Approaches to change tracking in XML

Robin La Fontaine

Abstract

There are many different approaches to tracking document changes in XML. This paper looks at different use cases for situations where change to XML is important. We then review the different approaches used by some of the more popular formats, including OpenDocument, Open XML, DocBook, DITA and editors including XMetaL, oXygen and Xopus. These different approaches are discussed in order to illustrate the advantages and disadvantages of each approach in the context of the different use cases. As a prerequisite to the need to represent change, we first consider what constitutes a change in an XML document, as opposed to a change in an unstructured text document. This sets the scene for our more detailed discussion on the representation(s) of change.

Table of Contents

Introduction

XML is widely used to represent documents and data, and is acknowledged to be the de-facto standard in this area. However, almost all documents and data are subject to change, but XML does not in itself provide a way of representing change. This has resulted in a number of different approaches to the representation of change in XML, in order to meet a wide variety of use cases.

In reviewing the way that change is represented in XML, it is first necessary to look in some detail at what can be considered to be a change in an XML document or data file. In order to do this, it is necessary to ask the more fundamental question which is, "when are two documents or XML data files the same?". This may seem a rather trivial question, but there are many factors that mean that it does need to be considered carefully.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Having established what is a change, it is then necessary to look at the reasons why changes are important, and why they need to be recorded. There are different use cases for this, including needs of editors, reviewers, and software testers.

This paper reviews a number of different approaches to representing change in XML. If there are different requirements, and different solutions, it is interesting to look at the advantages and disadvantages of each solution against each of the different use cases.

NOTE: When we refer to 'XML documents' this includes the use of XML for data, as the discussion applies to the use of XML for both data and documents.

When are two XML documents the same?

Although it is obvious that if two XML documents have exactly the same sequence of characters, then they must be equal, there are many differences that are legitimately allowed between two XML documents without changing their meaning. For example, the XML specification states that the order of attributes has no semantic meaning, and therefore if two documents differ only in the order of the attributes they can be considered to be the same. Similarly, a user would also consider that a pretty-printed version of a document is the same as the original document. Therefore the handling of whitespace and layout is important when considering equality. There are different ways of representing namespaces, and namespace prefixes may be different, without any semantic difference between the documents.

[Canonical XML] seeks to address some of these issues by specifying a defined way to write an XML file such that files that contain exactly the same information (or as the specification says, "that are logically equivalent within an application context") will be represented by the same sequence of characters. Canonical XML recognises "that two documents may have differing canonical forms yet still be equivalent in a given context based on application-specific equivalence rules for which no generalized XML specification could account.".

These issues are important in actual use cases and might include one or more of the following:

1. ID attributes: The use of ID attribute may be important, because these are defined as unique within a document, and are often used as internal pointers. Therefore if two documents are the same in terms of their structure, attributes and elements, except that the ID values are different (but consistent as internal pointers) then it may be reasonable to consider that the two documents are still the same.

2. Ordering: It is often the case that in a document some information, for example meta-data, may be presented in any order, again with no semantic difference.

3. Processing Instructions (PIs) and Comments: Differences in comments and processing instructions again do not constitute changes to the actual XML documents, and may or may not be important as a difference to a user.

4. Equivalent data: A specific XML format may allow different representations for the same information.

Therefore we can see that it is necessary to be very clear about how to determine whether or not two documents are the same, before we can make a reasonable assessment of how best to represent change.

ID attributes

An example of the use of an ID attribute within a document is shown in the listing below. This is an extract from an [OpenDocument] XML file, and the ID is used to relate the start of the index marker and the end of the index marker. In this example, the actual values of the ID attributes `text:id="IMark665666424"`do not matter, but it is important that they are the same. If ID attributes like this are generated each time a document is saved, typically they will be different even when there is no change to the underlying document.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
<text:p text:style-name="Standard">The quick brown fox
                <text:alphabetical-index-mark-start
                text:id="IMark665666424"
                /> jumped<text:alphabetical-index-mark-end
                text:id="IMark665666424"/> over the
                lazy dog.</text:p>
```

It is only possible to ignore changes like this using knowledge of the semantics of the document and the use of particular ID attributes.

Ordering

In the OpenDocument example shown below, a number of meta-data items are shown. As each individual item of data is enclosed in an element, the order of these items could be changed without changing the semantics of the document. We would like therefore to be able to say that some or all of the elements with in the `office:meta` element can appear in any order.

