# Canonical XML Version 1.1

## W3C Recommendation 2 May 2008

**This version:**
http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/
**Latest version:**
http://www.w3.org/TR/xml-c14n11/
**Previous version:**
http://www.w3.org/TR/2008/PR-xml-c14n11-20080129/
**Authors:**
John Boyer, IBM (formerly PureEdge Solutions Inc.) Version 1.0
Glenn Marcy, IBM

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.



---

## Abstract

Canonical XML Version 1.1 is a revision to Canonical XML Version 1.0 to address issues related to inheritance of attributes in the XML namespace when canonicalizing document subsets, including the requirement not to inherit `xml:id`, and to treat `xml:base` URI path processing properly.

Any XML document is part of a set of XML documents that are logically equivalent within an application context, but which vary in physical representation based on syntactic changes permitted by XML 1.0 [XML] and Namespaces in XML 1.0 [Names]. This specification describes a method for generating a physical representation, the canonical form, of an XML document that accounts for the permissible changes. Except for limitations regarding a few unusual cases, if two documents have the same canonical form, then the two documents are logically equivalent within the given application context. Note that two documents may have differing canonical forms yet still be equivalent in a given context based on application-specific equivalence rules for which no generalized XML specification could account.

Canonical XML Version 1.1 is applicable to XML 1.0 and defined in terms of the XPath 1.0 data model. It is not defined for XML 1.1.

## Status of this Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.*

This is a W3C Recommendation.

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Comments on this document should be sent to www-xml-canonicalization-comments@w3.org which is an automatically archived public email list.

The implementation report details CR implementation feedback from several implementations. It should be noted that this IR reflects results implemented against the CR as clarified based on issues raised during the CR period and subsequently reflected in the wording of this Recommendation.

This document has been produced by the W3C XML Core Working Group as part of the W3C XML Activity. The authors of this document are the members of the XML Core Working Group and invited experts from the Digital Signature community.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

The English version of this specification is the only normative version.

---

## Table of Contents

# 1 Introduction

The XML 1.0 Recommendation [XML] specifies the syntax of a class of resources called XML documents. The Namespaces in XML 1.0 Recommendation [Names] specifies additional syntax and semantics for XML documents. It is possible for XML documents which are equivalent for the purposes of many applications to differ in physical representation. For example, they may differ in their entity structure, attribute ordering, and character encoding. It is the goal of this specification to establish a method for determining whether two documents are identical, or whether an application has not changed a document, except for transformations permitted by XML 1.0 and Namespaces in XML 1.0.

Canonical XML Version 1.1 is a revision to Canonical XML Version 1.0 [C14N10] to address issues related to inheritance of attributes in the XML namespace when canonicalizing document subsets, including the requirement not to inherit `xml:id`, and to treat `xml:base` URI path processing properly. See also the Working Group Notes on [C14N-Issues] and [DSig-Usage] for further discussion of the relationship of Canonical XML Version 1.1 to Canonical XML Version 1.0.

Canonical XML Version 1.1 is applicable to XML 1.0 and defined in terms of the XPath 1.0 data model. It is not defined for XML 1.1.

## 1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [Keywords].

See [Names] for the definition of QName.

A *document subset* is a portion of an XML document indicated by a node-set that may not include all of the nodes in the document.

The *canonical form* of an XML document is physical representation of the document produced by the method described in this specification. The changes are summarized in the following list:

- The document is encoded in UTF-8
- Line breaks normalized to #xA on input, before parsing
- Attribute values are normalized, as if by a validating processor
- Character and parsed entity references are replaced
- CDATA sections are replaced with their character content
- The XML declaration and document type declaration are removed
- Empty elements are converted to start-end tag pairs
- Whitespace outside of the document element and within start and end tags is normalized
- All whitespace in character content is retained (excluding characters removed during line feed normalization)
- Attribute value delimiters are set to quotation marks (double quotes)
- Special characters in attribute values and character content are replaced by character references
- Superfluous namespace declarations are removed from each element
- Default attributes are added to each element
- Fixup of `xml:base` attributes [C14N-Issues] is performed
- Lexicographic order is imposed on the namespace declarations and attributes of each element

The term *canonical XML* refers to XML that is in canonical form. The *XML canonicalization method* is

the algorithm defined by this specification that generates the canonical form of a given XML document or document subset. The term *XML canonicalization* refers to the process of applying the XML canonicalization method to an XML document or document subset.

The XPath 1.0 Recommendation [XPath] defines the term *node-set* and specifies a data model for representing an input XML document as a set of nodes of various types (element, attribute, namespace, text, comment, processing instruction, and root). The nodes are included in or excluded from a node-set based on the evaluation of an expression. Within this specification, a node-set is used to directly indicate whether or not each node should be rendered in the canonical form (in this sense, it is used as a formal mathematical set). A node that is excluded from the set is not rendered in the canonical form being generated, even if its parent node is included in the node-set. However, an omitted node may still impact the rendering of its descendants (e.g. by augmenting the namespace context of the descendants or supplying a base URI through `xml:base`).

## 1.2 Applications

Since the XML 1.0 Recommendation [XML] and the Namespaces in XML 1.0 Recommendation [Names] define multiple syntactic methods for expressing the same information, XML applications tend to take liberties with changes that have no impact on the information content of the document. XML canonicalization is designed to be useful to applications that require the ability to test whether the information content of a document or document subset has been changed. This is done by comparing the canonical form of the original document before application processing with the canonical form of the document result of the application processing.

For example, a digital signature over the canonical form of an XML document or document subset would allow the signature digest calculations to be oblivious to changes in the original document's physical representation, provided that the changes are defined to be logically equivalent by the XML 1.0 or Namespaces in XML 1.0. During signature generation, the digest is computed over the canonical form of the document. The document is then transferred to the relying party, which validates the signature by reading the document and computing a digest of the canonical form of the received document. The equivalence of the digests computed by the signing and relying parties (and hence the equivalence of the canonical forms over which they were computed) ensures that the information content of the document has not been altered since it was signed.

**Note:** Although not stated as a requirement on implementations, nor formally proved to be the case, it is the intent of this specification that if the text generated by canonicalizing a document according to this specification is itself parsed and canonicalized according to this specification, the text generated by the second canonicalization will be the same as that generated by the first canonicalization.

## 1.3 Limitations

Two XML documents may have differing information content that is nonetheless logically equivalent within a given application context. Although two XML documents are equivalent (aside from limitations given in this section) if their canonical forms are identical, it is not a goal of this work to establish a method such that two XML documents are equivalent if *and only if* their canonical forms are identical. Such a method is unachievable, in part due to application-specific rules such as those governing unimportant whitespace and equivalent data (e.g. `<color>black</color>` versus `<color>rgb(0,0,0)</color>`). There are also equivalencies established by other W3C Recommendations and Working Drafts. Accounting for these additional equivalence rules is beyond the scope of this work. They can be applied by the application or become the subject of future specifications.

