XSLT 2.0 and XQuery 1.0 Serialization

W3C Recommendation 23 January 2007

This version:

http://www.w3.org/TR/2007/REC-xslt-xquery-serialization-20070123/

Latest version:

http://www.w3.org/TR/xslt-xquery-serialization/

Previous version:

http://www.w3.org/TR/2006/PR-xslt-xquery-serialization-20061121/

Authors and Contributors:

Scott Boag (IBM) <scott_boag@us.ibm.com>
Michael Kay (Saxonica) <http://www.saxonica.com>
Joanne Tong (IBM) <joannet@ca.ibm.com>
Norman Walsh (Sun Microsystems) <Norman.Walsh@Sun.COM>
Henry Zongaro (IBM) <zongaro@ca.ibm.com>

Abstract

This document defines serialization of an instance of the data model as defined in [XQuery 1.0 and XPath 2.0 Data Model] into a sequence of octets. Serialization is designed to be a component that can be used by other specifications such as [XSL Transformations (XSLT) Version 2.0] or [XQuery 1.0: An XML Query Language].

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 one document in a set of eight documents that have progressed to Recommendation together (XQuery 1.0, XQueryX 1.0, XSLT 2.0, Data Model, Functions and Operators, Formal Semantics, Serialization, XPath 2.0).

This is a Recommendation of the W3C. It has been jointly developed by the W3C XSL Working Group and the W3C XML Query Working Group, each of which is part of the XML Activity.

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.

No substantive changes have been made to this specification since its publication as a Proposed Recommendation.

Please report errors in this document using W3C's public Bugzilla system (instructions can be found at http://www.w3.org/XML/2005/04/qt-bugzilla). If access to that system is not feasible, you may send your comments to the W3C XSLT/XPath/XQuery public comments mailing list, public-qt-comments@w3.org. It will be very helpful if you include the string “[[Ser]]” in the subject line of your report, whether made in Bugzilla or in email. Each Bugzilla entry and email message should contain only one error report. Archives of the comments and responses are available at http://lists.w3.org/Archives/Public/public-qt-comments/.

This document was produced by groups 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 XML Query Working Group and also maintains a public list of any patent disclosures made in connection with the deliverables of the XSL Working Group; those pages also include 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.

Table of Contents

Appendices

This page is intentionally left blank.

1. Introduction

This document defines serialization of the W3C XQuery 1.0 and XPath 2.0 Data Model (XDM), which is the data model of at least [XML Path Language (XPath) 2.0], [XSL Transformations (XSLT) Version 2.0], and [XQuery 1.0: An XML Query Language], and any other specifications that reference it.

Serialization is the process of converting an instance of the [XQuery 1.0 and XPath 2.0 Data Model] into a sequence of octets. Serialization is well-defined for most data model instances.

1.1. Terminology

In this specification, where they appear in upper case, the words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "MAY", "REQUIRED", and "RECOMMENDED" are to be interpreted as described in [RFC2119].

As is indicated in § 10 – Conformance on page 26, conformance criteria for serialization are determined by other specifications that refer to this specification. A *serializer* is software that implements some or all of the requirements of this specification in accordance with such conformance criteria. A serializer is not REQUIRED to directly provide a programming interface that permits a user to set serialization parameters or to provide an input sequence for serialization. In this document, material labeled as "Note" and examples are provided for explanatory purposes and are not normative.

Certain aspects of serialization are described in this specification as implementation-defined or implementation-dependent.

Implementation-defined indicates an aspect that MAY differ between serializers, but whose actual behavior MUST be specified either by another specification that sets conformance criteria for serialization (see § 10 – Conformance on page 26) or in documentation that accompanies the serializer.

Implementation-dependent indicates an aspect that MAY differ between serializers, and whose actual behavior is not REQUIRED to be specified either by another specification that sets conformance criteria for serialization (see § 10 – Conformance on page 26) or in documentation that accompanies the serializer.

In some instances, the sequence that is input to serialization cannot be successfully converted into a sequence of octets given the set of serialization parameter (§ 3 – Serialization Parameters on page 4) values specified. A *serialization error* is said to occur in such an instance. In some cases, a serializer is REQUIRED to signal such an error. What it means to signal a serialization error is determined by the relevant conformance criteria (§ 10 – Conformance on page 26) to which the serializer conforms. In other cases, there is an implementation-defined choice between signaling a serialization error and performing a recovery action. Such a recovery action will allow a serializer to produce a sequence of octets that might not fully reflect the usual requirements of the parameter settings that are in effect.

Many terms used in this document are defined in the XPath specification [XML Path Language (XPath) 2.0] or the Data Model specification [XQuery 1.0 and XPath 2.0 Data Model]. Particular attention is drawn to the following:

- The term *atomization* is defined in . It is a process that takes as input a sequence of nodes and atomic values, and returns a sequence of atomic values, in which the nodes are replaced by their typed values as defined in [XQuery 1.0 and XPath 2.0 Data Model].
- The term *Node* is defined as part of . There are seven kinds of nodes in the data model: document, element, attribute, text, namespace, processing instruction, and comment.
- The term *sequence* is defined in . A sequence is an ordered collection of zero or more items.

- The term *string value* is defined in . Every node has a string value. For example, the string value of an element is the concatenation of the string values of all its descendant text nodes.
- The term *expanded QName* is defined in . An expanded QName consists of an optional namespace URI and a local name. An expanded QName also retains its original namespace prefix (if any), to facilitate casting the expanded QName into a string.
- An element or attribute that is in no namespace, or an expanded-QName whose namespace part is an empty sequence, is referred to as having a *null namespace URI*.
- An element or attribute that does not have a null namespace URI, is referred to as having a *non-null namespace URI*.

2. Sequence Normalization

An instance of the data model that is input to the serialization process is a sequence. Prior to serializing a sequence using any of the output methods whose behavior is specified by this document (§ 3 – Serialization Parameters on page 4), the serializer MUST first compute a normalized sequence for serialization; it is the normalized sequence that is actually serialized. The purpose of *sequence normalization* is to create a sequence that can be serialized as a well-formed XML document or external general parsed entity, that also reflects the content of the input sequence to the extent possible. The result of the sequence normalization process is a *result tree*.

The normalized sequence for serialization is constructed by applying all of the following rules in order, with the initial sequence being input to the first step, and the sequence that results from any step being used as input to the subsequent step. For any implementation-defined output method, it is implementation-defined whether this sequence normalization process takes place.

For any implementation-defined output method, it is implementation-defined whether sequence normalization process takes place.
Where the process of converting the input sequence to a normalized sequence indicates that a value MUST be cast to `xs:string`, that operation is defined in of [XQuery 1.0 and XPath 2.0 Functions and Operators]. Where a step in the sequence normalization process indicates that a node should be copied, the copy is performed in the same way as an XSLT `xsl:copy-of` instruction that has a `validation` attribute whose value is `preserve` and has a `select` attribute whose effective value is the node, as described in of [XSL Transformations (XSLT) Version 2.0], or equivalently in the same way as an XQuery content expression as described in Step 1e of of [XQuery 1.0: An XML Query Language], where the construction mode is `preserve`. The steps in computing the normalized sequence are:

1. If the sequence that is input to serialization is empty, create a sequence S_1 that consists of a zero-length string. Otherwise, copy each item in the sequence that is input to serialization to create the new sequence S_1.
2. For each item in S_1, if the item is atomic, obtain the lexical representation of the item by casting it to an `xs:string` and copy the string representation to the new sequence; otherwise, copy the item, which will be a node, to the new sequence. The new sequence is S_2.
3. For each subsequence of adjacent strings in S_2, copy a single string to the new sequence equal to the values of the strings in the subsequence concatenated in order, each separated by a single space. Copy all other items to the new sequence. The new sequence is S_3.

4. For each item in S_3, if the item is a string, create a text node in the new sequence whose string value is equal to the string; otherwise, copy the item to the new sequence. The new sequence is S_4.
5. For each item in S_4, if the item is a document node, copy its children to the new sequence; otherwise, copy the item to the new sequence. The new sequence is S_5.
6. For each subsequence of adjacent text nodes in S_5, copy a single text node to the new sequence equal to the values of the text nodes in the subsequence concatenated in order. Any text nodes with values of zero length are dropped. Copy all other items to the new sequence. The new sequence is S_6.
7. It is a serialization error if an item in S_6 is an attribute node or a namespace node. Otherwise, construct a new sequence, S_7, that consists of a single document node and copy all the items in the sequence, which are all nodes, as children of that document node.

S_7 is the normalized sequence.

The result tree rooted at the document node that is created by the final step of this sequence normalization process is the instance of the data model to which the rules of the appropriate output method are applied. If the sequence normalization process results in a serialization error, the serializer MUST signal the error.