```
 <office:meta>
        <meta:document-statistic meta:table-count="0"
            meta:image-count="0" meta:object-count="0"
            meta:page-count="1" meta:paragraph-count="1"
            meta:word-count="9" meta:character-count="45"/>
        <dc:date>2010-01-12T16:31:22</dc:date>
        <dc:creator>Robin </dc:creator>
        <meta:editing-duration>PT00H01M40S</meta:editing-duration>
        <meta:editing-cycles>1</meta:editing-cycles>
        <meta:generator>OpenOffice.org/3.1$Unix
            OpenOffice.org_project/310m19$Build-9420</meta:generator>
    </office:meta>
```

According to the definition of XML, the order of elements is always significant, and therefore it is not possible to cater for the situation described above in a standard way. However, it is simple to add an attribute to the parent element to indicate this, for example `dx:ordered="false"`.

Processing Instructions and Comments

Processing Instructions (PIs) allow documents to contain instructions for applications. PIs and comments do not form part of the document character data and therefore they may not be relevant when deciding if two documents are equal or not. Canonical XML may be generated either with or without comments, and these nodes may or may not be relevant to determining whether or not two documents are equal.

Equivalent data

A simple example of equivalent data is in XHTML where multiple whitespace characters are considered to be equivalent to a single whitespace character. Other examples might be different representations of the same numeric value, or different representations of a date. Two documents may be generated using different units of length, and then a length of "1000mm" would be the same as a length of "1m".

Similarly, it may be true that a `strong` (or `bold`) element which spans three words means exactly the same as if there were three `strong` elements each spanning one of the words. This of course is a function of the semantics of the particular XML schema in use.

The equivalence of two representations of the same data cannot be covered within a standard XML definition, but nevertheless it is very important when comparing documents to determine what has changed. If it is only the representation that has changed, then a user will generally prefer that this change is ignored.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Use cases for recording change in XML documents

There are many different use cases for the requirement to determine when documents have changed, or to represent the change in some way.

Documents

It is often important to see the changes between two or more versions of a document, either simply to review these or to accept or reject the changes. The reviewer may be interested in changes to the content and/or changes to the formatting. A reviewer probably does not expect to be able to see changes to some aspects of the document, for example meta-data.

One of the arguments for adopting XML is in order to separate content from layout, to allow repurposing of content into different types of media for publication. But XML is also extensively used to represent graphical items and layout. Although it is quite easy both to record and to show changes to the textual content of a document, it is much more difficult to record and show changes to formatting and layout.

Some document formats have built-in ways to represent changes. For example changes that are tracked or recorded will be recorded in the XML representation in OpenDocument or [Open XML]. Other mechanisms include revision or status flags ([DocBook] and [DITA]) to show where content has changed, even if the actual changes themselves are not represented. These built-in mechanisms generally do not record every sort of change, but only the changes that are deemed important to a user.

Editing

When an editor is working on a document, he or she may wish to track the changes that he is making. This is useful either as a record of the changes that have been made, or in introducing the ability to reverse one or more of those changes.

One of the issues with tracking changes is that the document may at certain times during the editing process be invalid XML or even not well-formed XML. As only well-formed XML can be saved in a file, and it is preferable not to save invalid XML, it only makes sense to track changes that are complete 'transactions' from one valid state of the document to another. Where the XML format is not used as the internal data representation for the document, the internal data structure of the editing system may be able to handle these intermediate states.

One other use case for editing needs to be mentioned: many editing systems allow two documents to be compared (typically in a side-by-side view) and each change accepted or rejected. However, as this only requires a representation of the changes within the editing system, it is not relevant to our discussions about representations of change in XML. XML does present challenges to editing systems here because it is all too easy for the acceptance/rejection of changes to destroy the tree-structure of the XML file.

Processing

It is often necessary to produce reports about how data has changed. If this data is represented as XML, then it becomes important to have the ability to determine changes and then process these changes in order to produce a report.

As a variation on this, it may be useful or necessary to generate actions as a result of change, for example to ensure that the changes in some XML data are replicated in another representation of the same data in a database. It may also be necessary to merge two XML documents, and in order to merge a comparison must be made in order to do determine which parts of the documents are the same.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

In this use case, any change to the XML content or attributes may be important and therefore potentially all changes to be represented.

Testing

Any software which is generating XML as an output will need to be tested. The need therefore arises to be able to compare the output from different versions of the software in order to determine whether or not anything has changed. Although much regression testing is satisfied simply by determining that two documents are the same, when they are different there is a need for some representation of the changes to be reviewed. The changes could be identified by line number, for example, or by XPath.

A Note on Comparison and Change Tracking

Editors often assume that they will get the same results from comparing two documents as they would have obtained by having change tracking 'on'. A comparison of two documents will show what has been changed, change tracking shows how the changes have been made. Therefore these are not the same, and the same result should not be expected. However, it often makes sense for the result of a comparison to be rendered into a change tracking format so that a user is then able to accept or reject the changes.

Representation of change in XML

There are several different ways of representing changes in XML. Although in general these are applied to the changes between two documents, some of them can be extended to show or represent changes between multiple documents, or multiple versions of the same document.

The example used here is to change "The **very** quick brown fox jumped over lazy dog." to "The quick brown fox jumped over **the** lazy dog.", where **bold** text shows changes. The examples have been shortened by removing some information that is not relevant to the discussion, and have been pretty-printed for clarity.

Line based diff

The traditional output of the UNIX `diff` utility shows changes between two text documents on a line by line basis. It is obviously possible to show differences between two XML documents in a similar way.

