XML Change Tracking
Representing Change Tracking in any XML Document

Version: DRAFT 8, 2013-03-21
This version has been updated to note the potential need for an external representation of the change tracking information as suggested at discussions during XML Prague 2012.
Version: DRAFT 7, 2012-08-14
This version has been updated to extend the introduction, re-organise some of the main content and to include a Processing Instruction (PI) representation so that the format can be used by XML editing applications that commonly use PIs to track changes.
Version: DRAFT 6, 2012-01-27
This version has been updated to emphasise the application of this proposal to XML generally.
Version: DRAFT 5, 2011-08-16
This version has been updated to move deleted text out from the content area and also to add meta-data so a change transaction can be identified as a particular type of edit operation.This version also provides an alternative representation for attribute changes. These updates are proposals and will need more detailed drafting if this approach is adopted.
Version: DRAFT 4, 2010-07-23
Author: Robin La Fontaine, Tristan Mitchell, DeltaXML Ltd.
This document has been sponsored by NLnet on behalf of the OpenDoc Society for submission to OASIS, ISO and W3C.

1 Abstract

This document describes a generic method for tracking changes in XML documents. The format described provides a way of representing successive changes or edits to an XML document, typically in one or more editing sessions. The document describes how changes may be represented in XML markup or in Processing Instructions. The tracked changes are designed to be used either as an independent addition to a file or integrated into the applicable schema. The OpenDocument Format (ODF) is used in the worked examples.

Table of Contents

2 Introduction

2.1 Background

The ability to track changes made to text documents is commonly available in document editing systems. These are now moving to XML, e.g. OOXML and ODF. At the same time, structured document formats all use XML and currently do not have any real capability to track changes. The change-tracking capability of XML editors is fairly basic, for example many do not track attribute changes, and there is no common standard. The lack of a standard means that documents with changes tracked cannot be moved between XML editors.

This is a real opportunity to make XML much more powerful. A standard way of tracking changes in XML documents would provide these benefits:

- documents with tracked changes could be moved from one XML editor to another
- XML editors could track-changes in any XML document type
- every XML document type could include a change history and the ability to roll-back to previous versions
- software designed to handle change in XML could be applied to many different XML document types

This document outlines a proposed standard that takes into account the current approaches, building on their strengths and addressing their weaknesses.

The state-of-the-art at present is that every XML document type takes its own approach to change tracking. OOXML is built on the underlying binary model within Microsoft Word. ODF has only a very limited capability to track some changes. DITA uses *rev* and *status* attributes to indicate changes and DocBook similarly has a *revisionflag* attribute, but neither can track attribute changes or structural changes.

XML editors track changes either by additional markup or using Processing Instructions (PI). Additional markup has the advantage of structure but at the cost of modifying the underlying schema. PIs have the advantage of preserving the latest state of the document in valid XML markup but the PIs do not have structure and so are limited in the changes they can track.

2.2 Evolution towards a standard

It is evident that significant improvements can be made with some reasonably small changes to the current situation. This is covered in the basic change tracking capability described as Level 1. Moving to a more powerful solution is also possible and this will give a better user experience but at the cost of increased complexity. This is covered in Level 2.

Level 1 provides the ability to modify attributes, add and delete elements, and add and delete text. It also enables changes to be grouped into transactions where a single transaction moves the document from one valid state to another. Changes can be represented as markup or PIs in a way that allows loss-less transformation between them, thus gaining the advantages of both.

Level 2 adds to this the ability to add or delete element structure around existing content and to split and merge elements in more complex ways. In XML documents, the content (typically text) takes priority over the structure (typically paragraphs, tables, text decoration) in terms of changes. In other words, an editor does not want to see change to content when he/she has only changed the structure or styling. As an example of this, when an editor inserts a newline in the middle of a paragraph, i.e. splits it into two paragraphs, he/she does not expect to see change to those paragraphs but rather the insertion of a new line. This does not always fit well with the underlying XML structure, and could not be represented in Level 1. Level 2 addresses this issue, but it is a

complex issue and the solution is therefore more complicated.

2.3 Influencing factors

A number of other factors have been taken into account, as follows.

The proposed change tracking format avoids complex semantic rules relating to the correctness of changes. Rather it takes a generic approach so that almost any change can be represented, and then defines correctness in terms of the validity of the state before and after the change.

The format takes account of the task of a reader application that may not be able to understand changes at all or in certain areas. For example, if a reader is unable to represent any changes, it must be easy to read in the latest version of a document. This would also apply to individual fragments or subtrees within the document.

A single action by an editor may generate changes in a number of different places in the document. For example, a global change or 'replace all' will generate changes throughout a document, or deleting a column in a table will generate several disjoint changes in the underlying XML representation. Therefore there is clearly a need to represent a number of small atomic changes as a single action. Also the format provides some flexibility in the way these can be grouped, so that for example a global change can be accepted/rejected in one action or each change handled separately.

An atomic change should have a simple, intuitive, and localised representation within a document. For example, when an attribute is changed the format should not generate a large amount of structure to represent that change. On the other hand, the format should not require a lot of parsing of attribute values or other information in order to determine the nature of a change. These criteria may conflict, and in such cases a balance between these issues should be sought and explained.

The format provides alternative representations for deleted content and for attribute changes. A choice between these alternatives could be made or both could be allowed and a standard XSLT style sheet provided to convert between these – they have syntactic differences but the same semantic content. These are provided because one or other may be much more convenient for a particular type of XML document, or a particular type of processing.

2.4 Status of this proposal

This proposal was originally developed for ODF and code has been written to demonstrate that the concepts of both Level 1 and Level 2 work in practice in being capable of representing any change to an ODF document. The ODF community has not taken this up because they felt that the ability to represent any change was beyond what was needed and too complex for ODF editing applications to handle[1].

This proposal includes alternative representations for the changes, i.e. markup and PIs. It also includes alternative representations for deleted content and attribute changes. These need further discussion and development in liaison with XML editor vendors and XML users. It is becoming accepted practice to have alternative formats for the same semantic information, e.g. the full form and the compact form of RelaxNG. This can be very useful for users because some operations are much easier in one representation than another.

It is envisaged that the normative form would be markup with the PI representation as an alternative. PI examples are shown as an illustration for many of the Level 1 examples.

