

## Subsetting and customizing DITA

This article explores ideas related to subsetting and customizing the DITA specification without the addition of new elements. Instead, we explore taking default rules and adapting them to meet the needs of specific writing and publishing environments.

Introductory information about the DITA specification and the difference between subsetting and specialization is provided.

The primary audience for this document is:

- ◆ People who are new to XML and the DITA specification and need to get up and running quickly
- ◆ Anyone seeking an understanding of why the DITA specification is gaining popularity and wants to learn about it;
- ◆ Users overwhelmed by the number of elements, attributes and values available in the DITA specification;
- ◆ Those familiar with structure and the DITA specification looking for ways to customize an implementation;
- ◆ Anyone seeking insight into limiting the number of tags used in a structured environment

This document contains the following sections:

- ◆ [About DITA on page 2](#)
- ◆ [Subsetting versus specialization on page 2](#)
- ◆ [Why subset and modify DITA on page 3](#)
- ◆ [Sample subsetting of a DITA element on page 4](#)
- ◆ [Subsetting tags on page 6](#)
- ◆ [Subsetting occurrence indicators and order on page 8](#)
- ◆ [Drawbacks to subsetting on page 9](#)
- ◆ [Conclusion on page 10](#)
- ◆ [Upcoming events on page 10](#)
- ◆ [Related materials on page 11](#)
- ◆ [Tools used to develop this article on page 11](#)
- ◆ [Author information on page 11](#)

## About DITA

The Darwin Information Typing Architecture (DITA) is an XML-based, end-to-end architecture for authoring, producing, and delivering technical information. This architecture consists of a set of design principles for creating “information-typed” modules at a topic level and for using that content in delivery modes such as online help and product support portals on the Web.

This architecture and DTD were designed by a cross-company workgroup representing user assistance teams from across IBM. After an initial investigation in late 1999, the workgroup developed the architecture collaboratively during 2000 through postings to a database and weekly teleconferences. The architecture has been placed on IBM's developerWorks Web site as an alternative XML-based documentation system, designed to exploit XML as its encoding format. With the delivery of these significant updates, which contain enhancements for consistency and flexibility, we consider the DITA design to be past its prototype stage.

## Subsetting versus specialization

To help clarify the difference in meaning between subsetting and specialization, consider an example that relates to the use of the 26 characters of the traditional English alphabet; that is, the letters A through Z. Generally, the idea of subsetting is similar to that of removing letters of the alphabet.

Consider a subset containing only the five principle vowels, AEIOU, and the letters RLSTN. Through the subset of these 10 letters we can create words like *stone*, *lesson*, *rail*, *station*, *institutional*, *satin* and so on. Each of these words is easily recognized as a well formed word by users of the full set of 26 letters. It is also recognized by users of the subset. However, the words *frequency*, *subtle* or *railway* would not be recognized by the subset.

A specialization example might be the addition of more characters to the core of 26 alphabetical characters. For example, an accounting firm may create a specialization that adds the Arabic numerals zero through nine (0 to 9) which is not a part of the default 26 alphabetical characters.

Therefore, *subsetting*, in the context of this document, means to remove or reorganize elements, attributes and attribute values to customize the way that options are presented, while ensuring all the DITA specifications are followed. *Specialization* is the general process by which additional elements are added to the DITA specification to allow for custom development of new information types.

## Why subset and modify DITA

There are numerous reasons to consider customizing the DITA specification either through subsetting or through specialization. The three main reasons to subset include changing default tags, modifying the order or elements and adjusting the frequency of element use.

### Default tag set

The default tags in the DITA specification include close to 200 elements. In many cases, tags provide features that are not required in your documentation or provide undesired redundancy. Many tags can be removed from the architecture and still leave all the structure authors need. This can be done without specialization.

As an example, 28 elements exist for use with prolog. The prolog element (and the 27 related subelements) exists to store metadata related to the document. However, if your document set does not need all of these elements, then removing some, or all, of the elements may make sense.