☞ The sequence normalization process for a sequence `$seq` is equivalent to constructing a document node using the XSLT instruction:

```
<xsl:document>

  <xsl:copy-of select="$seq" validation="preserve"/>

</xsl:document>
```

or the XQuery expression:

```
declare construction preserve;



document {

  for $s in $seq return

    if ($s instance of document-node())

    then $s/child::node()

    else $s

}
```

This process results in a serialization error if sequences contain parentless attribute and/or namespace nodes.

3. Serialization Parameters

There are a number of parameters that influence how serialization is performed. Host languages MAY allow users to specify any or all of these parameters, but they are not REQUIRED to be able to do so. However, the host language specification MUST specify how the value of all applicable parameters is to be determined.

It is a serialization error if a parameter value is invalid for the given parameter. It is the responsibility of the host language to specify how invalid values should be handled at the level of that language.

The following serialization parameters are defined:

Serialization parameter name	Permitted values for parameter
`byte-order-mark`	One of the enumerated values `yes` or `no`. This parameter indicates whether the serialized sequence of octets is to be preceded by a Byte Order Mark. (See Section 5.1 of [Unicode Encoding].) The actual octet order used is implementation-dependent. If the encoding defines no Byte Order Mark, or if the Byte Order Mark is prohibited for the specific Unicode encoding or implementation environment, then this parameter is ignored.
`cdata-section-elements`	A list of expanded QNames, possibly empty.
`doctype-public`	A string of PubidChar characters. This parameter may be absent.
`doctype-system`	A string of Unicode characters that does not include both an apostrophe (#x27) and a quotation mark (#x22) character. This parameter may be absent.
`encoding`	A string of Unicode characters in the range #x21 to #x7E (that is, printable ASCII characters); the value SHOULD be a charset registered with the Internet Assigned Numbers Authority [IANA], [RFC2278] or begin with the characters `x-` or `X-`.
`escape-uri-attributes`	One of the enumerated values `yes` or `no`.
`include-content-type`	One of the enumerated values `yes` or `no`.
`indent`	One of the enumerated values `yes` or `no`.
`media-type`	A string of Unicode characters specifying the media type (MIME content type) [RFC2046]; the charset parameter of the media type MUST NOT be specified explicitly in the value of the `media-type` parameter. If the destination of the serialized output is annotated with a media type, this parameter MAY be used to provide such an annotation. For example, it MAY be used to set the media type in an HTTP header.
`method`	An expanded QName with a null namespace URI, and the local part of the name equal to one of `xml`, `xhtml`, `html` or `text`, or having a non-null namespace URI. If the namespace URI is non-null, the parameter specifies an implementation-defined output method.
`normalization-form`	One of the enumerated values `NFC`, `NFD`, `NFKC`, `NFKD`, `fully-normalized`, `none` or an implementation-defined value.

Serialization parameter name	Permitted values for parameter
`omit-xml-declaration`	One of the enumerated values `yes` or `no`.
`standalone`	One of the enumerated values `yes`, `no` or `omit`.
`undeclare-prefixes`	One of the enumerated values `yes` or `no`.
`use-character-maps`	A list of pairs, possibly empty, with each pair consisting of a single Unicode character and a string of Unicode characters.
`version`	A string of Unicode characters.

The octet order of the serialized result sequence of octects is implementation-dependent.
The value of the `method` parameter is an expanded QName. If the value has a null namespace URI, then the local name identifies a method specified in this document and MUST be one of `xml`, `html`, `xhtml`, or `text`; in this case, the output method specified MUST be used for serializing. If the namespace URI is non-null, then it identifies an implementation-defined output method; the behavior in this case is not specified by this document.

If the namespace URI is non-null for the `method` serialization parameter, then the parameter specifies an implementation-defined output method.
In those cases where they have no important effect on the content of the serialized result, details of the output methods defined by this specification are left unspecified and are regarded as implementation-dependent. Whether a serializer uses apostrophes or quotation marks to delimit attribute values in the XML output method is an example of such a detail.

In those cases where they have no important effect on the content of the serialized result, details of the output methods defined by this specification are left unspecified and are regarded as implementation-dependent.
The detailed semantics of each parameter will be described separately for each output method.

Implementations MAY define additional serialization parameters, and MAY allow users to do so. For this purpose, the name of a serialization parameter is considered to be a QName; the parameters listed above are QNames in no namespace, while any additional serialization parameters must have names that are namespace-qualified. If the serialization method is one of the four methods `xml`, `html`, `xhtml`, or `text`, then the additional serialization parameters MAY affect the output of the serializer to the extent (but only to the extent) that this specification leaves the output implementation-defined or implementation-dependent. For example, such parameters might control whether namespace declarations on an element are written before or after the attributes of the element, or they might define the number of space or tab characters to be inserted when the `indent` parameter is set to `yes`; but they could not instruct the serializer to suppress the error that occurs when the HTML output method encounters illegal characters (see error).

The effect of additional serialization parameters on the output of the serializer, where the name of such a parameter must be namespace-qualified, is implementation-defined or implementation-dependent. The extent of this effect on the output must not override the provisions of this specification.

4. Phases of Serialization

Serialization comprises five phases of processing (preceded optionally by the sequence normalization process described in § 2 – Sequence Normalization on page 2).

For an implementation-defined output method, any of these phases MAY be skipped or MAY be performed in a different order than is specified here. For the output methods defined in this specification, these phases are carried out sequentially as follows:

1. A `meta` element is added to the normalized sequence along with discarding an existing `meta` element, as controlled by the `include-content-type` parameter for the XHTML and HTML output methods.
2. *Markup generation* produces the character representation of those parts of the serialized result that describe the structure of the normalized sequence. In the cases of the XML, HTML and XHTML output methods, this phase produces the character representations of the following:

   - the document type declaration;
   - start tags and end tags (except for attribute values, whose representation is produced by the character expansion phase);
   - processing instructions; and
   - comments.

   In the cases of the XML and XHTML output methods, this phase also produces the following:

   - the XML or text declaration; and
   - empty element tags (except for the attribute values);

   In the case of the text output method, this phase replaces the single document node produced by sequence normalization with a new document node that has exactly one child, which is a text node. The string value of the new text node is the string value of the document node that was produced by sequence normalization.
3. *Character expansion* is concerned with the representation of characters appearing in text and attribute nodes in the normalized sequence. For each text and attribute node, the following rules are applied in sequence.

   A. If the node is an attribute that is a URI attribute value and the `escape-uri-attributes` parameter is set to require escaping of URI attributes, apply URI escaping as defined below, and skip rules b-e. Otherwise, continue with rule b.

      URI escaping consists of the following three steps applied in sequence to the content of URI attribute values:

      i. normalize to NFC using the method defined in

      ii. percent-encode any special characters in the URI using the method defined in

      iii. escape according to HTML rules any characters (such as < and &) where HTML requires escaping, and any characters that cannot be represented in the selected encoding. For example, replace < with `&lt;`. (See also section § 7.3 – Writing Character Data on page 21)

      The values of attributes listed in Appendix C – List of URI Attributes on page 28 are *URI attribute values*. Attributes are not considered to be URI attributes simply because they are namespace declaration attributes or have the type annotation `xs:anyURI`.

   B. If the node is a text node whose parent element is selected by the rules of the `cdata-section-elements` parameter for the applicable output method, create CDATA sections as described below, and skip rules c-e. Otherwise, continue with rule c.

Apply the following two processes in sequence to create CDATA sections

i. Unicode Normalization if requested by the `normalization-form` parameter.

ii. apply changes as detailed in the description of the `cdata-section-elements` parameter for the applicable output method.

C. Apply character mapping as determined by the `use-character-maps` parameter for the applicable output method. For characters that were substituted by this process, skip rules d and e. For the remaining characters that were not modified by character mapping, continue with rule d.

D. Apply Unicode Normalization if requested by the `normalization-form` parameter.

Unicode Normalization is the process of removing alternate representations of equivalent sequences from textual data, to convert the data into a form that can be binary-compared for equivalence, as specified in [UAX #15: Unicode Normalization Forms]. For specific recommendations for character normalization on the World Wide Web, see [Character Model for the World Wide Web 1.0: Normalization].

The meanings associated with the possible values of the `normalization-form` parameter are defined in section § 5.1.8 – XML Output Method: the `normalization-form` Parameter on page 13.

Continue with step e.

E. Escape according to XML or HTML rules, as determined by the applicable output method, any characters (such as `<` and `&`) where XML or HTML requires escaping, and any characters that cannot be represented in the selected encoding. For example, replace `<` with `&lt;`. (See also section § 7.3 – Writing Character Data on page 21). For characters such as `>` where XML defines a built-in entity but does not require its use in all circumstances, it is implementation-dependent whether the character is escaped.