The canonical form of an XML document may not be completely operational within the application context, though the circumstances under which this occurs are unusual. This problem may be of concern in certain applications since the canonical form of a document and the canonical form of the canonical form of the document are equivalent. For example, in a digital signature application, it cannot be established whether the operational original document or the non-operational canonical

form was signed because the canonical form can be substituted for the original document without changing the digest calculation. However, the security risk only occurs in the unusual circumstances described below, which can all be resolved or at least detected prior to digital signature generation.

The difficulties arise due to the loss of the following information not available in the data model:

1. base URI, especially in content derived from the replacement text of external general parsed entity references
2. notations and external unparsed entity references
3. attribute types in the document type declaration

In the first case, note that a document containing a relative URI [URI] is only operational when accessed from a specific URI that provides the proper base URI. In addition, if the document contains external general parsed entity references to content containing relative URIs, then the relative URIs will not be operational in the canonical form, which replaces the entity reference with internal content (thereby implicitly changing the default base URI of that content). Both of these problems can typically be solved by adding support for the `xml:base` attribute [XBase] to the application, then adding appropriate `xml:base` attributes to document element and all top-level elements in external entities. In addition, applications often have an opportunity to resolve relative URIs prior to the need for a canonical form. For example, in a digital signature application, a document is often retrieved and processed prior to signature generation. The processing SHOULD create a new document in which relative URIs have been converted to absolute URIs, thereby mitigating any security risk for the new document.

In the second case, the loss of external unparsed entity references and the notations that bind them to applications means that canonical forms cannot properly distinguish among XML documents that incorporate unparsed data via this mechanism. This is an unusual case precisely because most XML processors currently discard the document type declaration, which discards the notation, the entity's binding to a URI, and the attribute type that binds the attribute value to an entity name. For documents that must be subjected to more than one XML processor, the XML design typically indicates a reference to unparsed data using a URI in the attribute value.

In the third case, the loss of attribute types can affect the canonical form in different ways depending on the type. Attributes of type ID, other than the `xml:id` attribute, cease to be ID attributes. Hence, any XPath expressions that refer to the canonical form using the `id()` function cease to operate. The attribute types ENTITY and ENTITIES are not part of this case; they are covered in the second case above. Attributes of enumerated type and of type ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, and NOTATION fail to be appropriately constrained during future attempts to change the attribute value if the canonical form replaces the original document during application processing. Applications can avoid the difficulties of this case by ensuring that an appropriate document type declaration is prepended prior to using the canonical form in further XML processing. This is likely to be an easy task since attribute lists are usually acquired from a standard external DTD subset, and any entity and notation declarations not also in the external DTD subset are typically constructed from application configuration information and added to the internal DTD subset.

While these limitations are not severe, it would be possible to resolve them in a future version of XML canonicalization if, for example, a new version of XPath were created based on the XML Information Set [Infoset] currently under development at the W3C.

# 2 XML Canonicalization

## 2.1 Data Model

The data model defined in the XPath 1.0 Recommendation [XPath] is used to represent the input XML document or document subset. Implementations SHOULD but need not be based on an XPath implementation. XML canonicalization is defined in terms of the XPath definition of a node-set, and

implementations MUST produce equivalent results.

The first parameter of input to the XML canonicalization method is either an XPath node-set or an octet stream containing a well-formed XML document. Implementations MUST support the octet stream input and SHOULD also support the document subset feature via node-set input. For the purpose of describing canonicalization in terms of an XPath node-set, this section describes how an octet stream is converted to an XPath node-set.

The second parameter of input to the XML canonicalization method is a boolean flag indicating whether or not comments should be included in the canonical form output by the XML canonicalization method. If a canonical form contains comments corresponding to the comment nodes in the input node-set, the result is called *canonical XML with comments*. Note that the XPath data model does not create comment nodes for comments appearing within the document type declaration. Implementations are REQUIRED to be capable of producing canonical XML excluding all comments that may have appeared in the input document or document subset. Support for canonical XML with comments is RECOMMENDED.

If an XML document must be converted to a node-set, XPath REQUIRES that an XML processor be used to create the nodes of its data model to fully represent the document. The XML processor performs the following tasks in order:

1. normalize line feeds
2. normalize attribute values
3. replace CDATA sections with their character content
4. resolve character and parsed entity references

The input octet stream MUST contain a well-formed XML document, but the input need not be validated. However, the attribute value normalization and entity reference resolution MUST be performed in accordance with the behaviors of a validating XML processor. As well, nodes for default attributes (declared in the ATTLIST with an AttValue but not specified) are created in each element. Thus, the declarations in the document type declaration are used to help create the canonical form, even though the document type declaration is not retained in the canonical form.

The XPath data model represents data using UCS characters. Implementations MUST use XML processors that support UTF-8 and UTF-16 and translate to the UCS character domain. For UTF-16, the leading byte order mark is treated as an artifact of encoding and stripped from the UCS character data (subsequent zero width non-breaking spaces appearing within the UTF-16 data are not removed) [UTF-16, Section 3.2]. Support for ISO-8859-1 encoding is RECOMMENDED, and all other character encodings are OPTIONAL.

All whitespace within the root document element MUST be preserved (except for any #xD characters deleted by line delimiter normalization). This includes all whitespace in external entities. Whitespace outside of the root document element MUST be discarded.

In the XPath data model, there exist the following node types: root, element, comment, processing instruction, text, attribute and namespace. There exists a single root node whose children are processing instruction nodes and comment nodes to represent information outside of the document element (and outside of the document type declaration). The root node also has a single element node representing the top-level document element. Each element node can have child nodes of type element, text, processing instruction, and comment. The attributes and namespaces associated with an element are not considered to be child nodes of the element, but they are associated with the element by inclusion in the element's attribute and namespace axes. Note that attribute and namespace axes may not directly correspond to the text appearing in the element's start tag in the original document.

**Note:** An element has attribute nodes to represent the non-namespace attribute declarations appearing in its start tag *as well as* nodes to represent the default attributes.

By virtue of the XPath data model, XML canonicalization is namespace-aware [Names]. However, it cannot and therefore does not account for namespace equivalencies using namespace prefix rewriting (see explanation in Section 4). In the XPath data model, each element and attribute has a name returned by the function `name()` which can, at the discretion of the application, be the QName appearing in the original document. XML canonicalization REQUIRES that the XML processor retain sufficient information such that the QName of the element as it appeared in the original document can be provided.