Further to this, the reduction of the number of elements also simplifies maintenance of your own file set. With fewer tags to define, modify, update, format, teach and more, the ability to quickly get a DITA work environment up and running is enhanced.

### Default element order

The order of elements in the DITA specification is incredibly flexible. This means that elements can be inserted in a variety of ways. The result is documents that allow writers the freedom to write. However, that freedom may result in writers skipping some elements or inserting others in an order that doesn't adhere to your style guide. Modifications to the default element order allow restrictions to the organization of information. If this still adheres to the principles of the DITA specification your content remains compliant and your authors have a guided workflow.

For example, within the `step` of a given `task`, the DITA specification allows numerous additional elements in any order. This includes:

- ◆ `info` (information to provide additional information about the step)
- ◆ `substeps` (to break a step down into a series of separate actions, recommended in the DITA specification to be used only if necessary)
- ◆ `tutorialinfo` (information that is included in a step when a task is part of a tutorial)
- ◆ `stepxmp` (used to provide a specific example of a step of a task)
- ◆ `choicetable` (contains a series of optional choices available within a step of a task)
- ◆ `choices` (used when the user will need to choose one of several actions while performing the steps of a task)

These six elements can be inserted in any order, according to the DITA specification. By customizing the rules you can decide to enforce a specific order, thereby ensuring users insert `info` before they can work with `choices`. This example would ensure that `info` is provided about why additional steps are needed before any choices are allowed.

Even if the order is modified, or child elements are completely removed from the default element options, nothing restricts the author from putting in an unlimited number of `info` elements. Therefore it is also important to consider modifications to the default element frequency.

## Default element frequency

Many of the elements in the DITA specification allow child elements to appear with no restrictions. This means that, basically, authors can insert a wide variety of elements as often as desired. Ironically, this may result in undesired content, such as a `step` made up of a `cmd`, followed by an unlimited number of `info` child elements.

For example, within the `step` of a given `task`, the DITA specification allows numerous additional elements with no restriction on frequency. By modifying the defaults you can configure the frequency of elements to restrict the amount of content that authors insert.

Again, it is important to combine modifications to the frequency of elements with the default element order for best results.

## Sample subsetting of a DITA element

As a practical example of subsetting within the DITA specification, consider the `step` element. This element has numerous default child elements with few limitations placed upon them. By defining a subset of the `step` element we allow authors to create content while ensuring specific guidelines are followed. This ensures clear content is created within the DITA specification. All output is also fully compliant with the DITA specification. We therefore enforce a custom style of writing while following the DITA specification.

In this example we compare the default and custom implementation of a step. It is generally used in task. You may be familiar with this type of content from your own writing.

We assume familiarity with the syntax of XML. If you are not familiar with XML a very primer is found at [www.publishingsmarter.com](http://www.publishingsmarter.com) in Books/Articles under Learn about XML.

## Default rule of `step`

Before we modify any elements, let's begin by reviewing the default rules that the DITA specification enforces when working with the element `step`. The `step` element specifies: `cmd` then ( `info` or `substeps` or `tutorialinfo` or `stepxmp` or `choicetable` or `choices` ) (0 or more) then ( `stepresult` ) (optional).

In DTD syntax this may be represented as:

```
cmd, (info | substeps | tutorialinfo | stepxmp | choicetable
| choices)*, stepresult?
```

The default allows a wide range of elements in virtually any order and with no limit to the number of elements inserted within the `step`.

Therefore, using the default a writer can create the following type of content:

- 
- 1 Select File > Save As.
  - 2 Select a filetype under Save as type.  
For example, choose Web for the internet or choose Document for a local file.  
This lets you specify the export format.
    - a select Web
    - b select a location
    - c enter a file name
 or you could
    - a select Document
    - b select a location
    - c enter a file name
 After you perform this substep continue to save your file.
  - 3 Click OK.
- 

In this example the second `cmd` element is structured as seen below:

```
step
  cmd (Select a filetype...)
  stepxmp (For example, choose...)
  info (This lets you...)
  choices (select...)
  info (or you could...)
  choices (select...)
  info (After you perform...)
```

The `step` contains numerous elements; some of which repeat and appear in an order that may not be repeated the next time the element `step` is used. By developing a custom rule additional restrictions can be enforced for consistency within your organization.