Discussions have also shown the potential need for an external representation of change tracking

1 They are therefore exploring another solution based on established methods for collaborative editing, but modified so that it is based on the editing operations that can be applied to a document, and using a new component-based addressing mechanism to map the external XML representation of changes to the document content. It is editor-application-centric, because it standardizes the edit operations, rather than document-centric and therefore is not as easily applied to other XML document types as the current proposal.

information, in other words an XML file that represents only the change tracking information and is separate from the document. Such an external representation could be combined with the document to generate a document with the change tracking embedded in it. If this conversion is bi-directional and loss-less then it would form a useful alternative that would make processing changes easier in some situations, for example where just changes need to be communicated. A note of caution here: the 'direction' of changes needs to be considered carefully. Tracked changes generally show the changes relative to the final version of a document, i.e. going 'back' in time. The most obvious use of an external representation would be to send changes to provide an update – but this would be a representation of changes moving 'forward' in time. Of course it should be possible to reverse the representation order automatically but it is not trivial.

3 Definitions and underlying rules

3.1 General concepts

1. An Atomic Change is a change such as the addition of an element or removal of an attribute, which represents a single syntactic change. The representation may involve more than one element or attribute.

   [Rationale: It is not appropriate to limit an Atomic Change to one that cannot be subdivided. For example, the deletion of an element and its contents (e.g. attributes and children) is considered to be atomic, whereas in principle this could be split into a collection of atomic changes that removes each leaf node in the XML structure. Further, even these leaf nodes could in principle have their textual content removed one character at a time. Forcing editors to record change at this level of detail is inappropriate.]

2. A Change Transaction (CT) consists of one or more Atomic Changes. If more than one Atomic Change is involved, there is no ordering of these, they are considered to happen as a single operation.

3. A CT is an indivisible change which is represented as a single transaction.

4. One CT depends on zero or many others. In other words, it may not be possible to apply a particular CT unless some other CT on which it depends is applied first.

   [Rationale: For example, if some text has been added, and then one of the words is deleted, it is not possible to accept the deletion if the addition has been rejected.]

5. Where a document has more than one CT, the order of the CTs must be defined.

   [Rationale: If we want to support an undo operation, then the ordering would be important. In general, changes made by an editor are done in a certain order, because a particular change may depend on a previous change. This ordering therefore represents the default dependency, i.e. each CT depends on all the previous CTs. A particular application may be able to provide more intelligent information on the dependencies.]

6. A change transaction (CT) is valid if the document before the CT is valid, and the document after the CT is applied is valid.

   [Rationale: This is a very simple definition of semantic correctness, and means therefore that we do not need a lot of complex rules about what combination of changes are correct.]

7. A document is valid if its final state is valid and all the CTs it contains are valid.

8. Every CT is uniquely identified by an ID.

9. Each atomic change is part of one and only one CT. This is enforced because each atomic change references the ID of the CT to which it belongs.

   [Rationale: This grouping is very important, because it means that we can form a change out of any number of atomic changes. This means that we only need a few atomic change operations, and these can be combined in complex ways to create CTs.]

10. CTs can be grouped either in a specific order (CT Stack) or as a set (CT Set). These groupings are for convenience, for example to allow a global edit (change all), or an editing session, to be undone in one operation. A CT group shall only reference previously-defined CTs or CT groups.

11. The final state of an element is the element and its attributes and the final state of all its content. When determining the final state, any deleted element is ignored, and the change history of any attributes is ignored. The format is designed so that the final state of a document can be determined by simply ignoring certain elements, elements with particular attributes and some attributes.

12. The final version of a document is the final state of its root element.
13. An element can only come into existence once (insert-with-content, insert-around-content or split) and go out of existence once (remove-with-content, remove-leaving-content or merge). Once an element has gone out of existence, or died, no further changes can be made to that element or its content.

    [Rationale: This is an important simplification because it means we do not need to cater for elements going out of existence and then coming back into existence again, which would make the format much more complex.]
14. Text and/or elements may be moved (deleted from one place and added elsewhere) to one or more other locations in a document. The change history of an element is not moved with the element. Content that has been moved from position A to position B can be moved again from B but it is deleted from A and so cannot be moved from A in a later operation.
15. There are two possible places for deleted content: in situ, i.e. where it was deleted, or in another place in the document.

    [Rationale: These alternatives need to be discussed and resolved.]
16. Similarly, there are two possible places for attribute changes: in situ, i.e. o their original element, or in the CT.

    [Rationale: These alternatives need to be discussed and resolved.]

3.2 Namespaces

See RelaxNG for formal definition of these.

The namespaces are defined as follows (the deltaxml.com namespace is only used an example):

xmlns:delta="http://www.deltaxml.com/ns/track-changes/delta-namespace"

xmlns:ac="http://www.deltaxml.com/ns/track-changes/attribute-change-namespace"

xmlns:split="http://www.deltaxml.com/ns/track-changes/split-namespace"

4 Change Transaction (CT) Structure

There must be a position in the document where the change transactions are defined, each being identified by an ID. Each will have some associated meta information such as the name of the author who made the change, and the date.

The ordering of the change transactions is important. If a user wishes to undo the changes one by one, then this can be achieved by undoing the change transaction at the end of the list and then moving up the list.

It is also possible to group CTs in a change transaction group (CT group). This will have similar meta information to a CT, and will reference CTs or other CT groups that it groups together, i.e. that are its members. Again, all the members must be previously-defined CT or CT groups. The effect of undoing a CT group will be to undo a number of CTs, which would then be removed from the list.

A software application that does not understand this grouping can ignore the groups, and the result will be some loss of structure but no effect on the underlying tracked changes. It is only a CT that has an effect on the document, the CT groups merely provide structure for user convenience.

A CT group may be ordered (CT stack, delta:change-transaction-stack) or unordered (CT set, delta:change-transaction-set). The members of a CT set can be accepted or rejected in any order. The members of a CT stack must be accepted or rejected in the defined order, i.e. undo last member first.

