the target and its data has been normalized.

16. External Entity References:

Recall the section on entity references in part 1, where we demonstrated how to canonicalize parsed internal entity references. In a similar fashion, parsed external entity references also need to be replaced with the content they refer to, as shown in Listing 4 (/2002/10/09/examples/Listing4.txt).

17. Comments

The canonical XML specification allows both retaining and removing comments from an XML file. An XML canonicalization engine will receive a boolean parameter (flag) along with the XML file to be canonicalized, which will tell the canonicalization engine whether to include or exclude XML comments in the canonical form.

For example, Listing 5 (/2002/10/09/examples/Listing5.txt) shows the removal of comments from Listing 4 (/2002/10/09/examples/Listing4.txt) (canonical XML without comments).

We are now ready to apply the thirteen steps to Listing 5 (/2002/10/09/examples/Listing5.txt) (as described in part 1). The result is shown in Listing 6 (/2002/10/09/examples/Listing6.txt).

18. Document Subsets

XML document subsets or fragments (portions of complete XML files) are an interesting case. When we extract a portion from an XML file, we essentially separate a child node form its parent (call it an *orphan* node). This separation may result in the invalidation of the child's namespace context if the namespace context of the orphan child was declared in the parent that has been omitted in the document subset.

The Canonical XML specification proposes a method to preserve the namespace context while extracting a document subset. However, there are application scenarios in which preserving the namespace context may create other problems. W3C has released a separate recommendation named Exclusive XML Canonicalization which deals with such scenarios.

The difference between the Canonical XML and Exclusive XML Canonicalization specifications is only about preserving or excluding the ancestor context.

Preserving the Ancestor Context

Have a look at Listing 7 (/2002/10/09/examples/Listing7.txt), which is a SOAP message. Let's assume we need to canonicalize the `booking` element in Listing 7 (/2002/10/09/examples/Listing7.txt) whose `unitCharge` attribute shows "50" as the value. The first step in doing this is to write an XPath expression that will extract the required document fragment from the XML file. While trying to identify which element I intend to canonicalize, I said "the booking element in Listing 7 whose unitCharge attribute shows '50' as the value". The equivalent XPath expression with the same meaning is

```
(//. | //@* | //namespace::*)[ancestor-or-self::bs:booking[@unitChar

(with namespace declaration xmlns:bs="http://www.FictitiousTourismInt
```

This XPath expression will extract the required booking element from the XML file of Listing 7 (/2002/10/09/examples/Listing7.txt). The expression in the first pair of brackets (`//. | //@* | //namespace::*`) selects all element, attribute, and namespace nodes of an XML file. The expression in the outer pair of square brackets (`ancestor-or-self::bs:booking`) selects all `booking` elements (along with their children) and the expression in the inner pair of square brackets (`@unitCharge="50"`) selects the `booking` element whose `unitCharge` attribute has the value "50".

Listing 8 (/2002/10/09/examples/Listing8.txt) is a subset of Listing 7 (/2002/10/09/examples/Listing7.txt) and consists of the booking element. Some readers might be tempted at this point to apply the thirteen steps of part 1 to canonicalize Listing 8 (/2002/10/09/examples/Listing8.txt). However, there are a couple of problems that require additional processing before we can apply those thirteen steps:

1. The namespace declarations for the `bs` and `hs` prefixes were made in booking element's parent tag, which is not included in the document subset shown in Listing 8 (/2002/10/09/examples/Listing8.txt).
2. The `xml:lang` attribute of the `bookingPackage` element of Listing 7 (/2002/10/09/examples/Listing7.txt) was applicable to all its children. This attribute is also missing in the document subset of Listing 8 (/2002/10/09/examples/Listing8.txt).

These problems clearly indicate that extracting document fragments should be accompanied by actions to preserve the namespace context and the effect of attributes from the `xml:` namespace. The Canonical XML specification requires the following measures to be taken while canonicalizing document subsets (in addition to all the requirements of canonicalizing complete XML files).

1. Namespace declarations in the omitted ancestors of the document subset are included in the canonical form.
2. Attributes in the xml namespace are also included in the canonical form, if they are not already present in the fragment being canonicalized.

These two steps are intended to conserve the ancestor context of a document subset. Have a look at Listing 9 (/2002/10/09/examples/Listing9.txt) (the required canonical form), which includes the four namespace declarations made in the ancestors of the booking element of Listing 7 (/2002/10/09/examples/Listing7.txt). Listing 9 (/2002/10/09/examples/Listing9.txt) also includes the `xml:lang` attribute. Also notice that the canonical form of document subsets does not have any line breaks (#xA) i.e. the entire file appears on the same line.