## Custom rule for `step`

The development of a custom rule should always be done with the confidence that output will still match the DITA specification. Basically, if the DITA specification allows an element to have numerous optional child elements, it is relatively simple to remove any of them. Since child elements are optional, removing them has no negative impact on output. If the child is required, then subsetting should not be done as the output will not meet the DITA specification.

An example of a custom definition of `step` is seen below:

```
cmd, (info, choices)?
```

This customization requires the author to insert a `cmd`. Once inserted the author has the option to insert `info` which can be followed by optional `choices`.

Therefore, using a custom definition a writer can create the following type of content:

- 
- 
- 1 Select File > Save As.
  - 2 Select a filetype under Save as type.  
For example, choose Web for the internet or choose Document for a local file.  
This lets you specify the export format.
    - a select Web or Document
    - b select a location
    - c enter a file name
  - 3 Click OK.
- 

In this example the second `cmd` element is structured as seen below:

```
step
  cmd (Select a filetype...)
  info (This lets you...)
  choices (select...)
```

The `step` contains a required `cmd` element. After the `cmd` there is a single `info` and a `choices` element. Nothing beyond this limited subset is allowed. Authors can not insert examples, multiple strings for `info`, numerous `choices` and more. The restriction helps to ensure consistency and provides more detailed guidance for each of the authors when working with a `step`.

Again, we see that by developing a custom rule, restrictions can be enforced for consistency within your organization.

## Result of subsetting `step`

The result of the customization is a document set that is more professional, consistent and easier to manage. Editing and translation are simplified as there are fewer decisions that need to be made based on writing style.

## Subsetting tags

There are close to 200 tags in the DITA specification. One of the easiest things you can do to make a DITA implementation simpler is to reduce the number of tags.

By identifying the elements and attributes that are not required, a major task associated with customization is accomplished. Aside from identifying these, it is also necessary to determine if they are required or optional. Fortunately, many of the tags are logically grouped together and therefore easier to work with. The DITA specification allows most content to be optional using the asterisk and question mark occurrence indicators, often further simplifying the removal of elements and remaining true to the DITA specification.

## High level DITA elements

Numerous high level elements exist in the DITA specification and several can be safely removed when subsetting. It is important to first plan your document set and then begin to subset as the removal of high level elements and all associated child element is difficult to undo later. Also ensure that any element that is removed is not required elsewhere in the DITA specification. If it is, ensure you make appropriate modifications in all locations.

The primary element groupings in the DITA specification include:

- ◆ Topic
- ◆ Concept
- ◆ Reference
- ◆ Task
- ◆ Body
- ◆ Table
- ◆ Domain
- ◆ Miscellaneous
- ◆ Prolog
- ◆ Related link
- ◆ Specialization
- ◆ Map

If you realize your documentation does not include, for example, tables or tasks, then the bulk of the elements associated with these two high level elements could be removed from your implementation. In the same way, if you do not use related links or maps, another set of elements can be removed.

## Common attributes

There are also several attributes that are commonly used throughout the DITA specification that may not be required. As with elements, it is important to plan your document set and then begin to subset. Remember that many of the attributes are reused throughout the DITA specification and it may be better to remove them on an element by element basis rather than removing them from the DITA specification completely.

The primary attribute groupings in the DITA specification include:

- ◆ Display attributes
- ◆ Global attributes
- ◆ ID attributes
- ◆ Relational attributes
- ◆ Select attributes
- ◆ Universal attributes

## Subsetting occurrence indicators and order

The frequency of elements in the DITA specification can be subset. Since the majority of elements are optional, removing them poses no significant impact in the compliance of your content with the DITA specification.

Several examples are provided that demonstrate a subset of tags, or a modification to the frequency and order of elements. This is not to say that any of these examples are the only way to subset the DITA specification, but they do ensure a more clearly defined order to content and custom limitations on the number of elements.

