Review P: Configuration

Table of contents

1 Configuration, specialization, generalization, constraints, and expansion

The extension facilities of DITA allow document-type shells, vocabulary modules, and element-configuration modules (constraint and expansion) to be combined to create specific DITA document types.

Comment by Kristen James Eberlein on 16 August 2022

Note that the title contains the term "generalization," although we are not including the generalization content in this review – It needs too much work. That content is slated for a future review.

On a different note, the title of this topic is too long. What would folks think about changing it to just "Configuration and specialization"?

Eliot Kimber, 29 August 2022

If we removed "generalization" from the title that make it sufficiently short. "Configuration and specialization" is accurate and I would support that option as well.

Robert Anderson, 29 August 2022

I also think removing "generalization" from the title is a good overall result here.

Disposition: Referred

Comment by dstevens73

I am uncomfortable leaving "constraints" out of the title. Many of my clients don't understand the distinction between constraints and specializations and leaving constraints out of the title seems like it might further the confusion

---

Kris Eberlein, 23 August 2022

Do remember that your clients are NOT the target audience for the spec.

We do carefully define both constraints and expansion as "element-type configuration".

Disposition: Referred

Comment by Scott Hudson

The shortdesc also refers to constraints and expansion. Should it be modified as well? I am OK with simplifying the title, but I think the shortdesc be updated to be consistent with the title.

Kris Eberlein, 24 August 2022

Both constraints and expansion are types of configuration: "element-type configuration". But yes, if we change the title, we also will need to consider whether the shortdesc will need editing. Personally, I think it's fine as-is.

Disposition: Closed

Comment by Zoë Lawson on 28 August 202

I agree that the title should be shortened. I'm having a lot of issues with "Configuration". I do see that there are "Document-type configuration" and "Element-type configuration", and they were described, but "document-type configurations" are implemented as document-type shells, and "element-type configurations" are implemented as modules? I'm not super familiar with how you actually implement a specialization, or with XML/DTD/RNG specific terms, so I could just be too ignorant to follow all the nuances.

I think I'd be happy with "Specialization and Generalization". In my brain (that's never really implemented a specialization), if you do anything so that you're not using DITA out of the box, you've specialized it. All the other terms (configuration, shell, constraint, expansion) are "types" of specialization.

Kris Eberlein, 28 August 2022

Zoe, since you are a voting member of the DITA TC, it's critical that you understand the differen ce between "configuration" and "specialization". I'll offer a 4-hour block of time in which we do a tutorial. Configuration, shell, constraint, expansion are NOT TYPES OF SPECIALIZATION.

Disposition: Referred

1.1 Overview of DITA extension facilities

DITA provides three extension facilities: Document-type configuration, specialization, and element-type configuration.

Document-type configuration

Document-type configuration enables the definition of DITA document types that include only the vocabulary modules that are required for a given set of documents. There is no need to modify the vocabulary modules. Document-type configurations are implemented as document-type shells.

Specialization

Specialization enables the creation of new element types in a way that preserves the ability to interchange those new element types with conforming DITA applications. Specializations are implemented as vocabulary modules, which are integrated into document-type shells.

Specializations declare the elements and entities that are unique to a specialization. The separation of the vocabulary and its declarations into modules makes it easy to extend existing modules, because new modules can be added without affecting existing document types. It also makes it easy to assemble elements from different sources into a single document-type shell and to reuse specific parts of the specialization hierarchy in more than one document-type shell.

Comment by Bill Burns

The first sentence here seems tautological. How about "A specialization module declares the elements and entities that are unique to that specialization"?

Kris Eberlein, 23 August 2022

Done

Disposition: Completed

DITA content that uses specializations can be treated as or converted to unspecialized markup through the process of generalization. The information about the original specialized form can be retained.

Comment by Gershon Joseph on 2022-08-24
can be treated as, or converted to, unspecialized

Added commas around "or converted to".

Kris Eberlein, 23 August 2022

Done

Disposition: Completed

Comment by Eliot Kimber
Do we need a link to the now-separate discussoin of generalization?

Kris Eberlein, 30 August 2022

Quite possibly, and I'll leave this comment here, and set the disposition to "Accepted".

Disposition: Accepted

Element-type configuration

Element-type configuration enables DITA architects to modify the content models and attribute lists for individual elements, without modifying the vocabulary modules in which the elements are defined.

There are two types of element configuration: Constraint

Comment by Gershon Joseph on 2022-08-24
s/Constraint/constraint/

Kris Eberlein, 25 August 2022

We need to keep these terms in the singular tense.

Disposition: Rejected

and expansion. Both constraint and expansion are implemented as modules that are integrated into document-type shells:

Constraint
Constraint modules enable the restriction of content models and attribute lists for individual elements.

Expansion
Expansion modules enable the expansion of content models and attribute lists for individual elements.

Comment by Eliot Kimber
lowercase Constraint and Expansion in the definition list terms.

Kris Eberlein, 30 August 2022

No, that does not follow our style guidelines. We standardize on the "IBM Style Guide."

Disposition: Rejected

Comment by Bill Burns

How is expansion different from specialization? Is this new to 2.0 (appears to be so)?

Kris Eberlein, 23 August 2022

Yes, expansion modules are new for DITA 2.0. They are essentially the opposite of constraints; instead of restricting the content model or attribute lists, they expand them. They work in conjunction with specializations.

Hopefully the content later in this chapter will answer your questions.

Disposition: Closed

1.2 Document-type configuration

Document-type configuration enables the definition of DITA document types that include only the vocabulary modules that are required for a given set of documents. There is no need to modify the vocabulary modules. Document-type configurations are implemented using document-type shells.

1.2.1 Overview of document-type shells

A document-type shell is an XML grammar file that specifies the elements and attributes that are allowed in a DITA document. The document-type shell integrates structural modules, domain modules, and element-configuration modules. In addition, a document-type shell specifies whether and how topics can nest.

A DITA document either must have an associated document-type definition or all required attributes must be made explicit in the document instances.

Comment by Stan Doherty
Should be qualified – a document-type shell, whether provided by OASIS or developed by you. As is, the sentence implies that most tooics use specilized doc shells.

Kris Eberlein, 25 August 2022

I don't think the qualification would add value, nor do I think that the current text implies that "most topics use specialized doc shells".

Disposition: Closed

Most DITA documents have an associated document-type shell. DITA documents that reference a document-type shell can be validated using most standard XML processors. Such validation enables processors to read the XML grammar files and determine default values for the `@specializations` and `@class` attributes.

Comment by Eliot Kimber
Technically it's not validation but simple grammar awareness that enables determining attribute defaults. Not sure it's worth splitting that hair here but I am compelled to make the comment. An XML processor can be grammar aware without being a validating processor.

The more precise wording would be something like "that reference a document shelll can be processed in a grammar-aware fashion using most standard XML processors. Such processing enables..."

But I think using "validation" as a shorthand for "grammar-aware processing" is fine.

Kris Eberlein, 30 August 2022

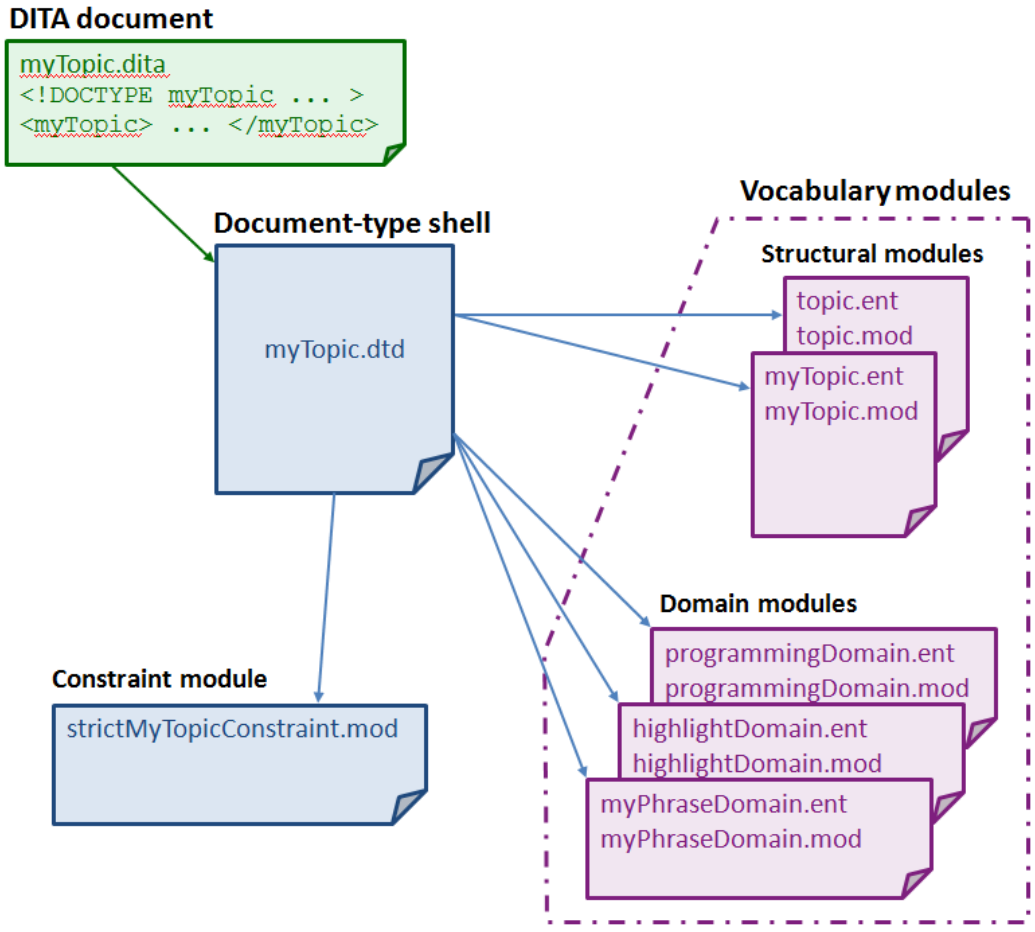
I have tried reworking this content previously to be more precise, and it inevitably ends up being almost unreadable. One of the issues is that we are talking about two different things here:

- Grammar-aware processing that determines default values for the `@class` and `@specialization` attributes
- Validation, which enables folks authoring DITA content to check that their objects validate against the referenced grammar files

Disposition: Referred

The following figure illustrates the relationship between a DTD-based DITA document, its document-type shell, the vocabulary modules that it uses, and the element-configuration modules (constraint and expansion) that it integrates. Similar structure applies to DITA documents that use other XML grammars.

Figure 1: Document type shell



Comment by Kristen J Eberlein on 28 March 2021

The illustration needs to be updated to include expansion modules.

Response by Gershon Joseph, 2022-08-24

I think we should redo examples like these using RelaxNG, since the RNG is the normative grammar of DITA 2.0. It seems odd to me that the spec has so many DTD-only example when the normative grammar is RelaxNG. Maybe we can move DTD equivalents out into a Committee Note or appendix?

Kris Eberlein, 24 August 2022

No, the spec supports both DTD and RNG. The fact that RNG is normative simply means that if there is a difference between a DTD grammar file and a matching RNG grammar file, the RNG version is the correct one. So, absolutely "no" to moving DTD content to a committee note or an appendix.

Regarding reworking the graphic to use RNG: That would be an unnecessary change, and we are experiencing such a resource strain right now concerning spec writing and editing.

Disposition: Rejected

Comment by Eliot Kimber
I'm pretty sure I drew this picture. I'll try to find the original source and update it or I will recreated it. Please give me the action item.

Kris Eberlein, 30 August 2022

You did, and the source for the graphic is in GitHub.

Disposition: Accepted

The DITA specification contains a starter set of document-type shells. These document-type shells are commented and can be used as templates for creating custom document-type shells.

While the OASIS-provided document-type shells can be used without any modification, creating custom document-type shells is a best practice. If the document-type shells need to be modified in the future, for example, to include a specialization or integrate an element-configuration module (constraint or expansion), the existing DITA documents will not need to be modified to reference a new document-type shell.

Comment by Gershon Joseph on 2022-08-24
I don't think this paragraph belongs in the spec. It should be moved out into a CN or appendix that discusses best practices for implementing DITA.

Kris Eberlein, 24 August 2022

Maybe, maybe not. In general, I agree that "best practices" content does not belong in the spec. But:

- This is such a critical point.
- The "overview" topics in this spec chapter are explicitly intended to be high-level overviews that should be accessible to as many people as possible (while still being technically accurate). If we ever manage to create the part 0 "Primer" document that we've had on our wish list, we'd use this content there.

Robert D Andewrson 29 August 2022 – agree with Kris (and Eliot below) that this is a really critical point and should not be removed. This is too easily missed, and really reflects how DITA grammar files are meant to be used (even more than a simple "best practice").

Eliot Kimber, 29 August 2022

I agree with Kris that this is a critical bit of guidance that should be retained. It could be put into a note so that it is explicitly not normative, but I'm not sure that's necessary.

Disposition: Referred

1.2.2 Rules for document-type shells

This topic collects the rules that concern DITA document-type shells.

