

XML Canonicalization

September 18, 2002

[Bilal Siddiqui \(/pub/au/140\)](#)

This two part series discusses the W3C Recommendations *Canonical XML* and *Exclusive XML Canonicalization*. In this first part I describe the process of XML canonicalization, that is, of finding the simplified form of an XML document, as defined by the Canonical XML specification. We'll start by illustrating when and why we would need to canonicalize an XML document.

Introduction

XML defines a format for structuring data so that information can be meaningfully interchanged between communicating parties. The rules for XML authoring are flexible in the sense that the same document structure and the same piece of information can be represented by different XML documents. Consider [Listings 1 \(/2002/09/18/examples/Listing1.txt\)](#) and [2 \(/2002/09/18/examples/Listing2.txt\)](#), which are logically equivalent, i.e. they follow the same document structure (the same XML Schema) and are meant to convey the same information. In spite of being logically equivalent, the XML files of Listings 1 and 2 do not contain the same sequence of characters (or sequence of bytes or octets).

In this case the character and octet sequences of the two XML files differ due to the order of attributes appearing in the `room` element. There can be other reasons for having different octet streams for logically equivalent XML documents. The purpose of finding the canonical (or simplified) form of an XML document is to determine logical equivalence between XML documents. W3C has defined canonicalization rules such that the canonical form of two XML documents will be the same if they are logically equivalent.

Whenever we are required to determine whether two XML documents are logically equivalent, we will canonicalize each of them and compare the canonical forms octet-by-octet. If the two canonical forms contain the same sequence of octets, we will conclude that the two XML files are logically equivalent.

Before we start exploring the technical details of the canonicalization process, let's see when and why you would need to test logical equivalence between XML documents.

The Need to Find Logical Equivalence between XML Documents

XML Digital Signatures is a W3C recommendation defining an XML format for signing XML (or non-XML) documents. Signing XML documents can be considered analogous to signing paper documents in ink. If you ever need to sign paper documents (e.g. bank checks) while

doing business in the conventional way, you'll likely need to sign digital documents while doing business electronically over the Internet.

Let's consider the process of signing a bank check. A bank check is part of a system that is designed to handle the following two critical requirements:

- **Integrity:** A check, once signed and issued cannot be edited by an unauthorized party. If someone tries to modify the amount written on a check, he will damage the background imprint on the check. The background imprint on the check, therefore, helps in maintaining confidence in the integrity of bank checks.
- **Non-repudiation:** The signing authority cannot deny the act of issuing the check. The signatures on the check are verifiable against sample signatures lying with the bank.

The two requirements are exactly the same in XML Digital Signatures. The background imprint on a bank check is analogous to a Message Digest algorithm, while the sample signatures lying with the bank are analogous to a private-public key pair in a PKI trust service.

Message Digests and PKI are very common terms used in literature covering security over the Internet. However, for those XML programmers who are new to security issues, the following is a brief description of these terms. For more details, I've included a reference at the end of the article.

Message Digests

Digest algorithms act upon and digest (consume) message octets to calculate a digest value. Digest values depend upon the message digest algorithm and the message itself. If an intruder modifies the message, its digest value can be used to detect the modification. Therefore message digests are the electronic way of enforcing integrity.

Public Key Infrastructure (PKI) Trust Services

Trust services produce and maintain private-public key pairs. Trust services are the custodians of private-public key pairs, just like your bank is the custodian of your money as well as your account details.

Private keys are kept confidential i.e. only the trust service and the user (signing authority) know the value of the private key. On the other hand public keys are open to everybody. A signer will use his private key to sign a message electronically along with the message digest value. Anyone can use the corresponding public key to verify the signature and the message digest. Public keys can only be used to verify signatures; they cannot be used to produce signatures. Therefore, as long as your private key is confidential, unauthorized signatures cannot be produced.

Signing Digital Data

When a user wants to sign an XML document, there is a two step procedure. First, the user digests the XML file to be signed and produces a digest value. Secondly, the user signs a reference to the XML message and the digest with the user's private key.

Verifying Signatures

An application receiving the signed message will validate the signature using signer's public key and will verify the integrity of the message by checking the message digest. The process of digesting an XML message relies on the sequence of bytes that represent an XML message. It is actually the sequence of bytes that produces a message digest value and the signatures. This means that Listings 1 and 2, although representing the same information, will not produce the same digest value. If you sign Listing 1 and the receiving party tries to verify Listing 2 against your signature (which makes perfect sense, since the two documents are carrying the same information), the verification process will fail.

Canonicalizing Before Signing and Verifying

Canonical XML finds its application in this scenario. The signing authority will digest and sign the canonical form of the document instead of the original form. Similarly, at the time of verification, the canonical form will be verified instead of the original XML. Thus all logically equivalent versions of the signed XML document will result in successful verification (validation) of XML digital signatures.