**Note:** An element ***E*** has namespace nodes that represent its namespace declarations *as well as* any namespace declarations made by its ancestors that have not been overridden in ***E***'s declarations, the default namespace if it is non-empty, and the declaration of the prefix `xml`.

**Note:** This specification supports the recent XML plenary decision to deprecate relative namespace URIs as follows: implementations of XML canonicalization MUST report an operation failure on documents containing relative namespace URIs. XML canonicalization MUST NOT be implemented with an XML parser that converts relative URIs to absolute URIs.

Character content is represented in the XPath data model with text nodes. All consecutive characters are placed into a single text node. Furthermore, the text node's characters are represented in the UCS character domain. The XML canonicalization method does not perform character model normalization (see explanation in Section 4). However, the XML processor used to prepare the XPath data model input is REQUIRED to use Unicode Normalization Form C [NFC, NFC-Corrigendum] when converting an XML document to the UCS character domain from any encoding that is not UCS-based (currently, UCS-based encodings include UTF-8, UTF-16, UTF-16BE, and UTF-16LE, UCS-2, and UCS-4).

Since XML canonicalization converts an XPath node-set into a canonical form, the first parameter MUST either be an XPath node-set or it must be converted from an octet stream to a node-set by performing the XML processing necessary to create the XPath nodes described above, then setting an initial XPath evaluation context of:

- A **context node**, initialized to the root node of the input XML document.
- A **context position**, initialized to 1.
- A **context size**, initialized to 1.
- Any **library of functions** conforming to the XPath Recommendation.
- An empty set of **variable bindings**.
- An empty set of **namespace declarations**.

and evaluating the following default expression:

| Comment Parameter Value | Default XPath Expression |
|---|---|
| Without (false) | `(//. \| //@* \| //namespace::*)[not(self::comment())]` |
| With (true) | `(//. \| //@* \| //namespace::*)` |

The expressions in this table generate a node-set containing every node of the XML document (except the comments if the comment parameter value is false).

If the input is an XPath node-set, then the node-set must explicitly contain every node to be rendered to the canonical form. For example, the result of the XPath expression `id("E")` is a node-set containing only the node corresponding to the element with an ID attribute value of "E". Since none of its descendant nodes, attribute nodes and namespace nodes are in the set, the canonical form would consist solely of the element's start and end tags, less the attribute and namespace declarations, with no internal content. Section 3.7 exemplifies how to serialize an identified element along with its internal content, attributes and namespace declarations.

## 2.2 Document Order

Although an XPath node-set is defined to be unordered, the XPath 1.0 Recommendation [XPath] defines the term *document order* to be the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities, except for namespace and attribute nodes whose document order is application-dependent.

The XML canonicalization method processes a node-set by imposing the following additional document order rules on the namespace and attribute nodes of each element:

- An element's namespace and attribute nodes have a document order position greater than the element but less than any child node of the element.
- Namespace nodes have a lesser document order position than attribute nodes.
- An element's namespace nodes are sorted lexicographically by local name (the default namespace node, if one exists, has no local name and is therefore lexicographically least).
- An element's attribute nodes are sorted lexicographically with namespace URI as the primary key and local name as the secondary key (an empty namespace URI is lexicographically least).

Lexicographic comparison, which orders strings from least to greatest alphabetically, is based on the UCS codepoint values, which is equivalent to lexicographic ordering based on UTF-8.

## 2.3 Processing Model

The XPath node-set is converted into an octet stream, the canonical form, by generating the representative UCS characters for each node in the node-set in ascending document order, then encoding the result in UTF-8 (without a leading byte order mark). No node is processed more than once. Note that processing an element node ***E*** includes the processing of all members of the node-set for which ***E*** is an ancestor. Therefore, directly after the representative text for ***E*** is generated, ***E*** and all nodes for which ***E*** is an ancestor are removed from the node-set (or some logically equivalent operation occurs such that the node-set's next node in document order has not been processed). Note, however, that an element node is not removed from the node-set until after its children are processed.

The result of processing a node depends on its type and on whether or not it is in the node-set. If a node is not in the node-set, then no text is generated for the node except for the result of processing its namespace and attribute axes (elements only) and its children (elements and the root node). If the node is in the node-set, then text is generated to represent the node in the canonical form in addition to the text generated by processing the node's namespace and attribute axes and child nodes.

**NOTE:** The node-set is treated as a set of nodes, not a list of subtrees. To canonicalize an element including its namespaces, attributes, and content, the node-set must actually contain all of the nodes corresponding to these parts of the document, not just the element node.

The text generated for a node is dependent on the node type and given in the following list:

- **Root Node-** The root node is the parent of the top-level document element. The result of processing each of its child nodes that is in the node-set in document order. The root node does not generate a byte order mark, XML declaration, nor anything from within the document type declaration.
- **Element Nodes-** If the element is not in the node-set, then the result is obtained by processing the namespace axis, then the attribute axis, then processing the child nodes of the element that are in the node-set (in document order). If the element is in the node-set, then the result is an open angle bracket (<), the element QName, the result of processing the namespace axis, the result of processing the attribute axis, a close angle bracket (>), the result of processing the child nodes of the element that are in the node-set (in document order), an open angle bracket, a forward slash (/), the element QName, and a close angle bracket.

- *Namespace Axis*- Consider a list ***L*** containing only namespace nodes in the axis and in the node-set in lexicographic order (ascending). To begin processing ***L***, if the first node is not the default namespace node (a node with no namespace URI and no local name), then generate a space followed by `xmlns=""` *if and only* if the following conditions are met:

  - the element ***E*** that owns the axis is in the node-set
  - The nearest ancestor element of ***E*** in the node-set has a default namespace node in the node-set (default namespace nodes always have non-empty values in XPath)

  The latter condition eliminates unnecessary occurrences of `xmlns=""` in the canonical form since an element only receives an `xmlns=""` if its default namespace is empty and if it has an immediate parent in the canonical form that has a non-empty default namespace. To finish processing ***L***, simply process every namespace node in ***L***, except omit namespace node with local name `xml`, which defines the `xml` prefix, if its string value is `http://www.w3.org/XML/1998/namespace`.

- *Attribute Axis*- In lexicographic order (ascending), process each node that is in the element's attribute axis and in the node-set.