XML grammars

001 (69)	While the DITA specification only defines coding requirements for DTD and RELAX NG, conforming DITA documents **MAY** use other document-type constraint languages, such as Schematron.

Comment by Eliot Kimber
Is it worth adding " or XSD"?

Kris Eberlein, 31 August 2022

Absolutely. Done!

Disposition: Closed

Defining element or attribute types

002 (69)	With two exceptions, a document-type shell **MUST NOT** directly define element or attribute types; it only includes vocabulary and element-configuration modules (constraint and expansion). The exceptions to this rule are the following: • The ditabase document-type shell directly defines the `<dita>` element. • RNG-based document-type shells directly specify values for the `@specializations` attribute. These values reflect the details of the attribute domains that are integrated by the document-type shell.

Document-type shells not provided by OASIS

003 (69)	Document-type shells that are not provided by OASIS **MUST** have a unique public identifier, if public identifiers are used.
004 (69)	Document-type shells that are not provided by OASIS **MUST NOT** indicate OASIS as the owner. The public identifier or URN for such document-type shells **SHOULD** reflect the owner or creator of the document-type shell.

For example, if `example.com` creates a copy of the document-type shell for topic, an appropriate public identifier would be "-//EXAMPLE//DTD DITA Topic//EN", where "EXAMPLE" is the component of the public identifier that identifies the owner. An appropriate URN would be "urn:example.com:names:dita:rng:topic.rng".

Comment by Eliot Kimber
The URN should have a "namespace ID" component between the "urn:" and specific part, i.e. "ns" for "namespace" or "pubid" or something similar that distinguishes the general type of URN this is. See

https://www.rfc-editor.org/rfc/rfc8141

Robert Anderson, 31 August 2022

Not sure I follow. The example given here matches what we have in our OASIS catalogs, replacing `oasis` with `example.com` – although ours also has a `tc:` that I do not think would be used outside of our TC. For example, the base topic is:

```
urn:oasis:names:tc:dita:rng:basetopic.rng:2.0
```

@Eliot, do o you have a specific example that would be better than the one above?

Kris Eberlein, 31 August 2022

I didn't follow either. We need more info from Eliot here.

Eliot provided more information. We should talk about this at the TC level; do we change our RNG catalog files to match what Eliot has suggested?

Disposition: Referred

1.2.3 Equivalence of document-type shells

Two distinct DITA document types that are taken from different tools or environments might be functionally equivalent.

Comment by Gershon Joseph on 2022-08-24
Is this topic really required? To me it's stating the obvious. If it is required, I'd say it belongs in an implementation guide and not in the spec

---

Kris Eberlein, 25 August 2022

I don't think this is obvious, and I think we need this in the spec. @Robert? @Eliot?

Robert D Anderson 29 August 2022 – agreed, it is not obvious but is rather unique to how DITA operates (as Eliot describes below). Other XML grammars are generally defined by The DTD / The RNG / etc, while DITA grammars are defined by the set of modules + specific configurations in a shell.

Eliot Kimber, 29 August 2022

I think this is an essential aspect of DITA that is not obvious for the simple reason that the vast majority of SGML and XML practice **and tools** assume that the document type declaration or grammar file (resource) defines the document type and if two documents use different document type declaration files then they are in different document types.

But that is explicitly not true for DITA but most non-DITA-aware XML practitioners would not necessarily realize it.

So worth stating explicitly. Gershon is correct that this aspect of DITA is implicit in the definition of "DITA document type" but it's still a pretty subtle implication.

.
Disposition: Referred

A DITA document type is defined by the following:

- The set of vocabulary and element-configuration modules (constraint and expansion) that are integrated by the document type shell
- The values of the `@class` attributes of all the elements in the document
- Rules for topic nesting

Two document-type shells define the same DITA document type if they integrate identical vocabulary modules, element-configuration modules (constraint and expansion), and rules for topic nesting. For example, a document type shell that is an unmodified copy of the OASIS-provided document-type shell for topic defines the same DITA document type as the original document-type shell. However, the new document-type shell has the following differences:

- It is a distinct file that is stored in a different location.
- It has a distinct system identifier.
- If it has a public identifier, the public identifier is unique.

Note The public or system identifier that is associated with a given document-type shell is not necessarily distinguishing. Two different people or groups might use the same modules and constraints to assemble equivalent document type shells, while giving them different names or public identifiers.

1.2.4 Conformance of document-type shells

DITA documents typically are governed by a conforming DITA document-type shell. However, the conformance of a DITA document is a function of the document instance, not its governing grammar. Conforming DITA documents are not required to use a conforming document-type shell.

Comment by Gershon Joseph on 2022-08-24
I don't know why the spec has to say this. It's obvious. I suggest we delete this topic.

Kris Eberlein, 25 August 2022

I don't think this is obvious, and I think we need this in the spec. @Robert? @Eliot?

Eliot Kimber, 29 August 2022

I agree that it's not obvious.

However, I thought we removed conformance of document types entirely--since we removed the conformance requirement of specific file organization for document types and modules, there's no other way in which a grammar can conform that I can see.

Because conformance devolves entirely to document instances if the coding rules are not conformance requirements then I'm not sure we can or should say anything about shell conformance.

Robert Anderson, 29 August 2022

I thought this was the topic that essentially backed us away from the 1.x approach of Grammar File Conformance, and read it generally as "We have all these patterns, but even if you don't follow them, what's important is the actual document conformance rather than the grammar file itself." I think that's an important message in the context of how we wrote 1.x rules, but I guess if we come at it from the "everything is new in 2.0" frame it seems less necessary. I think my position might come down to where we define all the rules.

Disposition: Referred

Conforming DITA documents are not required to have any governing document type declaration or schema. There might be compelling or practical reasons to use non-conforming document-type shells. For example, a document might use a document-type shell that does not conform to the DITA requirements for shells in order to meet the needs of a specific application or tool. Such a non-conforming document-type shell still might enable the creation of conforming DITA content.

1.3 Specialization

The specialization feature of DITA allows for the creation of new element types and attributes that are explicitly and formally derived from existing types. This facilitates interchange of conforming DITA content and ensures a minimum level of common processing for all DITA content. It also allows specialization-aware processors to add specialization-specific processing to existing base processing.

1.3.1 Overview of specialization

Specialization allows information architects to define new kinds of information (new structural types or new domains of information), while reusing as much of existing design and code as possible, and minimizing or eliminating the costs of interchange, migration, and maintenance.

Specialization modules enable information architects to create new element types and attributes. These new element types and attributes are derived from existing element types and attributes.

In traditional XML applications, all semantics for a given element instance are bound to the element type, such as `<para>` for a paragraph or `<title>` for a title. The XML specification provides no built-in mechanism for relating two element types to say "element type B is a subtype of element type A".

In contrast, the DITA specialization mechanism provides a standard mechanism for declaring that an element type or attribute is derived from an ancestor type. This means that a specialized type inherits the semantics and default processing behavior from its ancestor type. Additional processing behavior optionally can be associated with the specialized descendant type.

For example, the `<section>` element type is part of the DITA base. It represents an organizational division in a topic. Within the task information type (itself a specialization of `<topic>`), the `<section>` element type is further specialized to other element types (such as `<prereq>` and `<context>`) that provide more precise semantics about the type of organizational division that they represent. The specialized element types inherit both semantic meaning and default processing from the ancestor elements.

There are two types of DITA specializations:

Structural specialization

Structural specializations are developed from either topic or map types. Structural specializations enable information architect

Comment by Gershon Joseph on 2022-08-24
s/architect/architects/

Kris Eberlein, 24 August 2022

Done

Disposition: Completed

Comment by Eliot Kimber
I think singular "architect" requires "an " before it: "enable an information architect …"

Kris Eberlein, 31 August 2022

This is already handled; see Gershon's comment above.

Disposition: Closed

to add new document types to DITA. The structures defined in the new document types either directly use or inherit from

Comment by Gershon Joseph on 2022-08-24
...directly use, or inherit from, elements... (missing commas, which are needed to help non-native English speakers correctly parse and understand the content)

Kris Eberlein, 24 August 2022

Done

Disposition: Completed

elements found in other document types. For example;

Comment by Gershon Joseph on 2022-08-24
s/;/,

Kris Eberlein, 24 August 2022

Replaced semi-colon with a comma

Disposition: Completed

concept, task, and reference are specialized from topic, whereas bookmap is specialized from map.

Domain specialization

Domain specializations are developed from elements defined with topic or map, or from the `@props` or `@base` attributes. They define markup for a specific information domain or subject area. Domain specializations can be added to document-type shells.

Comment by Eliot Kimber
"defined within topic or map" (c/with/within/)

Kris Eberlein, 31 August 2022

Done

Disposition: Completed

Each type of specialization module represents an “is a” hierarchy, in object-oriented terms, with each structural type or domain being a subclass of its parent. For example, a specialization of task is still a task, and a specialization of the user interface domain is still part of the user interface domain. A given domain can be used with any map or topic type. In addition, specific structural types might require the use of specific domains.

Use specialization when you need a new structural type or domain. Specialization is appropriate in the following circumstances:

- You need to create markup to represent new semantics (meaningful categories of information). This might enable you to have increased consistency or descriptiveness in your content model.

- You have specific needs for output processing and formatting that cannot be addressed using the current content model.

Do not use specialization to simply eliminate element types from specific content models. Use constraint modules to restrict content models and attribute lists without changing semantics.

Comment by dstevens73
With the new expansion capability, do we need to change the sentence for "eliminate or add element types" and then "Use constraint modules to restrict and expansion modules to expand"

Kris Eberlein, 25 August 2022

No, because you would need a specialization in order to expand the content model or attribute list.

Disposition: Closed

Comment by Eliot Kimber
I think I agree with Dawn: I think expansion is an option that you might otherwise think required specialization, in particular specializing in order to allow two new element types specialized from a repeatable base type. That can now be formally done using an expansion module and a domain specialization (so I guess it still requires specialization, but not specialization simply to expand the structural content model).

Kris Eberlein, 31 August 2022

Yes, that's the fine point here. But I don't think that information belongs in this topic. If it's not adequately covered in the expansion topics, then we need to address that there. (We certainly cover this particular use case in "Example: Adding an existing domain attribute to certain elements."

Robert Anderson 31 August 2022

I think I agree that it does not exactly belong here. With or without expansion, domain modules can be used to expand the content of an element X without the need for specialization of that element X. In both cases, the specialization is done elsewhere, but I think this has always been the case - domains are there so you can expand the content model without speclizing the base, it's just that "expansion modules" give you a bit more control over that now.

Kris Eberlein, 01 September

From my point of view, expansion **requires** a specialization. It can be a brand new element specialization, a new attribute specialization, or an existing specialization not currently used in a company's grammar files (such as a domain specialization). I don't think we do our spec readers/users any benefit by obscuring this point!

It might well be that we need to state that clearly in the expansion topics, as several reviewers have tripped over this. Marking this as "accepted" to check on just that.

Disposition: Accepted

Comment by Zoë Lawson on 27 August 2022
I'm not quite sure this fits here, but do we want to be explicit somewhere that specializing part of a 'base' element can have unintended consequences? If you change what attribute values are allowed, that may be inherited strangely? I can't remember a specific example, but some of the weird "allowed ph here and now all elements based on it now have all the relatives of ph as well"

This is probably just a standard XML idiosyncrasy that I was flummoxed by and doesn't belong in the spec

---

Kris Eberlein, 31 August 2022

It's not a standard XML idiosyncrasy, but a function of how element-specialization domains work in DITA. Do we cover that clearly in this chapter of the spec?

Robert Anderson 31 August 2022

I think that should be made clear (if it's not already) in the sections that talk about element domain modules. The "I specialized a keyword and now it shows up in weird places" issue is explicitly tied to element domain modules, and trying to explain it here in the overview topic would be too much detail over one specific aspect of specialization.

Disposition: Accepted

1.3.2 Modularization

Modularization is at the core of DITA design and implementation. It enables reuse and extension of the DITA specialization hierarchy.

The DITA XML grammar files are a set of module files that declare the markup and entities that are required for each specialization. The document-type shell then integrates the modules that are needed for a particular authoring and publishing context.

Comment by Eliot Kimber
c/The document-type shell/A document type shell" (I don't think we normally hyphenate document type in "document type shell").

Robert D Anderson, 29 August 2022: we do hyphenate it pretty consistently (just checked my local clone of the spec and found 200+ hyphenations vs 19 not hyphenated. We've tried to consistently hyphenate when there are 2-word modifiers for the noun.

Kris Eberlein, 31 August 2022

Our style guide (and best practices for writing content for localization) call for avoiding noun strings. Most of the time we have the phrase "document-type" in a long noun string. DITA 1.3 reviewers found it difficult to make that fine distinction, so we decide to hyphenate "document-type" whenever it is a modifier.

We do not hyphenate "document type" or "document types" when the term is NOT used as a modifier.

Disposition: Rejected

Comment by Stan Doherty
Ouch. "easy"???

Response by Kris Eberlein, 23 August 2020:

How about changing "easy" to "simplified"?

Response by Gershon Joseph, 2022-08-24:

I agree with Kris's suggested change.

