Exchanger XML Editor - Canonicalization and XML Digital Signatures

Table of Contents

Exchanger XML Editor - Canonicalization and XML Digital Signatures

XML Canonicalization

Exchanger XML Editor supports Inclusive and Exclusive canonicalization of documents according to the W3C Canonicalization specifications.

Inclusive Canonicalization

The following list contains a summary of the possible changes the input document undergoes during Inclusive Canonicalization:

- The document is encoded in UTF-8
- Line breaks are normalized to #xA
- Attribute values are normalized (Character and parsed entity references are replaced; White space characters (#xD, #xA, #x9) are replaced by the space character #x20; If the attribute type is not CDATA, any leading and trailing space (#x20) characters are discarded, and any sequences of space (#x20) characters are replaced by a single space (#x20) character.) All attributes for which no declaration has been read should be treated by a non-validating processor as if declared CDATA. Note: a validating parser will produce different output to one that just checks well-formedness since leading/trailing spaces are discarded and sequences of spaces are collapsed for non-CDATA atrributes by the validating parser - the canonicalisation spec requires that attribute values are normalized "as if by a validating processor".
- Character and parsed entity references in content are replaced
- The XML declaration is removed
- Whitespace outside of the document element is normalized
- All whitespace in character content is retained (excluding characters removed during line feed normalization)
- Default attributes are added to each element
- A DOCUMENT_TYPE node is discarded.
- Processing Instructions (PIs) before the document element are output followed by a new line each. PIs after the document element are output preceeded by a new line each. PIs are output with a single space between the target name and the value, if there is a value otherwise no space is output. The character #x0d is replaced by the character entity
- Comments before the document element are output followed by a new line. Comments after the document element are output preceeded by a new line. The character #x0d is replaced by the character entity
- Text and CDATA nodes are output as text, with special characters replaced by character references: & replaced by &amp; < replaced by &lt; > replaced by &gt; and the character #x0d is replaced by the character entity &#xD;
- Empty elements are converted to start-end tag pairs.
- Whitespace within start and end tags is normalized.
- Attribute value delimiters are set to quotation marks (double quotes)

- Special characters in attribute values are replaced by character references: & replaced by &amp; < replaced by &lt; > replaced by &gt; and the character #x0d is replaced by the character entity &#xD;

- Relative namespaces are not allowed and will cause the canonicalization to fail.

- Namespaces and attributes: any superfluous redeclaration of namespaces or xml: attributes are dropped from the output - for example, if an element has an xml:space="preserve" and an ancestor has the same declaration without an intervening xml:space="default", then the xml:space attribute is dropped from the current element (and similarly for redeclared namespaces).

- Namespace and attribute ordering: namespaces are output before attributes. Namespaces are ordered based on the local name (prefix), with the default namespace, if it exists, being placed last because it has no local name. Attributes are ordered based on the namespace URI (not prefix!) as the primary key and the local name as the secondary key, with attributes in no namespace being placed last (remember, default namespace does not apply to attributes).

Inclusive Canonicalization Example

Open the file `input/books.xml` in the **Inclusive Canonicalization** project and select **Security->Canonicalize**. In the dialog, select **Inclusive**, send the output **To New Document** and click **OK**. This example shows up a number of canonicalization features, including entity and CDATA replacement, double quotes replacing single quotes around attribute values, attributes being ordered alphabetically, XML Declaration and DOCTYPE stripping, etc.

XPath and Canonicalization

XPaths can also be used to specify what portion of a document is to be canonicalized. Note that an XPath such as **//person** only identifies individual nodes (the apex nodes of subtrees starting at the **person** elements), whereas a construction similar to **(//. | //@* | //namespace::*)[ancestor-or-self::person]** is needed to describe the list of nodes contained within the **person** elements (for more details, see the Canonicalization specification). When using XPaths in Exchanger for Canonicalization or XML Signatures, you only need to enter the **predicate**, i.e. the part inside the square brackets **[]** which in the current example is **ancestor-or-self::person**.

