

A DEVELOPER'S GUIDE TO ARBORTEXT EDITOR'S CHANGE-TRACKING MARKUP

PTC/USER WORLD EVENT 2009

James Sulak
Developer, Jones McClure Publishing
jsulak@jonesmcclure.com
Blog: www.wordsinboxes.com
Twitter: jsulak

INTRODUCTION

The goal of this presentation is to show how Arbortext change tracking works and demonstrate the range of what can be accomplished with it. It will consist of three parts: (1) an exploration of the structure of change-tracking markup, (2) examples of how to enhance both its appearance and its utility within Arbortext Editor, and (3) demonstrations of how to programmatically manipulate it outside of Editor. At the end of this session, you will:

- Understand the different change-tracking elements and how they are used.
- Be able to modify the appearance of tracked changes using Arbortext Styler.
- Be able to use the `changetrackingafterhook` to add additional information to change-tracking markup.
- Be able to develop scripts to leverage change-tracking markup both within Editor and out.

This talk focuses exclusively on change-tracking in XML documents, not in SGML documents, which use processing instructions instead of elements.

ABOUT THE SPEAKER

James Sulak is a developer at Jones McClure Publishing. He converts legacy content to XML; writes and maintains Arbortext customizations; creates new, XML-based editorial processes; and develops stylesheets for multiple publishing platforms.

EXAMPLES

All examples, including a complete custom folder, are available from the PTC/User 2009 online portal. By modifying `editor-presentation.bat` to point to your `epic.exe` and the included example custom folder, you can launch Arbortext with these customizations and try out the examples yourself. This document and the slides are also available for download.

CHANGE-TRACKING XML

Arbortext Editor's change-tracking is stored as XML elements in the “<http://www.arbortext.com/namespace/atict>” namespace with the prefix “atict.” Each element records a specific type of change – for example, inserted text or deleted text. All of the elements are defined in the specification, which is available online at <http://www.arbortext.com/namespace/atict/change-tracking-markup-spec.html>. This specification is your most important resource, so bookmark it in your browser.

Let's go through the different types of markup, one by one. We'll start with the following paragraph, and then modify it, showing how the different change-tracking elements store revision information. As you read, think about (1) how this markup could be leveraged by custom applications and (2) how this markup would be interpreted by another (non-Arbortext) XML parser. Here's the paragraph:

```
<para>Wow! That presentation was the <i>very best</i> I saw -
almost as good as Paul's.</para>
```

Inserted Text: `<atict:add/>`

When a user inserts new text, it is surrounded by an **atict:add** element. The added content must be a valid XML fragment.

```
<para>Wow! That presentation was the <i>very best</i> I saw -
almost as good as Paul's.<atict:add> I filled out a
<u>glowing</u> presenter's evaluation. I hope you did
too.</atict:add></para>
```

Deleted Text: `<atict:del/>`

When a user deletes content from a document, that content is surrounded by an **atict:del** element. Depending on the view in Styler, that text is not shown, or shown as struck-out text. Note the similarity to **atict:add**.

```
<para>Wow! That presentation was the <i>very best</i> I
saw<atict:del> - almost as good as Paul's</atict:del>. I filled
out a <u>glowing</u> presenter's evaluation. I hope you did
too.</para>
```

Inserted Markup: `<atict:addm />`

When a user inserts a new tag into a document, a singleton **atict:addm** element is placed as the first child of the new tag. If the new tag is empty, then it is sometimes surrounded by an **atict:add** element instead.

```
<quote><atict:addm /><para>Wow! That presentation was the <i>very
best</i> I saw. I filled out a <u>glowing</u> presenter's
evaluation. I hope you did too.</para></quote>
```

Modified Markup: `<atict:chgm/>`

When a user modifies an existing tag in a document (changing either its name or its attributes), the original tag is surrounded by an **atict:chgm** element.

```
<quote><para>Wow! That presentation was the <i>very best</i> I
saw. I filled out a <i><atict:chgm><u/></atict:chgm> glowing</i>
presenter's evaluation. I hope you did too.</para></quote>
```