Kris Eberlein, 24 August 2022

Done

Disposition: Completed

Because all the pieces are modular, the task of developing a new information type or domain is easy. An information architect can start with existing base types (topic or map)—or with an existing specialization if it comes close to matching their business requirements—and only develop an extension that adds the extra semantics or functionality that is required. A specialization reuses elements from ancestor modules, but it only needs to declare the elements and attributes that are unique to the specialization. This saves considerable time and effort; it also reduces error, enforces consistency, and makes interoperability possible.

Comment by Gershon Joseph on 2022-08-24
Change the last sentence to: This saves considerable time and effort, reduces error, enforces consistency, and makes interoperability possible.

I deleted "; it also".

Kris Eberlein, 24 August 2022

Done

Disposition: Completed

Because all the pieces are modular, it is easy to reuse different modules in different contexts.

Comment by dstevens73
Here's another instance of "easy" – are we trying to remove that opinion and use a different word as above?

Kris Eberlein, 25 August 2022

Changed "easy" to "simple". (Does this really make a difference?)

Disposition: Completed

For example, a company that produces machines can use the hazard statement domain, while a company that produces software can use the software, user interface, and programming domains. A company that produces health information for consumers can avoid using the standard domains; instead,

Comment by Gershon Joseph on 2022-08-24
replace "standard domains; instead, it develops " with "standard domains. Instead, it develops ".

Kris Eberlein, 24 August 2022

Done

Disposition: Completed

it develops a new domain that contains the elements necessary for capturing and tracking the comments made by medical professionals who review information for accuracy and completeness.

Because all the pieces are modular, new modules can be created and put into use without affecting existing document-type shells.

For example, a marketing division of a company can develop a new specialization for message campaigns and have their content authors begin using that specialization, without affecting any of the other information types that they have in place.

Comment by Gershon Joseph on 2022-08-24
I don't see the value in this second example. It does not add anything we have not already said. I think we should delete it.

Kris Eberlein, 24 August 2022

It's intended to illustrate the preceding sentence.

Disposition: Rejected

1.3.3 Vocabulary modules

A DITA element type or attribute is declared in exactly one vocabulary module.

Comment by Zoë Lawson on 27 August 2022
why is this "element type"? Why isn't it just "element"? If it's supposed to be element type, why isn't it attribute type? If element type is correct, why isn't it element-type (like document-type)?

Again, if this is my XML ignorance, please ignore.

Eliot, Kimber 29 August 2022

Every XML element instance has an "element type", which may be declared in a grammar (RNG, DTD, etc.). The element type is indicated by the element's element type name (i.e, "p" for a `<p>` element).

The term "element" formally means "element instance" while "element type" means "The defined type of an element".

Informally we usually use "element" to mean either element type or element instance when it's clear from context which we mean.

Here we must be precise.

https://www.w3.org/TR/xml/#elemdecls, https://www.w3.org/TR/xml/#dt-element

Disposition: Closed

The following terminology is used to refer to DITA vocabulary modules:

structural module
: A vocabulary module that defines a top-level map or topic type.

element domain module
: A vocabulary module that defines one or more specialized element types that can be integrated into maps or topics.

attribute domain module
: A vocabulary module that defines exactly one specialization of either the `@base` or `@props` attribute.

For structural types, the module name is typically the same as the root element. For example, "task" is the name of the structural vocabulary module whose root element is `<task>`.

For element domain modules, the module name is typically a name that reflects the subject domain to which the domain applies, such as "highlight" or "software". Domain modules often have an associated short name, such as "hi-d" for the highlighting domain or "sw-d" for the software domain.

The name (or short name) of an element domain module is used to identify the module in `@class` attribute values. While module names need not be globally unique, module names must be unique within the scope of a given specialization hierarchy. The short name must be **a valid XML name token.**

Comment by Zoë Lawson on 27 August 2022
Are the above paragraphs "best practices"? The information definitely should be here, but I wonder if it should be highlighted as a best practice.

Kris Eberlein, 31 August 2022

The information about module names and associated short names are best practices. Other content is normative rules or just the way things have to be in order to work.

If we take all the best practices information out (and place it elsewhere) we run into difficulties, I think. The content becomes less useful, and some one trying to understand will have to flip back-and-forth between the body of the spec and the non-normative info.

That said, we do have a file-namings convention topic in the (current) appendix C. @Robert, @Eliot – Do we want remove file-naming conventions there? By using the phrase "is typically" here, I think we clearly indicate that they are conventions.

Robert Anderson, 31 August 2022

I think it's best to keep this info here. The "module names must be unique within the scope" part definitely needs to stay here, the rest is informative, but would be less useful if it appeared on its own somewhere else. You'd have to recreate a lot of the info here to even establish what you're talking about with the typical practice.

Disposition: Referred

005 (69)

> Structural modules based on topic **MAY** define additional topic types that are then allowed to occur as subordinate topics within the top-level topic. However, such subordinate topic types **MAY NOT** be used as the root elements of conforming DITA documents.

For example, a top-level topic type might require the use of subordinate topic types that would only ever be meaningful in the context of their containing type and thus would never be candidates for standalone authoring or aggregation using maps. In that case, the subordinate topic type can be declared in the module for the top-level topic type that uses it. However, in most cases, potential subordinate topics are best defined in their own vocabulary modules.

006 (69)

> Domain elements intended for use in topics **MUST** ultimately be specialized from elements that are defined in the topic module. Domain elements intended for use in maps **MUST** ultimately be specialized from elements defined by or used in the map module. Maps share some element types with topics but no map-specific elements can be used within topics.

Structural modules also can define specializations of, or reuse elements from, domain or other structural modules. When this happens, the structural module becomes dependent.

Comment by Gershon Joseph on 2022-08-24
The last sentence "When this happens..." is obvious. I think it should be deleted.

Zoe Lawson, 27 August 2022

The structural module becomes dependent on what? In my ignorance, what it becomes dependent on is lost to me, even though it might be obvious to others.

Eliot Kimber, 29 August 2022

Becomes dependent on the domain module, meaning that you cannot integrate the structural module without also integrating the domain module (if the structural domain uses element types defined in the domain module).

As this dependency is only formally reflected in the now-eliminated `@domains` attribute I'm sure the last sentence is necessary or useful.

If a structural module specializes from domain elements but does not use them directly there is no requirement to also integrate the domain module and if the structural module use element types defined in the domain module then the domain module must be integrated for instances would be valid, so I think it's not necessary to say anything.

Kris Eberlein, 31 August 2022

@Eliot, I think you are correct about element-specializiation domain modules, but not structural modules. Bookmap is dependent on map, and troubelshooting is dependent on task.

@Robert, do we want to make any changes to the language here?

Robert Anderson, 31 August 2022

I think it's an awkward end to the sentence (I think like Zoe, I expect to see another clause, "dependent on X"). I do think it's a little bit redundant overall, basically "When you use elements from X then you depend on X", but it's important to have the first sentence and I kind of feel like that first sentence, on its own, doesn't make a whole paragraph.

So I think my vague preference here would be to extend the sentence a bit and clarify that when the structural module reuses elements from another module, it depends on that module.

Eliot is right that if you are defining specializations of domain elements, but not using those domain elements, that dependency doesn't actually mean that you have to use the domain in your shell. But that idea makes me a bit uncomfortable, it's not something I've ever considered or run in to. If you have processing extensions for elements in the intermediate module, you would likely still be dependent on those, even if you could drop the grammar module from your shell.

Disposition: Referred

1.3.4 Specialization rules for element types

There are certain rules that apply to element type specializations.

A specialized element type has the following characteristics:

Comment by Kristen J Eberlein on 07 August 2022

Robert Anderson and I both think that the 2nd bullet point needs expansion, to cover the effects of expansion modules.

Comment by Gershon Joseph, 2022-08-24

Agreed.

Eliot Kimber, 29 August 2022

Maybe.

Note that because this section is about specialization, and not configuration, the statement is still true as stated even in the face of expansion.

I can't think of a better way to say this crisply without devolving into specific cases. The rule ultimately is "the specialized content module cannot allow anything that is not allowed by the base but may allow **or require** anything that is allowed by the base".

If the base allows "a*" and the specialization allows "(b|c)*" where b and c are specializations of a, it is still as constrained as the base, because the base allows "a*" and (b|c)* generalizes to "a*".

Likewise, the specialization could be a+, which is more constrained than the base.

Likewise, an expansion module could then allow d|e in place of or in addition to b and c. That extension does not change the degree of constraint of the specialization.

That is, "inclusive" in this context means "inclusive of base types or any specializations as allowed by repetition operators". Allowing additional specializations of an allowed base type is not more inclusive, it is exactly as inclusive for the purpose of determining correctness of specialized content models.

Disposition: Referred

- A properly-formed `@class` attribute that specifies the specialization hierarchy of the element
- A content model that is the same or less inclusive than that of the element from which it was specialized
- A set of attributes that are the same or a subset of those of the element from which it was specialized, except for specializations of `@base` or `@props`

Comment by dstevens73
Does expansion apply here? Next topic says I can expand attributes to a single element, why can't I do it here?

Kris Eberlein, 25 August 2022

You can expand the attribute list, but only with specializations of `@base` or `@props`, as stated here.

Disposition: Closed

Comment by Eliot Kimber

Because this section is just about specialization I don't think we need to mention expansion, but at the same time it might be worth a note about how expansion differs from specialization in this case.

In a specialization you can choose to entirely disallow attributes from the base type while with expansion and constraint you can choose add or remove attributes from a type as a matter of configuration.

Configuration is optional while specialization is irrevocable. If the specialization disallows attribute `@foo` it would be incorrect for an expansion module to subsequently allow it. If the specialization disallows `@base` for some reason it would again be incorrect for an expansion to then allow a specialization of `@base`.

Kris Eberlein, 31 August 2022

It's hard to figure out if where we would put this information (or whether it is better suited for a tutorial – or whether it is adequately covered in the expansion topics.

Setting the disposition to "Referred" because it is clear that many TC reviewers are struggling with the difference between specialization and expansion and configuration.

Disposition: Referred

- Values or value ranges of attributes that are the same or a subset of those of the element from which it was specialized

DITA elements are never in a namespace. Only the `@DITAArchVersion` attribute is in a DITA-defined namespace. All other attributes, except for those defined by the XML standard, are in no namespace.

This limitation is imposed by the details of the `@class` attribute syntax, which makes it impractical to have namespace-qualified names for either vocabulary modules or individual element types or attributes. Elements included as descendants of the DITA `<foreign>` element type can be in any namespace.

Note Domain modules that are intended for wide use should define element type names that are unlikely to conflict with names used in other domains, for example, by using a domain-specific prefix on all names.

Comment by Zoë Lawson on 27 August 2022
I find this namespace information very important. However, because its in paragraphs after a list, my brain semi-dismissed it as not as important as the bullets. Should the first sentenct (DITA elements are never in a namespace.) be a bullet so it stands out? Or is it worth making a specific "DITA Specializations and Namespaces" section?

Kris Eberlein, 31 August 2022

I'm not sure how we want to handle this. We don't use sections in topics (except element-reference topics), so that's not an option. The name space information could be moved into a separate topic, but it does really belong here.

I've used a definition list, with entries for "Characteristics" and "Namespaces".

Disposition: Completed

1.3.5 Specialization rules for attributes

There are certain rules that apply to attribute specializations.

A specialized attribute has the following characteristics:

- It is specialized from `@props` or `@base`.
- It can be integrated into a document-type shell either globally, which makes it available on all elements, or it can be assigned to specific elements by using an expansion module.
- It does not have values or value ranges that are more extensive than those of the attribute from which it was specialized.

Comment by Zoë Lawson on 27 August 2022
My ignorance is showing again. I read this bullet and it says to me if I want to say, add a new value to @type for note elements because I'm tired of using other and @othertype, I can't.

If that is correct, but I can use an expansion to solve that problem, can we state that?

I realize this is probably a request for user guide info, but I wonder if an example of to solve problem x, use a constraint (and why), to solve problem y, use a specialization (and why). would be useful. (or maybe that information can be added to existing examples?)

Eliot Kimber, 29 August 2022

You cannot add values to enumerations defined on a base type, so expansion does not provide a way around that limitation

Disposition: Closed

- Its values must be alphanumeric

Comment by Stan Doherty
Add comma between alphanumeric and space-delimited.

Response by Kris Eberlein, 23 August 2020:

Done

Disposition: Completed

space-delimited values.

Comment by Gershon Joseph on 2022-08-28
Move the sentence "In generalized form..." into a new list item. It's got nothing to do with the characters that make up the attribute value.

Response by Kris Eberlein, 3 August 2020:

Done

Disposition: Completed

In generalized form, the values must conform to the rules for attribute generalization.

1.3.6 @ class attribute rules and syntax

The specialization hierarchy of each DITA element is declared as the value of the `@class` attribute. The `@class` attribute provides a mapping from the current name of the element to its more general equivalents, but it also can provide a mapping from the current name to more specialized equivalents.

Comment by Gershon Joseph on 2022-08-28
Break the previous sentence into two sentences, as follows:

The@class attribute provides a mapping from the current name of the element to its more general equivalents. The @class attribute can also provide a mapping from the current name to more specialized equivalents.