- **Namespace Nodes-** A namespace node ***N*** is ignored if the nearest ancestor element of the node's parent element that is in the node-set has a namespace node in the node-set with the same local name and value as ***N***. Otherwise, process the namespace node ***N*** in the same way as an attribute node, unless it is the default namespace node. For the default namespace node, use `xmlns` for the text of the local name in place of the empty local name (in XPath, the default namespace node has an empty URI and local name).
- **Attribute Nodes-** a space, the node's QName, an equals sign, an open quotation mark (double quote), the modified string value, and a close quotation mark (double quote). The string value of the node is modified by replacing all ampersands (&) with `&`, all open angle brackets (<) with `<`, all quotation mark characters with `&quot;`, and the whitespace characters #x9, #xA, and #xD, with character references. The character references are written in uppercase hexadecimal with no leading zeroes (for example, #xD is represented by the character reference `&#xD;`).
- **Text Nodes-** the string value, except all ampersands are replaced by `&`, all open angle brackets (<) are replaced by `<`, all closing angle brackets (>) are replaced by `>`, and all #xD characters are replaced by `&#xD;`.
- **Processing Instruction (PI) Nodes-** The opening PI symbol (`<?`), the PI target name of the node, a leading space and the string value if it is not empty, and the closing PI symbol (`?>`). If the string value is empty, then the leading space is not added. Also, a trailing #xA is rendered after the closing PI symbol for PI children of the root node with a lesser document order than the document element, and a leading #xA is rendered before the opening PI symbol of PI children of the root node with a greater document order than the document element.
- **Comment Nodes-** Nothing if generating canonical XML without comments. For canonical XML with comments, generate the opening comment symbol (`<!--`), the string value of the node, and the closing comment symbol (`-->`). Also, a trailing #xA is rendered after the closing comment symbol for comment children of the root node with a lesser document order than the document element, and a leading #xA is rendered before the opening comment symbol of comment children of the root node with a greater document order than the document element. (Comment children of the root node represent comments outside of the top-level document element and outside of the document type declaration).

The QName of a node is either the local name if the namespace prefix string is empty or the namespace prefix, a colon, then the local name of the element. The namespace prefix used in the QName MUST be the same one which appeared in the input document.

## 2.4 Document Subsets

Some applications require the ability to create a physical representation for an XML document subset (other than the one generated by default, which can be a proper subset of the document if the

comments are omitted). Implementations of XML canonicalization that are based on XPath can provide this functionality with little additional overhead by accepting a node-set as input rather than an octet stream. The processing of an element node ***E*** MUST be modified slightly when an XPath node-set is given as input and the element's parent is omitted from the node-set. This is necessary because omitted nodes SHALL not break the inheritance rules of inheritable attributes [C14N-Issues] defined in the xml namespace.

[Definition:] **Simple inheritable attributes** are attributes that have a value that requires at most a simple redeclaration. This redeclaration is done by supplying a new value in the child axis. The redeclaration of a simple inheritable attribute ***A*** contained in one of ***E***'s ancestors is done by supplying a value to an attribute ***Ae*** inside ***E*** with the same name. Simple inheritable attributes are `xml:lang` and `xml:space`.

The method for processing the attribute axis of an element ***E*** in the node-set is hence enhanced. All element nodes along ***E***'s ancestor axis are examined for the nearest occurrences of simple inheritable attributes in the xml namespace, such as `xml:lang` and `xml:space` (whether or not they are in the node-set). From this list of attributes, any simple inheritable attributes that are already in ***E***'s attribute axis (whether or not they are in the node-set) are removed. Then, lexicographically merge this attribute list with the nodes of ***E***'s attribute axis that are in the node-set. The result of visiting the attribute axis is computed by processing the attribute nodes in this merged attribute list.

The `xml:id` attribute is not a simple inheritable attribute and no processing of these attributes is performed.

The `xml:base` attribute is not a simple inheritable attribute and requires special processing beyond a simple redeclaration. Hence the processing of ***E***'s attribute axis needs to be enhanced further. A "join-URI-References" function is used for `xml:base` fix up. It incorporates `xml:base` attribute values from omitted `xml:base` attributes and updates the `xml:base` attribute value of the element being fixed up.

An `xml:base` fixup is performed on an element ***E*** as follows. Let ***E*** be an element in the node set whose ancestor axis contains successive elements ***En*** ... ***E1*** (in reverse document order) that are omitted and ***E***=***En+1*** is included. (It is important to note that ***En*** ... ***E1*** is for contiguously omitted elements, for example only *e2* in the example in Section 3.8.) The fix-up is only performed if at least one of ***E1*** ... ***En*** had an `xml:base` attribute. In that case let ***X1*** ... ***Xm*** be the values of the `xml:base` attributes on ***E1*** ... ***En+1*** (in document order, from outermost to innermost, ***m*** <= ***n+1***). The sequence of values is reduced in reverse document order to a single value by first combining ***Xm*** with ***Xm-1***, then the result with ***Xm-2***, and so on by calling the "join-URI-References" function until the new value for ***E***'s `xml:base` attribute remains. The result may also be null or empty (`xml:base=""`) in which case `xml:base` MUST NOT be rendered.

Note that this `xml:base` fixup is only performed if an element with an `xml:base` attribute is removed. Specifically, it is not performed if the element is present but the attribute is removed.

The join-URI-References function takes an `xml:base` attribute value from an omitted element and combines it with other contiguously omitted values to create a value for an updated `xml:base` attribute. A simple method for doing this is similar to that found in sections 5.2.1, 5.2.2 and 5.2.4 of RFC 3986 with the following modifications:

- Perform RFC 3986 section 5.2.1. "Pre-parse the Base URI" modified as follows.
  - The scheme component is not required in the base URI (Base). (i.e. Base.scheme may be null)
  - Replace a trailing ".." segment with "../" segment before processing.
- Section 5.2.4. "Remove Dot Segments" is modified as follows:
  - Keep leading "../" segments
  - Replace multiple consecutive "/" characters with a single "/" character.
  - Append a "/" character to a trailing ".." segment
- The "Remove Dot Segments" algorithm is modified to ensure that a combination of two `xml:base`

attribute values that include relative path components (i.e., path components that do not begin with a '/' character) results in an attribute value that is a relative path component.
- Perform RFC 3986 section 5.2.2. "Transform References" modified as follows to ignore the fragment part of R
    - After parsing R set R.fragment = null

Then, lexicographically merge this fixed up attribute with the nodes of ***E***'s attribute axis that are in the node-set. The result of visiting the attribute axis is computed by processing the attribute nodes in this merged attribute list.

Attributes in the XML namespace other than `xml:base`, `xml:id`, `xml:lang`, and `xml:space` MUST be processed as ordinary attributes.