If you need to use namespace prefixes in your XPath predicates in Canonicalization, then you must declare them to the Editor either in the Canonicalize dialog or through the XML Preferences window (available via **File->Preferences** in the XML tab). See the section on using XPaths in Exchanger at the beginning of this document and the Preferences section at the end for more details on using namespace prefix mapping.

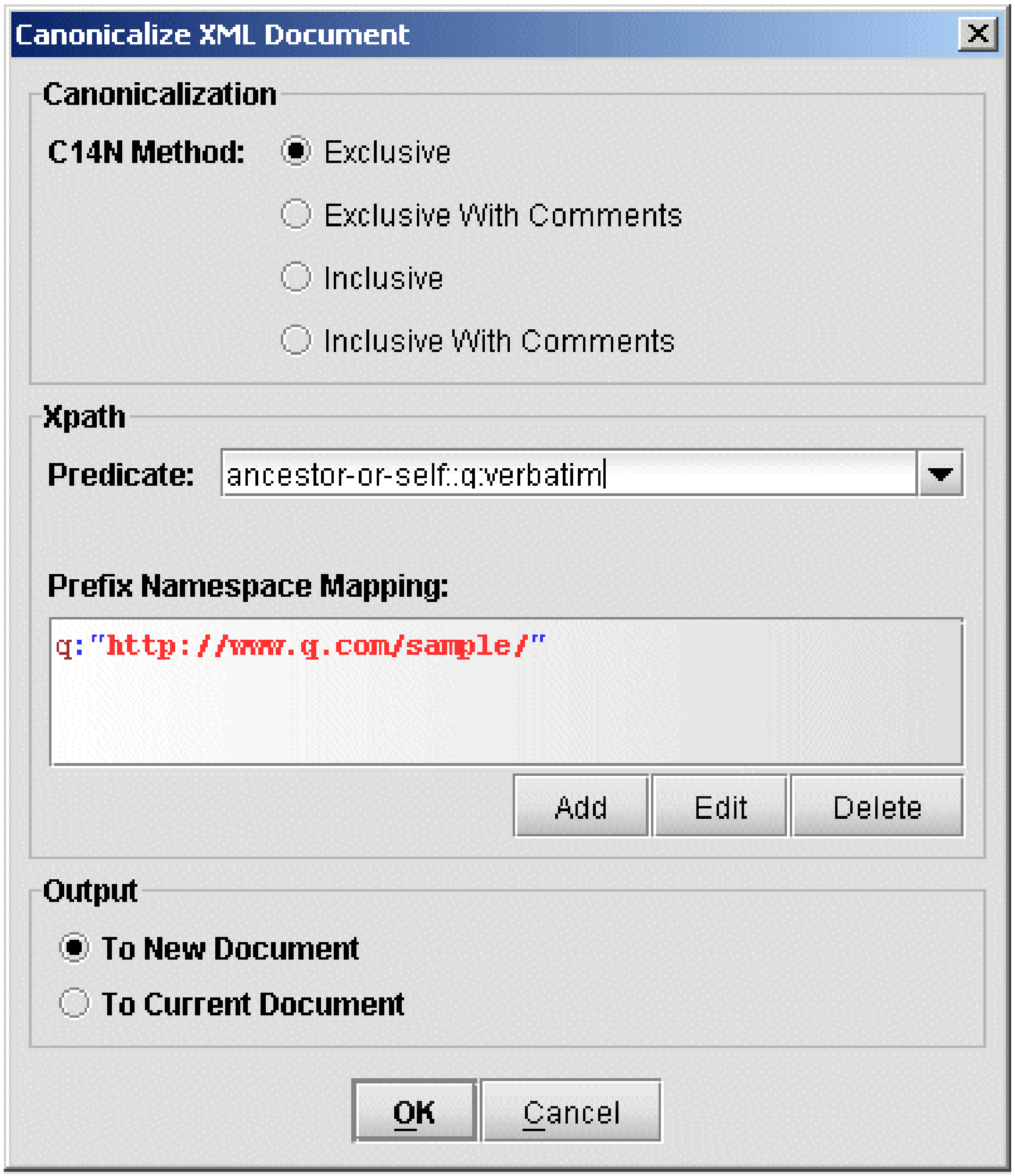
Exclusive Canonicalization

Exclusive canonicalization deals with the canonicalization of document subsets and attempts to isolate the subset from its context - this is very important when content of one XML document type is embedded (or "enveloped") within another document type such as SOAP. Exclusive canonicalization adds extra restrictions, above and beyond the regular canonicalization issues outlined above.

- Visibly utilized namespaces: Namespaces are not automatically output on an element unless they are actually used on the element itself - this holds both for namespaces delared on ancestor elements and also for those declared on the element itself. If an unused namespace is used on a subsequent descendant, the appropriate namespace declaration is only output at that point.
- Included namespaces: specific namespace declarations can be explicitly output by supplying them to the exclusive canonicalization algorithm in the InclusiveNamespaces PrefixList. This is useful when a prefix is used in an attribute value (say, as part of an XPath specification) and would not be "visibly used" by default.
- xml: attributes that have been used on ancestors that are not part of the document subset are NOT output on the apex nodes of the document subset.

