Network Working Group
Internet-Draft
Updates: 5261 (if approved)
Intended status: Informational
Expires: December 20, 2013

E. Wilde
EMC

June 18, 2013

A Media Type for XML Patch Operations
draft-wilde-xml-patch-05

Abstract

The XML Patch media type "application/xml-patch+xml" defines an XML document structure for expressing a sequence of patch operations that are applied to an XML document. The XML Patch document format's foundations are defined in RFC 5261, this specification defines a document format and a media type registration, so that XML Patch documents can be labeled with a media type, for example in HTTP conversations.

In addition to the media type registration, this specification also updates RFC 5261 in some aspects, limiting these updates to cases where RFC 5261 needed to be fixed, or was hard to understand.

Note to Readers

This draft should be discussed on the apps-discuss mailing list [14].

Online access to all versions and files is available on github [15].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 20, 2013.

Copyright Notice

Table of Contents

1. Introduction

The Extensible Markup Language (XML) [1] is a common format for the exchange and storage of structured data. HTTP PATCH [6] extends HTTP [7] with a method to perform partial modifications to resources. HTTP PATCH requires that patch documents are being sent along with the request, and it is therefore useful if there are standardized patch document formats (identified by media types) for popular media types.

The XML Patch media type "application/xml-patch+xml" is an XML document structure for expressing a sequence of operations to apply to a target XML document, suitable for use with the HTTP PATCH method. Servers can freely choose which patch formats they want to accept, and "application/xml-patch+xml" could be a simple default format that can be used unless a server decides to use a different (maybe more sophisticated) patch format for XML.

The format for patch documents is based on the XML Patch Framework defined in RFC 5261 [2]. While RFC 5261 does define a concrete syntax as well as the media type "application/patch-ops-error+xml" for error documents, it only defines XML Schema (XSD) [8] types for patch operations, and thus the concrete document format and the media type for patch operations are defined in an XSD defined in this specification.

Since RFC 5261 contains sections that need to be fixed, or are hard to understand, this specification updates RFC 5261. The updates are listed in Appendix A, and all references to RFC 5261 made in this specification should be read as referring to the updated version. The main reason for the changes are the problematic ways in which RFC 5261 relies on XPath as the expression language for selecting the location of a patch, while at the same time XPath's data model does not contain sufficient information to determine whether such a selector indeed can be used for a patch operation, or should result in an error. Specifically, the problem occurs with namespaces, where XPath does not expose namespace decalration attributes, while the patch model needs them to determine whether a namespace patch is allowed or not. Section 6 contains more information about the general problem, and Appendix A lists the resulting updates to RFC 5261 to make the model well-defined and the text easier to read and understand.

2. Patch Document Format

The XML patch document format is based on a simple schema that uses a "patch" element as the document element, and allows an arbitrary

sequence of "add", "remove", and "replace" elements as the children of the document element. These children follow the semantics defined in RFC 5261, which means that each element is treated as an individual patch operation, and the result of each patch operation is a patched XML document that is the target XML document for the next patch operation.

The following example patch document uses the example from RFC 5261, and simply uses a "patch" element and a new XML namespace. It shows the general structure of an XML patch document, as well as an example for each operation.

```
<p:patch xmlns="urn:ietf:params:xml:ns:xxx"
         xmlns:y="urn:ietf:params:xml:ns:yyy"
         xmlns:p="urn:ietf:rfc:XXXX">
    <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:patch>
```

As this example demonstrates, both the document element "patch" and the patch operation elements are in the same XML namespace. This is the result of RFC 5261 only defining types for the patch operation elements, which then can be reused in schemas to define concrete patch elements.