4. *Indentation*, as controlled by the `indent` parameter, MAY add or remove whitespace according to the rules defined by the applicable output method.

5. *Encoding*, as controlled by the `encoding` parameter, converts the character stream produced by the previous phases into an octet stream.

> Serialization is only defined in terms of encoding the result as a stream of octets. However, a serializer may provide an option that allows the encoding phase to be skipped, so that the result of serialization is a stream of Unicode characters. The effect of any such option is implementation-defined, and a serializer is not required to support such an option.
>
> The effect of providing an option that allows the encoding phase to be skipped, so that the result of serialization is a stream of Unicode characters, is implementation-defined. The serializer is not required to support such an option.

5. XML Output Method

The XML output method serializes the normalized sequence as an XML entity that MUST satisfy the rules for either a well-formed XML document entity or a well-formed XML external general parsed entity, or both. A serialization error results if the serializer is unable to satisfy those rules, except for content modified by the character expansion phase of serialization, as described in § 4 – Phases of Serialization on page 5. The effects of the character expansion phase could result in the serialized output being not well-formed, but will not result in a serialization error. If a serialization error results, the serializer MUST signal the error.

If the document node of the normalized sequence has a single element node child and no text node children, then the serialized output is a well-formed XML document entity, and the serialized output MUST conform to the appropriate version of the XML Namespaces Recommendation [XML Names] or [XML Names 1.1]. If the normalized sequence does not take this form, then the serialized output is a well-formed XML external general parsed entity, which, when referenced within a trivial XML document wrapper like this:

```
<?xml version="version"?>
<!DOCTYPE doc [
<!ENTITY e SYSTEM "entity-URI">
]>
<doc>&e;</doc>
```

where `entity-URI` is a URI for the entity, and the value of the `version` pseudo-attribute is the value of the `version` parameter, produces a document which MUST itself be a well-formed XML document conforming to the corresponding version of the XML Namespaces Recommendation [XML Names] or [XML Names 1.1].

A *reconstructed tree* may be constructed by parsing the XML document and converting it into an instance of the data model as specified in [XQuery 1.0 and XPath 2.0 Data Model]. The result of serialization MUST be such that the reconstructed tree is the same as the result tree except for the following permitted differences:

- If the document was produced by adding a document wrapper, as described above, then it will contain an extra `doc` element as the document element.
- The order of attribute and namespace nodes in the two trees MAY be different.
- The following properties of corresponding nodes in the two trees MAY be different:
  - the base-uri property of document nodes and element nodes;
  - the document-uri and unparsed-entities properties of document nodes;
  - the type-name and typed-value properties of element and attribute nodes;
  - the nilled property of element nodes;
  - the content property of text nodes, due to the effect of the `indent` and `use-character-maps` parameters.
- The reconstructed tree MAY contain additional attributes and text nodes resulting from the expansion of default and fixed values in its DTD or schema; also, in the presence of a DTD, non-CDATA attributes may lose whitespace characters as a result of attribute value normalization.
- The type annotations of the nodes in the two trees MAY be different. Type annotations in a result tree are discarded when the tree is serialized. Any new type annotations obtained by parsing the document will depend on whether the serialized XML document is assessed against a schema, and this MAY result in type annotations that are different from those in the original result tree.

☞ In order to influence the type annotations in the instance of the data model that would result from processing a serialized XML document, the author of the XSLT stylesheet, XQuery expression or other process might wish to create the instance of the data model that is input to the serialization process so that it makes use of mechanisms provided by [XML Schema], such as `xsi:type` and `xsi:schemaLocation` attributes.

> The serialization process will not automatically create such attributes in the serialized document if those attributes were not part of the result tree that is to be serialized.
>
> Similarly, it is possible that an element node in the instance of the data model that is to be serialized has the `nilled` property with the value `true`, but no `xsi:nil` attribute. The serialization process will not create such an attribute in the serialized document simply to reflect the value of the property. The value of the `nilled` property has no direct effect on the serialized result.

- Additional namespace nodes MAY be present in the reconstructed tree if the serialization process did not undeclare one or more namespaces, as described in § 5.1.7 – XML Output Method: the `undeclare-prefixes` Parameter on page 13, and the starting instance of the data model contained an element node with a namespace node that declared some prefix, but a child element of that node did not have any namespace node that declared the same prefix.

  The result tree MAY contain namespace nodes that are not present in the reconstructed tree, as the process of creating an instance of the data model MAY ignore namespace declarations in some circumstances. See and of [XQuery 1.0 and XPath 2.0 Data Model] for additional information.

- If the `indent` parameter has the value `yes`,

  - additional text nodes consisting of whitespace characters MAY be present in the reconstructed tree; and
  - text nodes in the result tree that contained only whitespace characters MAY correspond to text nodes in the reconstructed tree that contain additional whitespace characters that were not present in the result tree

  See § 5.1.3 – XML Output Method: the `indent` Parameter on page 11 for more information on the `indent` parameter.

- Additional nodes MAY be present in the reconstructed tree due to the effect of character mapping in the character expansion phase, and the values of attribute nodes and text nodes in the reconstructed tree MAY be different from those in the result tree, due to the effects of URI expansion, character mapping and Unicode Normalization in the character expansion phase of serialization.

> ☞ The `use-character-maps` parameter can cause arbitrary characters to be inserted into the serialized XML document in an unescaped form, including characters that would be considered to be part of XML markup. Such characters could result in arbitrary new element nodes, attribute nodes, and so on, in the reconstructed tree that results from processing the serialized XML document.

A consequence of this rule is that certain characters MUST be output as character references, to ensure that they survive the round trip through serialization and parsing. Specifically, CR, NEL and LINE SEPARATOR characters in text nodes MUST be output respectively as "`&#xD;`", "`&#x85;`", and "`&#x2028;`", or their equivalents; while CR, NL, TAB, NEL and LINE SEPARATOR characters in attribute nodes MUST be output respectively as "`&#xD;`", "`&#xA;`", "`&#x9;`", "`&#x85;`", and "`&#x2028;`", or their equivalents. In addition, the non-whitespace control characters #x1 through #x1F and #x7F through #x9F in text nodes and attribute nodes MUST be output as character references.

For example, an attribute with the value "x" followed by "y" separated by a newline will result in the output `"x&#xA;y"` (or with any equivalent character reference). The XML output cannot be "x" followed by a literal newline followed by a "y" because after parsing, the attribute value would be `"x y"` as a consequence of the XML attribute normalization rules.

☞ XML 1.0 did not permit an XML processor to normalize NEL or LINE SEPARATOR characters to a LINE FEED character. However, if a document entity that specifies version 1.1 invokes an external general parsed entity with no text declaration or a text declaration that specifies version 1.0, the external parsed entity is processed according to the rules of XML 1.1. For this reason, NEL and LINE SEPARATOR characters in text and attribute nodes must always be escaped using character references, regardless of the value of the `version` parameter.

XML 1.0 permitted control characters in the range #x7F through #x9F to appear as literal characters in an XML document, but XML 1.1 requires such characters, other than NEL, to be escaped as character references. An external general parsed entity with no text declaration or a text declaration that specifies a version pseudo-attribute with value `1.0` that is invoked by an XML 1.1 document entity must follow the rules of XML 1.1. Therefore, the non-whitespace control characters in the ranges #x1 through #x1F and #x7F through #x9F must always be escaped, regardless of the value of the `version` parameter.

It is a serialization error to specify the doctype-system parameter, or to specify the standalone parameter with a value other than `omit`, if the instance of the data model contains text nodes or multiple element nodes as children of the root node. The serializer MUST either signal the error, or recover by ignoring the request to output a document type declaration or `standalone` parameter.

5.1. The Influence of Serialization Parameters upon the XML Output Method

5.1.1. XML Output Method: the `version` Parameter

The `version` parameter specifies the version of XML and the version of Namespaces in XML to be used for outputting the instance of the data model. The version output in the XML declaration (if an XML declaration is not omitted) MUST correspond to the version of XML that the serializer used for outputting the instance of the data model. A serialization error results if the value of the `version` parameter specifies a version of XML that is not supported by the serializer; the serializer MUST signal the error.

If the serialized result would contain an that contains a character that is not permitted by the version of Namespaces in XML specified by the `version` parameter, a serialization error results. The serializer MUST signal the error.

If the serialized result would contain a character that is not permitted by the version of XML specified by the `version` parameter, a serialization error results. The serializer MUST signal the error.