```
<         <p>The very quick brown fox jumped over lazy dog.</p>
---
>         <p>The quick brown fox jumped over the lazy dog.</p>
```

This representation has a number of limitations for XML because of its sytnax and tree structure neither of which is reflected in the line-based structure. It may be useful to accept or reject changes based on lines for a regular text document, but this is unlikely to work for an XML document where the structure is often easily destroyed by moving lines from one document to another.

Processing instructions

Because processing instructions are in effect external to the main structure of an XML document, they are commonly used to mark additions and deletions in XML editors. Examples include [XMetaL], [Xopus], and [oXygen].

One of the great advantages of using processing instructions to represent changes is that the underlying XML file can still be validated by ignoring the processing instructions. This implies that any deleted content will be within a processing instruction, and any added content will be marked by a start and end marker, each of which is a processing instruction.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
<topic id="topic-1">
    <title>Topic title</title>
    <body>
        <p>The <?oxy_delete author="robin"
                timestamp="20100113T140621+0000"
                content="very"?> quick
            brown fox jumped over
            <?oxy_insert_start author="robin"
            timestamp="20100113T140625+0000"?>the
            <?oxy_insert_end?>lazy dog.</p>
    </body>
    </topic>
```

In this example, you can see that if all of the processing instructions are removed, the result is a valid file which represents all the changes being accepted.

Revision flags

Attributes are often used to show revisions to parts of an XML document in order to generate output showing where a document has been revised. This mechanism is built into the XML format itself, and any processor would need to know about this in order to reflect the changes. Examples include DocBook and DITA.

```
<topic xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/">
    <title>Topic title</title>
    <body>
        <p>The <ph rev="deltaxml-delete">very </ph>
                quick brown fox jumped over
                <ph rev="deltaxml-add">the </ph>lazy dog.</p>
    </body>
</topic>
```

In this example, the revision flags have been inserted by comparing two versions of a document, and putting revision flags around text that has been either added or deleted. The added or deleted text can then be decorated in the publishing pipeline, either to PDF or HTML.

Tracked changes

Some document formats use a more sophisticated version of revision flags to show where text has been added or deleted. These tracked changes are represented in the XML structure, and the editing system may enable the editor to accept or reject them. Tracked changes are typically not able to represent all the possible changes to a document, but will satisfy the needs of a typical editor. Examples include OpenDocument Format (ODF) and Open XML. The example below is OpenDocument text format, ODT.

```
<office:body>
 <office:text>
  <text:tracked-changes>
   <text:changed-region text:id="ct528047904">
    <text:deletion>
     <text:p text:style-name="Standard">
      very
     </text:p>
    </text:deletion>
   </text:changed-region>
   <text:changed-region text:id="ct645104016">
    <text:insertion />
   </text:changed-region>
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  </text:tracked-changes>
  <text:p text:style-name="Standard">
   The
   <text:change text:change-id="ct528047904" />
   <text:s />
   quick brown fox jumped over
   <text:change-start text:change-id="ct645104016" />
   the
   <text:change-end text:change-id="ct645104016" />
   lazy dog.
  </text:p>
 </office:text>
</office:body>
```

In this example of tracked changes, notice that the deleted text is held in a separate place from the main body text. This means that the main body of the document is very close to the new version of the document, i.e. with all the changes accepted. However, it is not trivial to reinsert the deleted text in its correct position, with all of the text decoration intact.

