SIMPLE WG
Internet-Draft
Expires: March 18, 2006

J. Urpalainen
Nokia Research Center

September 14, 2005

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

draft-urpalainen-simple-xml-patch-ops-01

## Status of this Memo



## Copyright Notice



## Abstract

Extensible Markup Language (XML) documents are widely used as containers for the exchange and storage of arbitrary data in today's systems. Updates to this data requires transporting of the entire XML instance document between hosts, unless there's a mechanism that allows transmitting only the changes of XML documents. This memo describes a framework utilizing XML Path language (XPath) selectors with the aid of which a set of patches can be applied to an existing XML document.

Table of Contents

## 1. Introduction

Extensible Markup Language (XML) [2] documents are widely used as containers for the exchange and storage of arbitrary data in today's systems. An example of such a system is the Common Presence Profile (CPP) [11] compatible presence system, in which presence data is represented using the XML based Presence Information Data Format (PIDF) [12]. Updates to this data requires transporting of the entire XML instance document between hosts, unless there's a mechanism that allows transmitting only the changes of a document. This memo describes a patch framework which utilizes XML Path language (XPath) [3] selectors. With the aid of these selector values and changed data content a set of patches can then be applied to an existing XML document.

These changes or synonymously patch operations as used in this memo, are described by defining XML Schema element types which can be embedded within an application specific XML diff document. The full content of these application specific documents is not defined in this document, but instead, specifications utilizing these element types MUST define the full format with an appropriate MIME [7] type for the application.

An XPath selector is used to locate a single unique node from the existing XML document. Once the node which pinpoints the target for the modification has been found, modifications like additions, removals or substitutions of elements and attributes can be done.

As an example, in the Session Initiation Protocol (SIP) [13] based presence system a partial PIDF XML document format [9] consists of the existing PIDF document format combined with the patch operations elements. In general, the patch operations can be used in any application that exchanges XML documents, e.g. in the SIP Events [8] framework.

The aim of this memo is to describe a deterministic framework where only a single possible canonical form [4] of the XML document exists once the patches have been applied onto it. Especially significant whitespace text nodes MUST be processed properly in order to fulfil this requirement.

## 2. Conventions

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant implementations.

The following terms are used in this document:

XML diff document: This is the document that will carry all the patch operations, namespace declarations and all the document content changes. It is the frame XML document which includes all the patch operation elements.

Patched document: This is the local copy of the XML document onto which all the patch operations in the XML diff document are being applied.

Patch operation: This is a single change that will be applied to the patched document. Defined operations in this memo are: add, remove and replace. Corresponding XML elements contain XPath selector values and also added or changed document content.

## 3. Overview of patch operations

The XML diff document contains a collection of patch operations: add, replace or remove which will be applied one-by-one in the given order. Each of these XML Schema types 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 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.

When the patched document has namespace declarations, QName [5] expansion within the location step is evaluated according to the namespace declarations of the XML diff document. Thus the namespace URIs for the prefixes within the evaluation step strings are easily found from the XML diff document. If the patch operation element has an in-scope default namespace declaration and there is no prefix used in a node test then that step is interpreted as if it had had a prefixed name associated with this same namespace URI. This is of course only relevant when there's a reference to an element within the location step. It should be emphasized that prefix names within the XML diff document do not have to be the same than that of the patched document as node matching is based on the equivalent namespace URIs and local names.

While the XPath recommendation specifies that prefixes can be used in location steps, it does not specify how associated namespace URIs are to be found during the evaluations. However, it allows using "namespace-uri()" and "local-name()" functions within XPath predicates. In practice, these functions may then be utilized if there are no other means to "register" prefixes with associated

namespace URIs. The Schema types defined in this memo do not allow using these functions.

When locating the target node for a patch operation, all XPath selections start from the root node of the document. Thus only relative location paths are used. When locating elements in the document tree, the node test can be either a "*" character or a QName. A "*" character selects all element children of the context node. Attribute value comparisons can be used as predicates. Also text content of the current "." or a child element can alternatively be used to identify elements in the tree. The character "." is an abbreviated form of "self::node()". Positional constraints can also be used as an additional predicate. Ordering of these positional constraints and value comparisons can interchange. Finally XPath id() function can also be used to identify unique elements from the document tree.