For example, if the `version` parameter has the value `1.0`, and the instance of the data model contains a non-whitespace control character in the range #x1 to #x1F, a serialization error results. If the `version` parameter has the value `1.1` and a comment node in the instance of the data model contains a non-whitespace control character in the range #x1 to #x1F or a control character other than NEL in the range #x7F to #x9F, a serialization error results.

5.1.2. XML Output Method: the `encoding` Parameter

The `encoding` parameter specifies the encoding to be used for outputting the instance of the data model. Serializers are REQUIRED to support values of `UTF-8` and `UTF-16`. A serialization error occurs if an output encoding other than `UTF-8` or `UTF-16` is requested and the serializer does not support that encoding. The serializer MUST signal the error, or recover by using `UTF-8` or `UTF-16` instead. The serializer MUST NOT use an encoding whose name does not match the production of the XML Recommendation [XML10].

When outputting a newline character in the instance of the data model, the serializer is free to represent it using any character sequence that will be normalized to a newline character by an XML parser, unless a specific mapping for the newline character is provided in a character map (see § 9 – Character Maps on page 25).

When outputting any other character that is defined in the selected encoding, the character MUST be output using the correct representation of that character in the selected encoding.

It is possible that the instance of the data model will contain a character that cannot be represented in the encoding that the serializer is using for output. In this case, if the character occurs in a context where XML recognizes character references (that is, in the value of an attribute node or text node), then the character MUST be output as a character reference. A serialization error occurs if such a character appears in a context where character references are not allowed (for example, if the character occurs in the name of an element). The serializer MUST signal the error.

For example, if a text node contains the character LATIN SMALL LETTER E WITH ACUTE (#xE9), and the value of the `encoding` parameter is `US-ASCII`, the character MUST be serialized as a character reference. If a comment node contains the same character, a serialization error results.

5.1.3. XML Output Method: the `indent` Parameter

If the `indent` parameter has the value `yes`, then the XML output method MAY output whitespace in addition to the whitespace in the instance of the data model in order to indent the result so that a person will find it easier to read; if the `indent` parameter has the value `no`, it MUST NOT output any additional whitespace. If the XML output method does output additional whitespace, it MUST use an algorithm to output additional whitespace that satisfies all of the following constraints:

- Whitespace characters MUST NOT be added adjacent to a text node that contains non-whitespace characters.
- Whitespace characters MUST NOT be added other than adjacent to an element node, that is, immediately before a start tag or immediately after an end tag.
- Whitespace characters MUST NOT be inserted in a part of the result document that is controlled by an `xml:space` attribute with value `preserve`. (See [XML10] for more information about the `xml:space` attribute.)
- Whitespace characters SHOULD NOT be added in places where the characters would constitute significant whitespace, for example, in the content of an element that is annotated with a type other than `xs:untyped` or `xs:anyType`, and whose content model is known to be mixed.

In any location where the above rules allow the addition of whitespace characters, existing whitespace characters MAY also be removed or replaced. For example, a tab MAY be inserted as a replacement for existing spaces.

☞ The effect of these rules is to ensure that whitespace is only added in places where (a) XSLT's `<xsl:strip-space>` declaration could cause it to be removed, and (b) it does not affect the string value of any element node with simple content. It is usually not safe to indent document types that include elements with mixed content.

☞ The whitespace added may possibly be based on whitespace stripped from either the source document or the stylesheet (in the case of XSLT), or guided by other means that might depend on the host language, in the case of an instance of the data model created using some other process.

5.1.4. XML Output Method: the `cdata-section-elements` Parameter

The `cdata-section-elements` parameter contains a list of expanded QNames. If the expanded QName of the parent of a text node is a member of the list, then the text node MUST be output as a CDATA section, except in those circumstances described below.

If the text node contains the sequence of characters `]]>`, then the currently open CDATA section MUST be closed following the `]]` and a new CDATA section opened before the `>`.

If the text node contains characters that are not representable in the character encoding being used to output the instance of the data model, then the currently open CDATA section MUST be closed before such characters, the characters MUST be output using character references or entity references, and a new CDATA section MUST be opened for any further characters in the text node.

CDATA sections MUST NOT be used except where they have been explicitly requested by the user, either by using the `cdata-section-elements` parameter, or by using some other implementation-defined mechanism.

☞ This is phrased to permit an implementor to provide an option that attempts to preserve CDATA sections present in the source document.

An serializer may provide an implementation-defined mechanism to place CDATA sections in the result tree.

5.1.5. XML Output Method: the `omit-xml-declaration` and `standalone` Parameters

The XML output method MUST output an XML declaration if the `omit-xml-declaration` parameter has the value `no`. The XML declaration MUST include both version information and an encoding declaration. If the `standalone` parameter has the value `yes` or the value `no`, the XML declaration MUST include a standalone document declaration with the same value as the value of the `standalone` parameter. If the `standalone` parameter has the value `omit`, the XML declaration MUST NOT include a standalone document declaration; this ensures that it is both an XML declaration (allowed at the beginning of a document entity) and a text declaration (allowed at the beginning of an external general parsed entity).

A serialization error results if the `omit-xml-declaration` parameter has the value `yes`, and

- the `standalone` parameter has a value other than `omit`; or
- the `version` parameter has a value other than `1.0` and the `doctype-system` parameter is specified.

The serializer MUST signal the error.

Otherwise, if the `omit-xml-declaration` parameter has the value `yes`, the XML output method MUST NOT output an XML declaration.

5.1.6. XML Output Method: the `doctype-system` and `doctype-public` Parameters

If the `doctype-system` parameter is specified, the XML output method MUST output a document type declaration immediately before the first element. The name following `<!DOCTYPE` MUST be the name of the first element, if any. If the `doctype-public` parameter is also specified, then the XML output method MUST output `PUBLIC` followed by the public identifier and then the system identifier; otherwise, it MUST output `SYSTEM` followed by the system identifier. The internal subset MUST be empty. The `doctype-public` parameter MUST be ignored unless the `doctype-system` parameter is specified.

5.1.7. XML Output Method: the `undeclare-prefixes` Parameter

The Data Model allows an element node that binds a non-empty prefix to have a child element node that does not bind that same prefix. In *Namespaces in XML 1.1* ([XML Names 1.1]), this can be represented accurately by undeclaring prefixes. For the undeclaring prefix of the child element node, if the `undeclare-prefixes` parameter has the value `yes`, the output method is XML or XHTML, and the `version` parameter value is greater than `1.0`, the serializer MUST undeclare its namespace. If the `undeclare-prefixes` parameter has the value `no` and the output method is XML or XHTML, then the undeclaration of prefixes MUST NOT occur.

Consider an element `x:foo` with four in-scope namespaces that associate prefixes with URIs as follows:

- `x` is associated with `http://example.org/x`
- `y` is associated with `http://example.org/y`
- `z` is associated with `http://example.org/z`
- `xml` is associated with `http://www.w3.org/XML/1998/namespace`

Suppose that it has a child element `x:bar` with three in-scope namespaces:

- `x` is associated with `http://example.org/x`
- `y` is associated with `http://example.org/y`
- `xml` is associated with `http://www.w3.org/XML/1998/namespace`

If namespace undeclaration is in effect, it will be serialized this way:

```
<x:foo xmlns:x="http://example.org/x"
       xmlns:y="http://example.org/y"
       xmlns:z="http://example.org/z">

       <x:bar xmlns:z="">...</x:bar>

</x:foo>
```

In *Namespaces in XML* 1.0 ([XML Names]), prefix undeclaration is not possible. If the output method is XML or XHTML, the value of the `undeclare-prefixes` parameter is `yes`, and the value of the `version` parameter is `1.0`, a serialization error results; the serializer MUST signal the error.

5.1.8. XML Output Method: the `normalization-form` Parameter

The `normalization-form` parameter is applicable to the XML output method. The values `NFC` and `none` MUST be supported by the serializer. A serialization error results if the value of the `normalization-form` parameter specifies a normalization form that is not supported by the serializer; the serializer MUST signal the error.

The meanings associated with the possible values of the `normalization-form` parameter are as follows:

- `NFC` specifies the serialized result will be in Normalization Form C, using the rules specified in [Character Model for the World Wide Web 1.0: Normalization].
- `NFD` specifies the serialized result will be in Normalization Form D, as specified in [UAX #15: Unicode Normalization Forms].

- `NFKC` specifies the serialized result will be in Normalization Form KC, as specified in [UAX #15: Unicode Normalization Forms].
- `NFKD` specifies the serialized result will be in Normalization Form KD, as specified in [UAX #15: Unicode Normalization Forms].
- `fully-normalized` specifies the serialized result will be in fully normalized text, as specified in [Character Model for the World Wide Web 1.0: Normalization].
- `none` specifies that no Unicode Normalization will be applied.
- An implementation-defined value has an implementation-defined effect.