Canonicalizing an XML document

The canonical XML specification has defined an algorithm to author the canonical form of XML documents. You will need to perform the following steps in order to canonicalize an XML document:

1. Encoding Scheme

All XML documents are composed of human readable text, which is a sequence of characters. Encoding schemes are meant to represent characters by octets. Therefore, the same XML file can be represented by an entirely different octet stream just by changing the character encoding of the XML file.

The canonical XML specification dictates that the canonical form of XML documents should be encoded in UTF-8 encoding. Therefore, if the XML file to be canonicalized has any other encoding, it should be changed to UTF-8.

2. Line Breaks

Line breaks in text files are normally represented either by hexadecimal A (decimal 10) or hexadecimal D (decimal 13) or a combination of these two octets. XML files are all simple text files, therefore #xA and #xD are used as line breaks in all XML files.


The canonical form of XML requires that all line breaks (`#xD` or a combination of `#xA` and `#xD`) be replaced with `#xA`. This should be done before starting to process the XML file.

3. Attribute values are normalized

All attributes are required to be normalized in canonical form, as if by a validating XML parser. The process of attribute value normalization is stated in the XML 1.0 recommendation by W3C (see Resources).

A simple example of attribute value normalization is demonstrated by [Listings 3 \(/2002/09/18/examples/Listing3.txt\)](#) and [4 \(/2002/09/18/examples/Listing4.txt\)](#). Listing 3 is the original XML file before attribute value normalization, while Listing 4 shows all attribute values in normalized form.

All attributes in [Listing 3 \(/2002/09/18/examples/Listing3.txt\)](#) are of string type without any entity and character references. In this case, attribute value normalization simply means the normalization of white space (all types of white space i.e. tabs, line breaks and normal non-breaking space should be converted to `#x20`, which is the octet that represents non-breaking white space). All `id` attributes in [Listing 3 \(/2002/09/18/examples/Listing3.txt\)](#) have two tabs in them. Each occurrence of tab in the `id` attribute in [Listing 3 \(/2002/09/18/examples/Listing3.txt\)](#) value has been changed to a space in [Listing 4 \(/2002/09/18/examples/Listing4.txt\)](#).

[Next Page](#)  [\(\)](#)

XML Canonicalization

by Bilal Siddiqui |

4. Double quotes for Attribute values

Only double quotes should be used to encapsulate attribute values in canonical form. Have a look at [Listing 4 \(/2002/09/18/examples/Listing4.txt\)](#), where the `name` attribute of the `product` element is enclosed inside single quotes. In [Listing 5 \(/2002/09/18/examples/Listing5.txt\)](#), we have replaced the single quotes around the `name` attribute value with double quotes.

5. Special Characters in Attribute Values and Character Content

When we replaced single quotes with double quotes, we introduced a problem in [Listing 5 \(/2002/09/18/examples/Listing5.txt\)](#). It is no longer a well formed XML file, as the string representing value of the `name` attribute already contained double quotes as part of the string value. In order to solve this problem, the Canonical XML specification requires that all special characters (e.g. double quotes) in attribute values and element content be replaced

with character entities (e.g. " for double quotes). [Listing 6 \(/2002/09/18/examples/Listing6.txt\)](#) is the result of applying this rule to [Listing 5 \(/2002/09/18/examples/Listing5.txt\)](#).

6. Entity References

[Listing 6 \(/2002/09/18/examples/Listing6.txt\)](#) contains a DTD declaration, which defines an entity named *testhistory*. The *testhistory* entity is referenced by the *comments* element content.

Canonical XML requires that all entity references be replaced with the content represented by the entity (e.g. in [Listing 6 \(/2002/09/18/examples/Listing6.txt\)](#), the *testhistory* entity represents the string "Part has been tested according to the specified standards."). [Listing 7 \(/2002/09/18/examples/Listing7.txt\)](#) is the resulting XML file after entity references in [Listing 6 \(/2002/09/18/examples/Listing6.txt\)](#) have been replaced.

7. Default Attributes

The DTD declaration in [Listing 7 \(/2002/09/18/examples/Listing7.txt\)](#) defines a default attribute named *approved* for each *part* element. None of the *part* tags in [Listing 7](#) contains this attribute.

Canonical XML requires that default attributes should be included in the canonical XML form. [Listing 8 \(/2002/09/18/examples/Listing8.txt\)](#) is the result of including the *approved* attribute with default value in [Listing 7 \(/2002/09/18/examples/Listing7.txt\)](#).

8. XML and DTD declarations

Canonical XML does not require XML and DTD declarations. Therefore XML and DTD declarations should be removed in the canonical form. Although we have used the DTD declaration while replacing entity references and adding default attributes, the actual XML and DTD declarations need to be removed as shown in [Listing 9 \(/2002/09/18/examples/Listing9.txt\)](#).

9. White Space outside the Document Element

A Canonical XML document starts with the '<' character. This means that there should be no white space before the first node.