Example:

```
<delta:tracked-changes>
        <delta:change-transaction delta:change-id="ct1">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:00</dc:date>
            </delta:change-info>
        </delta:change-transaction>
        <delta:change-transaction delta:change-id="ct2" delta:edit-operation="make-bold">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
        </delta:change-transaction>
        <delta:change-transaction delta:change-id="ct3"
               delta:edit-operation="text-edit">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
        </delta:change-transaction>
        <delta:change-transaction-set delta:change-group-id="cs4">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
            <delta:change-log>Global edit</delta:change-log>
            <delta:change-references>
                <delta:change-ref delta:change-idref="ct2"/>
                <delta:change-ref delta:change-idref="ct3"/>
            </delta:change-references>
        </delta:change-transaction-set>
        ...
</delta:tracked-changes>
```

Example for PI:

```
<?delta-tracked-changes
        <delta:change-transaction delta:change-id="ct1">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:00</dc:date>
            </delta:change-info>
```

```
        </delta:change-transaction>
        <delta:change-transaction delta:change-id="ct2" delta:edit-operation="make-bold">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
        </delta:change-transaction>
        <delta:change-transaction delta:change-id="ct3"
               delta:edit-operation="text-edit">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
        </delta:change-transaction>
        <delta:change-transaction-set delta:change-group-id="cs4">
            <delta:change-info>
                <dc:creator>Robin</dc:creator>
                <dc:date>2010-06-02T15:48:01</dc:date>
            </delta:change-info>
            <delta:change-log>Global edit</delta:change-log>
            <delta:change-references>
                <delta:change-ref delta:change-idref="ct2"/>
                <delta:change-ref delta:change-idref="ct3"/>
            </delta:change-references>
        </delta:change-transaction-set>
        ...
?>
```

5 Tracking Changes: Level 1

This section details the atomic changes which are the lowest level changes that can be represented. All changes can be represented using these atomic changes. It is possible to move back from the final version of a document through successive changes to previous versions of a document. It may not be easy to extract an arbitrary version, but it is always possible to undo the last CT and thus work back through versions, i.e. the state between each edit action or CT.

5.1 Change Tracking attributes: Level 1

Attribute	Values	Description
delta:insertion-type	'insert-with-content'	Indicates how an element was created. Absence means the element existed in the oldest version of the document.
delta:insertion-change-idref	References a delta:change-id	References the CT that brought this element into existence. Present on all elements with an delta:insertion-type attribute.
delta:removal-change-idref	References a delta:change-id	References the CT that removed some content from the document. Can appear on a delta:removed-content element.
ac:XXX	Details of the attribute change, comma separated	ac: is a defined namespace, XXX is a generated attribute name, each new XXX represents a change to one attribute.
delta:move-id	Defines an ID for a move	Can appear on a delta:removed-content or delta:merge element.
delta:move-idref	References an ID for a move	Can appear on an element with delta:insertion-type='insert-with-content' to indicate the element and content was moved from elsewhere to this place. Can appear on delta:inserted-text-start to indicate the text was moved from elsewhere to this place.
delta:change-id	Defines an ID for a CT	Identifies a CT
delta:inserted-text-end-id	Defines an ID for a delta:inserted-text-end	Identifies the end element of a text insertion.
delta:inserted-text-end-idref	Reference to delta:inserted-text-end-id	Identifies the end element for some inserted text.
delta:edit-operation	Values defined in the standard or by a particular editing application	Optional on CT, CT set and CT stack to identify the type of edit-operation that this represents, e.g. text-to-table, global-replace, make-bold, libreOffice:macro23
delta:removed-content-cache-id	Defines an ID for a delta:removed-content-cache element	Identifies a delta:removed-content-cache element
delta:removed-content-cache-idref	References an ID for a delta:removed-content-	References a delta:removed-content-cache element

	cache element	

5.2 Change Tracking Elements: Level 1

Element	Description
delta:removed-content	Contains element, PCDATA or mixed content that has been removed, or contains a delta:removed-content-cache-idref attribute which references a delta:removed-content-cache where the removed content is held, elsewhere in the file.
delta:inserted-text-start	Identifies the start point of some inserted text.
delta:inserted-text-end	Identifies the end point of some inserted text.
delta:removed-content-cache	This is used when removed content is not allowed to be held in its original location. Contains removed content and has an attribute delta:removed-content-cache-id which is referenced from a delta:removed-content-cache-idref in a delta:removed-content element.
delta:attribute-cache	This is used when changed attributes are cached away from their original location. Contains changed attributes and has an attribute IDREF which references the ID of the element whose attributes are changed in a particular CT.

5.3 Add an element and its content (insert-with-content)

5.3.1 Description

The whole element is added with its content.

5.3.2 Example

Addition of a paragraph.

```
<text:p delta:insertion-type="insert-with-content"
       delta:insertion-change-idref='ct1234'>
This paragraph is inserted.</text:p>
```

Example PI:

```
<text:p >
<?delta-tracked-change-attributes delta:insertion-type="insert-with-content"
       delta:insertion-change-idref='ct1234'?>
This paragraph is inserted.</text:p>
```

5.3.3 Comments and Rationale

An added item may contain changes within it, but the changes must all be after it was added.

5.4 Delete an element and its content (remove-with-content)

5.4.1 Description

The whole element is deleted with its content.

5.4.2 Example

Deletion of a paragraph.

```
<delta:removed-content delta:removal-change-idref='ct456'>
  <text:p>
   This paragraph is deleted.
  </text:p>
</delta:removed-content>
```

Addition and deletion of a paragraph is shown like this:

```
<delta:removed-content delta:removal-change-idref='ct456'>
  <text:p delta:insertion-type="insert-with-content"
       delta:insertion-change-idref='ct1234'>
  This paragraph is added then later deleted.
  </text:p>
</delta:removed-content>
```

Deletion of a paragraph – PI example.

```
<?delta:removed-content delta:removal-change-idref='ct456'>
  <text:p>
   This paragraph is deleted.
  </text:p>
?>
```

Addition and deletion of a paragraph is shown like this – PI example:

```
<?delta:removed-content delta:removal-change-idref='ct456'>
  <text:p delta:insertion-type="insert-with-content"
       delta:insertion-change-idref='ct1234'>
  This paragraph is added then later deleted.
  </text:p>
?>
```

5.4.3 Comments and Rationale

A deleted item may contain changes within it, but the changes must all be before its deletion.

5.5 Add an attribute to an element

5.5.1 Description

This construct provides the ability to add a new attribute to an element.

5.5.2 Example

If a fragment starts as

```
<text:p text:style-name="Standard">
How an attribute is added
</text:p>
```

and goes to

```
<text:p text:style-name="Standard" text:outline-level="3">
How an attribute is added
</text:p>
```

then this is represented as

```
<text:p text:style-name="Standard" text:outline-level="3"
ac:change001="ct1,insert,text:outline-level">
```

```
How an attribute is added
</text:p>
```

where change001 is a generated attribute name and the name is not significant – it must be different for each attribute change recorded for this element. The content is a comma separated list of:

1. The change transaction (CT) ID. This is a reference to the ID.
2. The type of change: insert, remove, modify
3. The name of the attribute that is changed
4. The old value of the attribute – this is not needed for an added attribute because the value will either be in the element or, if the attribute is later deleted it will be recorded there.