Rationale: the word "but" in the old version does not really fit. I think it's clearer as two separate sentences.

Kris Eberlein, 31 August

Done, as I agree with you here. I suspect, however, that some of this content will need to be removed from the short description.

Disposition: Completed

All specialization-aware processing can be defined in terms of `@class` attribute values.

The `@class` attribute tells a processor what general classes of elements the current element belongs to. DITA scopes elements by module type (for example topic type, domain type, or map type) instead of document type, which lets document type developers combine multiple module types in a single document without complicating transformation logic.

Comment by Gershon Joseph on 2022-08-28
Remove the parenthetical content from the previous sentence. Then add it back in this location as a separate sentence, as follows: Examples of module types include topic type, domain type, and map type.

Rationale: the parenthetical content makes it very difficualt for non-native English speaker to parse the sentence and understand it.

Kris Eberlein, 31 August 2021

Changed to read as follows:

The @class attribute tells a processor what general classes of elements the current element belongs to. DITA scopes elements by module type instead of document type. Examples of module types are topic type, domain type, or map type. This enables document-type developers to combine multiple module types in a single document without complicating transformation logic.

Disposition: Completed

The sequence of values in the `@class` attribute is important because it tells processors which value is the most general and which is most specific. This sequence is what enables both specialization aware processing and generalization.

Syntax

Values for the `@class` attribute have the following syntax requirements:

- An initial "-" or "+" character followed by one or more spaces. Use "-" for element types that are defined in structural vocabulary modules, and use "+" for element types that are defined in domain modules.
- A sequence of one or more tokens of the form "*modulename*/*typename*", with each token separated by one or more spaces, where *modulename* is the short name of the vocabulary module and *typename* is the element type name. Tokens are ordered left to right from most general to most specialized.

  These tokens provide a mapping for every structural type or domain in the ancestry of the specialized element. The specialization hierarchy for a given element type must reflect any intermediate modules between the base type and the specialization type, even those in which no element renaming occurs.
- At least one trailing space character (" "). The trailing space ensures that string matches on the tokens can always include a leading and trailing space in order to reliably match full tokens.

Rules

007 (69) Every DITA element (except the `<dita>` element that is used as the root of a ditabase document) **MUST** declare a `@class` attribute.

Comment by Eliot Kimber
I think "exhibit" is more correct than "declare" here in that "declare" implies "declared in a grammar", which is not required. But I'm not sure any reader would understand what "exhibit" means so I'm OK with keeping "declare".

Kris Eberlein, 31 August 2022

I agree with both your points. Marking this comment as "Closed."

Disposition: Closed

008 (69) When the `@class` attribute is declared in an XML grammar, it **MUST** be declared with a default value. In order to support generalization round-tripping (generalizing specialized content into a generic form and then returning it to the specialized form) the default value **MUST NOT** be fixed. This allows a generalization process to overwrite the default values that are defined by a general document type with specialized values taken from the document being generalized.

009 (69) A vocabulary module **MUST NOT** change the `@class` attribute for elements that it does not specialize, but simply reuses by reference from more generic levels.

010 (70) Authors **SHOULD NOT** modify the `@class` attribute.

Example: DTD declaration for @class attribute for the <step> element

The following code sample lists the DTD declaration for the `@class` attribute for the `<step>` element:

```
<!ATTLIST step         class  CDATA "- topic/li task/step ">
```

This indicates that the `<step>` element is specialized from the `<li>` element in a generic topic. It also indicates explicitly that the `<step>` element is available in a task topic;

Comment by Eliot Kimber
I would say " element from the topic module" rather than "in a generic topic".

Kris Eberlein, 31 August 2022

Done.

Disposition: Completed

Comment by Gershon Joseph on 2022-08-28
Break into two sentences at this point.

Kris Eberlein, 31 August 2022

Done.

Disposition: Completed

this enables round-trip migration between upper level and lower level types without the loss of information.

Example: Element with @class attribute made explicit

The following code sample shows the value of the `@class` attribute for the `<wintitle>` element:

```
<wintitle class="+ topic/keyword ui-d/wintitle ">A specialized keyword</wintitle>
```

The `@class` attribute and its value is generally not surfaced in authored DITA topics, although it might be made explicit as part of a processing operation.

Comment by Gershon Joseph on 2022-08-28
I don't think this sentence belongs here in an example. It should be in the syntax section above or some other location that makes sense. Examples should provide examples of what's said above. They should not add new information not mentioned in the main sections that precede the examples.

Kris Eberlein, 31 August 2022

I agree with you. I've moved this content into the paragraph that states "Authors SHOULD NOT modify the @class attribute."

Disposition: Completed

Example: @class attribute with intermediate value

The following code sample shows the value of a `@class` attribute for an element in the guiTask module, which is specialized from `<task>`. The element is specialized from `<keyword>` in the base topic vocabulary, rather than from an element in the task module:

```
<windowName class="- topic/keyword task/keyword guiTask/
```

```
windowname ">...</windowName>
```

The intermediate values are necessary so that generalizing and specializing transformations can map the values simply and accurately. For example, if `task/keyword`

Comment by Stan Doherty
Change "was" to "were".

Kris Eberlein, 31 October 2022

I'm leaving this as "was". I think this is an ambiguous area, but I'm going to interpret this as the phrase `task/keyword`, rather than as the phrases `task/keyword`.

Disposition: Rejected

was missing as a value, and a user decided to generalize this guiTask up to a task topic, then the transformation would have to guess whether to map to keyword (appropriate if task is more general than guiTask, which it is) or leave it as windowName (appropriate if task were more specialized, which it isn't). By always providing mappings for more general values, processors can then apply the simple rule that missing mappings must by default be to more specialized values than the one we are generalizing to, which means the last value in the list is appropriate. For example, when generalizing `<guitask>` to `<task>`, if a `<p>` element has no target value for `<task>`, we can safely assume that `<p>` does not specialize from `<task>` and does not need to be generalized.

Comment by dstevens73
I would want to revisit this when generalization is out for review, because right now, I'm not understanding it.

Kris Eberlein, 01 September 2022

@Dawn. I agree that this example should be reviewed as part of the generalization content. In addition, the example is not well written, nor is it immediately clear to me exactly what we are trying to say in this part of the topic.

I'm adding a draft comment that we need to review this example as part of the generalization content.

Disposition: Accepted

Comment by Zoë Lawson on 27 August 2022
I am confused by a lot of things in this entire review, but one specific question is how a processor determines if a thing is more general or more specialized.

Eliot Kimber, 29 August 2022

Degree of specialization is determined by where an element's name occurs in the @class value--further to the right means more specialized.

Disposition: Closed

1.3.7 @specializations attribute rules and syntax

The `@specializations` attribute enables processors to determine what attribute specializations are available in a document. The attribute is declared on the root element for each topic or map type. Each attribute domain defines a token to declare the extension

Comment by Gershon Joseph on 2022-08-28
Break the sentence up into two sentences.

Kris Eberlein, 31 August 2022

Done.

Disposition: Completed

; the effective value of the `@specializations` attribute is composed of these tokens.

Syntax and rules

The `@props` and `@base` attributes are the only two core attributes available for specialization.

011 (70) Each specialization of the `@props` and `@base` attributes **MUST** provide a token for use by the `@specializations` attribute.

The `@specializations` token for an attribute specialization begins with either `@props` or `@base` followed by a slash, followed by the name of the new attribute:

```
'@', props-or-base, ('/', attname)+
```

For example:

- If `@props` is specialized to create `@myNewProp`, this results in the following token: `@props/myNewProp`
- If `@base` is specialized to create `@myFirstBase`, this results in the following token: `@base/myFirstBase`

- If that specialized attribute `@myFirstBase` is further specialized to create `@mySecondBase`, this results in the following token: `@base/myFirstBase/mySecondBase`

Example: @specializations attribute for a task with multiple domains

In this example, a document-type shell integrates the task structural module and the following domain modules:

Domain	Domain short name
User interface	ui-d
Software	sw-d
`@deliveryTarget` attribute	deliveryTarget
`@platform` attribute	platform
`@product` attribute	product

The value of the `@specializations` attribute includes one value from each attribute module

Comment by Gershon Joseph on 2022-08-28
Break into two sentences at the semicolon.

Kris Eberlein, 31 August 2022

Done.

Disposition: Completed

; the effective value is the following:

```
specializations="@props/deliveryTarget @props/platform @props/product"
```

If the document-type shell also used a specialization of the `@platform` attribute that describes the hardware platform, the new `@hardwarePlatform` attribute domain would add an additional value to the `@specializations` attribute:

```
specializations="@props/deliveryTarget @props/platform @props/platform/hardwarePlatform
@props/product"
```

Note that the value for the `@specializations` attribute is not authored. Instead, the value is defaulted based on the modules that are included in the document type shell.

Comment by Bill Burns
Change "the value is defaulted based on the modules" to "the value is added by default to the modules...."

Kris Eberlein, 24 August 2022

I think that's missing a critical nuance. The value of the specializations attribute is not added to the modules. Grammar-aware processors will construct a value for the attribute based on the modules that are included in the document-type shell.

Disposition: Rejected

Comment by Zoë Lawson on 27 August 2022

I think this last paragraph is super important and maybe should be part of the short description. Knowing that @specializations shouldn't be authored up front would have helped me undestand the topic better. Maybe also that this is more important to processors that someone making a specialization? (Again, I'm probably too ignorant.)

Gershon Joseph, 28 August 2022

I was going to suggest we move this paragraph up into the spec text before the example. It definitely is not part of the example.

Kris Eberlein, 31 August 2022

Agree that this should not be in the example, but not sure if the short description is the correct place.

I've moved this paragraph to the end of the "Suntax and rules" section.

Disposition: Completed

Comment by Zoë Lawson on 27 August 2022
May want to do a search and replace at some point to make sure document-type is always hyphenated (or not).

Kris Eberlein, 31 August 2022

Agree, and I've done this (and it was painful). We hyphenate document-type when it is used as a modifier; we do not hyphenate the two words when they are a noun.

Disposition: Completed

1.3.8 Specializing to include non-DITA content

You can extend DITA to incorporate standard vocabularies for non-textual content, such as MathML and SVG, as markup within DITA documents. This is done by specializing the `<foreign>` or `<unknown>` elements.

There are three methods of incorporating foreign content into DITA.

Comment by Zoë Lawson on 27 August 2022
Do we need to define "foreign content"? Especially since "foreign" seems to mean something specific in RNG?

Robert Anderson, 31 August 2022

I think I'm too close to "foreign" content here, because I read that and just think "non-DITA XML". I'm not sure if we need to explain further (maybe add that as a parenthetical).

Kris Eberlein, 31 August 2022

OK, let's discuss this at the TC.

Disposition: Referred

- A domain specialization of the `<foreign>` or `<unknown>` element. This is the usual implementation.
- A structural specialization using the `<foreign>` or `<unknown>` element. This affords more control over the content

> **Comment by Gershon Joseph on 2022-08-28**
> Do we mean "This affords more control over the content **model**? If yes, then add the word "model". If not, then I don't understand what's being said here.
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> Yes, this should have been "content model." I've corrected it.
>
> **Disposition: Completed**

.

- Directly embedding the non-DITA content within `<foreign>` or `<unknown>` elements. If the non-DITA content has interoperability or vocabulary naming issues such as those that are addressed by specialization in DITA, they must be addressed by means that are appropriate to the non-DITA content.

Do not use `<foreign>` or `<unknown>` elements to include textual content or metadata in DITA documents, except where such content acts as an example or display, rather than as the primary content of a topic.

> **Comment by Zoë Lawson on 27 August 2022**
> I'm not following the previous paragraph, or why it's important.
>
> ---
>
> Gershon Joseph, 28 August 2022
>
> I too had to read it several times before I understood what's trying to be said. I think the best solution is to delete ", except where such content... of a topic." In other words, remove the exception. DITA has better elements to use for example and display anyway. The spec does not need to mention this exception.
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> Done.
>
> **Disposition: Completed**

Example: Creating an element domain specialization for SVG

The following code sample, which is from the `svgDomain.ent` file, shows the domain declaration for the SVG domain.

```
<!-- ============================================================= -->
<!--                    SVG DOMAIN ENTITIES                        -->
<!-- ============================================================= -->

<!-- SVG elements must be prefixed, otherwise they conflict with
     existing DITA elements (e.g., <desc> and <title>.
  -->
<!ENTITY % NS.prefixed "INCLUDE" >
<!ENTITY % SVG.prefix "svg" >

<!ENTITY % svg-d-foreign
   "svg-container
   "
>
```

> **Comment by Gershon Joseph on 2022-08-28**

I think all examples and instructions like this one should be in relax-ng format, since that's our normative format. Perhaps DTD-specific instructions and examples should be moved out into a committee note? I think the spec will be easier to read and understand if it only refers to relax-ng.

Kris Eberlein, 31 August 2022

I've mentioned previously why we cannot do this. We support both DTD and RNG. We've been moving towards ensuring that we have examples in both DTD and RNG, but we probably won't accomplish that for DITA 2.0.