  If the value of the `normalization-form` form parameter is not `NFC`, `NFD`, `NFKC`, `NFKD`, `fully-normalized`, or `none` then the meaning of the value and its effect is implementation-defined.

If the value of the parameter is `fully-normalized`, then no *relevant construct* of the parsed entity created by the serializer may start with a composing character. The term *relevant construct* has the meaning defined in section 2.13 of [XML11]. If this condition is not satisfied, a serialization error MUST be signaled.

☞ Specifying `fully-normalized` as the value of this parameter does not guarantee that the XML document output by the serializer will in fact be fully normalized as defined in [XML11]. This is because the serializer does not check that the text is `include normalized`, which would involve checking all external entities that it refers to (such as an external DTD). Furthermore, the serializer does not check whether any character escape generated using character maps represents a composing character.

5.1.9. XML Output Method: the `media-type` Parameter

The `media-type` parameter is applicable to the XML output method. See § 3 – Serialization Parameters on page 4 for more information.

5.1.10. XML Output Method: the `use-character-maps` Parameter

The `use-character-maps` parameter is applicable to the XML output method. The result of serialization using the XML output method is not guaranteed to be well-formed XML if character maps have been specified. See § 9 – Character Maps on page 25 for more information.

5.1.11. XML Output Method: the `byte-order-mark` Parameter

The `byte-order-mark` parameter is applicable to the XML output method. See § 3 – Serialization Parameters on page 4 for more information.

☞ The byte order mark may be undesirable under certain circumstances; for example, to concatenate resulting XML fragments without additional processing to remove the byte order mark. Therefore this specification does not mandate the `byte-order-mark` parameter to have the value `yes` when the encoding is UTF-16, even though the XML 1.0 and XML 1.1 specifications state that entities encoded in UTF-16 must begin with a byte order mark. Consequently, this specification does not guarantee that the resulting XML fragment, without a byte order mark, will not cause an error when processed by a conforming XML processor.

5.1.12. XML Output Method: the `escape-uri-attributes` Parameter

The `escape-uri-attributes` parameter is not applicable to the XML output method. It is the responsibility of the host language to specify whether an error occurs if this parameter is specified in combination with the XML output method, or if the parameter is simply dropped.

5.1.13. XML Output Method: the `include-content-type` Parameter

The `include-content-type` parameter is not applicable to the XML output method. It is the responsibility of the host language to specify whether an error occurs if this parameter is specified in combination with the XML output method, or if the parameter is simply dropped.

6. XHTML Output Method

The XHTML output method serializes the instance of the data model as XML, using the HTML compatibility guidelines defined in the XHTML specification.

It is entirely the responsibility of the person or process that creates the instance of the data model to ensure that the instance of the data model conforms to the [XHTML 1.0] or [XHTML 1.1] specification. It is not an error if the instance of the data model is invalid XHTML. Equally, it is entirely under the control of the person or process that creates the instance of the data model whether the output conforms to XHTML 1.0 Strict, XHTML 1.0 Transitional, or any other specific definition of XHTML.

The serialization of the instance of the data model follows the same rules as for the XML output method, with the general exceptions noted below and parameter-specific exceptions in § 6.1 – The Influence of Serialization Parameters upon the XHTML Output Method on page 16. These differences are based on the HTML compatibility guidelines published in Appendix C of [XHTML 1.0], which are designed to ensure that as far as possible, XHTML is rendered correctly on user agents designed originally to handle HTML.

- The following XHTML elements have an *EMPTY* content model: `area`, `base`, `br`, `col`, `hr`, `img`, `input`, `link`, `meta`, `basefont`, `frame`, `isindex`, and `param`. Given an empty instance of an XHTML element whose content model is not EMPTY (for example, an empty title or paragraph) the serializer MUST NOT use the minimized form. That is, it MUST output `<p></p>` and not `<p />`.
- If an element that has no children is an XHTML element with an EMPTY content model, the serializer MUST use the minimized tag syntax, for example `<br />`, as the alternative syntax `<br></br>` allowed by XML gives uncertain results in many existing user agents. The serializer MUST include a space before the trailing `/>`, e.g. `<br />`, `<hr />` and `<img src="karen.jpg" alt="Karen" />`.
- The serializer MUST NOT use the entity reference `&apos;` which, although legal in XML and therefore in XHTML, is not defined in HTML and is not recognized by all HTML user agents.
- The serializer SHOULD output namespace declarations in a way that is consistent with the requirements of the XHTML DTD if this is possible. The XHTML 1.0 DTDs require the declaration `xmlns="http://www.w3.org/1999/xhtml"` to appear on the `html` element, and only on the `html` element. The serializer MUST output namespace declarations that are consistent with the namespace nodes present in the result tree, but it MUST avoid outputting redundant namespace declarations on elements where the DTD would make them invalid.

☞ If the `html` element is generated by an XSLT literal result element of the form `<html xmlns="http://www.w3.org/1999/xhtml"> ... </html>`, or by an XQuery direct element constructor of the same form, then the `html` element in the result document will have a node name whose prefix is "", which will satisfy the requirements of the DTD. In other cases the prefix assigned to the element is implementation-dependent.

☞ Appendix C of [XHTML 1.0] describes a number of compatibility guidelines for users of XHTML who wish to render their XHTML documents with HTML user agents. In some cases, such as the guideline on the form empty elements should take, only the serialization process itself has the ability to follow the guideline. In such cases, those guidelines are reflected in the requirements on the serializer described above.

In all other cases, the guidelines can be adhered to by the instance of the data model that is input to the serialization process. The guideline on the use of whitespace characters in attribute values is one such example. Another example is that `xml:lang="..."` does not serialize to both `xml:lang="..."` and `lang="..."` as required by some legacy user agents. It is the responsibility of the person or process that creates the instance of the data model that is input to the serialization process to ensure it is created in a way that is consistent with the guidelines. No serialization error results if the input instance of the data model does not adhere to the guidelines.

6.1. The Influence of Serialization Parameters upon the XHTML Output Method

6.1.1. XHTML Output Method: the `version` Parameter

The behavior for `version` parameter for the XHTML output method is described in § 5.1.1 – XML Output Method: the `version` Parameter on page 10.

6.1.2. XHTML Output Method: the `encoding` Parameter

The behavior for `encoding` parameter for the XHTML output method is described in § 5.1.2 – XML Output Method: the `encoding` Parameter on page 10.

6.1.3. XHTML Output Method: the `indent` Parameter

If the `indent` parameter has the value `yes`, the serializer MAY add or remove whitespace as it serializes the result tree, if it observes the following constraints.

- Whitespace MUST NOT be added other than before or after an element, or adjacent to an existing whitespace character.
- Whitespace MUST NOT be added or removed adjacent to an inline element. The inline elements are those elements in the XHTML namespace in the %inline category of any of the XHTML 1.0 DTD's, in the %inline.class category of the XHTML 1.1 DTD, and elements in the XHTML namespace with local names `ins` and `del` if they are used as inline elements (i.e., if they do not contain element children).
- Whitespace MUST NOT be added or removed inside a formatted element, the formatted elements being those in the XHTML namespace with local names `pre`, `script`, `style`, and `textarea`.

☞ The effect of the above constraints is to ensure any insertion or deletion of whitespace would not affect how a conforming HTML user agent would render the output, assuming the serialized document does not refer to any HTML style sheets.

The HTML definition of whitespace is different from the XML definition: see section 9.1 of [HTML] 4.01 specification.

6.1.4. XHTML Output Method: the `cdata-section-elements` Parameter

The behavior for `cdata-section-elements` parameter for the XHTML output method is described in § 5.1.4 – XML Output Method: the `cdata-section-elements` Parameter on page 12.

6.1.5. XHTML Output Method: the `omit-xml-declaration` and `standalone` Parameters

The behavior for `omit-xml-declaration` and `standalone` parameters for the XHTML output method is described in § 5.1.5 – XML Output Method: the `omit-xml-declaration` and `standalone` Parameters on page 12.

☞ As with the XML output method, the XHTML output method specifies that an XML declaration will be output unless it is suppressed using the `omit-xml-declaration` parameter. Appendix C.1 of [XHTML 1.0] provides advice on the consequences of including, or omitting, the XML declaration.

6.1.6. XHTML Output Method: the `doctype-system` and `doctype-public` Parameters

The behavior for `doctype-system` and `doctype-public` parameters for the XHTML output method is described in § 5.1.6 – XML Output Method: the `doctype-system` and `doctype-public` Parameters on page 12.

6.1.7. XHTML Output Method: the `undeclare-prefixes` Parameter

The behavior for `undeclare-prefixes` parameter for the XHTML output method is described in § 5.1.7 – XML Output Method: the `undeclare-prefixes` Parameter on page 13.