PI example:

```
<text:p text:style-name="Standard" text:outline-level="3">
<?attribute-change "ct1,insert,text:outline-level" ?>
How an attribute is added
</text:p>
```

5.5.3 Comments and Rationale

All information on the change is local to the element changed. The attribute local name is generated because multiple changes are possible, and this avoids adding to a string (value) of some attribute and then parsing it. Minimal parsing of the ac:change001 attribute value is needed. The latest attributes are always listed in full, making extraction of the latest version simple.

5.6 Delete an attribute from an element

5.6.1 Description

This construct provides the ability to delete an attribute from an element.

5.6.2 Example

If a fragment starts as

```
<text:p text:style-name="Standard" text:outline-level="3">
How an attribute is deleted
</text:p>
```

and goes to

```
<text:p text:style-name="Standard" >
How an attribute is deleted
</text:p>
```

then this is represented as

```
<text:p text:style-name="Standard"
ac:change001="ct1,remove,text:outline-level,3">
How an attribute is deleted
</text:p>
```

PI example:

```
<text:p text:style-name="Standard" >
<?attribute-change "ct1,remove,text:outline-level,3" ?>
How an attribute is deleted
</text:p>
```

5.6.3 Comments and Rationale

This follows the same principles as an inserted attribute.

5.7 Change the value of an attribute

5.7.1 Description

This construct provides the ability to change the value of an attribute on an element.

5.7.2 Example

If a fragment starts as

```
<text:p text:style-name="Standard">
The style on the paragraph will be changed.
</text:p>
```

and goes to

```
<text:p text:style-name="Code">
The style on the paragraph will be changed.
</text:p>
```

then this is represented as

```
<text:p text:style-name="Code"
ac:change001="ct1,modify,text:style-name,Standard”>
The style on the paragraph will be changed.
</text:p>
```

PI example:

```
<text:p text:style-name="Code">
<?attribute-change "ct1,modify,text:style-name,Standard” ?>
The style on the paragraph will be changed.
</text:p>
```

5.7.3 Comments and Rationale

This follows the same principles as an added or deleted attribute.

5.8 Move an element (move)

5.8.1 Description

This construct describes the origin and the destination of content that is moved from one position in a document to another. Move provides a link between some removed content (move-from) and some inserted content (move-to), but this link simply provides additional information about the change transaction. If an application does not understand the concept of move, the move information can be ignored without compromising the content and structure of the document before the move or the content and structure of the document after the move.

The move representation allows content to be deleted and then inserted in one or more other positions in the document. A delta:move-id attribute must have one or more delta:move-idref references to it.

5.8.2 Example

If a fragment is moved from this position

```
<text:p>
This paragraph will be moved.
</text:p>
<text:h text:style-name="Heading_20_1" text:outline-level="1">
This is the heading for the paragraph
</text:h>
```

to this

```
<text:h text:style-name="Heading_20_1" text:outline-level="1">
This is the heading for the paragraph
```

```
</text:h>
<text:p>
This paragraph will be moved.
</text:p>
```

then this is represented as

```
<delta:removed-content delta:removal-change-idref=”ct123” delta:move-id=”mv33” >
       <text:p >
       This paragraph will be moved.
       </text:p>
</delta:removed-content>
<text:h text:style-name="Heading_20_1" text:outline-level="1">
This is the heading for the paragraph
</text:h>
<text:p delta:insertion-type=”insert-with-content” delta:move-idref=”mv33”
 delta:insertion-change-idref=”ct123”>
This paragraph will be moved.
</text:p>
```

PI example

```
<?delta:removed-content delta:removal-change-idref=”ct123” delta:move-id=”mv33” >
       <text:p >
       This paragraph will be moved.
       </text:p>
?>
<text:h text:style-name="Heading_20_1" text:outline-level="1">
This is the heading for the paragraph
</text:h>
<text:p >
<?delta-tracked-change-attributes delta:insertion-type=”insert-with-content”
delta:move-idref=”mv33” delta:insertion-change-idref=”ct123”?>
This paragraph will be moved.
</text:p>
```

5.8.3 Comments and Rationale

Move from and move to are linked by the delta:move-id attribute and delta:move-idref attributes. When content is moved, all its change history is reset, e.g. a move-from paragraph has the change history and the move-to has no history, it is as if it has been added new. This avoids duplicating history (causing ID duplicates etc) and the history is not lost because it is there in the original position.

The delta:move-id attribute appears on a delta:removed-content or delta:merge element and therefore there is no 1:1 relationship between a move-from element, whose parent has a delta:move-id attribute, and the move-to element. It would be possible to specify this relationship to a finer level of granularity by using multiple delta:removed-content elements rather than one.

5.9 Add text (PCDATA)

5.9.1 Description

This construct allows the insertion of text. It is similar to the existing mechanism. This construct shall only be used within an element that allows PCDATA content.

5.9.2 Example

If a fragment starts as

```
<text:p>
How text is added.
</text:p>
```

and goes to

```
<text:p>
How text is very easily added.
</text:p>
```

then this is represented as

```
<text:p>
How text is <delta:inserted-text-start delta:inserted-text-id="it632507360"
delta:insertion-change-idref=”ct1”/>very easily
<delta:inserted-text-end delta:inserted-text-idref="it632507360"/>added.
</text:p>
```

PI example:

```
<text:p>
How text is <?delta:inserted-text-start delta:inserted-text-id="it632507360"
delta:insertion-change-idref=”ct1” ?>very easily
<?delta:inserted-text-end delta:inserted-text-idref="it632507360" ?>added.
</text:p>
```

Second example: If a fragment starts as

```
<text:p>
How text is
</text:p>
```

and goes to

```
<text:p>
How text is very easily added.
</text:p>
<text:p>
And the addition is into a second paragraph.
</text:p>
```

then this is represented as

```
<text:p>
How text is <delta:inserted-text-start delta:inserted-text-id="it123" delta:insertion-
change-idref=”ct3”/>very easily added.<delta:inserted-text-end delta:inserted-text-
idref="it123"/>
</text:p>
<text:p delta:insertion-type="insert-with-content" delta:insertion-change-idref=”ct3”>
And the addition is into a second paragraph.
</text:p>
```

5.9.3 Comments and Rationale

Additions may not always be within a single element, but the delta:inserted-text-start and delta:inserted-text-end must both have the same parent element when they are created, and the content between them must be PCDATA only. Therefore when a second paragraph is added as per the second example, the first atomic change terminates and the paragraph is added in the normal way. The CT reference provides a link to indicate these occur at the same time as a single addition. This avoids having two ways to add an element and avoids the need to track across the element hierarchy to find the corresponding end of an addition.

