Network Working Group
Request for Comments: 5261
Category: Standards Track

J. Urpalainen
Nokia

September 2008

An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors

Status of This Memo

Abstract

Extensible Markup Language (XML) documents are widely used as containers for the exchange and storage of arbitrary data in today's systems. In order to send changes to an XML document, an entire copy of the new version must be sent, unless there is a means of indicating only the portions that have changed. This document describes an XML patch framework utilizing XML Path language (XPath) selectors. These selector values and updated new data content constitute the basis of patch operations described in this document. In addition to them, with basic <add>, <replace>, and <remove> directives a set of patches can then be applied to update an existing XML document.

Table of Contents

1. Introduction

Extensible Markup Language (XML) [W3C.REC-xml-20060816] documents are widely used as containers for the exchange and storage of arbitrary data in today's systems. In order to send changes to an XML document, an entire copy of the new version must be sent, unless there is a means of indicating only the portions that have changed (patches).

This document describes an XML patch framework that utilizes XML Path language (XPath) [W3C.REC-xpath-19991116] selectors. An XPath selector is used to pinpoint the specific portion of the XML that is the target for the change. These selector values and updated new data content constitute the basis of patch operations described in this document. In addition to them, with basic <add>, <replace>, and <remove> directives a set of patches can be applied to update an existing target XML document. With these patch operations, a simple semantics for data oriented XML documents [W3C.REC-xmlschema-2-20041028] is achieved, that is, modifications like additions, removals, or substitutions of elements and attributes can easily be performed. This document does not describe a full XML diff format, only basic patch operation elements that can be embedded within a full format that typically has additional semantics.

As one concrete example, in the Session Initiation Protocol (SIP) [RFC3903] based presence system a partial PIDF XML document format [RFC5262] consists of the existing Presence Information Data Format (PIDF) document format combined with the patch operations elements described in this document. In general, patch operations can be used in any application that exchanges XML documents, for example, within the SIP Events framework [RFC3265]. Yet another example is XCAP-diff [SIMPLE-XCAP], which uses this framework for sending partial updates of changes to XCAP [RFC4825] resources.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119] and indicate requirement levels for compliant implementations.

The following terms are used in this document:

Target XML document: A target XML document that is going to be updated with a set of patches.

XML diff document: An XML document that contains patch operation elements, namespace declarations, and all the document content changes that are needed in order to transform a target XML document into a new patched XML document.

Patched XML document: An XML document that results after applying one or more patch operations defined in the XML diff document to the target XML document.

Patch operation: A single change, i.e., a patch that is being applied to update a target XML document.

Patch operation element: An XML element that represents a single patch operation.

Type definition for an element: A World Wide Web Consortium (W3C) Schema type definition for an element that describes a patch operation content.

In-scope namespace declaration: A list of all in-scope namespace declarations within a context node. The QName (qualified name) expansion of a context node is based on mapping a prefix with one of these declarations. For an element, one namespace binding may have an empty prefix.

Positional constraint: A number enclosed with square brackets. It can be used as a location step predicate.

Located target node: A node that was found from the target XML document with the aid of an XPath selector value.

White space text node: A text node that contains only white space.

3. Basic Features and Requirements

In this framework, XPath selector values and new data content are embedded within XML elements, the names of which specify the modification to be performed: <add>, <replace>, or <remove>. These elements (patch operations) are defined by schema types with the W3C Schema language [W3C.REC-xmlschema-1-20041028]. XPath selectors pinpoint the target for a change and they are expressed as attributes of these elements. The child node(s) of patch operation elements contain the new data content. In general when applicable, the new content SHOULD be moved unaltered to the patched XML document.

XML documents that are equivalent for the purposes of many applications MAY differ in their physical representation. The aim of this document is to describe a deterministic framework where the

canonical form with comments [W3C.REC-xml-c14n-20010315] of an XML document determines logical equivalence. For example, white space text nodes MUST be processed properly in order to fulfill this requirement as white space is by default significant [W3C.REC-xml-c14n-20010315].

The specifications referencing these element schema types MUST define the full XML diff format with an appropriate MIME type [RFC3023] and a character set, e.g., UTF-8 [RFC3629]. For example, the partial PIDF format [RFC5262] includes this schema and describes additional definitions to produce a complete XML diff format for partial presence information updates.

As the schema defined in this document does not declare any target namespace, the type definitions inherit the target namespace of the including schema. Therefore, additional namespace declarations within the XML diff documents can be avoided.

It is anticipated that applications using these types will define <add>, <replace>, and <remove> elements based on the corresponding type definitions in this schema. In addition, an application may reference only a subset of these type definitions. A future extension can introduce other operations, e.g., with document-oriented models [W3C.REC-xmlschema-2-20041028], a <move> operation and a text node patching algorithm combined with <move> would undoubtedly produce smaller XML diff documents.

The instance document elements based on these schema type definitions MUST be well formed and SHOULD be valid.

The following XPath 1.0 data model node types can be added, replaced, or removed with this framework: elements, attributes, namespaces, comments, texts, and processing instructions. The full XML prolog, including for example XML entities [W3C.REC-xml-20060816] and the root node of an XML document, cannot be patched according to this framework. However, patching of comments and processing instructions of the root node is allowed. Naturally, the removal or addition of a document root element is not allowed as any valid XML document MUST always contain a single root element. Also, note that support for external entities is beyond the scope of this framework.

4. Patch Operations

An XML diff document contains a collection of patch operation elements, including one or more <add>, <replace>, and <remove> elements. These patch operations will be applied sequentially in the document order. After the first patch has been applied to update a target XML document, the patched XML document becomes a new

independent XML document against which the next patch will be applied. This procedure repeats until all patches have successfully been processed.