The following examples illustrate the modification of the "Remove Dot Segments" algorithm:

- `"abc/"` and `"../"` should result in `""`
- `"../"` and `"../"` are combined as `"../../"` and the result is `"../../"`
- `".."` and `".."` are combined as `"../../"` and the result is `"../../"`

To illustrate the last example, when the elements *b* and *c* are removed from the following sample XML document, the correct result for the `xml:base` attribute on element *d* would be `"../../x"`:

```
<a xml:base="foo/bar">
  <b xml:base="..">
    <c xml:base="..">
      <d xml:base="x">
      </d>
    </c>
  </b>
</a>
```

# 3 Examples of XML Canonicalization

The examples in this section assume a non-validating processor, primarily so that a document type declaration can be used to declare entities as well as default attributes and attributes of various types (such as ID and enumerated) without having to declare all attributes for all elements in the document. As well, one example contains an element that deliberately violates a validity constraint (because it is still well-formed).

## 3.1 PIs, Comments, and Outside of Document Element

| | |
|---|---|
| **Input Document** | `<?xml version="1.0"?>`<br><br>`<?xml-stylesheet href="doc.xsl"`<br>`type="text/xsl" ?>`<br><br>`<!DOCTYPE doc SYSTEM "doc.dtd">`<br><br>`<doc>Hello, world!<!-- Comment 1 --></doc>`<br><br>`<?pi-without-data ?>`<br><br>`<!-- Comment 2 -->`<br><br>`<!-- Comment 3 -->` |
| **Canonical Form (uncommented)** | `<?xml-stylesheet href="doc.xsl"`<br>`type="text/xsl" ?>`<br>`<doc>Hello, world!</doc>`<br>`<?pi-without-data?>` |

| | |
|---|---|
| **Canonical Form (commented)** | `<?xml-stylesheet href="doc.xsl"`<br>`  type="text/xsl"  ?>`<br>`<doc>Hello, world!<!-- Comment 1 --></doc>`<br>`<?pi-without-data?>`<br>`<!-- Comment 2 -->`<br>`<!-- Comment 3 -->` |

Demonstrates:

- Loss of XML declaration
- Loss of DTD
- Normalization of whitespace outside of document element (first character of both canonical forms is '<'; single line breaks separate PIs and comments outside of document element)
- Loss of whitespace between PITarget and its data
- Retention of whitespace inside PI data
- Comment removal from uncommented canonical form, including delimiter for comments outside document element (the last character in both canonical forms is '>')

## 3.2 Whitespace in Document Content

| | |
|---|---|
| **Input Document** | `<doc>`<br>`  <clean>  </clean>`<br>`  <dirty>  A  B  </dirty>`<br>`  <mixed>`<br>`  A`<br>`  <clean>  </clean>`<br>`  B`<br>`  <dirty>  A  B  </dirty>`<br>`  C`<br>`  </mixed>`<br>`</doc>` |
| **Canonical Form** | `<doc>`<br>`  <clean>  </clean>`<br>`  <dirty>  A  B  </dirty>`<br>`  <mixed>`<br>`  A`<br>`  <clean>  </clean>`<br>`  B`<br>`  <dirty>  A  B  </dirty>`<br>`  C`<br>`  </mixed>`<br>`</doc>` |

Demonstrates:

- Retain all whitespace between consecutive start tags, clean or dirty
- Retain all whitespace between consecutive end tags, clean or dirty
- Retain all whitespace between end tag/start tag pair, clean or dirty
- Retain all whitespace in character content, clean or dirty

**Note:** In this example, the input document and canonical form are identical. Both end with '>' character.

## 3.3 Start and End Tags

| | |
|---|---|
| | `<!DOCTYPE doc [<!ATTLIST e9 attr CDATA "default">]>`<br>`<doc>`<br>`  <e1  />` |

| | |
|---|---|
| **Input Document** | <e2 ></e2><br><e3 name = "elem3" id="elem3" /><br><e4 name="elem4" id="elem4" ></e4><br><e5 a:attr="out" b:attr="sorted" attr2="all" attr="I'm"<br>xmlns:b="http://www.ietf.org"<br>xmlns:a="http://www.w3.org"<br>xmlns="http://example.org"/><br><e6 xmlns="" xmlns:a="http://www.w3.org"><br><e7 xmlns="http://www.ietf.org"><br><e8 xmlns="" xmlns:a="http://www.w3.org"><br><e9 xmlns="" xmlns:a="http://www.ietf.org"/><br></e8><br></e7><br></e6><br></doc> |
| **Canonical Form** | <doc><br><e1></e1><br><e2></e2><br><e3 id="elem3" name="elem3"></e3><br><e4 id="elem4" name="elem4"></e4><br><e5 xmlns="http://example.org" xmlns:a="http://www.w3.org"<br>xmlns:b="http://www.ietf.org" attr="I'm" attr2="all" b:attr="sorted"<br>a:attr="out"></e5><br><e6 xmlns:a="http://www.w3.org"><br><e7 xmlns="http://www.ietf.org"><br><e8 xmlns=""><br><e9 xmlns:a="http://www.ietf.org" attr="default"></e9><br></e8><br></e7><br></e6><br></doc> |

Demonstrates:

- Empty element conversion to start-end tag pair
- Normalization of whitespace in start and end tags
- Relative order of namespace and attribute axes
- Lexicographic ordering of namespace and attribute axes
- Retention of namespace prefixes from original document
- Elimination of superfluous namespace declarations
- Addition of default attribute

**Note:** Some start tags in the canonical form are very long, but each start tag in this example is entirely on a single line.

**Note:** In `e5`, `b:attr` precedes `a:attr` because the primary key is namespace URI not namespace prefix, and `attr2` precedes `b:attr` because the default namespace is not applied to unqualified attributes (so the namespace URI for `attr2` is empty).

## 3.4 Character Modifications and Character References

| | |
|---|---|
| **Input Document** | <!DOCTYPE doc [<br><!ATTLIST normId id ID #IMPLIED><br><!ATTLIST normNames attr NMTOKENS #IMPLIED><br>]><br><doc><br><text>First line
Second line</text><br><value>2</value><br><compute><![CDATA[value>"0" && value<"10" ?"valid":"error"]]><br></compute><br><compute expr='value>"0" && value<"10" |

| | |
|---|---|
| | ?"valid":"error"'>valid</compute><br>  <norm attr=' '   
	  ' '/><br>  <normNames attr='  A   
	  B  '/><br>  <normId id=' '   
	  ' '/><br></doc> |
| **Canonical Form** | <doc><br>  <text>First line<br>Second line</text><br>  <value>2</value><br>  <compute>value>"0" && value<"10" ?"valid":"error"<br></compute><br>  <compute expr="value>"0" &&<br>value<"10" ?<br>"valid":"error"">valid</compute><br>  <norm attr=" '  
	  ' "></norm><br>  <normNames attr="A 
	 B"></normNames><br>  <normId id="' 
	 '"></normId><br></doc> |

Demonstrates:

- Character reference replacement
- Attribute value delimiters set to quotation marks (double quotes)
- Attribute value normalization
- CDATA section replacement
- Encoding of special characters as character references in attribute values (&, <, ", , 
, 	)
- Encoding of special characters as character references in text (&, <, >, )

**Note:** The last element, `normId`, is well-formed but violates a validity constraint for attributes of type ID. For testing canonical XML implementations based on validating processors, remove the line containing this element from the input and canonical form. In general, XML consumers should be discouraged from using this feature of XML.