Additions must therefore always be non-overlapping and the start and end of a change must be within a single element, when they are formed. Of course they may not be within a single element at some later stage due to other changes, but in this case it would not be possible to 'undo' it. This rule adds clarity at the slight cost to the writer application and the considerable gain for the reader. Since any number of atomic changes can be associated with a single CT, there is no loss of information.

5.10 Delete mixed or PCDATA content

5.10.1 Description

This construct allows the deletion of text. It is similar to the existing mechanism. This construct shall only be used within an element that allows PCDATA content.

5.10.2 Example

If a fragment starts as

```
<text:p>
How text is deleted or removed from a paragraph.
</text:p>
```

and goes to

```
<text:p>
How text is removed from a paragraph.
</text:p>
```

then this is represented as

```
<text:p>
How text is <delta:removed-content delta:removal-change-idref="ct2">deleted or
</delta:removed-content>removed from a paragraph.
</text:p>
```

Second example: If a fragment starts as

```
<text:p>
How text is deleted or <text:span text:style="bold">removed</text:span> like this from a
paragraph.
</text:p>
```

and goes to

```
<text:p>
How text is deleted from a paragraph.
</text:p>
```

then this is represented as

```
<text:p>
How text is deleted
<delta:removed-content delta:removal-change-idref="ct2">
  or
  <text:span text:style="bold"> removed</text:span>
  like this
</delta:removed-content>
from a paragraph.
</text:p>
```

5.10.3 Comments and Rationale

The deleted text is contained within a single element because it will never be subdivided or added to after its deletion. The deleted text element contains at least some deleted text, and may contain other elements.

A deleted item may contain changes within it, but the changes must all be before its deletion.

6 Tracking Changes: Level 2

6.1 Change Tracking attributes: Level 2

Attribute	Values	Description
delta:insertion-type	'insert-with-content' (also in Level 1) 'insert-around-content' 'split'	Indicates how an element was created. Absence means the element existed in the oldest version of the document.
delta:removal-change-idref	References a delta:change-id	References the CT that removed some content from the document. Can appear on a delta:removed-content (also in Level 1), delta:merge or delta:remove-leaving-content-start element.
delta:split-id	Defines an ID for an element formed in a split	Can appear on an element with delta:insertion-type='split'. Must be referenced by exactly one element with a split:XXX attribute.
split:XXX	References an ID for a split element	split: is a defined namespace, XXX is a generated attribute name, each new XXX represents a split and references the element which forms the second part of the split. An element may have more than one of these attributes.
delta:end-element-id	Defines an ID for an end element	Identifies a delta:remove-leaving-content-end element. Only one reference (delta:end-element-idref) allowed for each value of delta:end-element-id.
delta:end-element-idref	References an ID for an end element	Provides a reference to a delta:remove-leaving-content-end element. Occurs only on delta:remove-leaving-content-start.

6.2 Change Tracking Elements: Level 2

Element	Description
delta:merge	Merge of two elements, contains any deleted contents.
delta:leading-partial-content	Contains the content that was in the leading merged element.
delta:intermediate-content	Contains any elements that were between the leading and trailing merged elements
delta:trailing-partial-content	Contains the content that was in the trailing merged element. Always contains one element.
delta:remove-leaving-content-start	Contains the start tag of an element that has been removed leaving its content.

delta:remove-leaving-content-end	Represents the position of the end tag of an element that has been removed leaving its content.

6.3 Add an element around some existing content (insert-around-content)

6.3.1 Description

In document editing, it is common to add text decoration or structural information to existing content. As the content itself is not changed, we wish to reflect just the addition or change of the structure.

6.3.2 Example

In this example, a span is added into some text to make a word bold.

If a fragment start as

```
<text:p>
This text will be made bold.
</text:p>
```

and becomes

```
<text:p>
This text will be made <text:span text:style-name="bold-style">bold</text:span>.
</text:p>
```

then this is represented as

```
<text:p>
This text will be made <text:span delta:insertion-type='insert-around-content'
delta:insertion-change-idref='ct1234' text:style-name="bold-style">bold</text:span>.
</text:p>
```

6.3.3 Comments and Rationale

This facility allows wrapper elements to be added, where their content existed independently before the element was added.

6.4 Delete an element but not its content (remove-leaving-content)

6.4.1 Description

A good example of this situation is when a span element, which is there to provide some text decoration, is deleted. Clearly the text that was decorated remains there, it is just the span that has disappeared.

6.4.2 Example

If a fragment starts as

```
<text:p>
This text will be made <text:span text:style-name="bold-style">unbold</text:span>.
</text:p>
```

and becomes

```
<text:p>
This text will be made unbold.
</text:p>
```

then this is represented as

```
<text:p>
This text will be made
        <delta:remove-leaving-content-start
        delta:removal-change-idref='ct345' delta:end-element-idref='ee888'>
                <text:span text:style-name="bold-style" />
        </delta:remove-leaving-content-start>
unbold<delta:remove-leaving-content-end delta:end-element-id='ee888'/>.
</text:p>
```

Let's look at a more complex example.

First version: Here is some text **where the decoration is changed** several times.

```
<text:p>
Here is some text
<text:span text:style-name="bold-style">
        where the decoration is changed
</text:span>
several times.
</text:p>
```

Second version: Here is some text **where the *decoration* is changed** several times.

```
<text:p>
Here is some text
<text:span text:style-name="bold-style">
        where the
        <text:span delta:insertion-type='insert-around-content'
        delta:insertion-change-idref='ct1' text:style-name="italic-style">
                decoration
        </text:span>
        is changed
</text:span>
several times.
</text:p>
```

Third version: Here is some **text where the *decoration* is** changed several times.

```
<text:p>
Here is some
<text:span delta:insertion-type='insert-around-content'
delta:insertion-change-idref='ct2' text:style-name="bold-style">
        text
        <delta:remove-leaving-content-start delta:removal-change-idref='ct2'
                delta:end-element-idref='ee888'>
                <text:span text:style-name="bold-style"/>
        </delta:remove-leaving-content-start>
        where the
        <text:span delta:insertion-type='insert-around-content'
        delta:insertion-change-idref='ct1'
         text:style-name="italic-style">
                decoration
        </text:span>
        is
</text:span>
changed
<delta:remove-leaving-content-end delta:end-element-id='ee888'/>
several times.
</text:p>
```

6.4.3 Comments and Rationale

Since the element has been deleted, but the content remains, it is split it into a start and end element so that the content remains in position at the correct level. The split element is linked by an ID so that it can be reconstructed. The splitting of a wrapper element into its start element and end element means that deleted wrapper elements do not contribute to the hierarchical structure of the document. This is important because over time they may be split across element boundaries.