4.1. Locating the Target of a Patch

Each patch operation element contains a 'sel' attribute. The value of this attribute is an XPath selector with a restricted subset of the full XPath 1.0 recommendation. The 'sel' value is used to locate a single unique target node from the target XML document. This located node pinpoints the target for a change and usually it is an element, which is for example either updated itself or some child node(s) are added into it. It MAY also be, for instance, a comment node, after which some other sibling node(s) are inserted. In any case, it is an error condition if multiple nodes are found during the evaluation of this selector value.

The XPath selections of the 'sel' attribute always start from the root node of a document. Thus, relative location paths SHOULD be used so that the starting root node selection "/" can be omitted. When locating elements in a document tree, a node test can either be a "*" character or a QName [W3C.REC-xml-names-20060816]. A "*" character selects all element children of the context node. Right after the node test, a location step can contain one or more predicates in any order. An attribute value comparison is one of the most typical predicates. The string value of the current context node or a child element may alternatively be used to identify elements in the tree. The character ".", which denotes a current context node selection, is an abbreviated form of "self::node()". Lastly, positional constraints like "[2]" can also be used as an additional predicate.

An XPath 1.0 "id()" node-set function MAY also be used to identify unique elements from the document tree. The schema that describes the content model of the document MUST then use an attribute with the type ID [W3C.REC-xmlschema-2-20041028] or with non-validating XML parsers, an "xml:id" [W3C.WD-xml-id-20041109] attribute MUST have been used within an instance document.

4.2. Namespace Mangling

The normal model for namespace prefixes is that they are local in scope. Thus, an XML diff document MAY have different prefixes for the namespaces used in the target document. The agent parsing the diff document MUST resolve prefixes separately in both documents in order to match the resulting QNames (qualified name) from each.

The XML diff document MUST contain declarations for all namespaces used in the diff document. The diff document declarations are always used to determine what namespaces apply within the diff document.

4.2.1. Namespaces Used in Selectors

A selector in a diff document may use prefixes when naming elements. If it does use a prefix, the prefix must be looked up in the diff document namespace declarations.

> For example, the patch operation element of a diff document has an in-scope namespace declaration "xmlns:a='foo:'" with a selector "sel='a:bar'". The agent processing this patch MUST then look for a 'bar' element qualified with the 'foo:' namespace regardless of whether the 'foo:' namespace has a prefix assigned in the target document or what that prefix is.

Default namespaces make this model a little more complicated. When the diff document has a default namespace declaration, any element selector without a prefix MUST be evaluated using that namespace.

> For example, the patch operation element of a diff document has an in-scope namespace declaration "xmlns='foo:'" with a selector "sel='bar'". The agent processing this patch MUST then look for a 'bar' element qualified with the 'foo:' namespace, regardless of whether the 'foo:' namespace has a prefix assigned in the target document or what that prefix is.

Unqualified names are also possible. If there is no default namespace declared, and an element name appears without a prefix, then it is an unqualified element name. If this appears in a selector, it MUST match an unqualified element in the target document.

> For example, the patch operation element of a diff document has only one in-scope namespace declaration "xmlns:a='foo:'" with a selector "sel='bar'". Since the 'bar' element has no prefix, and there is no default namespace declaration in scope, the agent processing this patch can only match the selector against a 'bar' element that has no prefix and also no default namespace in scope.

4.2.2. Departures from XPath Requirements

The prefix matching rules described previously in this section are different from those required in XPath 1.0 and 2.0 [W3C.REC-xpath20-20070123]. In XPath 1.0, a "bar" selector always locates an unqualified <bar> element. In XPath 2.0, a "bar" selector not only matches an unqualified <bar> element, but also matches a

qualified <bar> element that is in scope of a default namespace declaration. In contrast, in this specification, a selector without a prefix only matches one element, and it may match an element with or without a prefix but only if the namespace it's qualified with (or none) is an exact match.

The XPath 1.0 recommendation specifies "namespace-uri()" and "local-name()" node-set functions that can be used within predicates. These functions may be utilized during XPath evaluations if there are no other means to "register" prefixes with associated namespace URIs. They can also be used when handling selections where default namespaces are attached to elements. However, this specification does not allow the usage of these functions.

4.2.3. Namespaces and Added/Changed Content

Elements within the changed data content are also in scope of namespace declarations. For example, when adding a new namespace qualified element to the target XML document, the diff document MUST contain a namespace declaration that applies to the element. The agent processing the diff document MUST ensure that the target document also contains the same namespace declaration. Similar to XPath, the same namespace declaration in this context means that the namespace URIs MUST be equal, but the prefixes MAY be different in the diff and target documents.

For example, if a new added <a:bar> element has a namespace declaration reference to "xmlns:a='foo:'" in the diff document and the target document has only a single in-scope namespace declaration "xmlns:b='foo:'" at the insertion point, the namespace reference MUST be changed so that a <b:bar> element will then exist in the patched document. The same rule applies although default namespaces were used in either or both of the documents, the namespace URIs determine what will be the correct references (prefixes) in the patched document.

When the new or changed content has elements that declare new namespaces (locally scoped), these declarations are copied unaltered (prefix and everything) from the XML diff document to the target XML document. Default namespace declarations can only be added in this way, but prefixed namespace declarations MAY be added or removed with XPath namespace axis semantics shown later in this document (look Section 4.3.3).