As all the namespace declarations relevant to the patch operations are within the XML diff document, the element names within the optional data content are also fully namespace qualified. As with location step strings, the prefixes of these elements are not significant only the namespace URIs MUST match. However, if overlapping in-scope namespaces exist within the evaluation context, i.e. there are several in-scope namespaces with the same namespace URI, then the namespace with the same prefix is selected. The process that performs all these patch operations does not add new or change existing namespace declarations based on the prefixes and declarations of the XML diff document. Instead, it keeps the declarations that already exist within the patched document and matches XML nodes according to the local names and namespace URIs. If the intention is to add new namespace declarations to the patched document then their declaration MUST be within the added or changed data content or they MUST be explicitly added by using XPath namespace axis semantics shown later in this document.

The XML Schema defines element types for these patch operations. The XML Schema is intended to be included into the other XML Schemas that utilize these operations: e.g. partial PIDF [9]. It will provide a relevant XML diff document context. As this Schema does not declare a target namespace, elements defined according to these types inherit the target namespace of the including Schema. Furthermore, it is anticipated that applications using these types will define <add>, <replace> and <remove> elements from the corresponding types defined in this Schema. Also applications MAY either extend or restrict the types described in this document as some applications MAY not need all the features described in this document. The instance document elements based on these types MUST be well formed and SHOULD be valid.

### 3.1 <add> element

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

The value of the 'sel' attribute is used to select a single unique element from the document to be patched. It is an error condition if multiple nodes will be found during the evaluation of this selector value.

The value of the optional 'type' attribute is used to describe the type of the new data content. The new data content exists as child node(s) of this <add> element. The value of 'type' attribute is also an XPath 1.0 compatible selector with a very limited set of XPath features. Once the new content within the <add> element contains elements, they will include all the attribute, namespace and descendant nodes. As the default value of the 'type' attribute is "node()" the new content can be element, text, comment or processing instruction nodes or a mixture of them. The "node()" selector value is an abbreviated form of "child::node()". A positional constraint can also be used with the "node()" selector when only a single node needs to be added. If the value of the 'type' attribute equals "@attr" the purpose is to add a new 'attr' attribute. The value of the 'attr' attribute is then the text content of the <add> element. The less frequently used, prefixed attributes can also be added. If the value of the 'type' attribute equals "namespace::pref" the aim is to add a new "pref" prefixed namespace declaration and the text content of the <add> element is then the corresponding namespace URI.

The value of the optional 'pos' attribute indicates the positioning of the new data content. As the default value is "to", the new content is then simply added onto the found element based on the value of 'sel' selector. For other node types than attribute and namespace nodes, new content is appended as the last child node(s). With the value of "before" the new content MUST be the closest preceding sibling node(s) and with "after" the closest following sibling node(s). Naturally the usage of "before" and "after" is only allowed with other types than attributes and namespaces. They can be used e.g. when a comment node is added just before or after a particular element which was located based on the 'sel' selector value.

Appending elements with all the descendant and attribute nodes is one of the most typical operations. The default values of 'type' and 'pos' attributes allow that the <add> element MAY then only contain the 'sel' attribute with the added content.

Some examples without any namespaces in XPath selectors or elements and patch operation elements are also not having any namespaces

attached:

```
<add sel="root"><elem id="ert4773">This is a new child</elem></add>
```

Once the <root> element has been found from the document, an <elem> element is appended as the last child of the <root> element.

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

This operation adds a new 'user' attribute to the <elem> element. The value of this attribute is "Bob".

It should be noted that as the 'sel' selector value MAY contain quotation marks, escaped forms: " or ' can then be used. 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.

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

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

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

This operation adds a new comment node just before the <elem> element as the closest preceding sibling node.