When created, the start and end elements must have the same parent element.

6.5 Split an element into two elements (split)

6.5.1 Description

The classic example of this is when a paragraph is split into two by the insertion of a new line. Similarly, a list item might be split into two list items. The element that is split is known as the parent of the split and the element that is created is known as the child of the split.

6.5.2 Example

If a fragment starts as

```
<text:p>
This paragraph will be split into two. This will be in the second paragraph.
</text:p>
```

and becomes

```
<text:p>
This paragraph will be split into two.
</text:p>
<text:p>
This will be in the second paragraph.
</text:p>
```

then this is represented as

```
<text:p split:split01='sp1'>
This paragraph will be split into two.
</text:p>
<text:p delta:insertion-type='split' delta:insertion-change-idref='ct1'
delta:split-id='sp1'>
This will be in the second paragraph.
</text:p>
```

Second example: if a fragment starts as

```
<text:p text:style-name="Standard">Here is a paragraph which is split in two with
                a new table. It was added before this.</text:p>
```

and goes to (represents pasting a table when the cursor is in the middle of a paragraph)

```
<text:p text:style-name="Standard">Here is a paragraph which is split in two with
                a new table. </text:p>
<table:table table:name="Table1" table:style-name="Table1">
   <table:table-column table:style-name="Table1.A" table:number-columns-repeated="2"/>
     <table:table-row>
       <table:table-cell table:style-name="Table1.A1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T1</text:p>
       </table:table-cell>
       <table:table-cell table:style-name="Table1.B1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T2</text:p>
       </table:table-cell>
     </table:table-row>
 </table:table>
 <text:p text:style-name="Standard">It was added before this.</text:p>
```

then this is represented as

```
<text:p text:style-name="Standard" split:split01='sp1'>
Here is a paragraph which is split in two with a new table. </text:p>
<table:table table:name="Table1" table:style-name="Table1"
       delta:insertion-type='insert-with-content' delta:insertion-change-idref='ct1'>
   <table:table-column table:style-name="Table1.A" table:number-columns-repeated="2"/>
     <table:table-row>
       <table:table-cell table:style-name="Table1.A1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T1</text:p>
       </table:table-cell>
       <table:table-cell table:style-name="Table1.B1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T2</text:p>
       </table:table-cell>
     </table:table-row>
</table:table>
<text:p text:style-name="Standard" delta:insertion-type='split'
 delta:insertion-change-idref='ct1' delta:split-id='sp1'>It was added before
this.</text:p>
```

6.5.3 Comments and Rationale

This facility allows the representation of quite a common editorial action. Note there may be elements between the split paragraph elements but these would all have been added in the same or a later CT. Therefore there is an attribute value pair to link the start and end of a split.

6.6 Merge two sibling elements into one (merge)

6.6.1 Description

This merge change is the opposite to a split, and would occur when, for example, an author deletes a newline, or deletes words at the end of one paragraph and some words from the following paragraph.

6.6.2 Example

If a fragment starts as

```
<text:p text:style-name="Standard">
These paragraphs will be merged into one.
</text:p>
<text:p text:style-name="Code">
This was in the second paragraph.
</text:p>
```

and goes to

```
<text:p text:style-name="Standard">
These paragraphs will be merged into one. This was in the second paragraph.
</text:p>
```

then this is represented as

```
<text:p text:style-name="Standard">
These paragraphs will be merged into one.
<delta:merge delta:removal-change-idref='ct2'>
  <delta:leading-partial-content/>
  <delta:intermediate-content/>
  <delta:trailing-partial-content>
    <text:p text:style-name="Code"/>
  </delta:trailing-partial-content>
</delta:merge>
This was in the second paragraph.
</text:p>
```

Second example: if a fragment starts as

```
<text:p text:style-name="Standard">Here is a paragraph which was split in two by
                a table. [this bit of text is deleted]</text:p>
 <table:table table:name="Table1" table:style-name="Table1">
   <table:table-column table:style-name="Table1.A" table:number-columns-repeated="2"/>
   <table:table-row>
       <table:table-cell table:style-name="Table1.A1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T1</text:p>
       </table:table-cell>
       <table:table-cell table:style-name="Table1.B1" office:value-type="string">
          <text:p text:style-name="Table_20_Contents">T2</text:p>
       </table:table-cell>
    </table:table-row>
   </table:table>
 <text:p text:style-name="Standard">[this is also deleted] It was split before
this.</text:p>
```

and goes to

```
<text:p text:style-name="Standard">Here is a paragraph which was split in two by
                a table. It was split before this.</text:p>
```

then this is represented as

```
<text:p text:style-name="Standard">Here is a paragraph which was split in two by
```

```
       a table.
 <delta:merge delta:removal-change-idref='ct2'>
   <delta:leading-partial-content>
      [this bit of text is deleted]
   </delta:leading-partial-content>
   <delta:intermediate-content>
     <table:table table:name="Table1" table:style-name="Table1">
        <table:table-column table:style-name="Table1.A"
           table:number-columns-repeated="2"/>
        <table:table-row>
            <table:table-cell table:style-name="Table1.A1" office:value-type="string">
               <text:p text:style-name="Table_20_Contents">T1</text:p>
            </table:table-cell>
            <table:table-cell table:style-name="Table1.B1" office:value-type="string">
               <text:p text:style-name="Table_20_Contents">T2</text:p>
            </table:table-cell>
        </table:table-row>
     </table:table>
   </delta:intermediate-content>
   <delta:trailing-partial-content>
     <text:p text:style-name="Standard">[this is also deleted]</text:p>
   </delta:trailing-partial-content>
 </delta:merge>
It was split before this.</text:p>
```

6.6.3 Comments and Rationale

In the first example, there is no structure or content to the deleted item, because no content has been deleted. In the second, the deleted table is included as content in the merged element delta:merge which records the content deleted from the leading and trailing merged paragraph and elements deleted between these.

Note that delta:trailing-partial-content always contains one element.

Note that the element in <delta:trailing-partial-content> could be of a different type to the one that encloses the delta:merge. This allows elements of different types to be merged.