RFC 5261 defines an XML Schema (XSD) [8] for the patch operation types, which is shown in Appendix A.4. The following schema for the XML Patch media type is based on the types defined in RFC 5261, which are imported as "rfc5261.xsd" in the following schema. The schema defines a "patch" document element, and then allows an unlimited (and possibly empty) sequence of the "add", "remove", and "replace" operation elements, which are directly based on the respective types from the schema defined in RFC 5261.

```
<xs:schema targetNamespace="urn:ietf:rfc:XXXX"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
    <xs:import schemaLocation="rfc5261.xsd"/>
    <xs:element name="patch">
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="add" type="add"/>
                <xs:element name="remove" type="remove"/>
                <xs:element name="replace" type="replace"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3. Patch Examples

Since the semantics of the XML patch operations are defined by RFC 5261, please refer to the numerous examples in that specification for concrete XML patch document examples. Most importantly, the examples in RFC 5261 can be taken literally as examples for the XML Patch media type, as long as it is assumed that the XML namespace for the operation elements in these examples is the URI "urn:ietf:rfc:XXXX".

4. IANA Considerations

The Internet media type [3] for an XML Patch Document is application/xml-patch+xml.

Type name: application

Subtype name: xml-patch+xml

Required parameters: none

Optional parameters: Same as charset parameter for the media type "application/xml" as specified in RFC 3023 [1].

Encoding considerations: Same as encoding considerations of media type "application/xml" as specified in RFC 3023 [1].

Security considerations: This media type has all of the security considerations described in RFC 3023 [1] and RFC 5261 [2], plus those listed in Section 5.

Interoperability considerations: N/A

Published specification: RFC XXXX

Applications that use this media type: Applications that manipulate XML documents.

Additional information:

Magic number(s): N/A

File extension(s): XML documents should use ".xml" as the file extension.

Macintosh file type code(s): TEXT

Person & email address to contact for further information: Erik Wilde <erik.wilde@emc.com>

Intended usage: COMMON

Restrictions on usage: none

Author: Erik Wilde <erik.wilde@emc.com>

Change controller: IETF

5. Security Considerations

Parsing XML may entail including information from external sources through XML's mechanism of external entities. Implementations therefore should be aware of the fact that standard parsers may resolve external entities, and thus include external information as a result of applying patch operations to an XML document.

6. Implementation Hints

This section is informative. It described some issues that might be interesting for implementers, but it might also be interesting for users of XML Patch that want to understand some of the differences between standard XPath 1.0 processing, and the processing model of selectors in RFC 5261.

6.1. Matching Namespaces

RFC 5261 defines standard rules for matching prefixed names in expressions: Any prefixes are interpreted according to the namespace bindings of the diff document (the document that the expression is applied against). This means that each prefixed name can be interpreted in the context of the diff document.

For unprefixed names in expressions, the rules depart from XPath 1.0 [9]. XPath 1.0 defines that unprefixed names in expressions match namespace-less names (i.e., there is no "default namespace" for names used in XPath 1.0 expressions). RFC 5261 requires, however, that unprefixed names in expressions must use the default namespace of the diff document (if there is one). This means that it is not possible to simply take a selector from a patch document and evaluate it in the context of the diff document according to the rules of XPath 1.0, because this would interpret unprefixed names incorrectly. As a consequence, it is not possible to simply take an XPath 1.0 processor and evaluate XMPL Patch selectors in the context of the diff document.

As an extension of XPath 1.0's simple model, XPath 2.0 [10] specifies different processing rules for unprefixed names: They are matched against the URI of the "default element/type namespace", which is defined as part of an expression's static context. In some XPath 2.0 applications, this can be set; XSLT 2.0 for example has the ability to define an "xpath-default-namespace", which then will be used to match unprefixed names in expressions. Thus, by using an XPath 2.0 implementation that allows to set this URI, and setting it to the default namespace of the diff document (or leaving it undefined if there is no such default namespace), it is possible to use an out-of-the-box XPath 2.0 implementation for evaluating XML Patch selectors.

Please keep in mind, however, that evaluating selectors is only one part of applying patches. When it comes to applying the actual patch operation, neither XPath 1.0 nor XPath 2.0 are sufficient, because they are not preserving some of the information from the XML syntax (specifically: namespace declarations) that is required to correctly apply patch operations. The following section described this issue in more detail.

Please note that RFC 5261's Section 4.2.2 on namespace matching explains XPath 2.0's rules incorrectly <http://tools.ietf.org/html/rfc5261#section-4.2.2>. For this reason, Appendix A.1 updates Section 4.2.2 of RFC 5261.

6.2. Patching Namespaces

One of the issues when patching namespaces based on XPath is that XPath exposes namespaces different than the XML 1.0 [11] syntax for XML Namespaces [12]. In the XML syntax, a namespace is declared with an attribute using the reserved name or prefix "xmlns", and this results in this namespace being available recursively through the document tree. In XPath, the namespace declaration is not exposed as an attribute (i.e., the attribute, although syntactically an XML attribute, is not accessible in XPath), but the resulting namespace nodes are exposed recursively through the tree.

RFC 5261 uses the terms "namespace declaration" and "namespace" almost interchangeably, but it is important to keep in mind that the namespace declaration is an XML syntax construct that is unavailable in XPath, while the namespace itself is a logical construct that is not visible in the XML syntax, but a result of a namespace declaration. The intent of RFC 5261 is to patch namespaces as if namespace declarations were patched, and thus it only allows to patch namespace nodes on the element nodes where the namespace has been declared.

Patching namespaces in XML Patch is supposed to "emulate" the effect of actually changing the namespace declaration (which is why a namespace can only be patched at the element where it has been declared). Therefore, when patching a namespace, even though XPath's "namespace" axis is used, implementations have to make sure that not only the single selected namespace node is being patched, but that all namespaces nodes resulting from the namespace declaration of this namespace are also patched accordingly.

This means that an implementation might have to descend into the tree, matching all namespace nodes with the selected prefix/URI pair recursively, until it encounters leaf elements or namespace declarations with the same prefix it is patching. Determining this requires access to the diff document beyond XPath, because in XPath itself namespace declarations are not represented, and thus such a recursive algorithm wouldn't know when to stop. Consider the following document:

```
<x xmlns:a="tag:42">
    <y xmlns:a="tag:42"/>
</x>
```

If this document is patched with a selector of /x/namespace::a, then only the namespace node on element x should be patched, even though the namespace node on element y has the same prefix/URI combination than the one on element x. However, determining that the repeated namespace declaration was present at all on element y is impossible

when using XPath alone, which means that implementations must have an alternative way to determine the difference between the document above, and this one:

```
<x xmlns:a="tag:42">
    <y/>
</x>
```

In this second example, patching with a selector of /x/namespace::a should indeed change the namespace nodes on elements x and y, because they both have been derived from the same namespace declaration.

The conclusion of these considerations is that for implementing XML Patch, access closer to the XML syntax (specifically: access to namespace declarations) is necessary. As a result, implementations attempting to exclusively use the XPath model for implementing XML Patch will fail to correctly address certain edge cases (such as the one shown above).

Note that XPath's specific limitations do not mean that it is impossible to use XML technologies other than XPath. The Document Object Model (DOM) [13], for example, does expose namespace declaration attributes as regular attributes in the document tree, and thus could be used to differentiate between the two variants shown above.

Please note that RFC 5261's Section 4.4.3 on replacing namespaces mixes the terms "namespace declaration" and "namespace". For this reason, Appendix A.2 updates Section 4.4.3 of RFC 5261.

7. Implementation Status

Note to RFC Editor: Please remove this section before publication.

As explained in a draft currently under development <http://tools.ietf.org/html/draft-sheffer-running-code>, this section contains information about implementation status, so that reviews of the draft document can take implementation reports into account as well. If you are implementing this draft, please contact this draft's author. Any implementation status reports are intended for draft publications only; the section will be removed when the draft is published in RFC form.

> EMC: We have implemented the selector part of the specification, which is the trickiest part (see Section 6.1 for an explanation). By reusing an existing XPath 1.0 implementation and changing it to match the changed default namespace processing model, the required behavior is fairly easy to implement. This does, however, require

that the implementation is available in source code, and also does require some changes to the implementation's code. The resulting implementation is closed source and will be made available, if released, as part of EMC's XML database product xDB <http://www.emc.com/products/detail/software2/documentum-xdb.htm>.

8. Change Log

Note to RFC Editor: Please remove this section before publication.

8.1. From -04 to -05

- Improved formatting of XML/XSD and ABNF code.
- Moving category from "std" to "info" (intended to become an informational RFC).

8.2. From -03 to -04

- Added text and section Appendix A about updating RFC 5261 (instead of relying on errata).

8.3. From -02 to -03

- Added section on "Implementation Status" (Section 7).
- Improved "Implementation Hints" (Section 6).

8.4. From -01 to -02

- Textual edits.
- Added section on "Implementation Hints" (Section 6).

8.5. From -00 to -01

- Removed Mark Nottingham from author list.
- Changed media type name to application/xml-patch+xml (added suffix per draft-ietf-appsawg-media-type-suffix-regs)
- Added ABNF grammar derived from XSD (Appendix A.5)

9. References

9.1. Normative References

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

[2] Urpalainen, J., "An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors", RFC 5261, September 2008.

[3] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

[4] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[5] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

9.2. Non-Normative References

[6] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[7] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[8] Thompson, H., Beech, D., 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>.