6.1.8. XHTML Output Method: the `normalization-form` Parameter

The behavior for `normalization-form` parameter for the XHTML output method is described in § 5.1.8 – XML Output Method: the `normalization-form` Parameter on page 13.

6.1.9. XHTML Output Method: the `media-type` Parameter

The behavior for `media-type` parameter for the XHTML output method is described in § 5.1.9 – XML Output Method: the `media-type` Parameter on page 14.

6.1.10. XHTML Output Method: the `use-character-maps` Parameter

The behavior for `use-character-maps` parameter for the XHTML output method is described in § 5.1.10 – XML Output Method: the `use-character-maps` Parameter on page 14.

6.1.11. XHTML Output Method: the `byte-order-mark` Parameter

The behavior for `byte-order-mark` parameter for the XHTML output method is described in § 5.1.11 – XML Output Method: the `byte-order-mark` Parameter on page 14.

6.1.12. XHTML Output Method: the `escape-uri-attributes` Parameter

If the `escape-uri-attributes` parameter has the value `yes`, the XHTML output method MUST apply URI escaping to URI attribute values, except that relative URIs MUST NOT be absolutized.

☞ This escaping is deliberately confined to non-ASCII characters, because escaping of ASCII characters is not always appropriate, for example when URIs or URI fragments are interpreted locally by the HTML user agent. Even in the case of non-ASCII characters, escaping can sometimes cause problems. More precise control of URI escaping is therefore available by setting `escape-uri-attributes` to `no`, and controlling the escaping of URIs by using methods defined in and .

6.1.13. XHTML Output Method: the `include-content-type` Parameter

If the instance of the data model includes a `head` element in the XHTML namespace, and the `include-content-type` parameter has the value `yes`, the XHTML output method MUST add a `meta` element as the first child element of the `head` element, specifying the character encoding actually used.

For example,

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-JP" />
...
```

The content type SHOULD be set to the value given for the `media-type` parameter.

☞ It is recommended that the host language use as default value for this parameter one of the MIME types ([RFC2046]) registered for XHTML. Currently, these are `text/html` (registered by [RFC2854]) and `application/xhtml+xml` (registered by [RFC3236]). Note that some user agents fail to recognize the charset parameter if the content type is not `text/html`.

If a `meta` element has been added to the `head` element as described above, then any existing `meta` element child of the `head` element having an `http-equiv` attribute with the value "Content-Type", making the comparison without consideration of casing and leading/trailing spaces, MUST be discarded.

☞ This process removes possible parameters in the attribute value. For example,

```
<meta http-equiv="Content-Type" content="text/html;version='3.0'" />
```

in the data model instance would be replaced by,

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
```

7. HTML Output Method

The HTML output method serializes the instance of the data model as HTML.

For example, the following XSL stylesheet generates html output,

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" version="4.0"/>
<xsl:template match="/">
  <html>
    <xsl:apply-templates/>
  </html>
</xsl:template>
...
</xsl:stylesheet>
```

In the example, the `version` attribute of the `xsl:output` element indicates the version of the HTML Recommendation [HTML] to which the serialized result is to conform.

It is entirely the responsibility of the person or process that creates the instance of the data model to ensure that the instance of the data model conforms to the HTML Recommendation [HTML]. It is not an error if the instance of the data model is invalid HTML. Equally, it is entirely under the control of the person or process that creates the instance of the data model whether the output conforms to HTML.

7.1. Markup for Elements

The HTML output method MUST NOT output an element differently from the XML output method unless the expanded QName of the element has a null namespace URI. An element whose expanded QName has a non-null namespace URI MUST be output as XML. This is known as an *XML Island*. If the expanded QName of the element has a null namespace URI, but the local part of the expanded QName is not recognized as the name of an HTML element, the element MUST be output in the same way as a non-empty, inline element such as `span`. In particular:

1. If the result tree contains namespace nodes for namespaces other than the XML namespace, the HTML output method MUST represent these namespaces using attributes named `xmlns` or `xmlns:`*prefix* in the same way as the XML output method would represent them when the `version` parameter is set to `1.0`.
2. If the result tree contains elements or attributes whose names have a non-null namespace URI, the HTML output method MUST generate namespace-prefixed QNames for these nodes in the same way as the XML output method would do when the `version` parameter is set to `1.0`.
3. Where special rules are defined later in this section for serializing specific HTML elements and attributes, these rules MUST NOT be applied to an element or attribute whose name has a non-null namespace URI. However, the generic rules for the HTML output method that apply to all elements and attributes, for example the rules for escaping special characters in the text and the rules for indentation, MUST be used also for namespaced elements and attributes.
4. When serializing an element whose name is not defined in the HTML specification, but that is in the null namespace, the HTML output method MUST apply the same rules (for example, indentation rules) as when serializing a `span` element. The descendants of such an element MUST be serialized as if they were descendants of a `span` element.
5. When serializing an element whose name is in a non-null namespace, the HTML output method MUST apply the same rules (for example, indentation rules) as when serializing a `div` element. The descendants of such an element MUST be serialized as if they were descendants of a `div` element. , except for the influence of the `cdata-section-elements` serialization parameter on any text node children of the element.

The HTML output method MUST NOT output an end-tag for an empty element if the element type has an empty content model. For HTML 4.0, the element types that have an empty content model are `area`, `base`, `basefont`, `br`, `col`, `frame`, `hr`, `img`, `input`, `isindex`, `link`, `meta` and `param`. For example, an element written as `<br/>` or `<br></br>` in an XSLT stylesheet MUST be output as `<br>`.

The HTML output method MUST recognize the names of HTML elements regardless of case. For example, elements named `br`, `BR` or `Br` MUST all be recognized as the HTML `br` element and output without an end-tag.

The HTML output method MUST NOT perform escaping for the content of the `script` and `style` elements.

For example, a `script` element created by an XQuery direct element constructor or an XSLT literal result element, such as:

```
<script>if (a &lt; b) foo()</script>
```

or

```
<script><![CDATA[if (a < b) foo()]]></script>
```

MUST be output as

```
<script>if (a < b) foo()</script>
```

A common requirement is to output a `script` element as shown in the example below:

```
<script type="application/ecmascript">
      document.write ("<em>This won't work</em>")
</script>
```

This is illegal HTML, for the reasons explained in section B.3.2 of the [HTML] 4.01 specification. Nevertheless, it is possible to output this fragment, using either of the following constructs:

Firstly, by use of a `script` element created by an XQuery direct element constructor or an XSLT literal result element:

```
<script type="application/ecmascript">
      document.write ("<em>This won't work</em>")
</script>
```

Secondly, by constructing the markup from ordinary text characters:

```
<script type="application/ecmascript">
      document.write ("&lt;em&gt;This won't work&lt;/em&gt;")
</script>
```

As the [HTML] specification points out, the correct way to write this is to use the escape conventions for the specific scripting language. For JavaScript, it can be written as:

```
<script type="application/ecmascript">
      document.write ("&lt;em&gt;This will work&lt;\/em&gt;")
</script>
```

The [HTML] 4.01 specification also shows examples of how to write this in various other scripting languages. The escaping MUST be done manually; it will not be done by the serializer.

7.2. Writing Attributes

The HTML output method MUST NOT escape "<" characters occurring in attribute values.

The HTML output method MUST output boolean attributes (that is attributes with only a single allowed value that is equal to the name of the attribute) in minimized form.

For example, a start-tag created using the following XQuery direct element constructor or XSLT literal result element

```
<OPTION selected="selected">
```

MUST be output as

```
<OPTION selected>
```

The HTML output method MUST NOT escape a & character occurring in an attribute value immediately followed by a { character (see Section B.7.1 of the HTML Recommendation [HTML]).

For example, a start-tag created using the following XQuery direct element constructor or XSLT literal result element