The two merged elements must be siblings.

Note that the merge operation could be represented using remove-leaving-content and insert-around-content but this leads to a more complex structure. Therefore delta:merge provides a special representation for this common editing action.

6.7 Change the type of an element

6.7.1 Description

There is no special form for this: use 'remove-leaving-content' and 'insert-around-content'. It may also sometimes be appropriate to use 'remove-with-content' and 'insert-with-content', though the meaning of this is different.

6.7.2 Example

If the fragment

```
<text:p text:style-name="Text_20_body">
What are the ground rules?
</text:p>
```

is changed to

```
<text:h text:style-name="Heading_20_1" text:outline-level="1">
What are the ground rules?
</text:h>
```

then this is represented as:

```
<delta:remove-leaving-content-start delta:removal-change-idref='ct1234'
       delta:end-element-idref='ee888'>
       <text:p text:style-name="Text_20_body"
</delta:remove-leaving-content-start>
<text:h text:style-name="Heading_20_1" text:outline-level="1"
delta:insertion-type='insert-around-content' delta:insertion-change-idref='ct1234'>
What are the ground rules?
</text:h>
<delta:remove-leaving-content-end delta:end-element-id='ee888'/>
```

6.7.3 Comments and Rationale

Since both atomic changes are associated with the same CT, they are linked as one and occur together. There is no need therefore for a new construct for this case. Note that the element start and end tags are outside the new element in this example but there is no rule to enforce this.

7 Miscellaneous

7.1 Alternative Representation for Deleted Content

Deleted content may be cached in situ or elsewhere in the document, for example within the CT.

The following example shows the deleted content cached in situ:

```
<delta:removed-content delta:removal-change-idref='ct456'>
  <text:p>
   This paragraph is deleted.
  </text:p>
</delta:removed-content>
```

The following example shows the deleted content cached in the CT:

```
<delta:change-transaction delta:change-id="ct456">
   <delta:change-info>
      <dc:creator>Robin</dc:creator>
      <dc:date>2010-06-02T15:48:00</dc:date>
   </delta:change-info>
   <delta:change-cache>
     <delta:removed-content-cache delta:removed-content-cache-id='cache456'>
     <text:p >
       This paragraph is deleted.
     </text:p>
     </delta:removed-content-cache>
   </delta:change-cache>
</delta:change-transaction>
...
<delta:removed-content delta:removal-change-idref='ct456'
  delta:removed-content-cache-idref='cache456'/>
```

7.2 Alternative Representation for Changed Attributes

Attribute changes may be stored in situ, i.e. on their original element, or in the CT. The element whose attributes are changed needs to be referenced in some way and an ID attribute is used here, but a specific attribute could be added if preferred (TBD). More than one CT might refer to an element to change its attributes, but the element needs only one ID attribute. The element delta-attribute-modification could be replaced by the requirement to add and delete an attribute to indicate a modification (in this case the old and new values could be kept).

An attribute can appear on at most once in a delta:attribute-cache.

The following example shows the attribute change cached in situ:

```
<text:p text:style-name="Standard" text:outline-level="3"
ac:change001="ct1,insert,text:outline-level"
```

```
ac:change002="ct1,modify,text:style-name,Code”
ac:change003="ct1,delete,text:class-names,X”>
How attributes are changed
</text:p>
```

The following example shows the attribute change cached in the CT:

```
<delta:change-transaction delta:change-id="ct1">
   <delta:change-info>
      <dc:creator>Robin</dc:creator>
      <dc:date>2010-06-02T15:48:00</dc:date>
   </delta:change-info>
   <delta:change-cache>
     <delta:attribute-cache IDREF='abcd'>
       <delta:attribute-addition text:outline-level="3"/>
       <delta:attribute-modification text:style-name="Code"/>
       <delta:attribute-deletion text:class-names="X"/>
     </delta:attribute-cache>
   </delta:change-cache>
</delta:change-transaction>
...
<text:p text:style-name="Standard" text:outline-level="3"
 ID='abcd'>
How attributes are changed
</text:p>
```

Note that the value of an attribute in attribute-addition is not needed, and as it stands differs from the cached-in-situ representation (this can be discussed).

7.3 'Suggested' Changes

In some situations an editor working on a document produced by another person may wish to flag a change as being a suggestion for a change rather than an actual change. A change transaction may be flagged in this way and a reader application may wish to treat this differently, for example by highlighting the change in a different way or having a default action to be to reject the change rather than accept it.

TBA: add delta:suggested-change as an attribute allowing value “true” or “false” where the default is false.

7.4 Edit-operation

It may be difficult for an editing application to determine the editing operation that was applied in order to achieve a particular set of changes represented by a CT, a CT group or a CT stack. Therefore editing operations can be defined in the standard and identified using the delta:edit-operation attribute.

For each definition of an edit operation, there will be rules to say what type of change is allowed. For example, an edit operation defined as a 'text-replacement' would not be allowed to include the deletion of a paragraph, or the change of a format attribute.

This mechanism could easily be extended so that a given editing application which has an operation that is not defined in the standard, would be able to create a new definition perhaps using its own namespace as a prefix.

7.5 Changes to External Objects

If a software application is able to track changes to external objects, then when an external object is changed the old version must be kept, and a new version with different identity produced. Then where that object is referenced there will be a change to the reference to indicate that it used to point to the old version of the object and it now points to the new version.

7.6 Handling ID attributes

ID attributes, such as xml:id, should not be re-used but a new value should be generated for any new element created with an ID. However, there are situations when the same ID may need to appear more than once in a tracked-change document. An example of this is when a list item is structurally changed by a remove-leaving-content and then insert-around-content. If the ID is left as it is, then there would be two IDs with the same value.

This can be solved quite simply by deleting the ID attribute in the removed element. The ID will appear again if the transaction is reversed.