A fairly difficult use case for these rules is found when the target document has several namespace declarations in scope for the same namespace. A target document might declare several different

prefixes for the same namespace. Normally, the agent applying the diff document chooses *the* appropriate prefix for adding new elements to the target document, but in this special case there's more than one. These requirements create deterministic behavior for this special and in practice rare case:

- If the diff document happens to use a prefix that is one of the prefixes declared for the same namespace in the evaluation context node of the target document, this prefix MUST be used in the resulting patched document. An empty evaluable prefix and an existing in-scope default namespace declaration means that the default namespace MUST be chosen. In other words, the expanded names are then equal within the diff and patched documents.

  In an <add> operation, the evaluation context node is the parent element of the inserted node, for example, with a selector "sel='*/ bar'" and without a 'pos' attribute directive (look Section 4.3), it is the <bar> element of the root document element. With modifications of elements, the evaluation context node is the parent element of the modified element, and in the previous example thus the root document element.

- Secondly, the prefix (also empty) of the evaluation context node MUST be chosen if the namespace URIs are equal.

- Lastly, if the above two rules still don't apply, first all in-scope namespace prefixes of the evaluation context node are arranged alphabetically in an ascending order. If a default namespace declaration exists, it is interpreted as the first entry in this list. The prefix from the list is then chosen that appears as the closest and just before the compared prefix if it were inserted into the list. If the compared prefix were to exist before the first prefix, the first prefix in the list MUST be selected (i.e., there's no default namespace).

  For example, if the list of in-scope prefixes in the target document is "x", "y" and the compared prefix in the diff document is "xx", then the "x" prefix MUST be chosen. If an "a" prefix were evaluated, the "x" prefix, the first entry MUST be chosen. If there were also an in-scope default namespace declaration, an evaluable "a" prefix would then select the default declaration. Note that unprefixed attributes don't inherit the default namespace declaration. When adding qualified attributes, the default namespace declaration is then not on this matching list of prefixes (see Section 4.3.2).

Note that these requirements might mean that a resulting patched document could contain unused and/or superfluous namespace declarations. The resulting patched document MUST NOT be "cleaned up" such that these namespace declarations are removed.

Note: In practice, the agent constructing a diff document can usually freely select the appropriate prefixes for the namespace declarations and it doesn't need to know or care about the actual prefixes in the target document unless there are overlapping declarations. In other words, the diff format content is typically independent of the target documents usage of namespace prefixes. However, it may be very useful to know where namespaces are declared in the target document. The most typical use case is such though, that the agent generating a diff has both the previous (target) and new (patched) documents available, and namespace declarations are thus exactly known. Note also, that in a case where the target document is not exactly known, it is allowed to use locally scoped namespace declarations, the consequences of which are larger and less human-readable patched documents.

4.3. <add> Element

The <add> element represents the addition of some new content to the target XML document: for example, a new element can be appended into an existing element.

The new data content exists as the child node(s) of the <add> element. When adding attributes and namespaces, the child node of the <add> element MUST be a single text node. Otherwise, the <add> element MAY contain any mixture of element, text, comment or processing instruction nodes in any order. All children of the <add> element are then copied into a target XML document. The described namespace mangling procedure applies to added elements, which include all of their attribute, namespace and descendant nodes.

The <add> element type has three attributes: 'sel', 'type', and 'pos'.

The value of the optional 'type' attribute is only used when adding attributes and namespaces. Then, the located target node MUST be an element into which new attributes and namespace declarations are inserted. When the value of this 'type' attribute equals "@attr", the string "attr" is the name of the actual attribute being added. The value of this new 'attr' attribute is the text node content of the <add> element. The less frequently used prefixed (i.e., namespace-qualified) attributes can also be added. If the value of the 'type' attribute equals "namespace::pref", "pref" is the actual

prefix string to be used for the namespace declaration in the patched document and the text node content of the <add> element contains the corresponding namespace URI.

> Note: The 'type' attribute is thus also an XPath selector, but it only locates attributes and namespaces. Attribute axis "attribute" has an abbreviated form "@" unlike the "namespace" axis, which doesn't have an abbreviated form. Double colons "::" are used as an axis separator in XPath.

The value of the optional 'pos' attribute indicates the positioning of new data content. It is not used when adding attributes or namespaces. When neither 'type' nor 'pos' attribute exist, the children of the <add> element are then appended as the last child node(s) of the located target element. When the value of 'pos' attribute is "prepend" the new node(s) are added as the first child node(s) of the located target element. With the value of "before", the added new node(s) MUST be the immediate preceding sibling node(s), and with "after", the immediate following sibling node(s) of the located target node.

Some examples follow that describe the use cases of these <add> element attributes. The nodes are not namespace qualified and prefixes are therefore not used, and the whole XML diff content is not shown in these examples, only patch operation elements. Full examples are given in an Appendix A.

4.3.1. Adding an Element

An example for an addition of an element:

```
<add sel="doc"><foo id="ert4773">This is a new child</foo></add>
```

Once the <doc> element has been found from the target XML document, a new <foo> element is appended as the last child node of the <doc> element. The located target node: the <doc> element is naturally the root element of the target XML document. The new <foo> element contains an 'id' attribute and a child text node.

4.3.2. Adding an Attribute

An example for an addition of an attribute:

```
<add sel="doc/foo[@id='ert4773']" type="@user">Bob</add>
```

This operation adds a new 'user' attribute to the <foo> element that was located by using an 'id' attribute value predicate. The value of this new 'user' attribute is "Bob".

A similar patched XML document is achieved when using a validating XML parser, if the 'sel' selector value had been 'id("ert4773")' and if the data type of the 'id' attribute is "ID" [W3C.REC-xmlschema-2-20041028].

Note that with namespace qualified attributes, the prefix matching rules within the 'type' attribute are evaluated with similar rules described in Section 4.2.3. Also, note that then the possible default namespace declaration of the context element isn't applicable.

