| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LSSerializer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/org/w3c/dom/ls/LSResourceResolver.html)   [**NEXT CLASS**](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) | [**FRAMES**](http://docs.google.com/index.html?org/w3c/dom/ls/LSSerializer.html)    [**NO FRAMES**](http://docs.google.com/LSSerializer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

## **org.w3c.dom.ls**

Interface LSSerializer

public interface **LSSerializer**

A LSSerializer provides an API for serializing (writing) a DOM document out into XML. The XML data is written to a string or an output stream. Any changes or fixups made during the serialization affect only the serialized data. The Document object and its children are never altered by the serialization operation.

During serialization of XML data, namespace fixup is done as defined in [[DOM Level 3 Core](http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407)] , Appendix B. [[DOM Level 2 Core](http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113)] allows empty strings as a real namespace URI. If the namespaceURI of a Node is empty string, the serialization will treat them as null, ignoring the prefix if any.

LSSerializer accepts any node type for serialization. For nodes of type Document or Entity, well-formed XML will be created when possible (well-formedness is guaranteed if the document or entity comes from a parse operation and is unchanged since it was created). The serialized output for these node types is either as a XML document or an External XML Entity, respectively, and is acceptable input for an XML parser. For all other types of nodes the serialized form is implementation dependent.

Within a Document, DocumentFragment, or Entity being serialized, Nodes are processed as follows

* Document nodes are written, including the XML declaration (unless the parameter "xml-declaration" is set to false) and a DTD subset, if one exists in the DOM. Writing a Document node serializes the entire document.
* Entity nodes, when written directly by LSSerializer.write, outputs the entity expansion but no namespace fixup is done. The resulting output will be valid as an external entity.
* If the parameter " [entities](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-entities)" is set to true, EntityReference nodes are serialized as an entity reference of the form " &entityName;" in the output. Child nodes (the expansion) of the entity reference are ignored. If the parameter " [entities](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-entities)" is set to false, only the children of the entity reference are serialized. EntityReference nodes with no children (no corresponding Entity node or the corresponding Entity nodes have no children) are always serialized.
* CDATAsections containing content characters that cannot be represented in the specified output encoding are handled according to the " [split-cdata-sections](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-split-cdata-sections)" parameter. If the parameter is set to true, CDATAsections are split, and the unrepresentable characters are serialized as numeric character references in ordinary content. The exact position and number of splits is not specified. If the parameter is set to false, unrepresentable characters in a CDATAsection are reported as "wf-invalid-character" errors if the parameter " [well-formed](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-well-formed)" is set to true. The error is not recoverable - there is no mechanism for supplying alternative characters and continuing with the serialization.
* DocumentFragment nodes are serialized by serializing the children of the document fragment in the order they appear in the document fragment.
* All other node types (Element, Text, etc.) are serialized to their corresponding XML source form.

**Note:** The serialization of a Node does not always generate a well-formed XML document, i.e. a LSParser might throw fatal errors when parsing the resulting serialization.

Within the character data of a document (outside of markup), any characters that cannot be represented directly are replaced with character references. Occurrences of '<' and '&' are replaced by the predefined entities < and &. The other predefined entities (>, ', and ") might not be used, except where needed (e.g. using > in cases such as ']]>'). Any characters that cannot be represented directly in the output character encoding are serialized as numeric character references (and since character encoding standards commonly use hexadecimal representations of characters, using the hexadecimal representation when serializing character references is encouraged).

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "'", and the double-quote character (") as """. New line characters and other characters that cannot be represented directly in attribute values in the output character encoding are serialized as a numeric character reference.

Within markup, but outside of attributes, any occurrence of a character that cannot be represented in the output character encoding is reported as a DOMError fatal error. An example would be serializing the element <LaCa�ada/> with encoding="us-ascii". This will result with a generation of a DOMError "wf-invalid-character-in-node-name" (as proposed in " [well-formed](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-well-formed)").

When requested by setting the parameter " [normalize-characters](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-normalize-characters)" on LSSerializer to true, character normalization is performed according to the definition of [fully normalized](http://www.w3.org/TR/2004/REC-xml11-20040204/#dt-fullnorm) characters included in appendix E of [[XML 1.1](http://www.w3.org/TR/2004/REC-xml11-20040204/)] on all data to be serialized, both markup and character data. The character normalization process affects only the data as it is being written; it does not alter the DOM's view of the document after serialization has completed.