10. White Space in Start and End Elements

Start and End elements should have normalized white space in canonical form. This means there should be:

- No white space between the left angle bracket ('<') and the name of a start element. Similarly there should be no space between a slash ('/') and the name of an end element.
- A single #x20 character between the element name and the first attribute name, if present.

- No white space before and after the equality sign in attribute-value pairs.
- A single #x20 character between attribute-value pairs.
- No white space following the closing double quote of the last attribute's value.
- If there are no attributes, there should be no white space between the element name and the right angle bracket '>'.

[Listing 10 \(/2002/09/18/examples/Listing10.txt\)](#) is the result of normalizing white space in start and end elements of [Listing 9 \(/2002/09/18/examples/Listing9.txt\)](#).

11. Empty Elements

Canonical XML requires start-end tag pairs for all elements, which includes empty elements as well. Therefore, all empty elements of the form `<emptyElement/>` need to be converted to `<emptyElement></emptyElement>`. [Listing 11 \(/2002/09/18/examples/Listing11.txt\)](#) shows the result of applying this rule to [Listing 10 \(/2002/09/18/examples/Listing10.txt\)](#).

12. Namespace Declarations

[Listing 11](#) contains three namespace declarations, two in the *product* element and one in the second *part* element. Canonical XML requires preserving all namespace declarations as such (along with the namespace prefixes) except superfluous namespace declarations (those namespace declarations that have no effect on the namespace context of any node in the XML file).

The namespace declared in the second *part* element in [Listing 11](#) is superfluous. You can remove it from the element with no effect on the namespace context of any node in the file. That's why [Listing 12 \(/2002/09/18/examples/Listing12.txt\)](#) does not include this namespace declaration, while preserving the rest of [Listing 11 \(/2002/09/18/examples/Listing11.txt\)](#) as such.

13. Ordering of Namespace Declarations and Attributes

Canonical XML requires the inclusion of namespace declarations and attributes in ascending lexicographic order. Inside an opening element, all namespace declarations should appear first, followed by the attribute-value pairs. [Listing 13 \(/2002/09/18/examples/Listing13.txt\)](#) shows how [Listing 12 \(/2002/09/18/examples/Listing12.txt\)](#) will look like after the ordering rule is applied.

[Listing 13 \(/2002/09/18/examples/Listing13.txt\)](#) is the final canonical form of all listings from 3 to 12.

Just to give you a bit of variety, we have provided another example in the [Listing 14 \(/2002/09/18/examples/Listing14.txt\)](#) and [15 \(/2002/09/18/examples/Listing15.txt\)](#) pair (Listing 15 is the canonical form of Listing 14). We didn't produce Listing 15 by hand. We rather generated it using the Canonical XML implementation by IBM alphaWorks, which is

part of their XML Security Suite (refer to resources). However, curious readers may start with [Listing 14 \(/2002/09/18/examples/Listing14.txt\)](/2002/09/18/examples/Listing14.txt) and follow the thirteen steps described above to arrive at [Listing 15 \(/2002/09/18/examples/Listing15.txt\)](/2002/09/18/examples/Listing15.txt).

Note: There is no DOCTYPE declaration in [Listing 14 \(/2002/09/18/examples/Listing14.txt\)](/2002/09/18/examples/Listing14.txt). Therefore, some of the canonicalization steps such as replacing entity references and adding default attributes are not relevant in this case.

Conclusion

In the second article in this series, we will take this concept further and discuss more advanced concepts such as dealing with parts of XML documents, CDATA sections, comments and processing instructions. We will also discuss tricky situations where canonicalization process renders XML documents useless for their intended function.

Resources

- The official specification of [Canonical XML \(http://www.w3.org/TR/xml-c14n\)](http://www.w3.org/TR/xml-c14n) is available at W3C.
- Download and try [IBM's XML Security Suite \(http://www.alphaworks.ibm.com/tech/xmlsecuritysuite\)](http://www.alphaworks.ibm.com/tech/xmlsecuritysuite).
- We mentioned [XML \(http://www.w3.org/TR/2000/REC-xml-20001006\)](http://www.w3.org/TR/2000/REC-xml-20001006), [Namespaces \(http://www.w3.org/TR/1999/REC-xml-names-19990114/\)](http://www.w3.org/TR/1999/REC-xml-names-19990114/) and [Schema \(http://www.w3.org/XML/Schema#dev\)](http://www.w3.org/XML/Schema#dev) in this article. Check them out at [W3's official web site \(http://www.w3.org\)](http://www.w3.org).
- Read the official specification of [XML Digital Signatures \(http://www.w3.org/TR/xmlsig-core/\)](http://www.w3.org/TR/xmlsig-core/).
- Check out this series of [articles \(http://www-106.ibm.com/developerworks/library/s-crypt07.html\)](http://www-106.ibm.com/developerworks/library/s-crypt07.html) on cryptography that discusses keys, digital signatures, message digests and other important aspects of security over the Internet.

Content licensed from and © 1998 - 2008 O'Reilly Media, Inc.