[9] 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>.

[10] Boag, S., Berglund, A., Kay, M., Simeon, J., Robie, J., Chamberlin, D., and M. Fernandez, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, December 2010, <http://www.w3.org/TR/2010/REC-xpath20-20101214>.

[11] Sperberg-McQueen, C., Yergeau, F., Paoli, J., Maler, E., and T.

Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[12] Hollander, D., Layman, A., Bray, T., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <http://www.w3.org/TR/2009/REC-xml-names-20091208>.

[13] Robie, J., Wood, L., Champion, M., Hegaret, P., Nicol, G., Le Hors, A., and S. Byrne, "Document Object Model (DOM) Level 3 Core Specification", World Wide Web Consortium Recommendation REC-DOM-Level-3-Core-20040407, April 2004, <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407>.

URIs

[14] <https://www.ietf.org/mailman/listinfo/apps-discuss>

[15] <https://github.com/dret/I-D/tree/master/xml-patch>

Appendix A. Updates to RFC 5261

This section is normative. It contains a list of updates to RFC 5261 [2]. These updates are limited to cases where RFC 5261 needed to be fixed, or was hard to understand.

A.1. Section 4.2.2

Section 4.2.2 of RFC 5261 [2] says:

> 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.

It should say:

> In XPath 2.0, a "bar" selector matches elements that have the URI of the "default element/type namespace", which is part of an XPath's static context. By setting this URI to the default namespace of the diff document (or leave it empty, if there is none), XPath 2.0's behavior matches the requirements of the

previous section.

Explanation: The original text is not easy to understand, but seems to assume that an unprefixed name in XPath 2.0 matches both unprefixed names, and prefixed ones that have the same namespace as the default namespace of the XPath static context. This is not the case: Matching depends on how the "default element/type namespace" of the XPath static context is defined, and then matches either namespace-less elements, or those in the "default element/type namespace", but never both. This context, however, is defined by the XPath itself, not by the document. Thus, it can be set externally and could be set to the diff document's default namespace (if there is one). In that case, XPath 2.0 can be used to evaluate XML Patch selectors.

A.2. Section 4.4.3

Section 4.4.3 of RFC 5261 [2] says:

> 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.

It should say:

> 4.4.3. Replacing a Namespace URI
>
> An example for a replacement of a namespace URI: <replace sel="doc/namespace::pref">urn:new:xxx</replace> This will replace the URI of the namespace associated with the 'pref' prefix with "urn:new:xxx". The parent node of the namespace declaration MUST be the <doc> element, otherwise an error occurs. Replacing the namespace at the element where it is declared MUST also change all namespace nodes derived from this declaration in descendant elements.

Explanation: The specification uses the terms "namespace declaration" and "namespace" almost interchangeably, which is incorrect. It is impossible to select namespace declarations using XPath. When selecting and replacing a namespace, then it should be taken into account that its associated namespace declaration very likely has resulted in numerous namespace nodes, attached to child elements of the element where the namespace was declared. It is likely that RFC 5261 intended to specify a "recursive replace" of the resulting

namespace nodes of a namespace declaration, and this is what the corrected text suggests. The original text is mixing terminology, hard to read, and ambiguous in its meaning.

Side note: If the original text indeed tried to specify that really only this one namespace node should be changed, then this could lead to rather strange effects in the resulting document, since the XPath tree now would have "orphan" namespace nodes, which then would need to be serialized, and there would be resulting namespace declarations in locations where previously no namespace declarations occurred.

A.3. Section 8

Section 8 of RFC 5261 [2] says:

```
<!ENTITY id "id\(('&ncname;')?\)|id\((&quot;&ncname;&quot;)?\)">
```

It should say:

```
<!ENTITY id "id\('&ncname;'\)|id\(&quot;&ncname;&quot;\)">
```

Explanation: The regex in the XSD suggests that "id()" would be a valid selector for a patch, but it would not make sense to specify such a selector, since it never would select a node (there's no identifier to locate in the document). This means that while "id()" is a valid XPath expression, it should not be allowed as a selector expression within an XML patch document.

A.4. XSD for RFC 5261

This section contains a modified copy of the XML Schema (XSD) [8] defining the add, replace, and remove types in RFC 5261 [2]. The modification is based on the grammar change made in Appendix A.3.

```
<!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>
```

A.5. ABNF for RFC 5261

RFC 5261 [2] does not contain an ABNF grammar for the allowed subset of XPath expressions, but includes an XSD-based grammar in its type definition for operation types (which is shown in Appendix A.4). In order to make implementation easier, this appendix contains an ABNF grammar that has been derived from the XSD expressions given in Appendix A.4. In the following grammar, "xpath" is the definition for the allowed XPath expressions for remove and replace operations, and "xpath-add" is the definition for the allowed XPath expressions for add operations. The names of all grammar productions are the ones used in the XSD-based grammar of RFC 5261.

```
   ncname     =  1*%x00-ffffffff
   qname      =  [ ncname ":" ] ncname
   aname      =  "@" qname
   pos        =  "[" 1*DIGIT "]"
   attr       =  ( "[" aname "='" 1*%x00-ffffffff "']" ) /
                 ( "[" aname "=" DQUOTE 1*%x00-ffffffff DQUOTE "]" )
   valueq     =  "[" ( qname / "." ) "=" DQUOTE 1*%x00-ffffffff DQUOTE "]"
   value      =  ( "[" ( qname / "." ) "='" 1*%x00-ffffffff "']" ) / valueq
   cond       =  attr / value / pos
   step       =  ( qname / "*" ) 0*( cond )
   piq        =  "processing-instruction(" [ DQUOTE ncname DQUOTE ] ")"
   pi         =  ( "processing-instruction(" [ "'" ncname "'" ] ")" ) / piq
   id         =  ( "id(" [ "'" ncname "'" ] ")" ) /
                 ( "id(" [ DQUOTE ncname DQUOTE ] ")" )
   com        =  "comment()"
   text       =  "text()"
   nspa       =  "namespace::" ncname
   cnodes     =  ( text / com / pi ) [ pos ]
   child      =  cnodes / step
   last       =  child / aname / nspa
   xpath      =  [ "/" ] ( ( id [ 0*( "/" step ) "/" last ] ) /
                 ( 0*( step "/" ) last ) )
   xpath-add  =  [ "/" ] ( ( id [ 0*( "/" step ) "/" child ] ) /
                 ( 0*( step "/" ) child ) )
```

Appendix B. Acknowledgements

Thanks for comments and suggestions provided by Bas de Bakker.

Author's Address

Erik Wilde
EMC
6801 Koll Center Parkway
Pleasanton, CA 94566
U.S.A.

Phone: +1-925-6006244
Email: erik.wilde@emc.com
URI: http://dret.net/netdret/