> Note: As the 'sel' selector value MAY contain quotation marks, escaped forms: """ or "'" can be used within attribute values. However, it is often more appropriate to use the apostrophe (') character as shown in these examples. An alternative is also to interchange the apostrophes and quotation marks.

4.3.3. Adding a Prefixed Namespace Declaration

An example for an addition of a prefixed namespace declaration:

```
<add sel="doc" type="namespace::pref">urn:ns:xxx</add>
```

This operation adds a new namespace declaration to the <doc> element. The prefix of this new namespace node is thus "pref" and the namespace URI is "urn:ns:xxx".

4.3.4. Adding Node(s) with the 'pos' Attribute

An example for an addition of a comment node:

```
<add
  sel="doc/foo[@id='ert4773']" pos="before"><!-- comment --></add>
```

This operation adds a new comment node just before the <foo> element as an immediate preceding sibling node. This is also an example how a 'pos' attribute directive can be used.

4.3.5. Adding Multiple Nodes

Some complexity arises when so-called white space text nodes exist within a target XML document. The XPath 1.0 data model requires that a text node MUST NOT have another text node as an immediate sibling node. For instance, if an add operation is like this:

```
<add sel="doc">
  <foo id="ert4773">This is a new child</foo></add>
```

The <add> element then has two child nodes: a white space text node (a linefeed and two spaces) and a <foo> element. If the existing last child of the <doc> element is a text node, its content and the white space text node content MUST then be combined together. Otherwise, (white space) text nodes can be added just like elements, and thus, the canonical form of the patched XML document easily remains deterministic. As several sibling nodes can be inserted with a single <add> operation, a "pretty printing" style can easily be maintained.

Still another example about the handling of text nodes. Consider this example:

```
<add sel="*/foo/text()[2]" pos="after">new<bar/>elem</add>
```

The second text node child of the <foo> element is first located. The added new content contains two text nodes and an element. As there cannot be immediate sibling text nodes, the located target text node content and the first new text node content MUST be combined together. In essence, if the 'pos' value had been "before", the second new text node content would effectively have been prepended to the located target text node.

> Note: It is still worth noting that text nodes MAY contain CDATA sections, the latter of which are not treated as separate nodes. Once these CDATA sections exist within the new text nodes, they SHOULD be moved unaltered to the patched XML document.

While XML entities [W3C.REC-xml-20060816] cannot be patched with this framework, the references to other than predefined internal entities can exist within text nodes or attributes when the XML prolog contains those declarations. These references may then be preserved if both the XML diff and the target XML document have identical declarations within their prologs. Otherwise, references may be replaced with identical text as long as the "canonically equivalent" rule is obeyed.

4.4. <replace> Element

The <replace> element represents a replacement operation: for example, an existing element is updated with a new element or an attribute value is replaced with a new value. This <replace> operation always updates a single node or node content at a time.

The <replace> element type only has a 'sel' attribute. If the located target node is an element, a comment or a processing instruction, then the child of the <replace> element MUST also be of the same type. Otherwise, the <replace> element MUST have text

content or it MAY be empty when replacing an attribute value or a text node content.

4.4.1. Replacing an Element

An example for a replacement of an element:

```
<replace sel="doc/foo[@a='1']"><bar a="2"/></replace>
```

This will update the <foo> element that has an 'a' attribute with value "1". The located target element is replaced with the <bar> element. So all descendant nodes, namespace declarations, and attributes of the replaced <foo> element, if any existed, are thus removed.

4.4.2. Replacing an Attribute Value

An example for a replacement of an attribute value:

```
<replace sel="doc/@a">new value</replace>
```

This will replace the 'a' attribute content of the <doc> element with the value "new value". If the <replace> element is empty, the 'a' attribute MUST then remain in the patched XML document appearing like <doc a=""/>.

4.4.3. Replacing a Namespace Declaration URI

An example for a replacement of a namespace URI:

```
<replace sel="doc/namespace::pref">urn:new:xxx</replace>
```

This will replace the URI value of 'pref' prefixed namespace node with "urn:new:xxx". The parent node of the namespace declaration MUST be the <doc> element, otherwise an error occurs.

4.4.4. Replacing a Comment Node

An example for a replacement of a comment node:

```
<replace sel="doc/comment()[1]"><!-- This is the new content
--></replace>
```

This will replace a comment node. The located target node is the first comment node child of the <doc> element.

4.4.5. Replacing a Processing Instruction Node

An example for a replacement of a processing instruction node:

```
<replace sel='doc/processing-instruction("test")'><?test bar="foobar"
?></replace>
```

This will replace the processing instruction node "test" whose parent is the <doc> element.

4.4.6. Replacing a Text Node

An example for a replacement of a text node:

```
<replace
sel="doc/foo/text()[1]">This is the new text content</replace>
```

This will replace the first text node child of the <foo> element. The positional constraint "[1]" is not usually needed as the element content is rarely of mixed type [W3C.REC-xmlschema-1-20041028] where several text node siblings typically exist.

If a text node is updated and the <replace> element is empty, the text node MUST thus be removed as a text node MUST always have at least one character of data.

4.5. <remove> Element

The <remove> element represents a removal operation of, for example, an existing element or an attribute.

The <remove> element type has two attributes: 'sel' and 'ws'. The value of the optional 'ws' attribute is used to remove the possible white space text nodes that exist either as immediate following or preceding sibling nodes of the located target node. The usage of 'ws' attribute is only allowed when removing other types than text, attribute and namespace nodes. If the value of 'ws' is "before", the purpose is to remove the immediate preceding sibling node that MUST be a white space text node and if the value is "after", the corresponding following node. If the 'ws' value is "both", both the preceding and following white space text nodes MUST be removed.