**Note:** Whitespace character references other than   are not affected by attribute value normalization [XML].

**Note:** In the canonical form, the value of the attribute named `attr` in the element `norm` begins with a space, an apostrophe (single quote), then *four* spaces before the first character reference.

**Note:** The `expr` attribute of the second `compute` element contains no line breaks.

## 3.5 Entity References

| | |
|---|---|
| **Input Document** | <!DOCTYPE doc [<br><!ATTLIST doc attrExtEnt ENTITY #IMPLIED><br><!ENTITY ent1 "Hello"><br><!ENTITY ent2 SYSTEM "world.txt"><br><!ENTITY entExt SYSTEM "earth.gif" NDATA gif><br><!NOTATION gif SYSTEM "viewgif.exe"><br>]><br><doc attrExtEnt="entExt"><br>  &ent1;, &ent2;!<br></doc><br><br><!-- Let world.txt contain "world" (excluding the quotes) --> |
| **Canonical Form (uncommented)** | <doc attrExtEnt="entExt"><br>  Hello, world!<br></doc> |

Demonstrates:

- Internal parsed entity reference replacement
- External parsed entity reference replacement (including whitespace outside elements and PIs)
- External unparsed entity reference

## 3.6 UTF-8 Encoding

| | |
|---|---|
| **Input Document** | `<?xml version="1.0" encoding="ISO-8859-1"?>`<br>`<doc>&#169;</doc>` |
| **Canonical Form** | `<doc>#xC2#xA9</doc>` |

Demonstrates:

- Effect of transcoding from a sample encoding to UTF-8

**Note:** The content of the doc element is NOT the string #xC2#xA9 but rather the two octets whose hexadecimal values are C2 and A9, which is the UTF-8 encoding of the UCS codepoint for the copyright sign (©).

## 3.7 Document Subsets

| | |
|---|---|
| **Input Document** | `<!DOCTYPE doc [`<br>`<!ATTLIST e2 xml:space (default\|preserve) 'preserve'>`<br>`<!ATTLIST e3 id ID #IMPLIED>`<br>`]>`<br>`<doc xmlns="http://www.ietf.org" xmlns:w3c="http://www.w3.org">`<br>`  <e1>`<br>`  <e2 xmlns="">`<br>`  <e3 id="E3"/>`<br>`  </e2>`<br>`  </e1>`<br>`</doc>` |
| **Document Subset Expression** | `<!-- Evaluate with declaration xmlns:ietf="http://www.ietf.org" -->`<br><br>`(//. \| //@* \| //namespace::*)`<br>`[`<br>`  self::ietf:e1 or (parent::ietf:e1 and not(self::text() or self::e2))`<br>`  or`<br>`  count(id("E3")\|ancestor-or-self::node()) = count(ancestor-or-self::node())`<br>`]` |
| **Canonical Form** | `<e1 xmlns="http://www.ietf.org" xmlns:w3c="http://www.w3.org"><e3 xmlns="" id="E3" xml:space="preserve"></e3></e1>` |

Demonstrates:

- Empty default namespace propagation from omitted parent element
- Propagation of attributes in the `xml` namespace in document subsets
- Persistence of omitted namespace declarations in descendants

**Note:** In the document subset expression, the subexpression `(//. | //@* | //namespace::*)` selects all nodes in the input document, subjecting each to the predicate expression in square brackets. The expression is true for `e1` and its implicit namespace nodes, and it is true if the element identified by E3

is in the `ancestor-or-self` path of the context node (such that ancestor-or-self stays the same size under union with the element identified by E3).

**Note:** The canonical form contains no line delimiters.

### 3.8 Document Subsets and XML Attributes

| | |
|---|---|
| **Input Document** | `<!DOCTYPE doc [`<br>`<!ATTLIST e2 xml:space (default\|preserve) 'preserve'>`<br>`<!ATTLIST e3 id ID #IMPLIED>`<br>`]>`<br>`<doc xmlns="http://www.ietf.org" xmlns:w3c="http://www.w3.org"`<br>`xml:base="something/else">`<br>`<e1>`<br>`<e2 xmlns="" xml:id="abc" xml:base="bar/">`<br>`<e3 id="E3" xml:base="foo"/>`<br>`</e2>`<br>`</e1>`<br>`</doc>` |
| **Document Subset Expression** | `<!-- Evaluate with declaration xmlns:ietf="http://www.ietf.org" -->`<br><br>`(//. \| //@* \| //namespace::*)`<br>`[`<br>`self::ietf:e1 or (parent::ietf:e1 and not(self::text() or`<br>`self::e2))`<br>`or`<br>`count(id("E3")\|ancestor-or-self::node()) = count(ancestor-or-`<br>`self::node())`<br>`]` |
| **Canonical Form** | `<e1 xmlns="http://www.ietf.org" xmlns:w3c="http://www.w3.org"`<br>`xml:base="something/else"><e3 xmlns="" id="E3"`<br>`xml:base="something/bar/foo" xml:space="preserve"></e3></e1>` |

Demonstrates:

- `xml:id` not inherited.
- simple inheritable XML attribute inherited (`xml:space`)
- `xml:base` fixup performed

## 4 Resolutions

This section discusses a number of key decision points as well as a rationale for each decision. Although this specification now defines XML canonicalization in terms of the XPath data model rather than XML Infoset, the canonical form described in this document is quite similar in most respects to the canonical form described in the January 2000 Canonical XML draft [C14N-20000119]. However, some differences exist, and a number of the subsections discuss the changes.

### 4.1 No XML Declaration

The XML declaration, including version number and character encoding is omitted from the canonical form. The encoding is not needed since the canonical form is encoded in UTF-8. The version is not needed since the absence of a version number unambiguously indicates XML 1.0.