Implementations are required to support the encodings "UTF-8", "UTF-16", "UTF-16BE", and "UTF-16LE" to guarantee that data is serializable in all encodings that are required to be supported by all XML parsers. When the encoding is UTF-8, whether or not a byte order mark is serialized, or if the output is big-endian or little-endian, is implementation dependent. When the encoding is UTF-16, whether or not the output is big-endian or little-endian is implementation dependent, but a Byte Order Mark must be generated for non-character outputs, such as LSOutput.byteStream or LSOutput.systemId. If the Byte Order Mark is not generated, a "byte-order-mark-needed" warning is reported. When the encoding is UTF-16LE or UTF-16BE, the output is big-endian (UTF-16BE) or little-endian (UTF-16LE) and the Byte Order Mark is not be generated. In all cases, the encoding declaration, if generated, will correspond to the encoding used during the serialization (e.g. encoding="UTF-16" will appear if UTF-16 was requested).

Namespaces are fixed up during serialization, the serialization process will verify that namespace declarations, namespace prefixes and the namespace URI associated with elements and attributes are consistent. If inconsistencies are found, the serialized form of the document will be altered to remove them. The method used for doing the namespace fixup while serializing a document is the algorithm defined in Appendix B.1, "Namespace normalization", of [[DOM Level 3 Core](http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407)] .

While serializing a document, the parameter "discard-default-content" controls whether or not non-specified data is serialized.

While serializing, errors and warnings are reported to the application through the error handler (LSSerializer.domConfig's " [error-handler](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-error-handler)" parameter). This specification does in no way try to define all possible errors and warnings that can occur while serializing a DOM node, but some common error and warning cases are defined. The types ( DOMError.type) of errors and warnings defined by this specification are:

"no-output-specified" [fatal] Raised when writing to a LSOutput if no output is specified in the LSOutput. "unbound-prefix-in-entity-reference" [fatal] Raised if the configuration parameter " [namespaces](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-namespaces)" is set to true and an entity whose replacement text contains unbound namespace prefixes is referenced in a location where there are no bindings for the namespace prefixes. "unsupported-encoding" [fatal] Raised if an unsupported encoding is encountered.

In addition to raising the defined errors and warnings, implementations are expected to raise implementation specific errors and warnings for any other error and warning cases such as IO errors (file not found, permission denied,...) and so on.

See also the [Document Object Model (DOM) Level 3 Load and Save Specification](http://www.w3.org/TR/2004/REC-DOM-Level-3-LS-20040407).

| **Method Summary** | |
| --- | --- |
| [DOMConfiguration](http://docs.google.com/org/w3c/dom/DOMConfiguration.html) | [**getDomConfig**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#getDomConfig())()            The DOMConfiguration object used by the LSSerializer when serializing a DOM node. |
| [LSSerializerFilter](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) | [**getFilter**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#getFilter())()            When the application provides a filter, the serializer will call out to the filter before serializing each Node. |
| [String](http://docs.google.com/java/lang/String.html) | [**getNewLine**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#getNewLine())()            The end-of-line sequence of characters to be used in the XML being written out. |
| void | [**setFilter**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#setFilter(org.w3c.dom.ls.LSSerializerFilter))([LSSerializerFilter](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) filter)            When the application provides a filter, the serializer will call out to the filter before serializing each Node. |
| void | [**setNewLine**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#setNewLine(java.lang.String))([String](http://docs.google.com/java/lang/String.html) newLine)            The end-of-line sequence of characters to be used in the XML being written out. |
| boolean | [**write**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#write(org.w3c.dom.Node,%20org.w3c.dom.ls.LSOutput))([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg, [LSOutput](http://docs.google.com/org/w3c/dom/ls/LSOutput.html) destination)            Serialize the specified node as described above in the general description of the LSSerializer interface. |
| [String](http://docs.google.com/java/lang/String.html) | [**writeToString**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#writeToString(org.w3c.dom.Node))([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg)            Serialize the specified node as described above in the general description of the LSSerializer interface. |
| boolean | [**writeToURI**](http://docs.google.com/org/w3c/dom/ls/LSSerializer.html#writeToURI(org.w3c.dom.Node,%20java.lang.String))([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg, [String](http://docs.google.com/java/lang/String.html) uri)            A convenience method that acts as if LSSerializer.write was called with a LSOutput with no encoding specified and LSOutput.systemId set to the uri argument. |

| **Method Detail** |
| --- |

### getDomConfig

[DOMConfiguration](http://docs.google.com/org/w3c/dom/DOMConfiguration.html) **getDomConfig**()

The DOMConfiguration object used by the LSSerializer when serializing a DOM node.