Disposition: Rejected

Note that the SVG-specific `%SVG.prefix;` parameter entity is declared. This establishes the default namespace prefix to be used for the SVG content embedded with this domain. The namespace can be overridden in a document-type shell by declaring the parameter entity before the reference to the `svgDomain.ent` file. Other foreign domains might need similar entities when required by the new vocabulary.

Comment by Gershon Joseph on 2022-08-28
I don't see the point of this example. Just refer the reader to the SVG domain's mod file in the DITA Technical Content. Nothing said here is of any relevance, or help, to the topic at hand. This information belongs in a CN that that's a tutorial on using foreign or unknown vocabularies with DITA.

Kris Eberlein, 31 August 2022

It's very unlikely that we will ever issue a tutorial about using foreign or unknown volcabularies with DITA. Given that, do we do the readers of the spec a better experience by retaining or removing this information?

Robert Anderson, 31 August 2022

Agree - realistically, there is a near-zero chance of a committee note on using foreign/unknown content, so if we remove it from here it's gone. Just referring to the TC package will not work, as that package is likely to come out after the base package, so it will not actually be available to reference (certainly the case for whenever this content goes out for public review). I think it's best to retain this, as it's been requested in past versions of the spec. (I see we do reference the TC package in the next paragraph, but I think that works better as a "for more info" reference rather than simply directing the reader.)

Disposition: Rejected

For more information, see the `svgDomain.mod` file that is shipped with the DITA Technical Content edition. For an example of including the SVG domain in a document-type shell, see `task.dtd`.

Comment by Zoë Lawson on 27 August 2022
Confirming okay to reference this since it's in the technical content bit.

Kris Eberlein, 31 August 2022

Yes, it's OK to reference stuff in the technical content package, if the base package does not include something relevant that could be referenced instead.

Disposition: Closed

1.3.9 Sharing elements across specializations

Specialization enables

Comment by Stan Doherty
Ouch. "easy" is still a stretch.

Response by Kris Eberlein, 23 August 2020:

Removed the word "easy".

Disposition: Completed

easy reuse of elements from ancestor specializations. However, it is also possible to reuse elements from non-ancestor specializations, as long as the dependency is properly declared in order to prevent invalid generalization or conref processing.

Comment by Eliot Kimber
I think the "properly declared" is no longer relevant as the declaration was on the `@domains` attribute, so there's no place to formally declare the dependencies among specialization modules, which is fine.

So I think we can delete ", as long as ….".

Response by Kris Eberlein, 23 August 2020:

Done.

Disposition: Completed

A structural specialization can incorporate elements from unrelated domains or other structural specializations by referencing them in the content model of a specialized element. The elements included in this manner must be specialized from ancestor content that is valid in the new context. If the reusing and reused specializations share common ancestry, the reused elements must be valid in the reusing context at every level they share in common.

Although a well-designed structural specialization hierarchy with controlled use of domains is still the primary means of sharing and reusing elements in DITA, the ability to also share elements declared elsewhere in the hierarchy allows for situations where relevant markup comes from multiple sources and would otherwise be developed redundantly.

Example: A specialization of <concept> reuses an element from the task module

A specialized concept topic could declare a specialized `<process>` section that contains the `<steps>` element that is defined in the task module. This is possible because of the following factors:

- The `<steps>` element is specialized from `<ol>`.
- The `<process>` element is specialized from `<section>`, and the content model of `<section>` includes `<ol>`.

The `<steps>` element in `<process>` always can be generalized back to `<ol>` in `<section>`.

Example: A specialization of <reference> reuses an element from the programming domain

A specialized reference topic could declare a specialized list (`<apilist>`) in which each `<apilistitem>` contains an `<apiname>` element that is borrowed from the programming domain.

Comment by Zoë Lawson
The previous example walked through and explained how things worked. Is it possible to explain more here? The only difference between the example and the title is the name of the elements. I'm not following how to do it, if there's anything funky I need to do, etc.

Eliot Kimber, 29 August 2022

The difference in the two examples is one involves two structural modules.

Disposition: Closed

1.4 Constraints

Constraint modules define additional constraints for vocabulary modules in order to restrict content models or attribute lists for specific element types, remove certain extension elements from an integrated domain module, or replace base element types with domain-provided, extension element types.

Comment by Bill Burns
Tautological - perhaps "constraint modules define additional *restrictions* for vocabulary modules"?

Kris Eberlein, 26 August 2022

Removed "define additional constraints for vocabulary modules in order to"; it now reads "Constraint modules restrict content modules …".

Disposition: Completed

1.4.1 Overview of constraints

Constraint modules enable information architects to restrict the content models or attributes of DITA elements. A constraint is a simplification of an XML grammar such that any instance that conforms to the constrained grammar also will conform to the original grammar.

A constraint module can perform the following functions:

Comment by Gershon Joseph on 2022-08-29
I wonder is an RNG example for each of these functions would help?

Kris Eberlein, 31 August 2022

Our current examples (both DTD and RNG) cover each of these use cases.

Disposition: Closed

Restrict the content model for an element

Constraint modules can modify content models by removing optional elements, making optional elements required, or requiring unordered elements to occur in a specific sequence. Constraint modules cannot make required elements optional or change the order of element occurrence for ordered elements.

For example, a constraint for `<topic>` can require `<shortdesc>`, can remove `<abstract>`, and can require that the first child of `<body>` be `<p>`. A constraint cannot allow `<shortdesc>` to follow `<prolog>`, because the content model for `<topic>` requires that `<shortdesc>` precedes `<prolog>`.

Restrict the attributes that are available on an element

Constraint modules can restrict the attributes that are available on an element. They also can limit the set of permissible values for an attribute.

For example, a constraint for `<note>` can limit the set of allowed values for the `@type` attribute to "note" and "tip". It also can omit the `@othertype` attribute, since it is needed only when the value of the `@type` attribute is "other".

Restrict the elements that are available in a domain

Constraint modules can restrict the set of extension elements that are provided in a domain. They also can restrict the content models for the extension elements.

For example, a constraint on the programming domain can reduce the list of included extension elements to `<codeph>` and `<codeblock>`.

Note For DITA implementations that use RNG-based grammar file

> **Comment by Gershon Joseph on 2022-08-29**
> change "file" to "files"
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> Done.
>
> **Disposition: Completed**

, restricting the set of extension elements that are provided in a domain can be handled simply by document-type configuration.

Replace base elements with domain extensions

Constraint modules can replace base element types with the domain-provided extension elements.

For example, a constraint module can replace the `<ph>` element with the domain-provided elements, making `<ph>` unavailable.

1.4.2 Constraint rules

There are certain rules that apply to the design and implementation of constraints.

Content model

> **Comment by Gershon Joseph on 2022-08-29**
> I wonder is an RNG example for each of these rules would help? (To reduce confusion, Please don't put DTD examples here, they can go in an appendix or CN. This comment applies to each rule in this topic. I'm happy to develop examples for these rules.
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> Every example in this topic is written in plain text, so it does not introduce any grammar-specific markup. I'm pretty sure that our current examples covers all of the use cases mentioned here.
>
> **Disposition: Rejected**

The content model for a constrained element must be at least as restrictive as the unconstrained content model for the element.

Domain constraints

When a domain module is integrated into a document-type shell, the base domain element can be omitted from the domain extension group or parameter entity. In such a case, there is no separate constraint declaration, because the content model is configured directly in the document-type shell.

A domain module can be constrained by only one constraint module. This means that all restrictions for the extension elements that are defined in the domain must be contained within that one constraint module.

Structural constraints

Each constraint module can constrain elements from only one vocabulary module. For example, a single constraint module that constrains `<refsyn>` from `reference.mod` and constrains `<context>` from `task.mod` is not allowed. This rule maintains granularity of reuse at the module level.

Constraint modules that restrict different elements from within the same vocabulary module can be combined with one another. Such combinations of constraints on a single vocabulary module have no meaningful order or precedence.

Aggregation of constraint modules

The content model of an element can be modified by either of the following element-configuration modules:

- Constraint module
- Expansion module

The content model of an element only can be modified by a single element-type configuration module. If multiple constraints or extensions need to be applied to a single element, the element configurations must be combined into a single module that reflects all the constraints and expansions that were defined in the original separate modules.

Comment by Bill Burns
Given that this content is repeated in the 1.5.2 Expansion module rules (51) topic, should it be a conref or conkeyref?

Kris Eberlein, 31 August 2022

Yes, if the content remains the same in both places. Often when content is in flux during development, I hesitate to implement a reuse mechanism.

Disposition: Accepted

Comment by Gershon Joseph on 2022-08-29
Or should we simply delete it from here? This topic is about constraints, not expansions. Maybe we need a new topic after the expansion topic that describes what to do if you need to both constrain and expand elements in an element-type configuration module. In any case, I had to read that last paragraph several times to understand what's being said... It's confusing...

Kris Eberlein, 31 August 2022

Part of the difficulty is that, under the covers, constraint and expansion are really the same thing – element configuration. We cover them separately here in the "Configuration" chapter, but treat them just as "Element configuration" in the "Coding practices" content.

I don't remember if you were on the TC when we made decisions about this.

Robert Anderson, 31 August 2022

I don't think we should delete this, as it can be easily overlooked. It's true if you're using 2 constraints, 2 expansions, or a combination of constraint and expansion.

It is really just sort of an artifact of how XML grammars work, that you can only define the literal content model once. So in that sense, perhaps it does not need to be stated. But given the detail we go into about creating these modules, and what is allowed, I think it's a useful thing to highlight.

Kris Eberlein, 01 September 2022

@Gershon, what did you find dificult to understand in the content? I did do a read through, and I think that it would be clearer if the 2nd paragraph started with "For any document-type shell, the content model of an element … I've made that change to the source topic.

I do think strongly that this content needs to be in both of the rules topics for constraint and expansion.

Disposition: Completed

1.4.3 Constraints, processing, and interoperability

Because constraints can make optional elements required, documents that use the same vocabulary modules might have incompatible constraints. Thus the use of constraints can affect the ability for content from one topic or map to be used in another topic or map.

A constraint does not change basic or inherited element semantics. The constrained instances remain valid instances of the unconstrained element type, and the element type retains the same semantics and `@class` attribute declaration. Thus, a constraint never creates a new case to which content processing might need to react.

For example, a document type constrained to require the `<shortdesc>` element allows a subset of the possible instances of the unconstrained document type with an optional `<shortdesc>` element. Thus, the content processing for topic still works when `<topic>` is constrained to require a short description.

Comment by Gershon Joseph on 2022-08-29
Should we put examples into <example> elements? We have two consecutive "For example" paragraphs here, which seems odd.

Also use of "However, " twice does not flow well. Recast to avoid this...

Kris Eberlein, 31 August 2022

I've reworded some to remove the awkwardness. I did not add example sections as it would be difficult to assign workable titles.

Disposition: Completed

For example, an unconstrained task is compatible with an unconstrained topic, because the `<task>` element can be generalized to `<topic>`. However, if the topic is constrained to require the `<shortdesc>` element, a document type with an unconstrained task is not compatible with the constrained document type, because some instances of the task might not have a `<shortdesc>` element. However, if the task document type also has been constrained to require the `<shortdesc>` element, it is compatible with the constrained topic document type.

1.4.4 Examples: Constraints implemented using DTDs

This section of the specification contains examples of constraints implemented using DTD.

Comment by Gershon Joseph on 2022-08-29
I think we should only talk RNG in the spec and move all DTD stuff into a committee note. I'm happy to drive this effort...

If we do keep the DTD examples in the spec, we should always have the RNG examples first, not the DTD examples, since RNG is normative.

Kris Eberlein, 31 August 2022

Because the DITA specification supports both DTT and RNG, we must keep DTD content in the spec. As discussed at a recent TC call, we are considering moving coding practices, constraint examples, and expansion examples into an appendix. (This would be both DTD and RNG content.

We alphabetize content in the spec, do DTD will always come before RNG.

Disposition: Rejected

1.4.4.1 Example: Redefine the content model for the <topic> element using DTD

In this scenario, the DITA architect for Acme

Comment by Stan Doherty
Remove the comma.

Response by Kris Eberlein, 23 August 2020:

Done

Disposition: Completed

, Incorporated wants to redefine the content model for the topic document type. They want to omit the `<abstract>` element and make the `<shortdesc>` element required; they also want to omit the `<related-links>` element and disallow topic nesting.

Comment by Gershon Joseph on 2022-08-29
Why not put all four things to be done in an unordered list? The current two sentences separated by a semicolon is clumsy. I think the following content is clearer to understand:

In this scenario, the DITA architect for Acme, Incorporated wants to redefine the content model for the topic document type as follows:

- Remove the <abstract> element
- Require the <shortdesc> element
- Remove the <related-links> element
- Prevent topic nesting

Response by Kris Eberlein, 23 August 2020:

I'm pretty sure that the current markup reflects a desire to keep the content-model information in the short description. We could change that, of course, but then the shortdesc will simply repeat what is contained in the title.

Robert Anderson, 31 August 2022

My understanding/expectation around the short descriptions for example topics is that we want to present the full scenario. That way, the link preview in a parent topic for the examples will tell you exactly what it is about, so you know whether it's the one you are looking for.