4.5.1. Removing an Element

An example of a removal of an element including all of its descendant, attribute, and namespace nodes:

```
<remove sel="doc/foo[@a='1']" ws="after"/>
```

This will remove the <foo> element as well as the immediate following sibling white space text node of the <foo> element. If the immediate following sibling node is not a white space text node, an error occurs.

4.5.2. Removing an Attribute

An example for a removal of an attribute node:

```
<remove sel="doc/@a"/>
```

This will remove the 'a' attribute node from the <doc> element.

4.5.3. Removing a Prefixed Namespace Declaration

An example for a removal of a prefixed namespace node:

```
<remove sel="doc/foo/namespace::pref"/>
```

This will remove the 'pref' prefixed namespace node from the <foo> element. Naturally, this prefix MUST NOT be associated with any node prior to the removal of this namespace node. Also, the parent node of this namespace declaration MUST be the <foo> element.

4.5.4. Removing a Comment Node

An example for a removal of a comment node:

```
<remove sel="doc/comment()[1]"/>
```

This will remove the first comment node child of the <doc> element.

4.5.5. Removing a Processing Instruction Node

An example for a removal of a processing instruction node:

```
<remove sel='doc/processing-instruction("test")'/>
```

This will remove the "test" processing instruction node child of the <doc> element.

4.5.6. Removing a Text Node

An example for a removal of a text node:

```
<remove sel="doc/foo/text()[1]"/>
```

This will remove the first text node child of the <foo> element.

When removing an element, a comment, or a processing instruction node that has immediate preceding and following sibling text nodes without the 'ws' directive, the content of these two text nodes MUST be combined together. The latter text node thus disappears from the document.

5. Error Handling

It is an error condition if any of the patch operations cannot be unambiguously fulfilled. In other words, once a particular patch operation fails, it is an error condition and processing of further patch operations is hardly sensible.

A new MIME error format is defined for applications that require deterministic error handling when patching cannot be applied. It is anticipated that these error elements can be used within other MIME types that allow extension elements.

5.1. Error Elements

The root element of the error document is <patch-ops-error>. The content of this element is a specific error condition. Each error condition is represented by a different element. This allows for different error conditions to provide different data about the nature of the error. All error elements support a "phrase" attribute, which can contain text meant for rendering to a human user. The optional "xml:lang" MAY be used to describe the language of the "phrase" attribute. Most of the error condition elements are supposed to contain the patch operation element that caused the patch to fail.

The following error elements are defined by this specification:

<invalid-attribute-value>: The validity constraints of 'sel', 'type', 'ws', or 'pos' attribute values MAY be indicated with this error, i.e., non-allowable content has been used. Also, this error can be used to indicate if an added or a modified attribute content is not valid, for example, CDATA sections were used when a new attribute was intended to be added.

<invalid-character-set>: The patch could not be applied because the diff and the patched document use different character sets.

<invalid-diff-format>: This indicates that the diff body of the request was not a well-formed XML document or a valid XML document according to its schema.

<invalid-entity-declaration>: An entity reference was found but corresponding declaration could not be located or resolved.

<invalid-namespace-prefix>: The namespace URI for the given prefix could not be located or resolved, e.g., within the 'sel' attribute a prefix was used but its declaration is missing from the target document.

<invalid-namespace-uri>: The namespace URI value is not valid or the target document did not have this declaration.

<invalid-node-types>: The node types of a <replace> operation did not match, i.e., for example, the 'sel' selector locates an element but the replaceable content is of text type. Also, a <replace> operation may locate a unique element, but replaceable content had multiple nodes.

<invalid-patch-directive>: A patch directive could not be fulfilled because the given directives were not understood.

<invalid-root-element-operation>: The root element of the document cannot be removed or another sibling element for the document root element cannot be added.

<invalid-xml-prolog-operation>: Patch failure related to XML prolog nodes.

<invalid-whitespace-directive>: A <remove> operation requires a removal of a white space node that doesn't exist in the target document.

<unlocated-node>: A single unique node (typically an element) could not be located with the 'sel' attribute value. Also, the location of multiple nodes can lead to this error.

<unsupported-id-function>: The nodeset function id() is not supported, and thus attributes with the ID type are not known.

<unsupported-xml-id>: The attribute xml:id as an ID attribute in XML documents is not supported.

Additional error elements can be indicated within the root <patch-ops-error> element from any namespace. However, the IETF MAY specify additional error elements in the "urn:ietf:params:xml:ns:patch-ops-error" namespace.

As an example, the following document indicates that it was attempted to add a new <note> element with white space into a document, but the parent element could not be located:

```
<?xml version="1.0" encoding="UTF-8"?>
<patch-ops-error
 xmlns:p="urn:ietf:params:xml:ns:pidf-diff"
 xmlns="urn:ietf:params:xml:ns:patch-ops-error">
 <unlocated-node
  phrase="a unique node could not be located with the id() function."
  ><p:add sel='id("ert4773")'>
    <p:note>some text added</p:note>
  </p:add></unlocated-node>
</patch-ops-error>
```

6. Usage of Patch Operations

An XML diff document SHOULD contain only the nodes that have been modified as the intention is to try to reduce bandwidth/storage requirements. However, when there's a large collection of changes it can be desirable to exchange the full document content instead. How this will be done in practice is beyond the scope of this document.

Some applications MAY require that the full versioning history MUST be indicated although the history had superfluous changes. This framework doesn't mandate any specific behavior, applications MAY decide the appropriate semantics themselves. Also, in practice, applications are free to select the proper algorithms when generating diff document content.