Deleted Markup: `<atict:delm/>`

When a user deletes an existing tag from the document, an **atict:delm** singleton element is inserted as the first child of the deleted tag. The deleted tag remains in the document until tracked changes are accepted. Note the similarity to **atict:admm**.

```
<quote><para>Wow! That presentation was the <i><atict:delm />very
best</i> I saw. I filled out a <i>glowing</i> presenter's
evaluation. I hope you did too.</para></quote>
```

Splitting an Element: `<atict:split1 />` and `<atict:split2 />`

When a single element is split into two adjacent elements of the same type (for example, by Edit > Split or the ACL split command), the first resulting element is indicated by an **atict:split1** singleton element, and the second by an **atict:split2** element. These two change-tracking elements are mutually linked with a “ref” attribute that shares a common id number.

```
<quote><para>Wow! That presentation was the very best I
saw.<atict:split1 ref='1'> </para><para><atict:split2 ref='1'>I
filled out a <i>glowing</i> presenter's evaluation. I hope you
did too.</para></quote>
```

Joining Two Elements: `<atict:join1 />` and `<atict:join2 />`

When two adjacent elements of the same type are merged into a single element (for example, by Edit > Join), the first joined element is marked by inserting an **atict:join1** element as its final child. The second joined element is marked with an **atict:join2** as its first child. These two change-tracking elements are mutually linked with a “ref” attribute that shares a common id number. Note how join elements are similar to split elements.

```
<quote><para>Wow! That presentation was the very best I
saw.<atict:join1 ref='1'> </para><para><atict:join2 ref='1'>I
filled out a <i>glowing</i> presenter's evaluation. I hope you
did too.</para></quote>
```

Additional Elements

There are a few additional change-tracking elements that don't specifically track revisions, but instead store metadata about the change-tracking markup in the document. These are stored at the beginning of the document as the first children of the root element. The first is **atict:info**:

```
<atict:info tracking="on" ref="0"/>
```

The **tracking** attribute indicates whether change tracking is currently activated in the current document. The documentation says that the **ref** attribute “uniquely identifies change tracking records,” but it is unclear what that means in the current context.

The second metadata element is **atict:user**:

```
<atict:user user="jsulak" fullname="James Sulak"/>
```

It maps a unique user name (which, as you will see below, is stored in each change-tracking element) to its associated full name.

CHANGE-TRACKING ATTRIBUTES

For simplicity, I didn’t show all of the attributes on the change-tracking markup in the examples above. Arbortext adds several attributes to each atict element. The first is **time**, which stores the time (in seconds since January 1, 1970) when the modification was made. The second is **user**, which stores the username of the person who made the modification. This attribute links the change to an **atict:info** element at the beginning of the document, which stores the user’s full name (if available).

ADDING ADDITIONAL METADATA TO ATICT MARKUP

Change-tracking markup stores hidden information, including the date the revision was made and the user who made it. It turns out that you can add additional custom information to the markup, by using ACL and `changetrackingafterhook`. This information can be accessed programmatically in your customizations.

So what is a hook? A hook is a user-defined function that is tied to a specific event in Editor. For example, there are hooks for when a new file is opened, when a parser error is reported, etc. A hook is activated by the `add_hook()` function, and deactivated by the `remove_hook()` function.

The event we are interested in is the insertion of an atict element, which is signified by `changetrackingafterhook`. This function returns an OID representing the new change-tracking element, which we can use to insert our own attribute. Here’s the code (it’s also in `scripts/workflow.acl`):

```
function workflow::change_hook(oid1, oid2, optimized) {
    oid_modify_attr($oid1, 'atict:editorialstep', $workflow::step);
    return;
}

local rc;
rc = add_hook('changetrackingafterhook', 'workflow::change_hook');
$workflow::step = workflow::get_current_step();
```

First, we declared the function `workflow::change_hook()`, and used the `add_hook` command to tie it to the change-tracking event. Now, when Arbortext inserts a new atict element, it will also call

workflow::change_hook, passing it predefined parameters, including the OID of the new element, which can be passed to the oid_modify_attr() function.