Response by Kris Eberlein, 31 August 2020:

I think we can accomplish both things here: have a clear shortdesc and have details of the constraint presented in a list. I've done this for the following topics:

- This topic (both DTD and RNG examples)
- "Constrain a domain module" (both DTD and RNG version)

Disposition: Completed

1. The DITA architect creates a constraint module: `acme-TopicConstraint.mod`.
2. They add the following content to `acme-TopicConstraint.mod`:

```
<!-- ============================================================= -->
<!--                    CONSTRAINED TOPIC ENTITIES                 -->
<!-- ============================================================= -->

<!-- Declares the entities referenced in the constrained content  -->
<!-- model.                                                       -->

<!ENTITY % title            "title">
<!ENTITY % shortdesc        "shortdesc">
<!ENTITY % prolog           "prolog">
<!ENTITY % body             "body">

<!-- Defines the constrained content model for <topic>.           -->

<!ENTITY % topic.content
                       "((%title;),
                         (%shortdesc;),
                         (%prolog;)?,
                         (%body;)?)"
>
```

3. They add the constraint module to the `catalog.xml` file.
4. They then integrate the constraint module into the document-type shell for topic by adding the following section above the "TOPIC ELEMENT INTEGRATION"

   Comment by Gershon Joseph on 2022-08-29
   Rather than tell to to add content "above" a section, tell them to add it to the appropriate section: In the ELEMENT-TYPE CONFIGURATION INTEGRATION section of the DTD.

   Kris Eberlein, 31 August 2020:

   This was originally written as it was because existing 1.x DTD shells did not have an "ELEMENT-TYPE CONFIGURATION INTEGRATION" section.

   I've changed the list item.

   Disposition: Completed

   comment:

```
<!-- ============================================================= -->
<!--           ELEMENT-TYPE CONFIGURATION INTEGRATION              -->
<!-- ============================================================= -->

<!ENTITY % topic-constraints-c-def
```

```
  PUBLIC "-//ACME//ELEMENTS DITA Topic Constraint//EN"
  "acme-TopicConstraint.mod">
%topic-constraints-c-def;
```

5. After checking the test topic to ensure that the content model is modified as expected, the work is done.

Comment by Bill Burns
Seems this last step should be a step followed by a result. The first clause is a dangling modifier as well. Suggest rewriting as, "They test topics to ensure that the content model is modified as expected." Next clause is a result or stepresult.

Kris Eberlein, 31 August 2020:

This is a concept (not task), so the markup that you suggest is not available. If we move these topics to an appendix, we could change them to task topics, but that would be a low priority.

I did change the list item to read as follows:

"They check their test topic to ensure that the content model is modified as expected."

Disposition: Completed

1.4.4.2 Example: Constrain attributes for the <section> element using DTD

In this scenario, a DITA architect wants to redefine the attributes for the `<section>` element. They want to make the `@id` attribute required.

Comment by Gershon Joseph on 2022-08-29
I think these examples should be written using task topics and the steps worded in second person as task steps should be. It also would read better than the "they" statements.

Kris Eberlein, 31 August 2020:

If we move these examples (both DTD and RNG) to an appendix, we could do that. But I think it is a very low priority.

Setting the disposition to "Deferred."

Disposition: Deferred

1. The DITA architect creates a constraint module: `idRequiredSectionContraint.mod`.
2. They add the following content to `idRequiredSectionContraint.mod`:

```
<!-- Declares the entities referenced in the constrained content  -->
<!-- model.                                                       -->

<!ENTITY % localization-atts
              "translate
                          (no |
                           yes |
                           -dita-use-conref-target)
                                    #IMPLIED
               xml:lang
                          CDATA
                                    #IMPLIED
               dir
                          (lro |
                           ltr |
                           rlo |
                           rtl |
                           -dita-use-conref-target)
                                    #IMPLIED"
```

```
>
<!ENTITY % filter-atts
             "props
                        CDATA
                                    #IMPLIED
              %props-attribute-extensions;"
>
<!ENTITY % select-atts
             "%filter-atts;
              base
                        CDATA
                                    #IMPLIED
              %base-attribute-extensions;
              importance
                        (default |
                         deprecated |
                         high |
                         low |
                         normal |
                         obsolete |
                         optional |
                         recommended |
                         required |
                         urgent |
                         -dita-use-conref-target)
                                    #IMPLIED
              rev
                        CDATA
                                    #IMPLIED
              status
                        (changed |
                         deleted |
                         new |
                         unchanged |
                         -dita-use-conref-target)
                                    #IMPLIED"
>
<!ENTITY % conref-atts
             "conref
                        CDATA
                                    #IMPLIED
              conrefend
                        CDATA
                                    #IMPLIED
              conaction
                        (mark |
                         pushafter |
                         pushbefore |
                         pushreplace |
                         -dita-use-conref-target)
                                    #IMPLIED
              conkeyref
                        CDATA
                                    #IMPLIED"
>
<!-- Redefines the attributes available on section  -->

<!ENTITY % section.attributes
             "id
                        ID
                                    #REQUIRED
              %conref-atts;
              %select-atts;
              %localization-atts;
              outputclass
                        CDATA
                                    #IMPLIED"
>
```

Note

Comment by Gershon Joseph on 2022-08-29

I don't think this needs to be in a note. It's information about the example, explaining the code in the file.

Kris Eberlein, 31 August 2020:

Changed the markup to a `<p>`.

Disposition: Completed

The DITA architect had to declare all the parameter entities that are referenced in the redefined attributes for `<section>`. If they did not do so, none of the attributes that are declared in the parameter entities would be available on the `<section>` element. Furthermore, since the `%select-atts;` parameter entity references the `%filter-atts;` parameter entity, the `%filter-atts;` must be declared and it must precede the declaration for the `%select-atts;` parameter entity. The `%props-attribute-extensions;` and `%base-attribute-extensions;` parameter entities do not need to be declared in the constraint module, because they are declared in the document-type shells before the inclusion of the constraint module.

3. They add the constraint module to the `catalog.xml` file.
4. They then integrate the constraint module into the applicable document-type shells.

Comment by Zoë Lawson on 27 August 2022
Previous example showed the code. Should this one?

Kris Eberlein, 01 September 2020:

Done.

Disposition: Completed

5. After checking the test topics to ensure that the content model is modified as expected, the work is done.

Comment by Bill Burns
Same issue as previous topic.

Kris Eberlein, 31 August 2020:

Done.

Disposition: Completed

1.4.4.3 Example: Constrain a domain module using DTD

In this scenario, a DITA architect wants to use only a subset of the elements defined in the highlighting domain. They want to use `<b>` and `<i,>` but not `<line-through>`, `<overline>`, `<sup>`, `<sup>`, `<tt>`, or `<u>`. They want to integrate this constraint into the document-type shell for task.

Comment by Stan Doherty
In the first step of the following procedure, insert a space between "The" and "DITA".

Response by Kris Eberlein, 23 August 2020:

Done

Disposition: Completed

1. TheDITA architect creates a constraint module: `reducedHighlightingDomainConstraint.mod`.
2. They add the following content to `reducedHighlightingDomainConstraint.mod`:

```
<!-- ============================================================= -->
<!--      CONSTRAINED HIGHLIGHT DOMAIN ENTITIES                    -->
<!-- ============================================================= -->

<!ENTITY % HighlightingDomain-c-ph     "b | i"                >
```

3. They add the constraint module to the `catalog.xml` file.
4. They then integrate the constraint module into the company-specific, document-type shell for the task topic by adding the following section directly before the "DOMAIN ENTITY DECLARATIONS" comment:

> **Comment by Gershon Joseph on 2022-08-29**
> Delete "before the "DOMAIN ENTITY DECLARATIONS" comment" and replace with " in the DOMAIN CONSTRAINT INTEGRATION section"
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> Again, this content was developed for DITA 1.3, and we knew that most people were working with DITA 1.1 or 1.2 document-type shell, which did not have a "Domain Constraint Integration" section. And the constraint would NOT work if the domain constraint after integrated AFTER the "DOMAIN ENTITY DECLARATIONS" section.
>
> I've modified the list item.
>
> **Disposition: Completed**

```
<!-- ============================================================= -->
<!--                    DOMAIN CONSTRAINT INTEGRATION              -->
<!-- ============================================================= -->

<!ENTITY % HighlightDomain-c-dec
  PUBLIC "-//ACME//ENTITIES DITA Highlighting Domain Constraint//EN"
  "acme-HighlightDomainConstraint.mod"
>%HighlightDomain-c-dec;
```

5. In the "DOMAIN EXTENSIONS" section, they replace the parameter entity for the highlighting domain with the parameter entity for the constrained highlighting domain:

```
<!ENTITY % ph           "ph |
                         %HighlightDomain-c-ph; |
                         %sw-d-ph; |
                         %ui-d-ph;
                        ">
```

6. After checking the test topic to ensure that the content model is modified as expected, the work is done.

> **Comment by Gershon Joseph on 2022-08-29**
> Does it make sense for all these examples to be in a spec? Maybe even the RNG ones should be in a separate committee note or appendix?
>
> ---
>
> Kris Eberlein, 31 August 2022
>
> We've discussed this at a recent TC meeting. I generated a PDF with this content in an appendix, and sent it out for the TC to review.
>
> **Disposition: Closed**

1.4.4.4 Example: Replace a base element with the domain extensions using DTD

In this scenario, a DITA architect wants to remove the <ph> element but allow the extensions of <ph> that exist in the highlighting, programming, software, and user interface domains.

1. In the "DOMAIN EXTENSIONS" section, the DITA architect removes the reference to the <ph> element:

```
<!-- Removed "ph | " so as to make <ph> not available, only the domain extensions. -->
<!ENTITY % ph           "%pr-d-ph; |
                         %sw-d-ph; |
                         %ui-d-ph;
                        ">
```

Note Because no other entities are modified or declared outside of the usual "DOMAIN EXTENSIONS" section, this completes the architect's task. Because no new grammar file or entity is created that would highlight this change, adding a comment to highlight the constraint becomes particularly important (as shown in the example above).

Comment by Gershon Joseph on 2022-08-29
Change

particularly important (as shown in the example above).

to

particularly important, as shown in the example above.

Rationale: remove the parentheses.

Response by Kris Eberlein, 23 August 2020:

Done

1.4.4.5 Example: Apply multiple constraints to a single document-type shell using DTD

You can apply multiple constraints to a single document-type shell. However, there can be only one constraint for a given element or domain.

Here is a list of constraint modules and what they do:

File name	What it constrains	Details
example-TopicConstraint.mod	<topic>	• Removes <abstract> • Makes <shortdesc> required • Removes <related-links> • Disallows topic nesting
example-SectionConstraint.mod	<section>	Makes @id required
example-HighlightingDomainConstraint.mod	Highlighting domain	Reduces the highlighting domain elements to <b> and <i>

File name	What it constrains	Details
N/A	<ph>	Remove the <ph> element, allowing only domain extensions (does not require a .mod file)

All of these constraints can be integrated into a single document-type shell for <topic>, since they constrain distinct element types and domains. The constraint for the highlighting domain must be integrated before the "DOMAIN ENTITIES" section, but the order in which the other constraints are listed does not matter.

Comment by Gershon Joseph on 2022-08-29
State the section in which to add this. Also state the reason to do this.

Robert Anderson, 31 August 2022

Not sure I follow entirely:

- "State the section in which to add this" – it says it must be integrated before the "domain entities", so is the request to phrase that differently – "it must go in this section" rather than "it must go before that section"? Strictly from a DTD perspective, for the XML to work, it has to go before that section, but could go anywhere before it, regardless of whether section headers exist.
- "State the reason" – based on the short description, isn't the reason just, you want to apply constraints on more than one element? Not sure that it needs to be repeated here.

Kris Eberlein, 01 September 2022

I changed the second sentence of the paragraph to read as follows:

"The constraint for the highlighting domain typically is located in the "DOMAIN CONSTRAINT INTEGRATION" section, and it must be integrated before the "DOMAIN ENTITIES" section. The other constraints typically are located in the "ELEMENT-TYPE CONFIGURATION INTEGRATION" section, and the order in which they are listed does not matter."

Disposition: Completed

1.4.5 Examples: Constraints implemented using RNG

This section of the specification contains examples of constraints implemented using RNG

1.4.5.1 Example: Redefine the content model for the <topic> element using RNG

In this scenario, a DITA architect for Acme, Incorporated wants to redefine the content model for the topic document type. They want to omit the <abstract> element and make the <shortdesc> element required; they also want to omit the <related-links> element and disallow topic nesting.

1. The DITA architect creates a constraint module: acme-TopicConstraintMod.rng.
2. They update the catalog.xml file to include the new constraint module.
3. They add the following content to acme-TopicConstraint.mod:

```
<div>
  <a:documentation>CONTENT MODEL OVERRIDES</a:documentation>
  <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
    <define name="topic.content" combine="interleave">
      <ref name="title"/>
      <ref name="shortdesc"/>
      <optional>
        <ref name="prolog"/>
      </optional>
```

```
      <optional>
        <ref name="body"/>
      </optional>
    </define>
  </include>
</div>
```

Comment by Kristen J Eberlein on 21 April 2021

I know that the override won't happen without `combine="interleave"`, but I don't know if we cover that in the coding requirements topic. If people start with copying-and-pasting from the module that they are overriding, they won't have that and will get errors.

