White Paper: Release Management Domain

Contents

The problem

In general

Documenting the workings of complex machines - a jet fighter or a CAT scanner - requires the marshalling of thousands of facts. Even with a very low error rate, corrections will be required; in a large document, a revision could mean hundreds of changes. Most readers will complain if given an update without some list of significant changes.

The key word here is significant. Machines are notoriously bad at recognizing the significance of human language, so it is nearly impossible for a computer to distinguish between trivial changes and significant ones. For large books, automatic lists or difference documents risk obscuring the significant changes in a snowstorm of trivial ones.

Recording these changes then becomes the responsibility of the content worker.

In DITA

As the use of DITA spreads into such realms, a bit of help for its practicioners is welcome. Up to now, those of us who use DITA for such documents have had to resort to workarounds: spreadsheets, text files, or maybe a special-purpose topic. Some lucky few might have a content management system with good metadata facilities. But it's always external to the content.

But why not record the changes in the topic itself? This would offer numerous advantages:

- For authors, it eliminates the time-consuming and error-prone step of getting the separate topic and recording the change there. If a cross reference is desired, this would no longer be a separate process.
- It eliminates reliance on the CMS metadata, if such is used.
- For readers, it means in all likelihood more accurate documents, since compliance will improve.
- In a topic-based display, such as infocenter or wiki, it would facilitate a tabbed display: one tab for content, one for history. (Wikipedia has such a display.)

Thus was born the new DITA release management domain. This document will introduce the domain, describe its elements, and give examples of its use.

Available with this document is an XQuery and some sample files. The XQuery follows a DITA map and extracts release notes from the dita topics in the map; these release notes are put into a table in a newly generated topic that can be included in a publication or used alone.

The solution

Introducing the release management domain

Although I referred earlier to the release management domain as new, it's really based on prior DITA work (on bookmap's bookchangehistory element, but release management works in both maps and topics). It can be used in the prolog of topic types and in the metadata element of maps.

Release management is an element-only domain; current processing simply ignores the new elements. Those who want to use it will need to supply their own processing, such as that of the sample XQuery. In addition, all the elements are optional. They all support select attributes as well, since the domain was intended for use in shared document environments.

For some organizations, the use of select attributes is insufficient by itself. Release notes may belong in one and only one document. In other words, once the release note has appeared in print, it should never appear again. Note, however, that this model is not imposed by the release management domain, which can also easily support cumulative documents.

Release management domain elements

Here are the elements of the release management domain. <change-historylist> is a child of prolog. It can be included in maps as a child of <metadata>.

[how to present?]

```
change-historylist?
  change-item*
    ( change-person | change-organization )*
    change-revisionid?
    change-request-reference?
      change-request-system?
      change-request-id?
    change-started?
    change-completed?
    change-summary?
    data*
```

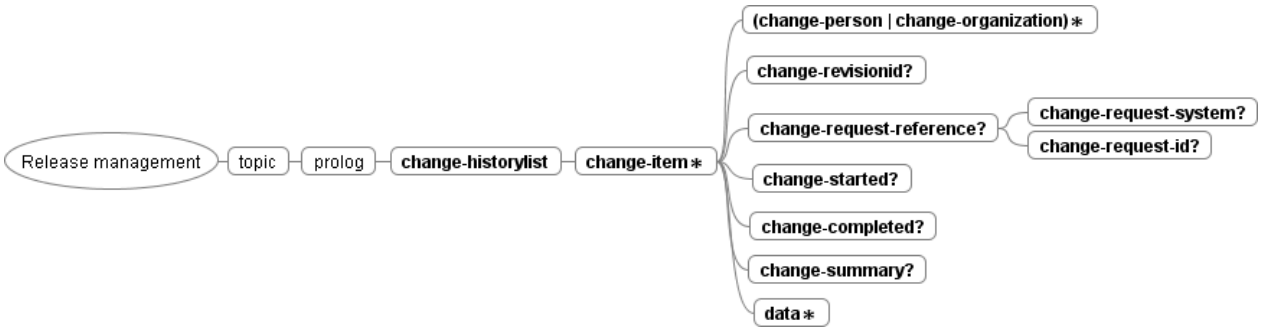

Figure 1: Release Management Elements

[[show both?]]

All these elements are derived from <data>; thus, except for the containers change-historylist, change-item, and change-request-reference, they have CDATA content models. All the elements are optional; the user is free to use as little or as much of the domain as is needed. Additional data elements of any number may be used as is or specialized to meet requirements not foreseen. All release management elements support select attribution.