7. Usage of Selector Values

It is up to the application to decide what kind of selector values to use. Positional element selectors like "*/*[3]/*[2]" provide the shortest selectors, but care must to taken when using them. When there are several removals of sibling elements, the positional element indexes change after each update. Likewise these indexes change when new elements are inserted into the tree. Using names with possible attribute predicates like "doc[@sel='foo']" is usually easier for an application, be it for example an auto diff tool, but it leads to larger diff documents.

8. XML Schema Types of Patch Operation Elements

The schema types for the patch operation elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema [
 <!ENTITY ncname "\i\c*">
 <!ENTITY qname  "(&ncname;:)?&ncname;">
 <!ENTITY aname  "@&qname;">
 <!ENTITY pos    "\[\d+\]">
```

```
    <!ENTITY attr   "\[&aname;='(.)*'\]|\[&aname;=&quot;(.)*&quot;\]">
    <!ENTITY valueq "\[(&qname;|\.)=&quot;(.)*&quot;\]">
    <!ENTITY value  "\[(&qname;|\.)='(.)*'\]|&valueq;">
    <!ENTITY cond   "&attr;|&value;|&pos;">
    <!ENTITY step   "(&qname;|\*)(&cond;)*">
    <!ENTITY piq    "processing-instruction\((&quot;&ncname;&quot;)\)">
    <!ENTITY pi     "processing-instruction\(('&ncname;')?\)|&piq;">
    <!ENTITY id     "id\(('&ncname;')?\)|id\((&quot;&ncname;&quot;)?\)">
    <!ENTITY com    "comment\(\)">
    <!ENTITY text   "text\(\)">
    <!ENTITY nspa   "namespace::&ncname;">
    <!ENTITY cnodes "(&text;(&pos;)?)|(&com;(&pos;)?)|((&pi;)(&pos;)?)">
    <!ENTITY child  "&cnodes;|&step;">
    <!ENTITY last   "(&child;|&aname;|&nspa;)">
   ]>
   <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified">

    <xsd:simpleType name="xpath">
     <xsd:restriction base="xsd:string">
      <xsd:pattern
       value="(/)?((&id;)((/&step;)*(/&last;))?|(&step;/)*(&last;))"/>
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="xpath-add">
     <xsd:restriction base="xsd:string">
      <xsd:pattern
       value="(/)?((&id;)((/&step;)*(/&child;))?|(&step;/)*(&child;))"/>
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="pos">
     <xsd:restriction base="xsd:string">
      <xsd:enumeration value="before"/>
      <xsd:enumeration value="after"/>
      <xsd:enumeration value="prepend"/>
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="type">
     <xsd:restriction base="xsd:string">
      <xsd:pattern value="&aname;|&nspa;"/>
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="add">
```

```
     <xsd:complexContent mixed="true">
      <xsd:restriction base="xsd:anyType">
       <xsd:sequence>
        <xsd:any processContents="lax" namespace="##any"
                 minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="sel" type="xpath-add"
                      use="required"/>
       <xsd:attribute name="pos" type="pos"/>
       <xsd:attribute name="type" type="type"/>
      </xsd:restriction>
     </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="replace">
     <xsd:complexContent mixed="true">
      <xsd:restriction base="xsd:anyType">
       <xsd:sequence>
        <xsd:any processContents="lax" namespace="##any"
                 minOccurs="0" maxOccurs="1"/>
       </xsd:sequence>
       <xsd:attribute name="sel" type="xpath" use="required"/>
      </xsd:restriction>
     </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="ws">
     <xsd:restriction base="xsd:string">
      <xsd:enumeration value="before"/>
      <xsd:enumeration value="after"/>
      <xsd:enumeration value="both"/>
     </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="remove">
     <xsd:attribute name="sel" type="xpath" use="required"/>
     <xsd:attribute name="ws" type="ws"/>
    </xsd:complexType>

   </xsd:schema>
```

9. XML Schema of Patch Operation Errors

The patch operation errors definitions.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="urn:ietf:params:xml:ns:patch-ops-error"
```

```
      xmlns:tns="urn:ietf:params:xml:ns:patch-ops-error"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      attributeFormDefault="unqualified">

    <!-- This import brings in the XML language attribute xml:lang-->
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
                schemaLocation="http://www.w3.org/2001/xml.xsd"/>

    <!-- ROOT document element for signaling patch-ops errors -->
    <xsd:element name="patch-ops-error">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:any namespace="##any" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:anyAttribute processContents="lax"/>
     </xsd:complexType>
    </xsd:element>

    <!-- patch-ops error elements:
         not intended to be used as root documnet elements -->
    <xsd:element name="invalid-attribute-value"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-character-set"
                 type="tns:patch-error-simple"/>
    <xsd:element name="invalid-diff-format"
                 type="tns:patch-error-simple"/>
    <xsd:element name="invalid-entity-declaration"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-namespace-prefix"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-namespace-uri"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-node-types"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-patch-directive"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-root-element-operation"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-xml-prolog-operation"
                 type="tns:patch-error"/>
    <xsd:element name="invalid-whitespace-directive"
                 type="tns:patch-error"/>
    <xsd:element name="unlocated-node"
                 type="tns:patch-error"/>
    <xsd:element name="unsupported-id-function"
                 type="tns:patch-error"/>
```

```
    <xsd:element name="unsupported-xml-id"
                 type="tns:patch-error"/>

    <!-- simple patch-ops error type  -->
    <xsd:complexType name="patch-error-simple">
     <xsd:attribute name="phrase" type="xsd:string"/>
     <xsd:attribute ref="xml:lang"/>
     <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

    <!-- error type which includes patch operation -->
    <xsd:complexType name="patch-error">
     <xsd:sequence>
      <xsd:any namespace="##any" processContents="lax"/>
     </xsd:sequence>
     <xsd:attribute name="phrase" type="xsd:string"/>
     <xsd:attribute ref="xml:lang"/>
     <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>

   </xsd:schema>
```

10. IANA Considerations

IANA has completed the following actions:

- registered a new XML namespace URN according to the procedures of RFC 3688 [RFC3688].