Future versions of XML will be required to include an XML declaration to indicate the version number. However, canonicalization method described in this specification may not be applicable to future versions of XML without some modifications. When canonicalization of a new version of XML is required, this specification could be updated to include the XML declaration as presumably the

absence of the XML declaration from the XPath data model can be remedied by that time (e.g. by reissuing a new XPath based on the Infoset data model).

## 4.2 No Character Model Normalization

The Unicode standard [Unicode] allows multiple different representations of certain "precomposed characters" (a simple example is "ç"). Thus two XML documents with content that is equivalent for the purposes of most applications may contain differing character sequences. The W3C is preparing a normalized representation [CharModel]. The C14N-20000119 Canonical XML draft used this normalized form. However, many XML 1.0 processors do not perform this normalization. Furthermore, applications that must solve this problem typically enforce character model normalization at all times starting when character content is created in order to avoid processing failures that could otherwise result (e.g. see example from Cowan). Therefore, character model normalization has been moved out of scope for XML canonicalization. However, the XML processor used to prepare the XPath data model input is required (by the Data Model) to use Normalization Form C [NFC, NFC-Corrigendum] when converting an XML document to the UCS character domain from any encoding that is not UCS-based (currently, UCS-based encodings include UTF-8, UTF-16, UTF-16BE, and UTF-16LE, UCS-2, and UCS-4).

## 4.3 Handling of Whitespace Outside Document Element

The C14N-20000119 Canonical XML draft placed a #xA after each PI outside of the document element as well as a #xA after the end tag of the document element. The method in this specification performs the same function except for omitting the final #xA after the last PI (or comment or end tag of the document element). This technique ensures that PI (and comment) children of the root are separated from markup by a line feed even if root node or the document element are omitted from the output node-set.

## 4.4 No Namespace Prefix Rewriting

The C14N-20000119 Canonical XML draft described a method for rewriting namespace prefixes such that two documents having logically equivalent namespace declarations would also have identical namespace prefixes. The goal was to eliminate dependence on the particular namespace prefixes in a document when testing for logical equivalence. However, there now exist a number of contexts in which namespace prefixes can impart information value in an XML document. For example, an XPath expression in an attribute value or element content can reference a namespace prefix. Thus, rewriting the namespace prefixes would damage such a document by changing its meaning (and it cannot be logically equivalent if its meaning has changed).

More formally, let D1 be a document containing an XPath in an attribute value or element content that refers to namespace prefixes used in D1. Further assume that the namespace prefixes in D1 will all be rewritten by the canonicalization method. Let D2 = D1, then modify the namespace prefixes in D2 and modify the XPath expression's references to namespace prefixes such that D2 and D1 remain logically equivalent. Since namespace rewriting does not include occurrences of namespace references in attribute values and element content, the canonical form of D1 does not equal the canonical form of D2 because the XPath will be different. Thus, although namespace rewriting normalizes the namespace declarations, the goal eliminating dependence on the particular namespace prefixes in the document is not achieved.

Moreover, it is possible to prove that namespace rewriting is harmful, rather than simply ineffective. Let D1 be a document containing an XPath in an attribute value or element content that refers to namespace prefixes used in D1. Further assume that the namespace prefixes in D1 will all be rewritten by the canonicalization method. Now let D2 be the canonical form of D1. Clearly, the canonical forms of D1 and D2 are equivalent (since D2 is the canonical form of the canonical form of D1), yet D1 and D2 are not logically equivalent because the aforementioned XPath works in D1 and

doesn't work in D2.

Note that an argument similar to this can be leveled against the XML canonicalization method based on any of the cases in the Limitations, the problems cannot easily be fixed in those cases, whereas here we have an opportunity to avoid purposefully introducing such a limitation.

Applications that must test for logical equivalence must perform more sophisticated tests than mere octet stream comparison. However, this is quite likely to be necessary in any case in order to test for logical equivalencies based on application rules as well as rules from other XML-related recommendations, working drafts, and future works.

## 4.5 Order of Namespace Declarations and Attributes

The C14N-20000119 Canonical XML draft alternated between namespace declarations and attribute declarations. This is part of the namespace prefix rewriting scheme, which this specification eliminates. This specification follows the XPath data model of putting all namespace nodes before all attribute nodes.

## 4.6 Superfluous Namespace Declarations

Unnecessary namespace declarations are not made in the canonical form. Whether for an empty default namespace, a non-empty default namespace, or a namespace prefix binding, the XML canonicalization method omits a declaration if it determines that the immediate parent element *in the canonical form* has an equivalent declaration in scope. The root document element is handled specially since it has no parent element. All namespace declarations in it are retained, except the declaration of an empty default namespace is automatically omitted.

Relative to the method of simply rendering the entire namespace context of each element, implementations are not hindered by more than a constant factor in processing time and memory use. The advantages include:

- Eliminates overrun of `xmlns=""` from canonical forms of applications that may not even use namespaces, or support them only minimally.
- Eliminates namespace declarations from elements where they may not belong according to the application's content model, thereby simplifying the task of reattaching a document type declaration to a canonical form.

Note that in document subsets, an element with omissions from its ancestral element chain will be rendered to the canonical form with namespace declarations that may have been made in its omitted ancestors, thus preserving the meaning of the element.

## 4.7 Propagation of Default Namespace Declaration in Document Subsets

The XPath data model represents an empty default namespace with the absence of a node, not with the presence of a default namespace node having an empty value. Thus, with respect to the fact that element `e3` in the following examples is not namespace qualified, we cannot tell the difference between `<e1 xmlns="a:b"><e2 xmlns=""><e3/></e2></e1>` versus `<e1 xmlns="a:b"><e2><e3 xmlns=""/></e2></e1>`. All we know is that `e3` was not namespace qualified on input, so we preserve this information on output if `e2` is omitted so that `e3` does not take on the default namespace qualification of `e1`.

## 4.8 Sorting Attributes by Namespace URI

Given the requirement to preserve the namespace prefixes declared in a document, sorting attributes with the prefix, rather than the namespace URI, as the primary key is viable and easier to implement. However, the namespace URI was selected as the primary key because this is closer to the intent of

the Namespaces in XML 1.0 specification, which is to identify namespaces by URI and local name, not by a prefix and local name. The effect of the sort is to group together all attributes that are in the same namespace.

# 5 References


**C14N10**
*Canonical XML Version 1.0*, W3C Recommendation. ed. J. Boyer. 15 March 2001.http://www.w3.org/TR/xml-c14n10/.

**C14N-20000119**
*Canonical XML Version 1.0*, W3C Working Draft. T. Bray, J. Clark, J. Tauber, and J. Cowan. January 19, 2000. http://www.w3.org/TR/2000/WD-xml-c14n-20000119.html.