---

### Example: Subset step

As seen earlier in this article, the element `step` can be subset as required.

Another example of a subsetting of the default definition of the element `step` may appear as seen below:

```
cmd, info?
```

This new rule still matches the DITA specification. However, it has been customized to specify that a `cmd` must be inserted. Then, if required, `info` may be added, but only once.

---



---

### Example: Subset paragraph

Consider an example of the default definition for a paragraph, represented in the DITA specification through the use of the element `p`.

```
(<TEXT> | ph | codeph | synph | filepath | msgph | userinput |
systemoutput | b | u | i | tt | sup | sub | uicontrol | menucascade
| term | xref | cite | q | boolean | state | keyword | option |
parmname | apiname | cmdname | msgnum | varname | wintitle | tm |
lq | note | dl | parml | ul | ol | sl | pre | codeblock | msgblock
| screen | lines | fig | syntaxdiagram | imagemap | image | object
| table | simpletable | draft-comment | required-cleanup | fn |
indextermref | indexterm)*
```

This allows huge variations in the content of a paragraph. One of the first things that can be done with this element is a subsetting of content. Perhaps a review of the elements identifies a lot of options that may not apply in your environment.

The paragraph allows numerous elements for text formatting (such as `b`, `u`, `i`, `tt`). It also allows for elements that are more clearly defined in function such as `filepath`, `userinput`, `uicontrol`, `cmdname` and so on. If we decide that formatting on the fly (that is, applying bold, underline, italic, `typetext`) is restricted in our document we can drop these. In their place elements such as `filepath`, `userinput`, `uicontrol` and `cmdname` may be more useful. These elements define the purpose as opposed to arbitrary formatting.

Further review may identify other elements that we do not need to use in a paragraph and we end up with a subset that might appear as follows:

```
(<TEXT> | filepath | userinput | systemoutput | uicontrol | xref
| keyword | cmdname | image | table | indexterm)*
```

This subset has removed many of the elements that we do not need, but has left the order up to the author. At present, the author is limited in the number of elements, but not in the frequency. Therefore, we may also consider modifications to the order and frequency of the remaining elements. If we rewrite the rule once more we may end up with:

```
(<TEXT> | filepath | userinput | systemoutput | uicontrol | xref
| keyword | cmdname | indexterm)*, (image | table)?
```

This custom rule matches the DITA specification. However, it now specifies text and text objects must occur first. After text, if desired, a single image or table may be added. This still meets the rule set by the DITA specification, but is stricter.

## Drawbacks to subsetting

There are two key drawbacks to consider before subsetting: *tag limitation* and *stricter rule requirements*. If a DITA implementation is well planned neither should be a major problem in managing the way DITA is used.

### Tag limitation

While subsetting helps to implement a stricter implementation of the DITA standard, it also deviates from it. By only supporting a key set of tags you restrict the ability to import other content that complies with the DITA specification.

For example, if you redefine the element `p` to not allow `b`, `i`, `u`, `tt`, `sub` and `sup`, you may have to manage these elements when importing content from other sources. If your implementation clearly defines that `p` should contain `varname`, `uicontrol`, `keyword` and other tags, you can review the use of other elements and decide on how to manage their import. In this example, you may decide that, on import, `b` should be converted to `varname`, `i` to `uicontrol` and so on.

While there may be additional work when first importing content, the use of logical named objects (as opposed to format driven conventions) provides more value to content. Logical application of conventions generally supports far more reuse and clearly defines content based on purpose rather than on appearance.

## Stricter rule requirements

By redefining the order of elements and their frequency, you effectively rule out some combinations of elements that others may use. In doing so, you may be limiting the usefulness of content that others provide that match the DITA specification.

When content from other sources that adhere to the DITA specification is imported to your system it may be necessary to review the content and, perhaps, reorganize it. This must be done to ensure that your stricter rules are followed.

The net result may be a limitation in the organization of information and the need to rework content provided to you.