Exclusive Canonicalisation Examples

Open the file `input/xc14ntest.xml` in the **Exclusive Canonicalization** project and select **Security->Canonicalize**. In this example, we are going to exclusively canonicalize the **verbatim** element contents and see how the namespaces are affected. In the dialog, select **Exclusive** and set the XPath Predicate to **ancestor-or-self::q:verbatim**. (Note: you will need to add a Namespace Prefix Mapping entry with Prefix equals to **q** and URI equals **http://www.q.com/sample/** and send the output **To New Document**.



Canonicalize XML Document

Canonicalization

C14N Method: [✓ Exclusive] Exclusive With Comments Inclusive Inclusive With Comments

Xpath

Predicate: ancestor-or-self::q:verbatim

Prefix Namespace Mapping:

```
q:"http://www.q.com/sample/"
```

Output

[✓ To New Document] To Current Document

Figure 1. Exclusive Canonicalization

In the output (also available in the `output` directory in the project) notice how namespace declarations are only output when needed. Also note that while the **w** prefix is used in an attribute value on the **b** element, the namespace declartion is not output on the **b** element as the namepsace is not "visibly used" but is output on the **i** element where it is visible.

XML Signatures

It is possible to create and verify a digital signature in Exchanger XML Editor according to the XML Digital Signature specification.

How to Create and Verify a Signature

Open the file `input/full.xml` in the **Envelope XML Signature** project. To create a signature for this document, select **Security->Sign Document...** and the following dialog will appear.