```
<BODY bgcolor='&amp;{{randomrbg}};'>
```

MUST be output as

```
<BODY bgcolor='&{randomrbg};'>
```

See § 7.4 – The Influence of Serialization Parameters upon the HTML Output Method on page 21 for additional directives on how attributes may be written.

7.3. Writing Character Data

The HTML output method MAY output a character using a character entity reference in preference to using a numeric character reference, if an entity is defined for the character in the version of HTML that the output method is using. Entity references and character references SHOULD be used only where the character is not present in the selected encoding, or where the visual representation of the character is unclear (as with `&nbsp;`, for example).

When outputting a sequence of whitespace characters in the instance of the data model, within an element where whitespace is treated normally (but not in elements such as `pre` and `textarea`), the HTML output method MAY represent it using any sequence of whitespace that will be treated in the same way by an HTML user agent. See section 3.5 of [XHTML Modularization] for some additional information on handling of whitespace by an HTML user agent.

Certain characters, specifically the control characters #x7F-#x9F, are legal in XML but not in HTML. It is a serialization error to use the HTML output method when such characters appear in the instance of the data model. The serializer MUST signal the error.

The HTML output method MUST terminate processing instructions with `>` rather than `?>`. It is a serialization error to use the HTML output method when `>` appears within a processing instruction in the data model instance being serialized.

7.4. The Influence of Serialization Parameters upon the HTML Output Method

7.4.1. HTML Output Method: the `version` Parameter

The `version` attribute indicates the version of the HTML Recommendation [HTML] to which the serialized result is to conform. If the serializer does not support the version of HTML specified by this parameter, it MUST signal a serialization error .

7.4.2. HTML Output Method: the `encoding` Parameter

The `encoding` parameter specifies the encoding to be used. Serializers are REQUIRED to support values of `UTF-8` and `UTF-16`. A serialization error occurs if an output encoding other than `UTF-8` or `UTF-16` is requested and the serializer does not support that encoding. The serializer MUST signal the error.

It is possible that the instance of the data model will contain a character that cannot be represented in the encoding that the serializer is using for output. In this case, if the character occurs in a context where HTML recognizes character references, then the character MUST be output as a character entity reference or decimal numeric character reference; otherwise (for example, in a `script` or `style` element or in a comment), the serializer MUST signal a serialization error .

See § 7.4.13 – HTML Output Method: the `include-content-type` Parameter on page 23 regarding how this parameter is used with the `include-content-type` parameter.

7.4.3. HTML Output Method: the `indent` Parameter

If the `indent` parameter has the value `yes`, then the HTML output method MAY add or remove whitespace as it serializes the result tree, if it observes the following constraints.

- Whitespace MUST NOT be added other than before or after an element, or adjacent to an existing whitespace character.
- Whitespace MUST NOT be added or removed adjacent to an inline element. The inline elements are those included in the `%inline` category of any of the HTML 4.01 DTD's, as well as the `ins` and `del` elements if they are used as inline elements (i.e., if they do not contain element children).
- Whitespace MUST NOT be added or removed inside a formatted element, the formatted elements being `pre`, `script`, `style`, and `textarea`.

☞ The effect of the above constraints is to ensure any insertion or deletion of whitespace would not affect how a conforming HTML user agent would render the output, assuming the serialized document does not refer to any HTML style sheets.

Note that the HTML definition of whitespace is different from the XML definition (see section 9.1 of the [HTML] specification).

7.4.4. HTML Output Method: the `cdata-section-elements` Parameter

The `cdata-section-elements` parameter is not applicable to the HTML output method, except in the case of XML Islands.

7.4.5. HTML Output Method: the `omit-xml-declaration` and `standalone` Parameters

The `omit-xml-declaration` and `standalone` parameters are not applicable to the HTML output method.

7.4.6. HTML Output Method: the `doctype-system` and `doctype-public` Parameters

If the `doctype-public` or `doctype-system` parameters are specified, then the HTML output method MUST output a document type declaration immediately before the first element. The name following `<!DOCTYPE` MUST be `HTML` or `html`. If the `doctype-public` parameter is specified, then the output method MUST output `PUBLIC` followed by the specified public identifier; if the `doctype-system` parameter is also specified, it MUST also output the specified system identifier following the public identifier. If the `doctype-system` parameter is specified but the `doctype-public` parameter is not specified, then the output method MUST output `SYSTEM` followed by the specified system identifier.

7.4.7. HTML Output Method: the `undeclare-prefixes` Parameter

The `undeclare-prefixes` parameter is not applicable to the HTML output method.

7.4.8. HTML Output Method: the `normalization-form` Parameter

The `normalization-form` parameter is applicable to the HTML output method. The values `NFC` and `none` MUST be supported by the serializer. A serialization error results if the value of the `normaliza-`

`tion-form` parameter specifies a normalization form that is not supported by the serializer; the serializer MUST signal the error.

7.4.9. HTML Output Method: the `media-type` Parameter

The `media-type` parameter is applicable to the HTML output method. See § 3 – Serialization Parameters on page 4 for more information. See § 7.4.13 – HTML Output Method: the `include-content-type` Parameter on page 23 regarding how this parameter is used with the `include-content-type` parameter.

7.4.10. HTML Output Method: the `use-character-maps` Parameter

The `use-character-maps` parameter is applicable to the HTML output method. See § 9 – Character Maps on page 25 for more information.

7.4.11. HTML Output Method: the `byte-order-mark` Parameter

The `byte-order-mark` parameter is applicable to the HTML output method. See § 3 – Serialization Parameters on page 4 for more information.

7.4.12. HTML Output Method: the `escape-uri-attributes` Parameter

If the `escape-uri-attributes` parameter has the value `yes`, the HTML output method MUST apply URI escaping to URI attribute values, except that relative URIs MUST NOT be absolutized.

☞ This escaping is deliberately confined to non-ASCII characters, because escaping of ASCII characters is not always appropriate, for example when URIs or URI fragments are interpreted locally by the HTML user agent. Even in the case of non-ASCII characters, escaping can sometimes cause problems. More precise control of URI escaping is therefore available by setting `escape-uri-attributes` to `no`, and controlling the escaping of URIs by using methods defined in and .

7.4.13. HTML Output Method: the `include-content-type` Parameter

If there is a `head` element, and the `include-content-type` parameter has the value `yes`, the HTML output method MUST add a `meta` element as the first child element of the `head` element specifying the character encoding actually used.

For example,

```
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=EUC-JP">
...
```

The content type MUST be set to the value given for the `media-type` parameter.

If a `meta` element has been added to the `head` element as described above, then any existing `meta` element child of the `head` element having an `http-equiv` attribute with the value "Content-Type", making the comparison without consideration of case and leading or trailing spaces, MUST be discarded.

☞ This process removes possible parameters in the attribute value. For example,

```
<meta http-equiv="Content-Type" content="text/html;version='3.0'"/>
```

in the data model instance would be replaced by,

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
```

8. Text Output Method

The Text output method serializes the instance of the data model by outputting the string value of the document node created by the markup generation step of the without any escaping.

A newline character in the instance of the data model MAY be output using any character sequence that is conventionally used to represent a line ending in the chosen system environment.

8.1. The Influence of Serialization Parameters upon the Text Output Method

8.1.1. Text Output Method: the `version` Parameter

The `version` parameter is not applicable to the Text output method.

8.1.2. Text Output Method: the `encoding` Parameter

The `encoding` parameter identifies the encoding that the Text output method MUST use to convert sequences of characters to sequences of bytes. Serializers are REQUIRED to support values of `UTF-8` and `UTF-16`. A serialization error occurs if the serializer does not support the encoding specified by the `encoding` parameter. The serializer MUST signal the error. If the instance of the data model contains a character that cannot be represented in the encoding that the serializer is using for output, the serializer MUST signal a serialization error .

8.1.3. Text Output Method: the `indent` Parameter

The `indent` parameter is not applicable to the Text output method.

8.1.4. Text Output Method: the `cdata-section-elements` Parameter

The `cdata-section-elements` parameter is not applicable to the Text output method.

8.1.5. Text Output Method: the `omit-xml-declaration` and `standalone` Parameters

The `omit-xml-declaration` and `standalone` parameters are not applicable to the Text output method.

8.1.6. Text Output Method: the `doctype-system` and `doctype-public` Parameters

The `doctype-system` and `doctype-public` parameters are not applicable to the Text output method.

8.1.7. Text Output Method: the `undeclare-prefixes` Parameter

The `undeclare-prefixes` parameter is not applicable to the Text output method.

8.1.8. Text Output Method: the `normalization-form` Parameter

The `normalization-form` parameter is applicable to the Text output method. The values `NFC` and `none` MUST be supported by the serializer. A serialization error results if the value of the `normaliza-tion-form` parameter specifies a normalization form that is not supported by the serializer; the serializer MUST signal the error.

8.1.9. Text Output Method: the `media-type` Parameter

The `media-type` parameter is applicable to the Text output method. See § 3 – Serialization Parameters on page 4 for more information.

8.1.10. Text Output Method: the `use-character-maps` Parameter

The `use-character-maps` parameter is applicable to the Text output method. See § 9 – Character Maps on page 25 for more information.

8.1.11. Text Output Method: the `byte-order-mark` Parameter

The `byte-order-mark` parameter is applicable to the Text output method. See § 3 – Serialization Parameters on page 4 for more information.

8.1.12. Text Output Method: the `escape-uri-attributes` Parameter

The `escape-uri-attributes` parameter is not applicable to the Text output method.