Generic XML deltas

A delta file can be defined such that it represents the differences between two arbitrary XML documents, in XML. Any XML format that is capable of updating one XML document into another could be described as a generic delta, for example [XQuery] Update Facility, [XSLT] or [DeltaXML]. Typically this will operate in one direction only (XQuery Update Facility, XSLT),though a symmetrical representation is also possible (DeltaXML). The delta may be a transformation, defining how to get from one document to another, or a data representation, defining what is different between the documents.

XQuery Update Facility and XSLT are both declarative transformations and can represent complex changes. They are intended to be executable transformations between two XML documents and will, in conjunction with the execution engine, convert one document into another. They are not intended to be used as a change-tracking mechanism. They do not meet any of the needs of the use case scenarios described here.

A delta file that is a data representation describes in some way the differences between two documents. A useful derivation of such a generic XML delta file would be one that contained not only the changes but also the original data, all in XML. Both of the original documents could be generated from such a delta representation. Ideally such a delta representation would not duplicate content that is common to the two documents. It is possible to transform this type of data representation into any of the other types of representation listed above, although the reverse is in general not possible. Therefore this generic delta representation is very versatile. An example of this is the full-context delta used by DeltaXML, shown in the example below.

```
<para deltaxml:deltaV2="A!=B"
        deltaxml:version="2.0"
        deltaxml:content-type="full-context">
 The
 <deltaxml:textGroup deltaxml:deltaV2="A">
  <deltaxml:text deltaxml:deltaV2="A">
   very
  </deltaxml:text>
 </deltaxml:textGroup>
 quick brown fox jumped over
 <deltaxml:textGroup deltaxml:deltaV2="B">
  <deltaxml:text deltaxml:deltaV2="B">
   the
  </deltaxml:text>
 </deltaxml:textGroup>
```