**C14N-Issues**
*Known Issues with Canonical XML 1.0*, W3C Working Group Note. J. Kahan, K. Lanz. December 2006. http://www.w3.org/TR/C14N-issues/.

**CharModel**
*Character Model for the World Wide Web*, W3C Working Draft. eds. Martin J. Dürst, François Yergeau, Misha Wolf, Asmus Freytag and Tex Texin. http://www.w3.org/TR/charmod/.

**Cowan**
*Example of Harmful Effect of Character Model Normalization*, Letter in XML Signature Working Group Mail Archive. John Cowan, July 7, 2000. http://lists.w3.org/Archives/Public/w3c-ietf-xmldsig/2000JulSep/0038.html.

**DSig-Usage**
*Using XML Digital Signatures in the 2006 XML Environment*, W3C Working Group Note. Thomas Roessler. December 2006. http://www.w3.org/TR/DSig-usage/.

**Infoset**
*XML Information Set*, W3C Working Draft. eds. John Cowan and Richard Tobin. http://www.w3.org/TR/xml-infoset/.

**ISO-8859-1**
*ISO-8859-1 Latin 1 Character Set*. http://www.utoronto.ca/webdocs/HTMLdocs/NewHTML/iso_table.html or http://www.iso.org/iso/iso_catalogue.htm.

**Keywords**
*Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119. S. Bradner. March 1997. http://www.ietf.org/rfc/rfc2119.txt.

**Namespaces**
*Namespaces in XML 1.0 (Second Edition)*, W3C Recommendation. eds. Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. http://www.w3.org/TR/REC-xml-names/.

**NFC**
*TR15, Unicode Normalization Forms.* M. Davis, M. Dürst. Revision 18: November 1999. http://www.unicode.org/unicode/reports/tr15/tr15-18.html.

**NFC-Corrigendum**
*Normalization Corrigendum*. The Unicode Consortium. http://www.unicode.org/unicode/uni2errata/Normalization_Corrigendum.html.

**Unicode**
*The Unicode Standard, version 3.0.* The Unicode Consortium. ISBN 0-201-61633-5. http://www.unicode.org/unicode/standard/versions/Unicode3.0.html.

**UTF-16**
*UTF-16, an encoding of ISO 10646*, IETF RFC 2781. P. Hoffman , F. Yergeau. February 2000. http://www.ietf.org/rfc/rfc2781.txt.

**UTF-8**
*UTF-8, a transformation format of ISO 10646*, IETF RFC 2279. F. Yergeau. January 1998. http://www.ietf.org/rfc/rfc2279.txt.

**URI**
*Uniform Resource Identifiers (URI): Generic Syntax*, IETF RFC 3986. T. Berners-Lee, R. Fielding, L. Masinter. January 2005 http://www.ietf.org/rfc/rfc3986.txt.

**XBase**
*XML Base* ed. Jonathan Marsh. 27 June 2001. http://www.w3.org/TR/xmlbase/.
**XML**
*Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation. eds. Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau and Eve Maler. 16 August 2006. http://www.w3.org/TR/REC-xml/.
**XML DSig**
*XML-Signature Syntax and Processing*, IETF Draft/W3C Candidate Recommendation. D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, and E. Simon. 31 October 2000. http://www.w3.org/TR/xmldsig-core/.
**XML ID**
*xml:id Version 1.0*, W3C Recommendation. eds. Norman Walsh, Daniel Veillard and Jonathan Marsh. 9 September 2005. http://www.w3.org/TR/xml-id/.
**XML Plenary Decision**
*W3C XML Plenary Decision on relative URI References In namespace declarations*, W3C Document. 11 September 2000. http://lists.w3.org/Archives/Public/xml-uri/2000Sep/0083.html.
**XPath**
*XML Path Language (XPath) Version 1.0*, W3C Recommendation. eds. James Clark and Steven DeRose. 16 November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

## A Appendix

The following informative table outlines example results of the modified Remove Dot Segments algorithm described in Section 2.4.

| Input | Output |
|---|---|
| no/.././/pseudo-netpath/seg/file.ext | pseudo-netpath/seg/file.ext |
| no/..//.///pseudo-netpath/seg/file.ext | pseudo-netpath/seg/file.ext |
| yes/no//..//.///pseudo-netpath/seg/file.ext | yes/pseudo-netpath/seg/file.ext |
| no/../yes | yes |
| no/../yes/ | yes/ |
| no/../yes/no/.. | yes/ |
| ../../no/../.. | ../../../ |
| no/../.. | ../ |
| no/.. | |
| no/../ | |
| /a/b/c/./../../g | /a/g |
| mid/content=5/../6 | mid/6 |
| ../../.. | ../../../ |
| no/../../ | ../ |
| ..yes/..no/..no/..no/../../../..yes | ..yes/..yes |
| ..yes/..no/..no/..no/../../../..yes/ | ..yes/..yes/ |

| | |
|---|---|
| ../.. | ../../ |
| ../../../ | ../../../ |
| . | |
| ./ | |
| ./. | |
| //no/.. | / |
| ../../no/.. | ../../ |
| ../../no/../ | ../../ |
| yes/no/../ | yes/ |
| yes/no/no/../.. | yes/ |
| yes/no/no/no/../../.. | yes/ |
| yes/no/../yes/no/no/../.. | yes/yes/ |
| yes/no/no/no/../../../yes | yes/yes |
| yes/no/no/no/../../../yes/ | yes/yes/ |
| /no/../ | / |
| /yes/no/../ | /yes/ |
| /yes/no/no/../.. | /yes/ |
| /yes/no/no/no/../../.. | /yes/ |
| ../../..no/.. | ../../ |
| ../../..no/../ | ../../ |
| ..yes/..no/../ | ..yes/ |
| ..yes/..no/..no/../.. | ..yes/ |
| ..yes/...no/..no/..no/../../.. | ..yes/ |
| ..yes/..no/../..yes/..no/..no/../.. | ..yes/..yes/ |
| /..no/../ | / |
| /..yes/..no/../ | /..yes/ |
| /..yes/..no/..no/../.. | /..yes/ |
| /..yes/..no/..no/..no/../../.. | /..yes/ |
| / | / |
| /. | / |
| /./ | / |
| /./. | / |

| | |
|---|---|
| /./../ | / |
| /.. | / |
| /../.. | / |
| /../../.. | / |
| /../../.. | / |
| //.. | / |
| //..//.. | / |
| //..//..//.. | / |
| /./.. | / |
| /./../../.. | / |
| /./../../../../.. | / |
| . | |
| ./ | |
| ./. | |
| .. | ../ |
| ../ | ../ |