Elements in detail

This section describes the elements in greater detail.

change-item	Contains a single release note. It holds information about when and by whom the topic was edited during its history.
change-person	The person making the change to the document
change-organization	An organization that requires or instigates a change. Examples include government agencies or standards bodies.
change-revisionid?	Contains an identifier associated with the change described by the release note
change-request-reference?	Changes may result from tickets filed in defect tracking systems or other databases. This element is a container for the next two elements
change-request-system?	The tracking system or database from which the change originated (see change-request-reference)
change-request-id?	The id or other key number linking the change back to the tracking system or database (see change-request-reference)
change-started?	The date work on the change began. Recommended date format is ISO-8601, with or without time information, (for example 2014-06-17) unless a machine timestamp is used.
change-completed?	The date work on the change was completed. Recommended date format is ISO-8601, with or without time information, (for example 2014-06-17) unless a machine timestamp is used.
change-summary?	A text description of the change. This should represent the actual text describing the change as presented to the reader.

Example release notes

This figure shows three simple release notes added to a single topic. This topic is used in documentation for two products, A and B.

```
<prolog>
...
  <changehistory-list>
    <change-item product="productA productB">
      <change-person>Bill Carter</change-person>
      <change-completed>2013-03-23</change-completed>
      <change-summary>Made change 1 to both products</change-summary>
```

```
      <data>Details of change 1</data>
    </change-item>
    <change-item product="productA">
      <change-person>Phil Carter</change-person>
      <change-completed>2013-06-07</change-completed>
      <change-summary>Made change 2 to product A</change-summary>
      <data>Details of change 2</data>
    </change-item>
    <change-item product="productA productB">
      <change-person>Bill Carter</change-person>
      <change-completed>2013-07-20</change-completed>
      <change-summary>Made change 3 to both products</change-summary>
      <data>Details of change 3</data>
    </change-item>
  </changehistory-list>
...
</prolog>
```

Figure 2: Excerpt from prolog of topic myTopic

Sample output showing date filtering

One presentation of the data from the above release notes might be as a table. The example XQuery outputs a topic containing such a table.

To illustrate the use of date filtering, in this scenario revision 5 of product A's manual was published on June 1, while product B's manual hasn't been published since February 10 (revision 2). Then, on September 3, both manuals are being published. Here is a timeline of events:

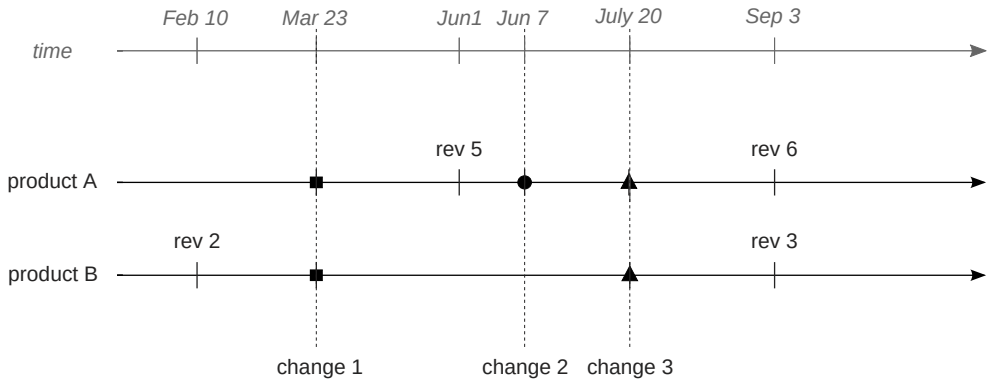

Figure 3: Example timeline

Thus, product A's release notes for revision 6 should include only those changes since June 2, while those for revision 2 of product B should start with changes made on February 11. Here is what these documents' release notes should contain for this topic:

Table 1: Excerpt from product A's revision 6 release notes, September 3 (last published June 1)

Change Site	details
Topic X	Made change 2 to product A
Topic X	Made change 3 to both products

Table 2: Excerpt from product B's revision 3 release notes, September 3 (last published February 10)

Change Site	details
Topic X	Made change 1 to both products
Topic X	Made change 3 to both products

Note that change 1 already appeared in the revision 5 release notes of product A on June 1. Therefore, it should *not* appear in the revision 6 release notes, or it will confuse and annoy customers by alerting them to something that hasn't actually changed since the previous revision.