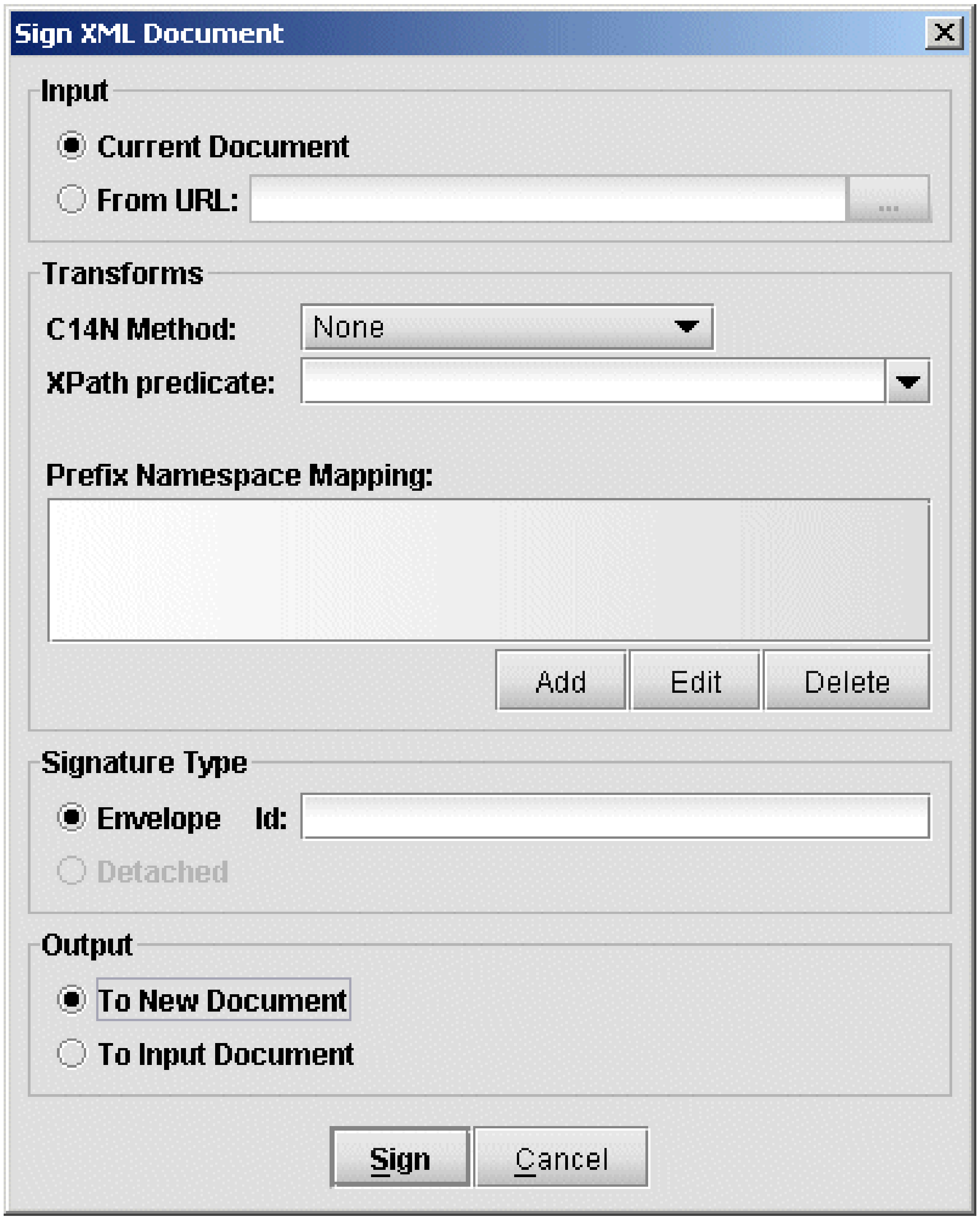


Figure 2. Sign Document

Ensure that **Current Document** is selected as Input and in the Transforms section of the dialog, leave the C14N Method with its default setting (None) and leave the XPath field blank for now. Make sure that the output is set **To New Document** so as to not overwrite the existing file and then press **Sign**. A new document will open, consisting of the original document and a new Signature element made up of:

- A SignedInfo element with details of how the signature was generated.
- A SignatureValue element.
- A KeyInfo element with details of the key used. In this instance, the document was signed using a key that is shipped with the Exchanger XML Editor as can be seen in the SubjectName element: <ds:X509SubjectName>CN=Exchanger,OU=Development,O=Cladonia,C=IE</ds:X509SubjectName>

To verify that the signature is correct, just select **Security->Verify Signature** and a dialog should popup to say that the signature is valid. Now change the data in the new document (for example, change the second person's name from Jane Smith to Jane Jones) and verify the signature again. This time you will be notified that the signature is no longer valid.