In addition to the parameters recognized by the  [DOMConfiguration](http://www.w3.org/TR/DOM-Level-3-Core/core.html#DOMConfiguration) interface defined in [[DOM Level 3 Core](http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407)] , the DOMConfiguration objects for LSSerializer adds, or modifies, the following parameters: "canonical-form" true [*optional*] Writes the document according to the rules specified in [[Canonical XML](http://www.w3.org/TR/2001/REC-xml-c14n-20010315)]. In addition to the behavior described in " [canonical-form](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-canonical-form)" [[DOM Level 3 Core](http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407)] , setting this parameter to true will set the parameters "format-pretty-print", "discard-default-content", and "xml-declaration ", to false. Setting one of those parameters to true will set this parameter to false. Serializing an XML 1.1 document when "canonical-form" is true will generate a fatal error. false [*required*] (*default*) Do not canonicalize the output. "discard-default-content" true [*required*] (*default*) Use the Attr.specified attribute to decide what attributes should be discarded. Note that some implementations might use whatever information available to the implementation (i.e. XML schema, DTD, the Attr.specified attribute, and so on) to determine what attributes and content to discard if this parameter is set to true. false [*required*]Keep all attributes and all content. "format-pretty-print" true [*optional*] Formatting the output by adding whitespace to produce a pretty-printed, indented, human-readable form. The exact form of the transformations is not specified by this specification. Pretty-printing changes the content of the document and may affect the validity of the document, validating implementations should preserve validity. false [*required*] (*default*) Don't pretty-print the result. "ignore-unknown-character-denormalizations" true [*required*] (*default*) If, while verifying full normalization when [[XML 1.1](http://www.w3.org/TR/2004/REC-xml11-20040204/)] is supported, a character is encountered for which the normalization properties cannot be determined, then raise a "unknown-character-denormalization" warning (instead of raising an error, if this parameter is not set) and ignore any possible denormalizations caused by these characters. false [*optional*] Report a fatal error if a character is encountered for which the processor cannot determine the normalization properties. "normalize-characters" This parameter is equivalent to the one defined by DOMConfiguration in [[DOM Level 3 Core](http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407)] . Unlike in the Core, the default value for this parameter is true. While DOM implementations are not required to support [fully normalizing](http://www.w3.org/TR/2004/REC-xml11-20040204/#dt-fullnorm) the characters in the document according to appendix E of [[XML 1.1](http://www.w3.org/TR/2004/REC-xml11-20040204/)], this parameter must be activated by default if supported. "xml-declaration" true [*required*] (*default*) If a Document, Element, or Entity node is serialized, the XML declaration, or text declaration, should be included. The version (Document.xmlVersion if the document is a Level 3 document and the version is non-null, otherwise use the value "1.0"), and the output encoding (see LSSerializer.write for details on how to find the output encoding) are specified in the serialized XML declaration. false [*required*] Do not serialize the XML and text declarations. Report a "xml-declaration-needed" warning if this will cause problems (i.e. the serialized data is of an XML version other than [[XML 1.0](http://www.w3.org/TR/2004/REC-xml-20040204)], or an encoding would be needed to be able to re-parse the serialized data).

### getNewLine

[String](http://docs.google.com/java/lang/String.html) **getNewLine**()

The end-of-line sequence of characters to be used in the XML being written out. Any string is supported, but XML treats only a certain set of characters sequence as end-of-line (See section 2.11, "End-of-Line Handling" in [[XML 1.0](http://www.w3.org/TR/2004/REC-xml-20040204)], if the serialized content is XML 1.0 or section 2.11, "End-of-Line Handling" in [[XML 1.1](http://www.w3.org/TR/2004/REC-xml11-20040204/)], if the serialized content is XML 1.1). Using other character sequences than the recommended ones can result in a document that is either not serializable or not well-formed).

On retrieval, the default value of this attribute is the implementation specific default end-of-line sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by XML 1.0 or XML 1.1, depending on the serialized content. Setting this attribute to null will reset its value to the default value.

### setNewLine

void **setNewLine**([String](http://docs.google.com/java/lang/String.html) newLine)

The end-of-line sequence of characters to be used in the XML being written out. Any string is supported, but XML treats only a certain set of characters sequence as end-of-line (See section 2.11, "End-of-Line Handling" in [[XML 1.0](http://www.w3.org/TR/2004/REC-xml-20040204)], if the serialized content is XML 1.0 or section 2.11, "End-of-Line Handling" in [[XML 1.1](http://www.w3.org/TR/2004/REC-xml11-20040204/)], if the serialized content is XML 1.1). Using other character sequences than the recommended ones can result in a document that is either not serializable or not well-formed).

On retrieval, the default value of this attribute is the implementation specific default end-of-line sequence. DOM implementations should choose the default to match the usual convention for text files in the environment being used. Implementations must choose a default sequence that matches one of those allowed by XML 1.0 or XML 1.1, depending on the serialized content. Setting this attribute to null will reset its value to the default value.