- registered a new MIME type 'application/patch-ops-error+xml' according to the procedures of RFC 4288 [RFC4288] and guidelines in RFC 3023 [RFC3023].

- registered two XML Schemas according to the procedures of RFC 3688 [RFC3688].

10.1. URN Sub-Namespace Registration

This specification registers a new XML namespace, as per the guidelines in RFC 3688 [RFC3688].

URI: The URI for this namespace is urn:ietf:params:xml:ns:patch-ops-error

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jari Urpalainen (jari.urpalainen@nokia.com).

```
XML:

BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
  "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Patch-Ops Error Namespace</title>
</head>
<body>
  <h1>Namespace for Patch-Ops Error Documents</h1>
  <h2>urn:ietf:params:xml:ns:patch-ops-error</h2>
  <p>See <a
  href="http://www.rfc-editor.org/rfc/rfc5261.txt">RFC5261</a>.</p>
</body>
</html>
END
```

10.2. application/patch-ops-error+xml MIME Type

MIME media type name: application

MIME subtype name: patch-ops-error+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [RFC3023].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [RFC3023].

Security considerations: See Section 10 of RFC 3023 [RFC3023].

Interoperability considerations: none.

Published specification: RFC 5261

Applications which use this media type: This document type has been used to support transport of Patch-Ops errors in RFC 5261.

Additional Information:

Magic Number: None

File Extension: .xer

Macintosh file type code: "TEXT"

Personal and email address for further information: Jari Urpalainen, jari.urpalainen@nokia.com

Intended usage: COMMON

Author/Change controller: The IETF

10.3. Patch-Ops-Types XML Schema Registration

This section registers a new XML Schema, the sole content of which is shown in Section 8.

URI:
urn:ietf:params:xml:schema:patch-ops

Registrant Contact:
IETF, SIMPLE working group, <simple@ietf.org>
Jari Urpalainen, <jari.urpalainen@nokia.com>

10.4. Patch-Ops-Error XML Schema Registration

This section registers a new XML Schema, the sole content of which is shown in Section 9.

URI:
urn:ietf:params:xml:schema:patch-ops-error

Registrant Contact:
IETF, SIMPLE working group, <simple@ietf.org>
Jari Urpalainen, <jari.urpalainen@nokia.com>

11. Security Considerations

Security considerations depend very much on the application that utilizes this framework. Since each application will have different needs, threat models, and security features, it will be necessary to consider these on an application-by-application basis.

However, this framework utilizes a limited subset of XPath 1.0. Applications may thus be vulnerable to XPath injection attacks that can reveal some non-allowable content of an XML document. Injection attacks are most likely with shareable resources where access to a resource is limited to only some specific parts for a user, contrary to a typical use case of this framework. To defend against those attacks the input MUST be sanitized which can be done, for example, by validating the diff formats with these restrictive schemas.

12. Acknowledgments

The author would like to thank Lisa Dusseault for her efforts including BoF arrangements, comments and editing assistance. The author would also like to thank Eva Leppanen, Mikko Lonnfors, Aki Niemi, Jonathan Rosenberg, Miguel A. Garcia, Anat Angel, Stephane Bortzmeyer, Dave Crocker, Joel Halpern, Jeffrey Hutzelman, David Ward, and Chris Newman for their valuable comments and Ted Hardie for his input and support.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[W3C.REC-xml-20060816] Maler, E., Paoli, J., Bray, T., Yergeau, F., and C. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", World Wide Web Consortium Recommendation REC-xml-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml-20060816>.

[W3C.REC-xpath-19991116] DeRose, S. and J. Clark, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[W3C.REC-xml-names-20060816]
Hollander, D., Bray, T., Layman, A., and R. Tobin, "Namespaces in XML 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xml-names-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml-names-20060816>.

[W3C.REC-xmlschema-1-20041028]
Beech, D., Thompson, H., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[W3C.REC-xml-c14n-20010315]
Boyer, J., "Canonical XML Version 1.0", World Wide Web Consortium Recommendation REC-xml-c14n-20010315, March 2001, <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>.

[W3C.REC-xmlschema-2-20041028]
Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[W3C.WD-xml-id-20041109]
Veillard, D., Walsh, N., and J. Marsh, "xml:id Version 1.0", W3C LastCall WD-xml-id-20041109, November 2004.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

13.2. Informative References

[W3C.REC-xpath20-20070123]
Berglund, A., Fernandez, M., Chamberlin, D., Boag, S., Robie, J., Kay, M., and J. Simeon, "XML Path Language (XPath) 2.0", World Wide Web Consortium Recommendation REC-xpath20-20070123, January 2007, <http://www.w3.org/TR/2007/REC-xpath20-20070123>.

[RFC4825] Rosenberg, J., "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", RFC 4825, May 2007.

[RFC3265] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[RFC5262] Lonnfors, M., Leppanen, E., Khartabil, H., and J. Urpalainen, "Presence Information Data format (PIDF) Extension for Partial Presence", RFC 5262, September 2008.

[SIMPLE-XCAP]
Urpalainen, J. and J. Rosenberg, "An Extensible Markup Language (XML) Document Format for Indicating A Change in XML Configuration Access Protocol (XCAP) Resources", Work in Progress, May 2008.

[RFC3903] Niemi, A., Ed., "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, October 2004.

Appendix A. Informative Examples

All following examples assume an imaginary XML diff document including these patch operation elements.