Signing using an ID attribute value

Instead of signing an entire document, it is sometimes desirable to sign a particular subsection, for example, a manager may wish to sign the part of an employee's purchase order that deals with approval. One way to do this is using an ID attribute on the element that should be signed. Open the file `input/id.xml` in the **Envelope XML Signature** project - notice that this document is the same as in the preceding example, except now each Person element has an extra "id" attribute. Select **Security->Sign Document...** but this time enter the value **123** in the Id section of the Signature Type field in order to sign the first Person element in the document (containing the details of John Doe). Again, select **To New Document** for the output so as not to modify the original file and press Sign. In the new document, modify the second Person element, changing the name from Jane Smith to Jane Jones and select **Security->Verify Signature**. The signature is still reported as valid even though the overall document has been changed. Now modify the first Person element, changing the name from John Doe to Tom Doe and again select **Security->Verify Signature**. This time the signature is invalid because the element to which it relates has been changed. Notice in the SignedInfo section that the **<ds:Reference URI="#123">** element identifies which element has been signed. By default, when a schema/DTD is not present, only attributes named "id", "ID" and "Id" can be used for generating signtures in this way.

Signing using an XPath

XPaths can also be used to specify what portion of a document is to be signed. Note that an XPath such as **//section** only identifies individual **section** nodes, whereas a construction similar to **(//. | //@* | //namespace::*)[ancestor-or-self::section]** is needed to describe the list of nodes in each **section** element (for more details, see the Canonicalization specification and the XML Signature specification). To create a signature based on an XPath, open the file `input/xpath.xml` in the **Envelope XML Signature** project and select **Security->Sign Document...**. Ensure that the Id field is blank and in the XPath field in the Transform section, enter **ancestor-or-self::Salary** - this predicate is used by the signature code to refine the path **(//. | //@* | //namespace::*)**. Again, select To New Document for the output so as not to modify the original file and press Sign. Now modifying the Salary element will invalidate the signature while changes to the Name or SSN element will not.

Signature Type: Envelope versus Detached

In the preceding examples, we chose by default to create Envelope signatures, where the signature information is combined with the input data in a single file. It is also possible to create Detached signatures where the signature information lives in an entirely separate file from the input. Because the detached signature needs a URI for the file being signed, there is a slight restriction in Exchanger XML Editor that only files on disk can be signed using detached signatures (simply save any changes to an open document before creating a detached signature). Select **Security->Sign Document...** and set the input to **From URL** and select the desired XML file (for example, `http://www.exchangerxml.com/Cladonia.gif`). Set the Signature Type to Detached, set the appropriate XPath if required (the output is automatically set To New Document) and press Sign. The detached signature is displayed in a new window and can be saved to file for future use. (An example of a detached signature is available in the **Detached XML Signature** project in the `signed` folder.)

Note: Exchanger XML Editor does not currently support Enveloping signatures, the third type of signature defined in the XML Signature specification.

Signing large or non-XML documents

Any file can be signed, irrespective of whether it is XML or not, by specifying the URI and creating a detached signature with C14N Method set to None and not specifying an XPath transformation. Note that if an XML file is signed in this way, then no changes can be made to it (even non-significant whitespace changes) without invalidating the signature. Also, the URI of the input file is encoded in the detached signature so the file cannot be moved without invalidating the signature also. However, using a detached signature with extremely large XML files is sometimes the only possibility as the requirement to read the entire document into memory for a canonicalization or XPath transform can cause significant performance problems and with larger documents will cause out of memory errors (an expansion factor of between 4 and 10 is typical when building a DOM from an XML file on disk.)

C14N Method

The **C14N Method** drop down in the Transform section of the **Sign Document...** dialog specified which canonicalization method should be used. The choices are Inclusive Canonicalization removing comments, Inclusive Canonicalization preserving comments, Exclusive Canonicalization removing comments, Exclusive Canonicalization preserving comments and None. By default, the method is set to None. For more information on Canonicalization in general, see the earlier section.