### getFilter

[LSSerializerFilter](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) **getFilter**()

When the application provides a filter, the serializer will call out to the filter before serializing each Node. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.

The filter is invoked after the operations requested by the DOMConfiguration parameters have been applied. For example, CDATA sections won't be passed to the filter if " [cdata-sections](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-cdata-sections)" is set to false.

### setFilter

void **setFilter**([LSSerializerFilter](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) filter)

When the application provides a filter, the serializer will call out to the filter before serializing each Node. The filter implementation can choose to remove the node from the stream or to terminate the serialization early.

The filter is invoked after the operations requested by the DOMConfiguration parameters have been applied. For example, CDATA sections won't be passed to the filter if " [cdata-sections](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-cdata-sections)" is set to false.

### write

boolean **write**([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg,  
 [LSOutput](http://docs.google.com/org/w3c/dom/ls/LSOutput.html) destination)  
 throws [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html)

Serialize the specified node as described above in the general description of the LSSerializer interface. The output is written to the supplied LSOutput.

When writing to a LSOutput, the encoding is found by looking at the encoding information that is reachable through the LSOutput and the item to be written (or its owner document) in this order:

1. LSOutput.encoding,
2. Document.inputEncoding,
3. Document.xmlEncoding.

If no encoding is reachable through the above properties, a default encoding of "UTF-8" will be used. If the specified encoding is not supported an "unsupported-encoding" fatal error is raised.

If no output is specified in the LSOutput, a "no-output-specified" fatal error is raised.

The implementation is responsible of associating the appropriate media type with the serialized data.

When writing to a HTTP URI, a HTTP PUT is performed. When writing to other types of URIs, the mechanism for writing the data to the URI is implementation dependent.

**Parameters:**nodeArg - The node to serialize.destination - The destination for the serialized DOM. **Returns:**Returns true if node was successfully serialized. Return false in case the normal processing stopped but the implementation kept serializing the document; the result of the serialization being implementation dependent then. **Throws:** [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html) - SERIALIZE\_ERR: Raised if the LSSerializer was unable to serialize the node. DOM applications should attach a DOMErrorHandler using the parameter " [error-handler](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-error-handler)" if they wish to get details on the error.

### writeToURI

boolean **writeToURI**([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg,  
 [String](http://docs.google.com/java/lang/String.html) uri)  
 throws [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html)

A convenience method that acts as if LSSerializer.write was called with a LSOutput with no encoding specified and LSOutput.systemId set to the uri argument.

**Parameters:**nodeArg - The node to serialize.uri - The URI to write to. **Returns:**Returns true if node was successfully serialized. Return false in case the normal processing stopped but the implementation kept serializing the document; the result of the serialization being implementation dependent then. **Throws:** [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html) - SERIALIZE\_ERR: Raised if the LSSerializer was unable to serialize the node. DOM applications should attach a DOMErrorHandler using the parameter " [error-handler](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-error-handler)" if they wish to get details on the error.

### writeToString

[String](http://docs.google.com/java/lang/String.html) **writeToString**([Node](http://docs.google.com/org/w3c/dom/Node.html) nodeArg)  
 throws [DOMException](http://docs.google.com/org/w3c/dom/DOMException.html),  
 [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html)

Serialize the specified node as described above in the general description of the LSSerializer interface. The output is written to a DOMString that is returned to the caller. The encoding used is the encoding of the DOMString type, i.e. UTF-16. Note that no Byte Order Mark is generated in a DOMString object.

**Parameters:**nodeArg - The node to serialize. **Returns:**Returns the serialized data. **Throws:** [DOMException](http://docs.google.com/org/w3c/dom/DOMException.html) - DOMSTRING\_SIZE\_ERR: Raised if the resulting string is too long to fit in a DOMString. [LSException](http://docs.google.com/org/w3c/dom/ls/LSException.html) - SERIALIZE\_ERR: Raised if the LSSerializer was unable to serialize the node. DOM applications should attach a DOMErrorHandler using the parameter " [error-handler](http://www.w3.org/TR/DOM-Level-3-Core/core.html#parameter-error-handler)" if they wish to get details on the error.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/LSSerializer.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/org/w3c/dom/ls/LSResourceResolver.html)   [**NEXT CLASS**](http://docs.google.com/org/w3c/dom/ls/LSSerializerFilter.html) | [**FRAMES**](http://docs.google.com/index.html?org/w3c/dom/ls/LSSerializer.html)    [**NO FRAMES**](http://docs.google.com/LSSerializer.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).