Some complexity arises when so called whitespace text nodes exist within the patched document. The XPath 1.0 data model requires that a text node can not have another text node as a sibling node. For instance, if an add operation is like this:

```
<add sel="root">
  <elem id="ert4773">This is a new child</elem></add>
```

The <add> element in this example has then two child nodes: a whitespace text node and an <elem> element. If the last child of the <root> element is a text node, it's content and the whitespace text node content MUST then be catenated together. Otherwise whitespace text nodes can be added just like elements and thus the canonical form of the patched XML document easily remains deterministic.

It is worth noting that if text nodes contain XML Entities there declarations MUST be within the prologue of the framing XML diff document.

### 3.2 <replace> element

The <replace> element type has only one attribute: 'sel'. The value of this attribute is used to select a single unique node from the document. If the target node which was found based on the 'sel' selector value is an element, then the child of the <replace> element MUST also be an element. Otherwise the <replace> element MUST have text content. Examples for replace operations:

```
<replace sel="root/elem[@a='1']"><update a="2"/></replace>
```

This will replace the <elem> with the <update> element.

```
<replace sel="root/@a">new content</replace>
```

This will replace the attribute 'a' of the <root> element with the value "new content".

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

This will replace the URI value of 'pref' prefixed namespace node with "urn:new:xxx". It should be noted that the namespace declaration MUST then also exist within the <root> element.

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

This will replace the comment node content with the above string.

```
<replace sel="root/processing-instruction('foo')">bar="foobar"</
replace>
```

This will replace the content of the processing instruction node "foo".

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

This will replace the first text node content of the <elem> element. Usually the positional constraint e.g. "[1]" is not needed as the element content is rarely of mixed type [6].

### 3.3 <remove> element

The <remove> element type has two attributes: 'sel' and 'ws'. The value of the 'sel' attribute is used to select a single unique node from the document. The value of the optional 'ws' attribute is used to remove the possible whitespace text nodes that are either the closest following or preceding sibling nodes of the found node. The

usage of 'ws' attribute is only meaningful when removing other types than text, attribute and namespace nodes. As the default value of 'ws' attribute is "none", removal of whitespace nodes is thus not requested. If the value of 'ws' is "before", the purpose is to remove the closest preceding sibling node which MUST be a whitespace text node and if the value is "after", the corresponding following node. If the 'ws' value is "both", both the preceding and following whitespace text nodes MUST be removed. Examples for remove operations:

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

This will remove the <elem> element as well as the closest following sibling node of the <elem> element. This sibling node MUST be a whitespace text node.

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

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

```
<remove sel="root/namespace::pref"/>
```

This will remove the 'pref' prefixed namespace node from the <root> element. Naturally this prefix MAY not be used by any node prior to the removal of this namespace node. Also the namespace declaration MUST thus exist within the <root> element.

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

This will remove the first comment node from the <root> element.

```
<remove sel="root/processing-instruction('foo')"/>
```

This will remove the processing instruction node "foo" from the <root> element.

```
<remove sel="root/elem/text()[1]"/>
```

This will remove the first text node content from the <elem> element.

If for example an element, a comment node or a processing instruction node which has a whitespace text node as both the closest preceding and following node, is removed without a request to remove whitespaces, the content of these two whitespace nodes MUST then be catenated together to the remaining single whitespace node.

## 4. Error handling

It is an error condition if any of the given operations can not be unambiguously fulfilled. However, it is beyond the scope of this document to describe a generic error response.

## 5. Usage of patch operations

The XML diff document SHOULD contain only those nodes which have been modified. However, when there's a large collection of changes it MAY be desirable to transport the full document content instead. How this will be done in practice is beyond the scope of this document.