If a fragment starts as

```
<list xml:id=”id1”>
  <list-item><p>Some [list item</p></list-item>
</list>
<list xml:id=”id2”>
  <list-item><p>Another ]later paragraph in a list</p></list-item>
</list>
```

and goes to

```
<list xml:id=”id1”>
  <list-item><p>Some later paragraph in a list</p></list-item>
</list>
```

then this is represented as

```
<list xml:id="id1" delta:insertion-change-idref="ct1" delta:insertion-type="insert-
around-content">
    <delta:remove-leaving-content-start delta:removal-change-idref="ct1"
        delta:end-element-idref="end1">
        <list ac:change001="ct1,remove,xml:id,id1"/>
    </delta:remove-leaving-content-start>
    <list-item delta:insertion-change-idref="ct1"
              delta:insertion-type="insert-around-content">
        <delta:remove-leaving-content-start delta:end-element-idref="end3">
            delta:removal-change-idref="ct1" <list-item/>
        </delta:remove-leaving-content-start>
        <p delta:insertion-change-idref="ct1"
              delta:insertion-type="insert-around-content">
            <delta:remove-leaving-content-start delta:end-element-idref="end4"
                delta:removal-change-idref="ct1">
                <p/></delta:remove-leaving-content-start>
                Some <delta:removed-content delta:removal-change-idref="ct1">
                [list item</delta:removed-content>
            <delta:remove-leaving-content-end delta:end-element-id="end4"/>
            <delta:remove-leaving-content-end delta:end-element-id="end3"/>
            <delta:remove-leaving-content-end delta:end-element-id="end1"/>
            <delta:remove-leaving-content-start delta:end-element-idref="end2"
                delta:removal-change-idref="ct1">
                <list ac:change002="ct1,remove,xml:id,id2"/>
            </delta:remove-leaving-content-start>
            <delta:remove-leaving-content-start delta:end-element-idref="end5"
                delta:removal-change-idref="ct1">
                <list-item/></delta:remove-leaving-content-start>
            <delta:remove-leaving-content-start delta:end-element-idref="end6"
                delta:removal-change-idref="ct1">
                <p/></delta:remove-leaving-content-start>
            <delta:removed-content delta:removal-change-idref="ct1"> Another
                ]</delta:removed-content> paragraph in a list
                <delta:remove-leaving-content-end delta:end-element-id="end6"/>
        </p>
        <delta:remove-leaving-content-end delta:end-element-id="end5"/>
    </list-item>
    <delta:remove-leaving-content-end delta:end-element-id="end2"/>
</list>
```

Note that it would also be possible to have a simpler Level 1 representation of this, although this would involve the deletion and addition of text which, arguably, has not been changed. The representation is as follows:

```
<list xml:id="id1">
  <list-item><p>Some <delta:removed-content delta:removal-change-idref="ct1">
              [list item</delta:removed-content>
       <delta:inserted-text-start delta:inserted-text-id="it123"
              delta:insertion-change-idref="ct1"/>
       later paragraph in a list
       <delta:inserted-text-end delta:inserted-text-idref="it123"/>
              </p>
       </list-item>
</list>
<delta:removed-content delta:removal-change-idref="ct1">
       <list xml:id="id2"  >
              <list-item><p>Another ]later paragraph in a list</p></list-item>
       </list>
</delta:removed-content>
```

8 Integration with a host format

We have identified two different ways of integrating this track change format, discussed below.

8.1 Stand-alone use of 'XML Track Changes'

The format can be used as an independent addition to an existing XML host format. In this scenario no changes are made to the schema of the host format, but the track change elements and attributes are used to represent changes and edits to a document. The following tools are provided to extract different versions of the document, and to validate a version of the document that has tracked changes represented.

1.Schematron checker to check a change-tracked document (Schematron Checker)

2.XSLT stylesheet to extract the final document from a change-tracked document (XSLT Extractor)

3.XSLT stylesheet to roll back the last change transaction from a change-tracked document (XSLT Roll-back)

These tools allow a complete integrity check as follows:

1.Execute the Schematron Checker to check the document.

2.Use XSLT Extractor to extract the last version of the document

3.Check the last version of the document against the normal document schema and/or other integrity checks.

4.If there are no Change Transactions in the document, the checking is finished.

5.Use XSLT Roll-back to roll back the tracked change

6.Return to 1 to continue checking.

An application reading the change-tracked document would need to recognise the change tracking elements and treat these in a special way so that the final version of the document ends up in memory with some ancillary in-memory data structure to denote the changes.

8.2 Host-integrated 'XML Track Changes'

In this scenario there will be a RelaxNG schema which specifies the host format with change tracking schema integrated with it. The stand-alone testing mentioned above would still be valid and work, but as well as that the change-tracked document could be checked against a schema.

Note: More work is needed to develop this integration for a particular host format, and it has a significant impact on the RelaxNG schema.

8.2.1 Schema Integration

Integration of Level 1 is simpler than integration of Level 2.

It is possible to represent any changes to a document in each level, but Level 2 provide a more natural representation of typical document editing actions. Level 2 seek to make minimal changes to the document content or text while allowing complex changes to the structure surrounding that textual content.

8.2.2 Schema Integration Level 1

This level is provided as a guide for other use cases of this tracked-change representation. Level 1 has not been fully tested independently of Level 2.

Rule 1: The element delta:tracked-changes must be allowed at one point in the document.

Rule 2: Any element in the host format that has one or more attributes which can be added, deleted or values changed, need to allow attributes in the ac: namespace.

Rule 3: All elements that can be added or deleted with their content (including any element that allows no content, i.e. is always empty) need to allow the attribute delta:insertion-type with value 'insert-with-content' and be permitted as a child of delta:removed-content (unless this allows any elment, see RelaxNG for details). (Note that this is not necessarily all elements, for example an element that is only used as a required item and never in a choice would not be in this category.)

Rule 4: All elements that allow element content must have their content model modified so that they allow delta:removed-content to appear anywhere as a child element.

Rule 5: All elements that allow PCDATA content, including elements that allow mixed content, need to allow for text content to be added (Rule 4 allows text to be deleted).

8.2.3 Schema Integration Level 2

Rules 1, 2, 3 and 5 are as Level 1. Rule 4 is replaced with Rule 8.

Rule 6: Any element that can be added as a wrapper around existing content, or removed as a wrapper (in this situation it is often true that the content model of the element is a subset of the content model of its parent): These elements need to allow delta:insertion-type='insert-around-content' and supporting attributes (see RelaxNG for details).

Rule 7: For any Rule 6 element, if any content is required then the content model must be changed to make an empty element (no content) allowed when its parent is delta:remove-leaving-content-start.

Rule 8: This is an extension to Rule 4: Any element that allows content must allow as child elements delta:remove-leaving-content-start, delta:remove-leaving-content-end, delta:removed-content and delta:merge to appear zero or more times anywhere.

Rule 9: Elements where it is useful to represent a split or merge. Typically these will have mixed content, though this is not a condition. These elements need to allow delta:insertion-type='split' (and supporting attributes).

8.2.4 Schema Integration: RelaxNG

The above rules have been coded in RelaxNG in a way that enables integration with a host format (see delta-format.rng for details).