A.1. Adding an Element

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <add sel="doc"><foo id="ert4773">This is a new child</foo></add>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<foo id="ert4773">This is a new child</foo></doc>
```

A.2. Adding an Attribute

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<foo id="ert4773">This is a new child</foo></doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <add sel="doc/foo[@id='ert4773']" type="@user">Bob</add>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<foo id="ert4773" user="Bob">This is a new child</foo></doc>
```

A.3. Adding a Prefixed Namespace Declaration

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<foo id="ert4773">This is a new child</foo></doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <add sel="doc" type="namespace::pref">urn:ns:xxx</add>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:pref="urn:ns:xxx">
  <note>This is a sample document</note>
<foo id="ert4773">This is a new child</foo></doc>
```

A.4. Adding a Comment Node with the 'pos' Attribute

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<foo id="ert4773">This is a new child</foo></doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <add sel="doc/foo[@id='ert4773']" pos="before"><!-- comment --></add>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
<!-- comment --><foo id="ert4773">This is a new child</foo></doc>
```

A.5. Adding Multiple Nodes

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <add sel="doc">
  <foo id="ert4773">This is a new child</foo></add>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <note>This is a sample document</note>

  <foo id="ert4773">This is a new child</foo></doc>
```

A.6. Replacing an Element

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <replace sel="doc/foo[@a='1']"><bar a="2"/></replace>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <bar a="2"/>
</doc>
```

A.7. Replacing an Attribute Value

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc a="test">
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <replace sel="doc/@a">new value</replace>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc a="new value">
  <foo a="1">This is a sample document</foo>
</doc>
```

A.8. Replacing a Namespace Declaration URI

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:pref="urn:test">
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <replace sel="doc/namespace::pref">urn:new:xxx</replace>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:pref="urn:new:xxx">
  <foo a="1">This is a sample document</foo>
</doc>
```

A.9. Replacing a Comment Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:pref="urn:test">
  <foo a="1">This is a sample document</foo>
  <!-- comment -->
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <replace sel="doc/comment()[1]"><!-- This is the new content
   --></replace>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:pref="urn:test">
  <foo a="1">This is a sample document</foo>
  <!-- This is the new content
   -->
</doc>
```

A.10. Replacing a Processing Instruction Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  <?test foo="bar"?>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <replace sel='doc/processing-instruction("test")'
    ><?test bar="foobar"?></replace>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  <?test bar="foobar"?>
</doc>
```

A.11. Replacing a Text Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
 <replace sel="doc/foo/text()[1]"
   >This is the new text content</replace></diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is the new text content</foo>
</doc>
```

A.12. Removing an Element

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
 <remove sel="doc/foo[@a='1']" ws="after"/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  </doc>
```

A.13. Removing an Attribute

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc a="foo">
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
 <remove sel="doc/@a"/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

A.14. Removing a Prefixed Namespace Declaration

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1" xmlns:pref="urn:test"
   >This is a sample document</foo>
  <!-- comment -->
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <remove sel="doc/foo/namespace::pref"/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  <!-- comment -->
</doc>
```

A.15. Removing a Comment Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  <!-- comment -->
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <remove sel="doc/comment()[1]" ws="after"/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  </doc>
```

A.16. Removing a Processing Instruction Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
  <?test?>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <remove sel='doc/processing-instruction("test")'/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

A.17. Removing a Text Node

An example target XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1">This is a sample document</foo>
</doc>
```

An XML diff document:

```
<?xml version="1.0" encoding="UTF-8"?>
<diff>
  <remove sel="doc/foo/text()[1]"/>
</diff>
```

A result XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc>
  <foo a="1"/>
</doc>
```

A.18. Several Patches With Namespace Mangling

An example target XML document where namespace qualified elements exist:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns="urn:ietf:params:xml:ns:xxx"
     xmlns:z="urn:ietf:params:xml:ns:yyy">
  <note>This is a sample document</note>
  <elem a="foo">
    <child/>
  </elem>
  <elem a="bar">
    <z:child/>
  </elem>
</doc>
```

An imaginary XML diff document where prefix "p" corresponds the targetNamespace of this imaginary schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<p:diff xmlns="urn:ietf:params:xml:ns:xxx"
        xmlns:y="urn:ietf:params:xml:ns:yyy"
        xmlns:p="urn:ietf:params:xml:ns:diff">

<p:add sel="doc/elem[@a='foo']">  <!-- This is a new child -->
    <child id="ert4773">
      <y:node/>
    </child>
  </p:add>

<p:replace sel="doc/note/text()">Patched doc</p:replace>

<p:remove sel="*/elem[@a='bar']/y:child" ws="both"/>

<p:add sel="*/elem[@a='bar']" type="@b">new attr</p:add>

</p:diff>
```

One possible form of the result XML document after applying the patches:

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns="urn:ietf:params:xml:ns:xxx"
     xmlns:z="urn:ietf:params:xml:ns:yyy">
  <note>Patched doc</note>
  <elem a="foo">
    <child/>
    <!-- This is a new child -->
    <child id="ert4773">
      <z:node/>
    </child>
  </elem>
  <elem a="bar" b="new attr"/>
</doc>
```

The <node> and removed <child> element prefixes within the XML diff document are different than what are the "identical" namespace declarations in the target XML document. If the target XML document had used a prefixed namespace declaration instead of the default one, the XML diff document could still have been the same. The added new qualified elements would just have inherited that prefix.

Author's Address

Jari Urpalainen
Nokia
Itamerenkatu 11-13
Helsinki 00180
Finland

Phone: +358 7180 37686
EMail: jari.urpalainen@nokia.com

Full Copyright Statement

Intellectual Property