## 6. Examples

A document to be patched:

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

An imaginary XML diff document where prefix "p" corresponds to the targetNamespace of this imaginary XML 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="root/elem[@a='foo']">  <!-- This is a new child -->
    <child id="ert4773">
      <y:node>
    </child>
  </p:add>

<p:replace sel="root/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 resulting document after applying the patches:

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

## 7. XML Schema

The XML schema types for the patch operations.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE schema [
  <!ENTITY ncname  "[^:\I][^:\C]*">
  <!ENTITY qname   "(&ncname;:)?&ncname;">
  <!ENTITY aname   "@&qname;">
```

```
    <!ENTITY pos_t   "\[\d+\]">
    <!ENTITY attr_t  "\[&aname;=('|&quot;)(.)*('|&quot;)\]">
    <!ENTITY name_t  "\[(&qname;|\.)=('|&quot;)(.)*('|&quot;)\]">
    <!ENTITY cond    "(&attr_t;|&name_t;)?(&pos_t;)?|
                      (&pos_t;)?(&attr_t;|&name_t;)?">
    <!ENTITY step    "(&qname;|\*)(&cond;)?">

    <!ENTITY pi      "processing-instruction\
                      (('|&quot;)&qname;('|&quot;))?\)">
    <!ENTITY id      "id\((('|&quot;)&ncname;('|&quot;))?\)">
    <!ENTITY comm    "comment\(\)">
    <!ENTITY text    "text\(\)">
    <!ENTITY nspace  "namespace::&ncname;">
    <!ENTITY last    "&step;|&aname;|&nspace;|(&comm;(&pos_t;)?)|
                     &text;(&pos_t;)?|&pi;(&pos_t;)?">
  ]>
  <xsd:schema
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">

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

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

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

    <xsd:simpleType name="type">
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="node\(\)(&pos_t;)?"/>
        <xsd:pattern value="&aname;"/>
        <xsd:pattern value="&nspace;"/>
      </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-elem"
                         use="required"/>
          <xsd:attribute name="pos" type="pos"
                         default="to"/>
          <xsd:attribute name="type" type="type"
                         default="node()"/>
        </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="none"/>
        <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" default="none"/>
    </xsd:complexType>
```

```
</xsd:schema>
```

## 8. IANA Considerations

### 8.1 XML Schema Registration

This section registers a new XML Schema.

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

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

## 9. Security considerations

Information transported within these patch operations can be highly sensitive. Thus systems need to protect the integrity and confidentiality of this data. Especially, the transport protocol SHOULD have capabilities to protect from possible threats.

## 10. Acknowledgments

The author would like to thank Eva Leppanen, Mikko Lonnfors, Aki Niemi, Jonathan Rosenberg and Miguel A. Garcia for their valuable comments.

## 11. References

### 11.1 Normative references

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

[2] "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation REC-xml-20040204 , February 2004.

[3] "XML Path Language (XPath) Version 1.0", W3C Recommendation REC-xpath-19991116 , November 1999.

[4] "Canonical XML 1.0", W3C Recommendation REC-xml-c14n-20010315 , March 2001.

[5] "Namespaces in XML", W3C Recommendation REC-xml-names-19990114 , January 1999.

[6] "XML Schema Part 1: Structures Second Edition", W3C Recommendation REC-xmlschema-1-20041028 , October 2004.

## 11.2 Informative references

[7] Murata, M., "XML media types", RFC 3023, January 2001.

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

[9] Lonnfors, M., Leppanen, E., Khartabil, H., and J. Urpalainen, "Presence Information Data format (PIDF) Extension for Partial Presence", draft-ietf-simple-partial-pidf-format-05 (work in progress), September 2005.

[10] Lonnfors, M., Costa-Requena, J., Leppanen, E., and H. Khartabil, "Session Initiation Protocol (SIP) extension for Partial Notification of Presence Information", draft-ietf-simple-partial-notify-04 (work in progress), February 2005.

[11] Peterson, J., "Common Profile for Presence (CPP)", RFC 3859, August 2004.

[12] Sugano, H., "CPIM presence information data format", RFC 3863, May 2003.

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

## Author's Address

Jari Urpalainen
Nokia Research Center
Itamerenkatu 11-13
Helsinki 00180
Finland

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

## Intellectual Property Statement



## Disclaimer of Validity



## Copyright Statement



## Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.