STYLING ATICT MARKUP

You can style change-tracking markup the same way you can any other elements in Styler – for example, to highlight inserted text instead of underlining it. Or you can style the markup conditionally based on attributes – for example, by user (using the user attribute), or by which editorial step a document is in (using the custom editorialstep attribute we added in the previous section).

To see an example of styled change tracking, use Styler to open doctypes/simple/simple.style in the example custom folder. Make sure that Options > List All Elements is checked. Otherwise, you will not be able to see change-tracking elements in Styler.

To add atict:add to a stylesheet, go to Elements > New Element, and type in atict:add. You can do the same for atict:del.

MANIPULATING ATICT MARKUP OUTSIDE OF EDITOR

ACCEPTING TRACKED CHANGES PROGRAMATICALLY

While you can successfully validate a document with change-tracking markup inside of Arbortext, you won't be able to validate it in any other tool – for example, an XSLT processor. The reason, of course, is that atict elements generally aren't defined in a DTD. Arbortext turns out to be an exception to this rule because it specifically knows that it should ignore them when validating a document.

As long as you work with your documents exclusively inside of Arbortext, this isn't a problem. But inevitably, the situation will arise when you'll need to parse and validate your document outside of Editor. So what do you do?

Usually, the answer is to accept the tracked changes. However, opening up 50 documents one by one in Editor to accept changes is tedious.

Fortunately, accepting tracked changes without Editor isn't difficult. I've included three scripts in the presentation materials – an XSL transform, a Python script, a .NET executable. These can be invoked from the command line, and with a little work, by right-clicking a file in Windows Explorer. Of the three, I will only cover the XSL transform in detail, but the source code for all three is available in the “src” folder.

The XSL transform is brief, but demonstrates several interesting XSLT techniques, including modes and named templates.

CREATING AN “ACCEPT CHANGES” EXPLORER RIGHT-CLICK MENU OPTION

The console application accept.exe, along with a batch script, can be used to create a right-click context menu option in Windows Explorer. Here's how:

1. Put `accept.exe`, `XMLUtils.dll`, and `accept_all.bat` somewhere where you can find it, such as `My Documents/scripts`.
2. Open Windows Explorer. Go to `Tools > Folder Options`.
3. Select the `File Types` tab.
4. Scroll down until you find the `XML` extension, then select it.
5. Click the `Advanced` button. If it's not there, then click `Restore`. This will restore the default file association and allow you to bring up the advanced properties.
6. Click `New...`
7. Type "Accept Changes" into the `Action` box. Click `Browse`, and browse to where you saved `accept_all.bat`.
8. You're done. Click `OK` until you have closed all the dialog boxes.

SEARCH/HIGHLIGHT CUSTOMIZATION

As an example of a creative use of change-tracking markup, I've included a search/highlight customization that is in current production use at Jones McClure. The purpose of the customization is to allow editors to quickly highlight a predefined list of search terms in a document, and then review them one by one using Arbortext's built-in tracked-changes review capability (`Tools > Change Tracking > Accept or Reject Changes`). Generally, it's used to assist proofreaders in finding common mistakes.

To try it out yourself, first open a sample search list, `searchlist.xml`, in the `/examples` folder of the presentation materials. It's a table in three columns. The first column defines a regular expression, the second, a context, and the third, a comment. Using the `Searchlist` menu, you can validate that the regular expressions and the `XPath` expressions are valid.

Next, open up `down_and_out.xml`. Go to `Searchlist > Search/Highlight Document`, and when the open file dialog box opens, choose `searchlist.xml`. Next, it asks what name to save the resulting document under. Choose `output.xml`, and click `okay`. Open the resulting document to observe the results.

I'll leave the implementation details as an exercise for the interested reader. But generally speaking, the work is done by a console `.NET` application, `highlight.exe`, which is called using the `system() ACL` function. The `.NET` source is in the `/src` folder, and the Arbortext customization source is in `/scripts/highlight.js`.

MORE INFORMATION

Change Tracking Markup Specification

<http://www.arbortext.com/namespace/atict/change-tracking-markup-spec.html>