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
 lazy dog.
</para>
```

In this example, all of the data from both documents, A and B, is present and has the same look and feel as the original documents. The delta element wrappers and attributes indicate where the documents differ. Either version of the document can quite easily be extracted from this representation, for example using XSLT or XQuery. Note that the actual delta file will not comply with the original DTD/schema because of the additional delta wrapper elements and attributes, but each version that is extracted will be valid against the DTD/schema. Although not shown in this example, the format is capable of representing changes to attributes and elements as well as text. The format also extends to represent changes between more than two documents, for example the changes between two concurrent edits and the document from which the edits are derived.

Discussion

Having established that there are different use cases for needing to have a representation of change in XML, we can look at the advantages and disadvantages of the different representations for each of these use cases.

The issues here in include:

1. Does it allow all changes to be represented, e.g. changes to attributes?
2. Is it generic to all XML or specific to a particular format?
3. Does it allow processing using standard XML processors, e.g. XSLT?
4. Does it support accept/reject changes?
5. Is the result, after accepting or rejecting one or more changes, still valid and/or well-formed?

For a particular use case, not all of these features will be relevant. The table below provides a summary of which features are needed within each particular use case.

Table 1. Features Required for each Use case

Feature	**Required for Documents?**	**Required for Editing?**	**Required for Processing?**	**Required for Testing?**
All changes need to be represented	Not needed because it is mainly the content changes that need to be shown	Not needed when editing documents, though could be useful for a generic XML editor	Yes	Useful though not necessary
Needs to be generic to all XML not specific to a particular format	No, a document can have its own 'tracked change' format	Useful	Yes	Yes
Processable using standard XML processors	Useful	Not needed	Yes	Not needed
Supports accept/reject changes	Yes	Yes	Yes	Not needed
Result, after accepting or rejecting one or more changes, must be valid and/or well-formed	Valid	Valid	Well-formed only needed because output may be different XML	Not needed

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

We can also look at how each change represention performs against these features.

Table 2. Features of each Change Tracking format

Feature	Tracked changes or Revision flags	Processing instructions	Generic XML deltas	Line based diff
All changes need to be represented	No	No, cannot handle attributes	Yes	Yes but not XML-aware
Needs to be generic to all XML not specific to a particular format	No	Yes	Yes	Yes but not XML-aware
Processable using standard XML processors	Yes	Possible but not easy as deleted items need to be re-parsed	Yes	No
Supports accept/reject changes	Yes	Yes	Yes	No
Result, after accepting or rejecting one or more changes, must be valid and/or well-formed	Valid	Valid	Well-formed, may be valid	No

The two tables have been deliberately lined up so that the middle three columns show a good match between the requirements for particular use cases, and specific change tracking representations:

The feature requirements for documents are met by tracked changes and revision flags.
The requirements for editing are a good match with processing instructions.
The requirements for general processing are met by generic XML deltas.
The line based diff does not really meet any requirements for XML documents.

It is also possible to see from the second table that it would be possible to convert the processing instructions into tracked changes or revision flags, although this may not be easy. However the generic XML delta could be converted into any of the other formats because it is a more complete and generic representation of changes, as well as being processable.

Design of XML for change

Given the wide range of use cases, someone who is designing an XML format may want to consider some of these at the design stage. For example, is it always clear when two documents are the same? Is there a clear distinction between the textual content and the formatting? Is it easy to represent changes to either or both of these?

It is possible to draw some conclusions and design guidelines for a DTD/schema designer such that the resultant DTD/schema specifies documents that are easy to compare, or where change can be represented well.

Do not use attibutes for data that may change. If some information is liable to change and the change needs to be represented, then it is better to include the information as content rather than as an attribute. This is because it is easier to show changes to content than to show changes to attributes. [SVG] is an example of an XML format which uses attributes inappropriately for information content, e.g. a list of geometric points. When comparing SVG, it is difficult to show which point in a list has changed, because XML structure cannot be introduced into an attribute. For documents, traditionally the text content is held as PCDATA and all formating and other information tends to be held as attributes. This works as a convention for documents, but for data representation consider having all content in PCDATA and use attributes sparingly. Remember you can never develop an attribute later to give it more XML structure, whereas you can always add substructure to elements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ID attributes should be persistent. This cannot always be achieved, but it does make comparison - and therefore testing - much easier if applications that read and write the XML are required to maintain the value of ID attributes.

Unordered content should be enclosed in a single element. We have discussed the fact that some content can appear in any order, and it is best if such content is not mixed with ordered content. Therefore it is good practice to enclose such content in a container element. In particular, do not put data in attributes just because data is unordered, because that is a very poor reason for using attributes.

Consider which attributes or content constitute keys, and make these persistent. Keys are needed in many situations and help cross-references within and from outside a document. If you keep them persistent, then you can use them to reference from outside a document and comparison and testing is easier. Keys can be used to control alignment in comparison and therefore more accurate and predictable results can be obtained.

Avoid mixed content. For documents in XML this is almost impossible, but for data it is almost always possible to avoid mixed content, i.e. having both PCDATA and elements as siblings. Representing change to mixed content is difficult because the PCDATA needs to be parsed and divided, for example into words, in order to get sensible results.

Conclusions

There are several different ways of tracking change to XML documents. Each representation has its advantages in a particular use case scenario, except line-based diff which is inappropriate for XML. Many document formats have change tracking built into the format, and these range from basic revision flags on added or deleted text to a richer recording of changes to text and formatting. For editing documents, processing instructions enable a degree of 'undo' support for changes made to the document. Generic delta representations enable sophisticated processing of all changes to any XML document, generating reports or update scripts in almost any format. XML therefore has many advantages over line-based formats for generic change processing.

It is possible to convert from one change representation into another, in particular the generic delta can be converted into the other representations.

It is useful for anyone involved in designing or processing XML to have a basic understanding of how change can be detected and represented in order to choose appropriate solutions either at the DTD/schema design stage or when processing XML.

Bibliography

[CanonicalXML] Canonical XML Version 1.0: W3C Recommendation 15 March 2001. http://www.w3.org/TR/xml-c14n

[OpenDocument] ISO/IEC 26300:2006 Open Document Format for Office Applications (OpenDocument). http://www.iso.org/iso/catalogue_detail.htm?csnumber=43485

[Open XML] Standard ECMA-376 Office Open XML File Formats. http://www.ecma-international.org/publications/standards/Ecma-376.htm

[DocBook] Norman, Walsh: The DocBook Schema. Working Draft 5.0a1, OASIS, 29 June 2005. http://www.docbook.org/specs/wd-docbook-docbook-5.0a1.html

[DITA] Darwin Information Typing Architecture (DITA) http://dita.xml.org/

[XMetaL] XMetaL authoring systemhttp://na.justsystems.com/content-xmetal [http://na.justsystems.com/content-xmetall]

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

[Xopus] Xopus online editinghttp://xopus.com/xopus-web-based-wysiwyg-xml-editor.html

[oXygen] oXygen XML editor http://www.oxygenxml.com/

[XQuery] XQuery Update Facility 1.0. http://www.w3.org/TR/xquery-update-10/

[XSLT] XSL Transformations (XSLT). http://www.w3.org/TR/xslt

[DeltaXML] DeltaXML: Managing change in an XML environment http://www.deltaxml.com/

[SVG] Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/