Gershon Joseph, 29 August 2022

We must give examples that can be copied as-is into an RNG file and work as-is.

Kris Eberlein, 23 August 2022

That's not always going to work, since our code blocks include bold and line-through highlighting.

Robert Anderson, 31 August 2022

I don't think we can hold to a "it must work when you paste it into your RNG file" rule. It would mean that we cannot have any partial examples (which I think is the issue here) – we're trying to give an example of the actual redefinition, which is not going to work without the usual updates in other areas.

The extra markup we use for highlighting is also an issue; we have a test script that runs on our spec to make sure the examples are valid (in the case of RNG, just valid XML), and it had to be updated specifically to ignore our highlighting markup within these.

Kris Eberlein, 01 September 2022

I'm marking this comment as "Accepted," since we need to make sure that we cover combine="interleave" in the "Coding practices" topics.

However, we need to reject the idea that our code samples can be copy-and-pasted – and then actually used. There are a number of reasons:

- The code samples use extra markup for highlighting, using bold and line-through.
- Most code samples are incomplete snippets, and often they include commented-out ellipses to indicate where code is smipped.
- The validation process that is run when after a pull request is opened only checks that the code is valid XML; it does not ensure that something is a valid DTD or RNG.

4. They then integrate t

Comment by Stan Doherty
The transitions "then" and "finally" are unnecessary in an ordered list.

Response by Kris Eberlein, 23 August 2020:

I agree. Thanks for catching this.

Disposition: Completed

he constraint module into the document-type shell for topic by adding an `<include>` element in the "ELEMENT-TYPE CONFIGURATION INTEGRATION" section:

```
<div>
  <a:documentation>ELEMENT-TYPE CONFIGURATION INTEGRATION</a:documentation>
  <include href="acme-TopicConstraintMod.rng"/>
</div>
```

5. They then remove the `<include>` statement that references `topicMod.rng` from the "MODULE INCLUSIONS" section:

```
<div>
  <a:documentation>MODULE INCLUSIONS </a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:audienceAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:deliveryTargetAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:platformAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:productAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:otherpropsAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:highlightDomain.rng:2.0"/>
  </div>
```

6. After checking the test topic to ensure that the content model is modified as expected, the work is done.

1.4.5.2 Example: Constrain attributes for the <section> element using RNG

In this scenario, a DITA architect wants to redefine the attributes for the `<section>` element. They want to make the `@id` attribute required.

1. The DITA architect creates a constraint module: `id-requiredSectionContraintMod.rng`.
2. They update the `catalog.xml` file to include the new constraint module.
3. They add the following content to the constraint module:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <div>
    <a:documentation>ATTRIBUTE LIST OVERRIDES</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
      <define name="section.attributes">
          <attribute name="id">
            <data type="NMTOKEN"/>
          </attribute>
        <ref name="conref-atts"/>
        <ref name="select-atts"/>
        <ref name="localization-atts"/>
        <optional>
          <attribute name="outputclass"/>
        </optional>
      </define>
    </include>
  </div>

</grammar>
```

Note that unlike a constraint module that is implemented using DTD, this constraint module did not need to re-declare the patterns that are referenced in the redefinition of the content model for `<section>`

Comment by Gershon Joseph on 2022-08-29
Why is this reference to DTDs here? We're talking RNG here. This could go into an RNG tutorial, but I don't think we need it in the spec here.

Robert Anderson, 31 August 2022

I agree that it's not strictly necessary, but coming from a long history of DTD configuration I certainly found this useful in learning how to adapt to RNG. The patterns match in so many areas that I think it's useful to note the differences. Curious what others think here.

Kris Eberlein, 31 August 2022

I agree with Robert. And I suspect that many people reading these RNG examples will already have experience with DTDs, so knowing where things differ is very, very useful.

Disposition: Rejected

4. They then integrate the constraint module into the document-type shell for topic by adding an `<include>` element in the "CONTENT CONSTRAINT INTEGRATION" section:

```
<div>
  <a:documentation>CONTENT CONSTRAINT INTEGRATION</a:documentation>
  <include href="id-requiredSectionConstraintMod.rng"/>
</div>
```

5. They then remove the `<include>` statement that references `topicMod.rng` from the "MODULE INCLUSIONS" section:

```
<div>
  <a:documentation>MODULE INCLUSIONS </a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:audienceAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:deliveryTargetAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:platformAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:productAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:otherpropsAttDomain.rng:2.0"/>
    <include href="urn:oasis:names:tc:dita:rng:highlightDomain.rng:2.0"/>
  </div>
```

Comment by Bill Burns
Not all that familiar with RNG, but is this step correct for an attribute constraint?

Kris Eberlein, 31 August 2022

Yes. All examples have been thoroughly tested.

Disposition: Closed

6. After checking the test topic to ensure that the attribute list is modified as expected, the work is done.

1.4.5.3 Example: Constrain a domain module using RNG

In this scenario, a DITA architect wants to use only a subset of the elements defined in the highlighting domain. They want to use `<b>` and `<i>` but not `<line-through>`, `<overline>`, `<sup>`, `<sup>`, `<tt>`, or `<u>`. Their implementation uses RNG for its grammar files.

Comment by Zoë Lawson
None of the other short descriptions in the RNG section state "Their implementation uses RNG for its grammar files." Ditto the DTD examples. I think we can remove it (or add similar approprate statements to all short descriptions.

Kris Eberlein, 23 August 2020:

I agree. Done.

Disposition: Completed

When using RNG, domains can be constrained directly in the document-type shells.

1. They open the document-type shell for topic in an XML editor, and then they modify the "MODULE INCLUSIONS" division to exclude the elements that they do not want the implementation to use:

```
<div>
  <a:documentation>MODULE INCLUSIONS</a:documentation>
  ...
  <include href="highlightDomain.rng">
    <define name="line-through.element">
      <notAllowed/>
    </define>
    <define name="overline.element">
      <notAllowed/>
    </define>
    <define name="sub.element">
      <notAllowed/>
    </define>
    <define name="sup.element">
      <notAllowed/>
    </define>
    <define name="tt.element">
      <notAllowed/>
    </define>
    <define name="u.element">
      <notAllowed/>
    </define>
  </include>
  ..
</div>
```

Note The DITA architect made a choice as to where in the document-type shell they would implement the constraint. It can be placed either in the "Element-type configuration integration" or the "Module inclusions" section.

Comment by Zoë Lawson on 27 August 2022
Is this an RNG thing? A best practice?

Gershon Joseph, 28 August 2022

Agreed. Are there any implications as to where this is placed? If yes, let's state them. If not, why not just specify one place to put it? Don't add any ambiguity to the spec if we don't need to. Let's make the process as straightforward and possible. The less choices the IA has to make, the better.

Kris Eberlein, 31 August 2022

We try not to be overly prescriptive if something is not required. People can build their grammar files in multiple different ways.

Personally, I think it is useful to know that the constraint module could be placed in multiple places.

Robert Anderson, 31 August 2022

I have rather strong feelings on this one - in the past, these sections were presented as "something must go in section X", when putting it in section Y would work just as well (in fact, I am under no obligation to create these section headers at all in my own shell). We should not

say something must go in section X if it works just as well in section Y - that's not avoiding ambiguity, it's giving incorrect information.

I'm willing to accept that we give names to these sections (and happy to do that consistenly / with headers in the grammar files we deliver). I think it's helpful guidance for everyone to describe placement in relation to those sections. But we can't arbitrarily decide you must use one over the other, when there is no actual difference.

To answer Zoe's original question – I think the answer is that this is not a best practice, it is just "it will work in either of those two locations".

Disposition: Rejected

2. They make similar changes to all the other document-type shells in which they want to constrain the highlighting domain.

1.4.5.4 Example: Replace a base element with the domain extensions using RNG

In this scenario, the DITA architect wants to remove the <ph> element but allow the extensions of <ph> that exist in the highlight, programming, software, and user interface domains.

1. They open the document-type shell for topic in an XML editor, and then they modify the "MODULE INCLUSIONS" division to exclude <ph>:

```
<div>
  <a:documentation>MODULE INCLUSIONS</a:documentation>
  <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
    <define name="ph.element">
      <notAllowed/>
    </define>
  </include>
  ...
</div>
```

2. They make similar changes to all the other document-type shells in which they want <ph> to not be available

1.4.5.5 Example: Apply multiple constraints to a single document-type shell using RNG

In this scenario, the DITA architect wants to apply multiple constraints to a document-type shell.

Here is a list of the constraint modules and what they do:

File name	What it constrains	Details
example-TopicConstraint.mod	<topic>	• Removes <abstract> • Makes <shortdesc> required • Removes <related-links> • Disallows topic nesting
example-SectionConstraint.mod	<section>	Makes @id required
Not applicable	Highlighting domain	Reduces the highlighting domain elements to <b> and <i>
Not applicable	<ph>	Remove the <ph> element, allowing only domain extensions

The constraint modules that target the <topic> and <section > elements must be combined, since both elements are defined in topicMod.rng. The other constraints can be implemented directly in the document-type shell.

1. The DITA architect creates a constraint module that combines the constraints from example-TopicConstraint.mod and example-SectionConstraint.mod:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <div>
    <a:documentation>CONTENT MODEL AND ATTRIBUTE LIST OVERRIDES</a:documentation>
    <include href="topicMod.rng">
      <define name="section.attributes">
          <attribute name="id">
            <data type="NMTOKEN"/>
          </attribute>
        <ref name="conref-atts"/>
        <ref name="select-atts"/>
        <ref name="localization-atts"/>
        <optional>
          <attribute name="outputclass"/>
        </optional>
      </define>
      <define name="topic.content">
        <ref name="title"/>
        <ref name="shortdesc"/>
        <optional>
          <ref name="prolog"/>
        </optional>
        <optional>
          <ref name="body"/>
        </optional>
      </define>
    </include>
  </div>
</grammar>
```

2. In the document-type shell, they integrate the constraint module (and remove the inclusion statement for topicMod.rng):

```
<div>
  <a:documentation>ELEMENT-TYPE CONFIGURATION INTEGRATION</a:documentation>
  <include href="acme-SectionTopicContraintMod.rng"/>
</div>
```

3. To constrain the highlight domain, they modify the include statement for the domain module:

```
<div>
  <a:documentation>MODULE INCLUSIONS</a:documentation>
  ...
  <include href="highlightDomain.rng">
    <define name="line-through.element">
      <notAllowed/>
    </define>
    <define name="overline.element">
      <notAllowed/>
    </define>
    <define name="sub.element">
      <notAllowed/>
    </define>
    <define name="sup.element">
      <notAllowed/>
    </define>
    <define name="tt.element">
      <notAllowed/>
    </define>
```

```
    <define name="u.element">
      <notAllowed/>
    </define>
  </include>
  ..
</div>
```

4. Finally, to disallow <ph>, they add the following statement to the constraint module:

```
    <define name="ph.element">
      <notAllowed/>
    </define>
```

1.5 Expansion modules

Expansion modules enable the extension of content models and attribute lists for individual elements. Expansion modules are the opposite of constraints. They add elements and attributes to specific content models and attribute lists, rather than removing them.

1.5.1 Overview of expansion modules

Expansion modules enable information architects to include specialized attributes or elements in specific element types, without making them globally available.

Comment by Eliot Kimber
"them" antecedant is ambiguous here although obvously the intented antecedant is the specialized element or attribute. Might be clear to repeat that: without making the specialized element or attribute globally available.

Response by Kris Eberlein, 31 August 2020:

I agree. Thanks for catching this.

Disposition: Completed

An expansion module can perform the following functions:

Expand content models

Expansion modules extend the content models of specific elements, without making the specialized elements available wherever the specialization base is permitted.

Comment by Eliot Kimber
I think we need to indicate that this is specifically for domain elements, I.e, "extend the content models of specific elements to allow elements from domain specializations without making the specialized elements available wherever their specialization bases are are permitted, as would be the case for normal domain integration."

Kris Eberlein, 31August 2020:

I agree that this wording should be recast, but I don't want to simply use your suggested wording.

Disposition: Accepted

For example, an expansion for <section> can make a new element (<sectionDesc>) available as an optional, child element. The <sectionDesc> element is specialized from <p>, but it is available only within <section>.

Comment by Zoë Lawson on 27 August 2022
Should we explicitly state "for example, <sectionDesc> doesn't appear in <context> ?

Kris Eberlein, 31 August 2020

We don't want to bring the task topic into the discussion here. This example covers expanding the content of the section element for (generic) topics. You'd have to do something entirely different if you want to add sectionDesc to a general task topic. (You could not do this in a strict task, which does not permit the section element.

Disposition: Closed

Comment by Zoë Lawson on 27 August 2022
So I have to use Specializations AND Expansion modules together? That didn't seem obvious to me, and I was surprised when I got to the examples.

Kris Eberlein, 31 August 2020:

Yes, you need to use expansion modules in conjunction with specialization modules. The specialization modules are necessary; they contain the information about the new element or attribute. You could use an existing specialization module, for example, one of the domain specializations shipped with DITA 2.0, if you want to make an element or attribute available ONLY in certain places.