Once the ancestor context has been included, the ordering of namespace declarations and attributes is the same as for canonicalizing the complete XML file.

Next Page ()

XML Canonicalization, Part 2
by Bilal Siddiqui |

Excluding the Ancestor Context

We have seen that the ancestor context is included while canonicalizing XML document subsets. However, doing this may introduce problems under certain circumstances. In order to elaborate the scenarios in which including the ancestor context creates problems, we first need to discuss *Enveloping*, a concept that is of paramount importance in web services interoperability.

Enveloping

SOAP is fast becoming the de-facto standard for XML messaging over the Internet. SOAP defines the format to wrap XML data inside envelopes.

Look at Listing 7 (/2002/10/09/examples/Listing7.txt), which wraps the `PackageBooking` element inside the `SOAP:Body` element. Listing 7 (/2002/10/09/examples/Listing7.txt) demonstrates a simple enveloping mechanism, in which the message payload (i.e. the message that needs to be sent across the Internet) is wrapped inside a `SOAP:Body` element and the entire `SOAP:Body` is wrapped by the `SOAP:Envelope` element.

The advantage of this simple enveloping lies in its ability to enable vertical stacking of XML-based protocols. Vertical stacking means that protocols and message formats can be defined for specific low-level tasks (such as signing, encrypting, routing etc.) and higher protocol layers will use the services provided by lower layers. For example, WS-Security, a high-level XML security protocol being developed by an OASIS Technical Committee, uses the SOAP format to utilize the signing and encrypting mechanisms provided by W3C's XML Digital Signature and XML Encryption specifications respectively.

Listing 7 (/2002/10/09/examples/Listing7.txt) also contains a `SOAP:Header` element in addition to the `SOAP:Body` element. The `SOAP:Header` element is optional and is meant to contain protocol-specific information. This effectively means that the message payloads are contained inside `SOAP:Body` elements and protocol headers are contained in the `SOAP:Header` element. For instance, WS-Security uses the SOAP Header to wrap signature related information.

Envelope Handling

The application which receives a SOAP message is likely to tear the envelope (wrapper) and extract the XML payload (the XML message) in order to be able to process the message received. This tearing of a SOAP envelope and extracting the XML payload is referred to as *de-enveloping*. Further, the receiving application might need to *re-envelope* the XML message received in a new envelope.

The need for re-enveloping emerges in federated web services, which rely on partner applications to do part of a job, thus integrating, for example, an entire supply chain into interoperable and loosely coupled systems.

As an example of federated applications, let's consider a tourism industry B2B scenario. A tourist wants to know the details of a vacation tour being offered by a tour operator's web service. The tourist sends an XML message containing information about the places he would like to visit and the dates on which he is planning to travel.

Naturally the tourist's XML message will be authored by some client-side XML- and SOAP-aware application, which will author and wrap all information inside a SOAP envelope without requiring the tourist to know anything about XML and SOAP.

Upon receipt of this SOAP message, the tour operator's web service will extract the information related to time and place of travel from the SOAP envelope. The tour operator's service will need to send pieces of the travel information to different partner hotels and car rental companies. Therefore, the tour operator's service will author fresh SOAP envelopes containing the relevant pieces of information and forward them to partner hotels and car rental companies.

In a similar fashion, upon receipt of the response from partner hotels and car rental companies, the tour operator's service will re-envelope the information received before sending the fresh envelope back to the tourist.

Exclusive XML Canonicalization

With the above discussion in mind, have a look at Listing 10 (/2002/10/09/examples/Listing10.txt), which is a SOAP response message that a fictitious partner hotel has just sent back to the tour operator's web service.

The tour operator would have also received SOAP response messages from other partner hotels and car rental companies. These messages need to be combined to form a complete packaged vacation tour.

You may now have a look again at Listing 7 (/2002/10/09/examples/Listing7.txt), which is actually a packaged vacation tour that the tour operator's web service will ultimately send back to the tourist. The first `booking` element of Listing 7 (/2002/10/09/examples/Listing7.txt) (whose `unitCharge` attribute shows a value of "50" and which we canonicalized in Listing 9 (/2002/10/09/examples/Listing9.txt)) is the same as the booking element of Listing 10 (/2002/10/09/examples/Listing10.txt).