## Conclusion

Subsetting the DITA specification and modifying the default rules can provide many benefits to an organization. A restricted set of elements reduces the need to develop formatting and transformation rules for all possible combinations of elements. It also allows organizations to further control the types of content used and the way that they are used. This results in far more consistent documentation.

As long as any subsetting and modification of the rules is done in such a way that compliance with the DITA specification is assured in your output, then subsetting can be beneficial. The key is to plan based on your current documentation environment and to also plan for any future implementations that are expected.

By subsetting, you still follow the DITA specification and therefore ensure all content is 100% compliant with the DITA specification. This allows you to use default transformations, multiple third party tools and any future utilities that manage DITA content with very little modification. Software and hardware that works with DITA will, by default, work with your subset. Very little, if any, additional effort is required.

Custom implementations of most XML architectures from DocBook to S1000D to the DITA specification happen all the time. By restricting tags and enforcing custom order your DITA implementation can be done quicker, with more reliable results and at a lower overall cost of development, training and implementation.

## Upcoming events

The author of this article is involved in several events in 2006.

- ◆ 2006 DITA Conference, March 23 to 25  
Presenting a case study on DITA implementation using a subset.  
[conf.travelthepath.com](http://conf.travelthepath.com)
- ◆ FrameMaker and DITA Seminar, March 3, May 5  
Presenting an online seminar on developing a DITA and FrameMaker interface.  
[www.pubsnet.com](http://www.pubsnet.com)
- ◆ Toronto STC Spring Conference, March 27 to 29

Preparing for the Content Management Tipping Point: conference features five half-day workshops for intermediate and senior technical communicators.

[www.stctoronto.org](http://www.stctoronto.org)

- ◆ DITA Summit, June 7

An online summit bringing together original designers of DITA, and early adopters who have successfully implemented DITA in their organizations.

[www.pubsnet.com](http://www.pubsnet.com)

## Related materials

A variety of related materials can be found online, including a set of FrameMaker specific documents for developing and publishing DITA content using a custom subset:

- ◆ Introduction to DITA

[www-128.ibm.com/developerworks/library/x-dita1/docs.oasis-open.org/dita/v1.0/archspec/ditaintro.html](http://www-128.ibm.com/developerworks/library/x-dita1/docs.oasis-open.org/dita/v1.0/archspec/ditaintro.html)

- ◆ Overview of XML

[www.w3.org/XML/](http://www.w3.org/XML/)  
[www.publishingsmarter.com/pages/teach/xml\\_primer.html](http://www.publishingsmarter.com/pages/teach/xml_primer.html)

- ◆ Author Site, including a fully functional DITA and FrameMaker sample document set.

[www.publishingsmarter.com](http://www.publishingsmarter.com)

## Tools used to develop this article

As the saying goes “we eat our own dog food”. In an effort to prove that DITA can be used to author content, and to deliver it in numerous formats, we created this entire article using readily available tools. Content was converted as required via numerous transforms provided with the DITA toolkit.

This article is created using *Adobe FrameMaker 7.2* and a subset of the DITA specification created by the author; it is freely available by request. The DITA XML output was tested using *XMetaL DITA Edition* for compliance and to review content online. The XML files are transformed using *OxygenXML* for output to HTML. As a proof of concept, the source code of the article is available in XML that is compliant with the DITA specification. It may be downloaded and tested as required.

## Author information

A recognized publishing technologies expert, Bernard Aschwanen presents at conferences and events across Europe and North America. Bernard is an Adobe Certified Expert, a Certified Technical Trainer and the author of numerous publications on publishing and single sourcing including *Advanced FrameMaker*, published by TIPS Publishing.

February 21, 2006

Provided by [www.publishingsmarter.com](http://www.publishingsmarter.com)

The founder of Publishing Smarter, a senior member of the Society for Technical Communication, the Vice President of the Toronto STC and Past President of the Computer Trainers Network, Bernard has helped hundreds of companies implement successful publishing solutions. Bernard is focused on publishing better, publishing faster and publishing smarter.

Home Page: <http://www.publishingsmarter.com>

Email: [dita@publishingsmarter.com](mailto:dita@publishingsmarter.com)