8.1.13. Text Output Method: the `include-content-type` Parameter

The `include-content-type` parameter is not applicable to the Text output method.

9. Character Maps

The `use-character-maps` parameter is a list of characters and corresponding string substitutions.

Character maps allow a specific character appearing in a text or attribute node in the instance of the data model to be replaced with a specified string of characters during serialization. The string that is substituted is output "as is," and the serializer performs no checks that the resulting document is well-formed. This mechanism can therefore be used to introduce arbitrary markup in the serialized output. See of [XSL Transformations (XSLT) Version 2.0] for examples of using character mapping in XSLT.

Character mapping is applied to the characters that actually appear in a text or attribute node in the instance of the data model, before any other serialization operations such as escaping or Unicode Normalization are applied. If a character is mapped, then it is not subjected to XML or HTML escaping, nor to Unicode Normalization. The string that is substituted for a character is not validated or processed in any way by the serializer, except for translation into the target encoding. In particular, it is not subjected to XML or HTML escaping, it is not subjected to Unicode Normalization, and it is not subjected to further character mapping.

Character mapping is not applied to characters in text nodes whose parent elements are listed in the `cdata-section-elements` parameter, nor to characters for which output escaping has been disabled (disabling output escaping is an [XSL Transformations (XSLT) Version 2.0] feature), nor to characters in attribute values that are subject to URI escaping defined for the HTML and XHTML output methods, unless URI escaping has been disabled using the `escape-uri-attributes` parameter in the output definition.

On serialization, occurrences of a character specified in the `use-character-maps` in text nodes and attribute values are replaced by the corresponding string from the `use-character-maps` parameter.

Using a character map can result in non-well-formed documents if the string contains XML-significant characters. For example, it is possible to create documents containing unmatched start and end tags, references to entities that are not declared, or attributes that contain tags or unescaped quotation marks.

If a character is mapped, then it is not subjected to XML or HTML escaping.

A serialization error occurs if character mapping causes the output of a string containing a character that cannot be represented in the encoding that the serializer is using for output. The serializer MUST signal the error.

10. Conformance

Serialization is intended primarily as a component of a *host language* such as [XSL Transformations (XSLT) Version 2.0] or [XQuery 1.0: An XML Query Language]. Therefore, this document relies on specifications that use it to specify conformance criteria for Serialization in their respective environments. Specifications that set conformance criteria for their use of Serialization MUST NOT change the semantic definitions of Serialization as given in this specification, except by subsetting and/or compatible extensions. It is the responsibility of the host language to specify how serialization errors should be handled.

Certain facilities in this specification are described as producing implementation-defined results. A claim that asserts conformance with this specification MUST be accompanied by documentation stating the effect of each implementation-defined feature. For convenience, a non-normative checklist of implementation-defined features is provided at Appendix D – Checklist of Implementation-Defined Features on page 29.

Appendix A. References

A.1. Normative References

Character Model for the World Wide Web 1.0: Normalization

XQuery 1.0 and XPath 2.0 Data Model

XQuery 1.0 and XPath 2.0 Functions and Operators

HTML

IANA

RFC2046

RFC2119

RFC2278

RFC2854

RFC3236

Unicode Encoding

Unicode Character Encoding Model, Unicode Consortium. Unicode Standard Annex #17.

UAX #15: Unicode Normalization Forms

Unicode Normalization Forms, Unicode Consortium. Unicode Standard Annex #15.

XHTML 1.0

XHTML 1.1

XML10

XML11

XML Names

XML Names 1.1

XML Schema

XML Path Language (XPath) 2.0

XQuery 1.0: An XML Query Language

XSL Transformations (XSLT) Version 2.0

A.2. Informative References

XHTML Modularization

Appendix B. Summary of Error Conditions

This document uses the `err` prefix which represents the same namespace URI (http://www.w3.org/2005/xqt-errors) as defined in [XML Path Language (XPath) 2.0]. Use of this namespace prefix binding in this document is not normative.

It is an error if an item in S_6 in sequence normalization is an attribute node or a namespace node.

It is an error if the serializer is unable to satisfy the rules for either a well-formed XML document entity or a well-formed XML external general parsed entity, or both, except for content modified by the character expansion phase of serialization.

It is an error to specify the doctype-system parameter, or to specify the standalone parameter with a value other than `omit`, if the instance of the data model contains text nodes or multiple element nodes as children of the root node.

It is an error if the serialized result would contain an that contains a character that is not permitted by the version of Namespaces in XML specified by the `version` parameter.

It is an error if the serialized result would contain a character that is not permitted by the version of XML specified by the `version` parameter.

It is an error if an output encoding other than `UTF-8` or `UTF-16` is requested and the serializer does not support that encoding.

It is an error if a character that cannot be represented in the encoding that the serializer is using for output appears in a context where character references are not allowed (for example if the character occurs in the name of an element).

It is an error if the `omit-xml-declaration` parameter has the value `yes`, and the `standalone` attribute has a value other than `omit`; or the `version` parameter has a value other than `1.0` and the `doctype-system` parameter is specified.

It is an error if the output method is `xml`, the value of the `undeclare-prefixes` parameter is `yes`, and the value of the `version` parameter is 1.0.

It is an error if the value of the `normalization-form` parameter specifies a normalization form that is not supported by the serializer.

It is an error if the value of the `normalization-form` parameter is `fully-normalized` and any relevant construct of the result begins with a combining character.

It is an error if the serializer does not support the version of XML or HTML specified by the `version` parameter.

It is an error to use the HTML output method when characters which are legal in XML but not in HTML, specifically the control characters #x7F-#x9F, appear in the instance of the data model.

It is an error to use the HTML output method when `>` appears within a processing instruction in the data model instance being serialized.

It is a an error if a parameter value is invalid for the defined domain.

Appendix C. List of URI Attributes

The following list of attributes are declared as type `%URI` or `%UriList` for a given HTML or XHTML element, with the exception of the `name` attribute for element `A` which is not a URI type. The `name` attribute for element `A` should be escaped as is recommended by the HTML Recommendation [HTML] in Appendix B.2.1.

Attributes	Elements
action	FORM
archive	OBJECT
background	BODY
cite	BLOCKQUOTE, DEL, INS, Q
classid	OBJECT
codebase	APPLET, OBJECT
data	OBJECT
datasrc	BUTTON, DIV, INPUT, OBJECT, SELECT, SPAN, TABLE, TEXTAREA
for	SCRIPT
href	A, AREA, BASE, LINK
longdesc	FRAME, IFRAME, IMG
name	A
profile	HEAD
src	FRAME, IFRAME, IMG, INPUT, SCRIPT

Attributes	Elements
usemap	IMG, INPUT, OBJECT

Appendix D. Checklist of Implementation-Defined Features (Non-Normative)

This appendix provides a summary of Serialization features whose effect is explicitly implementation-defined. The conformance rules (see § 10 – Conformance on page 26) require vendors to provide documentation that explains how these choices have been exercised.

Appendix E. Revision Log (Non-Normative)

No substantive changes have been made to this document since the Proposed Recommendation Draft of 21 November 2006.

Appendix F. Changes since the First Edition (Non-Normative)

The changes made to this document are described in detail in the Errata to the first edition. The rationale for each erratum is explained in the corresponding Bugzilla database entry. The following table summarizes the errata that have been applied.

Erratum	Bugzilla	Category	Description
E1	4372	substantive	This erratum places constraints on the type of string that is valid for the doctype-public attribute of xsl:output.
E2	4557	editorial	This erratum corrects an editorial error concerning the number of phases of serialization.
E3	5066	editorial	This erratum corrects an editorial error concerning the currently registered XHTML media types.
E4	5433	substantive	This erratum clarifies how descendant elements of an XML island must be serialized according to the HTML output method.
E5	5439	substantive	This erratum aligns the description of the effect of the include-content-type serialization parameter of the HTML output method with that of the XHTML output method.
E6	5458	substantive	This erratum ensures that the sequence normalization process preserves any type annotations associated with nodes in the input sequence.
E7	5300	substantive	This erratum clarifies how elements with empty content models are to be serialized under the HTML and XHTML output methods.
E8	5441	substantive	This erratum ensures that Unicode normalization applies to all characters that might be adjacent in the serialized result produced by the text output method, including those that are in text nodes that are separated by element nodes in the data model instance.

Erratum	Bugzilla	Category	Description
E9	5993	substantive	This erratum makes previously non-normative text that describes how the xhtml and html output methods must behave if the indent parameter has the value yes into normative text.
E10	6466	substantive	This erratum specifies the syntactic constraints on the values of the doctype-public and doctype-system serialization parameters.
E11	6376	editorial	This erratum makes clear which parts of the recommendation are not considered to be normative.