In order to demonstrate the role of canonicalization in federated web service applications, let's assume that the partner hotel wanted to sign the booking element of Listing 10 (/2002/10/09/examples/Listing10.txt) while sending the SOAP message to the tour operator, thus allowing the tourist to verify that the booking is not fake.

Listing 11 (/2002/10/09/examples/Listing11.txt) shows the canonical form of the booking element of Listing 10. This is the canonical form that the partner hotel will use to create a message digest. On the other hand, recall that Listing 9 (/2002/10/09/examples/Listing9.txt) was the canonical form of the same booking element, when it was part of Listing 7 (/2002/10/09/examples/Listing7.txt). Therefore, the tourist will use Listing 9 (/2002/10/09/examples/Listing9.txt) to verify the message digest of the partner hotel.

Compare Listing 11 (/2002/10/09/examples/Listing11.txt) with Listing 9 (/2002/10/09/examples/Listing9.txt) and you will find that they are different from each other. The difference comes from the fact that we conserved the ancestor contexts from two different XML documents while canonicalizing the same booking element.

Therefore, message digest and signature verification will fail at the tourist's client application end. This clearly establishes the need to *exclude* ancestor context while employing canonicalization concepts in federated web service applications. W3C has released the Exclusive XML Canonicalization recommendation for this purpose.

Exclusive canonicalization applies only while canonicalizing fragments of XML files and differs from (inclusive) canonical XML in the following two ways:

1. Attributes from the xml namespace are not imported from ancestors into orphan nodes.
2. Omitted namespace declarations are included in exclusive canonical form to an element only if:
   - The namespace declaration is used by the element or any or its child attributes.
   - The namespace declaration is not already in effect in the exclusive canonical form.

Note that the second point above also applies to empty default namespace declarations (`xmlns=""`). This means that the exclusive canonical form of an element will include the `xmlns=""` declaration if the elements belongs to the default empty namespace, and the nearest ancestor occurrence of the default namespace declaration in the exclusive canonical form has some non-empty default namespace (`xmlns="http://someURI..."`).

Applying these rules to the booking element of Listings 10 (/2002/10/09/examples/Listing10.txt), the exclusive canonical form comes to be as shown in Listing 12 (/2002/10/09/examples/Listing12.txt). You may notice that the exclusive canonical form of the first booking element of Listing 7 (/2002/10/09/examples/Listing7.txt) (whose `unitCharge` attribute has the value "50") is also exactly the same as Listing 12 (/2002/10/09/examples/Listing12.txt).

Problematic Scenarios

I should point at two important problems that may result by applying the Canonical XML specification:

1. If the XML file being canonicalized contains external parsed entity references, the external entity references will be replaced with external content during canonicalization as already discussed above under the heading "External Entity References." However, if the external content contains some relative URIs, the URIs may become non-operational after the replacement (since the DTD declaration will be removed during canonicalization and there will be no way to reach the external content after XML canonicalization).

   Non-operational URIs may create problems in signature applications, as there is no way to detect whether the original operational XML document or its non-operational canonical form was intended to be signed. If an application thinks that the purpose of signature applications may be defeated by this ambiguity, such scenarios should be resolved prior to canonicalization (e.g. relative URIs be converted to absolute URIs before starting to canonicalize).

2. There may be some application specific equivalence criteria that cannot be covered in a generalized specification. For example, an XML file carrying an invoice with all prices in French francs will not produce the same canonical form as that of the same invoice with equivalent prices in Euros (although the two invoices will be logically equivalent). Therefore, such application specific issues need to be resolved in an application specific manner.

Resources

- Read the first part (http://www.xml.com/pub/a/ws/2002/09/18/c14n.html) of this series.
- Check out the official specification for Exclusive XML Canonicalization (http://www.w3.org/TR/xml-exc-c14n) at W3C.
- We mentioned SOAP (http://www.w3.org/TR/2000/NOTE-SOAP-20000508/) and WS-Security (http://www-106.ibm.com/developerworks/webservices/library/ws-secure/). Interested readers may download and read their official specifications to find how WS-Security wraps XML Digital Signature (http://www.w3.org/TR/xmldsig-core/) info in SOAP headers.
- This page (http://www.w3.org/Signature/#Code) at W3C provides links to some implementations of Canonical XML specification.
- Implementing XPath for Wireless Devices (http://www.xml.com/pub/a/2002/06/05/wirelessxpath1.html), an article by the same author, which describes XPath syntax and its simple implementation

---