1. HOW DITA SPECIALIZATION WORKS

The specialization mechanism sets DITA apart from all other XML standards (as explained here). In this blog post, I want to clarify this mechanism in a way that non-technical people can understand. This may help you see opportunities for DITA specialization in your own content production and publication environment.

Note: This blog post does not explain how to create a specialization. That information can best be found on Eliot Kimber's site.

The need for specialization

It all starts with semantics, which is why XML exists in the first place. The semantic tags add meaningful metadata to your content. Instead of using format-oriented tags like <heading1> or <bold>, semantic tags like <title> or <keyword> convey meaningful metadata. How the <title> and <keyword> are rendered in published content may depend on various factors. A <title> might turn out bold or in italics depending on the position of the title in the content hierarchy. A <keyword> should stand out from the content around it, so it might be shown in bold in normal running text, but would be in bold italics in a title that is already bold.

But even if there is no visible difference in the rendering of tagged content there is a reason for the tags. If a piece of content is tagged as <dimension> with a number of attributes or child tags that specify the unit and size, this element may be automatially transformed from one measurement system into another, without human intervention, e.g. <dimension><size>1</size><unit>inch</unit><dimension> will be rendered as 1 or as 2.54cm depending on the target locale of the publication.

You may be asking yourself "but wait, I did not see the <dimension> tag in the DITA specs. Have I missed something?" No, you have not. This is exactly where DITA stands out from other XML standards. You can simply add your own semantics to DITA and use it in your content without breaking the rules. This process is called specialization and it is not as complex as many of you might think.

specialization is evolution of an existing element

The reason for 'D' in DITA relates to the specialization mechanism, as it is short for Darwin. Every specialised element in DITA is derived from a more generic ancestor, and this ancestry determines where the specialised element can be used but also what its boundaries are. If you are specialising a <step> to a <superstep> element, you will not be able to use it outside the <steps>, <steps-unordered> or <steps-informal> elements, as the <step> ancestor of your <superstep> is only valid there.

The specialised element inherits the conditions and characteristics of its ancestor. It just gets a different tag name, so that it can be distinguished as a special case of its ancestor. When you create a specialization, you need to pick the right ancestor, so that you will be able to use your specialization as intended. For the <dimension> example above, the right choice could be <keyword> or <ph> or even <propvalue>. The choice of ancestor is determined by the usage you are going to make of the new tag. Should it only be used in a <properties> table, then the <propvalue> would be the right choice. But normally you would want to make it more generically available, so you may want to choose <ph>.

You can of course restrict the usage of your new specialization by imposing constraints, but that is a topic that will be covered in another blog post. The main takeaway from this section is that you cannot make your specialised element appear outside the boundaries that are already defined by its ancestor.

How does the specialization add value?

First of all, the specialised tags add more specific meaning to your content. In many cases, this is all that is required. I call this 'semantic specialization' as it does not entail anything but having a more specific name for content. One reason to create this type of specialization lies in a subsequent structural specialization. If you are assembling a library of recipes, you may want to have specialised sections for ingredients, preparation and allergy information.

One level up in sophistication (and complexity) is structural specialization, where you (re-)define the contents of an existing element to constrain the freedom that authors have to order its components. An example would be an <api> as specialization of <topic>. You may want to have a fixed number of sections in a fixed order, e.g. <description>, <syntax>, <inputs>,

<outputs>, <exceptions> and <example>. All of these sections would be semantic specializations, except for <example>, which already exists. To string them together in a fixed order, you specialise <body> to <apibody> and define the content of <apibody> to call out the specialised sections in the desired order. The <api> topic will now be a guideline for authors that only validates when all the sections are present.

The last type of specialization I want to mention here is one where the specialised element should be treated differently when the content is rendered. The <dimension> that I mentioned earlier would be such a case, where the rendering software does different things to the <dimension> element depending on the target locale. This type of specialization brings us to the mechanism of specialization that ensures all your extra tags are valid and can be processed by DITA compliant software.

How are specialised elements rendered?

DITA content is meant to be rendered in one of a variety of output formats. The rendering software follows rules about how all kinds of elements are to be rendered. When a specialised element is encountered, there is no rule about its rendering. Instead of breaking down in an ugly way (as would be the case with non-DITA content that has non-conforming tags in it), the DITA processing software looks for the ancestry of an unknown element to see how the element must be treated.

This is what the class attribute of DITA elements is about: it shows the ancestry of a specialised element, allowing the rendering software to treat the unknown specialization in the same way as its known ancestor. In the <api> example mentioned above, each of the <description>, <syntax>, <inputs>, <outputs> and <exceptions> elements have a class attribute that points to the <section> element, allowing the rendering software to treat them as such.

You also have the option to change the rendering behaviour for your specialised elements. Adding specific pieces of XSLT and/or XSL-FO code to the rendering software makes this happen. This falls outside the scope of this blog post, but there are online resources that describe this in detail for the DITA Open Toolkit. The bottom line is that even if your specialised content is rendered on a computer that does not have the special treatment installed, it will still work. The result may be sub-optimal, as the specialised element is treated like its direct ancestor, but the rendering process will not break down.

Conclusion

With DITA’s unique specialization mechanism, you can enrich your content with semantics that are valid and valuable for your organisation, without losing the ability to exchange that content with others who may not have the need, and the specific rendering, for your specializations. This is a unique feature that supports interoperability via the exchange of semantically rich content.