Disposition: Closed

The elements are defined in a separate element domain that declares the content models and attribute lists for the new elements.

Expand attribute lists

Expansion modules extend the attribute lists of specific elements by adding attributes specialized from either `@base` or `@props`.

For example, an expansion for `<entry>`, `<row>`, and `<colspec>` can make `@cell-purpose` available only on those elements. The `@cell-purpose` attribute is specialized from `@base`.

The additional attribute can be either defined directly within the expansion module, or it can be defined in a separate attribute-specialization module. In either case, the token used as value for the `@specializations` attribute must be defined.

Comment by Eliot Kimber
insert "the" between "as" and "value": the token used as the value

---

Kris Eberlein, 31 August 2020:

Done.

1.5.2 Expansion module rules

There are certain rules that apply to the design and implementation of expansion modules. These rules all stem from the requirement that the content model of a specialized element must be consistent with the content model of the specialization base. After generalization, the content model of an element affected by an expansion module must match the original content model for that element.

Comment by Eliot Kimber
I think this is the first place where use generalization as the test for correctness. It's a good test but we haven't defined generalization at this point.

While it's more verbose, I think we need to use the same definitional language we use for constraints and for configuration generally--the result must be consistent with and no less constrained than the base.

Kris Eberlein, 31 August 2020:

I'm assuming that we'll be reintroducing some topics about generalization into this chapter of the spec, so I'll wait to make any changes here until we have done that.

Disposition: Accepted

Specialization base of expanded elements

Elements that are added to content models by expansion

Comment by Gershon Joseph on 2022-08-29
models should be "modules"

Kris Eberlein, 31 August 2020:

No, we really mean "content models" here.

Disposition: Rejected

must be specializations of existing elements that are permitted in the original content model.

Content model of expanded elements

Elements that are added to content models by expansion

Comment by Gershon Joseph on 2022-08-29
models should be "modules"

Kris Eberlein, 31 August 2020:

Duplicate comment.

Disposition: Rejected

must be allowed only where their specialization base is allowed.

For example, when creating an expansion

Comment by Gershon Joseph on 2022-08-29
models should be "modules"

Kris Eberlein, 31 August 2020:

Duplicate comment.

Disposition: Rejected

that adds a specialization of `<data>` to `<ol>`, the specialization of `<data>` must only be allowed before any `<li>` elements, as that is the only place that the `<data>` element is allowed in the content model for an ordered list.

Ordinality of expanded elements

Elements that are added to content models by expansion modules must not violate the ordinality of the original content model. If the original content model requires a child element to occur at least once, then the expanded content model cannot break this requirement. If the original content model only permits a child element to occur once, then the expanded content model cannot break this requirement.

For example, in the expansion module that adds a specialization of `<data>` to `<ol>`, the redefined content model for `<ol>` cannot make the `<li>` element optional.

However, a

Comment by Gershon Joseph on 2022-08-29
an

Kris Eberlein, 31 August 2020:

Done

Disposition: Completed

expansion module that adds a specialization of `<li>` (`<listIntro>`) to `<ol>` can redefine the content model of `<ol>` in the following ways:

- Make `<listIntro>` the first child element and be required
- Make `<li>` the second child element and optional

When a DITA topic affected by this expansion module is generalized, the resulting markup would be valid; the content model of `<ol>` would be respected.

Comment by Eliot Kimber
Another mention of generalization.

Kris Eberlein, 31 August 2020:

I'm assuming that we'll be reintroducing some topics about generalization into this chapter of the spec, so I'll wait to make any changes here until we have done that.

Disposition: Accepted

Aggregation of expansion modules

The content model of an element can be modified by either of the following element-configuration modules:

- Constraint module
- Expansion module

The content model of an element can be modified only by a single element-type configuration module. If multiple constraints or extensions need to be applied to a single element, the element configurations must be combined into a single module that reflects all the constraints and expansions that were defined in the original separate modules.

1.5.3 Examples: Expansion implemented using DTDs

This section of the specification contains examples of extension modules that are implemented using DTDs.

1.5.3.1 Example: Adding an element to the <section> element using DTDs

In this scenario, a DITA architect wants to modify the content model for the `<section>` element. The DITA architect wants to add an optional `<sectionDesc>` element that is specialized from `<p>`.

To accomplish this, the DITA architect needs to create the following modules and integrate them into the document-type shell:

- An element specialization module that defines the `<sectionDesc>` element

  Comment by Eliot Kimber
  An element **domain** specialization module.

  Kris Eberlein, 31 August 2020:

  Done. Hyphenated "element-domain".

  Disposition: Completed

- An expansion module that adds the `<sectionDesc>` element to the content model for `<section>`

1. First, the DITA architect creates the element specialization module: `sectionDescDomain.mod`. This single `.mod` file defines the parameter entity, content model, attributes, and value for the `@class` attribute for `<sectionDesc>`.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ENTITY % sectionDesc "sectionDesc">

<!ENTITY % sectionDesc.content "(%para.cnt;)*">
<!ENTITY % sectionDesc.attributes "%univ-atts;">

<!ELEMENT sectionDesc %sectionDesc.content;>
<!ATTLIST sectionDesc %sectionDesc.attributes;>

<!ATTLIST sectionDesc    class CDATA "+ topic/p sectionShortdesc-d/sectionDesc ">
```

   Comment by Bill Burns
   Is the inconsistency in the domain name for a reason? If not, I think sectionDesc-d would make more sense.

   Eliot Kimber, 29 August 2022

   I agree that sectionDesc-d is the better short name.

   Kris Eberlein, 31 August 2022

   Good catch. Done.

   Disposition: Completed

2. The DITA architect adds the element specialization module to the `catalog.xml` file.

3. Next, the DITA architect modifies the applicable document-type shell to integrate the applicable element specialization module:

```
<!-- ============================================================= -->
<!--                    DOMAIN ELEMENT INTEGRATION                 -->
<!-- ============================================================= -->

<!-- ... other domains ... -->

<!ENTITY % sectionDesc-d-def
  PUBLIC "-//ACME//ELEMENTS DITA 2.0 Section Description Domain//EN"
         "sectionDescDomain.mod"
>%sectionDesc-d-def;
```

At this point, the new domain is integrated into the topic document-type shell. However, the new element is not added to the content model for <section>.

4. Next, the DITA architect creates an expansion module: acme-SectionExpansion.mod. This module adds the <sectionDesc> element to the content model of <section>.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Declares the entities referenced in the modified content  -->
<!-- model.                                                    -->

<!ENTITY % dl "dl">
<!ENTITY % div "div">
<!ENTITY % fig "fig">
<!ENTITY % image "image">
<!ENTITY % lines "lines">
<!ENTITY % lq "lq">
<!ENTITY % note "note">
<!ENTITY % object "object">
<!ENTITY % ol "ol">
<!ENTITY % p "p">
<!ENTITY % pre "pre">
<!ENTITY % simpletable "simpletable">
<!ENTITY % sl "sl">
<!ENTITY % table "table">
<!ENTITY % ul "ul">
<!ENTITY % cite "cite">
<!ENTITY % include "include">
<!ENTITY % keyword "keyword">
<!ENTITY % ph "ph">
<!ENTITY % q "q">
<!ENTITY % term "term">
<!ENTITY % text "text">
<!ENTITY % tm "tm">
<!ENTITY % xref "xref">
<!ENTITY % state "state">
<!ENTITY % data "data">
<!ENTITY % foreign "foreign">
<!ENTITY % unknown "unknown">
<!ENTITY % title "title">
<!ENTITY % draft-comment "draft-comment">
<!ENTITY % fn "fn">
<!ENTITY % indexterm "indexterm">
<!ENTITY % required-cleanup "required-cleanup">
<!ENTITY % sectionDesc "sectionDesc">

<!-- Defines the modified content model for <section>.       -->

<!ENTITY % section.content
              "(#PCDATA |
               %dl; |
               %div; |
               %fig; |
               %image; |
               %lines; |
               %lq; |
               %note; |
               %object; |
```

```
                %ol; |
                %p; |
                %pre; |
                %simpletable; |
                %sl; |
                %table; |
                %ul; |
                %cite; |
                %include; |
                %keyword; |
                %ph; |
                %q; |
                %term; |
                %text; |
                %tm; |
                %xref; |
                %state; |
                %data; |
                %foreign; |
                %unknown; |
                %title; |
                %draft-comment; |
                %fn; |
                %indexterm; |
                %required-cleanup; |
                %sectionDesc;)*"
>
```

Comment by Eliot Kimber
Bold the addition of `%sectionDesc;`

Kris Eberlein, 31 August 2020:

Done.

Disposition: Completed

Note that the DITA architect needed to explicitly declare all the elements, rather than using the `%section.cnt;` parameter entity that is used in the definition of `<section>`. Because the element-configuration modules are integrated into the document-type shell before the base grammar modules, none of the parameter entities that are used in the base DITA vocabulary modules are available.

5. Finally, the DITA architect integrates the expansion module into the document-type shell:

```
<!-- ============================================================= -->
<!--           ELEMENT-TYPE CONFIGURATION INTEGRATION               -->
<!-- ============================================================= -->

<!-- Other constraint and expansion modules -->

<!ENTITY % acmeSection-def
  PUBLIC "-//ACME//ELEMENTS DITA 2.0 Section Expansion//EN"
         "acme-SectionExpansion.mod"
>%acmeSection-def;
```

6. After updating the `catalog.xml` file

Comment by Stan Doherty
Recommend - "to include the expansion module and to test it,"

Response by Kris Eberlein, 23 August 2020:

Done

Disposition: Completed

to include the expansion module and testing, the work is done.

1.5.3.2 Example: Adding an attribute to certain table elements using DTDs

In this scenario, a company makes extensive use of complex tables to present product listings. They occasionally highlight individual cells, rows, or columns for various purposes. The DITA architect wants to implement a semantically meaningful way to identify the purpose of various table elements.

The DITA architect decides to create a new attribute (`@cell-purpose`) and add it to the attribute lists of the following elements:

- `<colspec>`
- `<entry>`
- `<row>`
- `<stentry>`
- `<strow>`

The new attribute will be specialized from `@base`, and it will take a small set of tokens as values.

The DITA architect decides to integrate the attribute declaration and its assignment to elements into a single expansion module. An alternate approach would be to put each `<!ATTLIST` declaration in its own separate expansion module, thus allowing DITA architects who construct document-type shells to decide the elements to which to apply the attribute.

1. First, the DITA architect creates the expansion module for the `@cell-purpose` attribute: `acme-cellPurposeAttExpansion.ent`.

```
<!-- Define the attribute -->
<!ENTITY % cellPurposeAtt-d-attribute-expansion
  "cell-purpose  (sale | out-of-stock | new | last-chance | inherit | none)  #IMPLIED"
>

<!-- Declare the entity to be used in the @specializations attribute -->
<!ENTITY cellPurposeAtt-d-att "@base/cell-purpose" >

<!-- Add the attribute to the elements. -->
<!ATTLIST entry %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST row %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST colspec %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST strow %cellPurposeAtt-d-attribute-expansion;>
<!ATTLIST stentry %cellPurposeAtt-d-attribute-expansion;>
```

   Note The attribute definition entity is optional. It is used here to enable the DITA architect to add the same attribute with the same tokens to several elements.

2. They then update the `catalog.xml` file to include the expansion module.
3. They integrate this module into the applicable document-type shell.

```
<!-- ============================================================= -->
<!--              DOMAIN ATTRIBUTES DECLARATIONS                   -->
<!-- ============================================================= -->

<!-- ... other domains ... -->

<!ENTITY % cellPurposeAttExpansion-d-dec
  PUBLIC "-//ACME//ENTITIES DITA Cell Purpose Attribute Expansion//EN"
         "cellPurposeAttExpansion.ent"
>%cellPurposeAttExpansion-d-dec;
```

4. They add the entity for the contribution to the `@specializations` attribute.

```
<!-- ============================================================= -->
<!--                  SPECIALIZATIONS ATTRIBUTE OVERRIDE           -->
<!-- ============================================================= -->
```

```
<!ENTITY included-domains
                          "&audienceAtt-d-att;
                           &cellPurposeAtt-d-att;
                           &deliveryTargetAtt-d-att;
                           &otherpropsAtt-d-att;
                           &platformAtt-d-att;
                           &productAtt-d-att;"
>
```

5. After checking the test topic to ensure that the attribute lists are modified as expected, the work is done.

1.5.3.3 Example: Adding an existing domain attribute to certain elements using DTDs

In this scenario, a company wants to use the `@otherprops` attribute specialization. However, they want to make the attribute available **only** on certain elements: `<p>`, `<div>`, and `<section>`.

The DITA architect will need to create an extension module and integrate it into the appropriate document-type shells.

1. The DITA architect creates an expansion module that adds the `@otherprops` attribute to the selected elements: `acme-otherpropsAttExpansion.mod`. The expansion module contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Add the otherprops attribute to certain elements -->
<!ATTLIST p %otherpropsAtt-d-attribute;>
<!ATTLIST div %otherpropsAtt-d-attribute;>
<!ATTLIST section %otherpropsAtt-d-attribute;>
```

   Note that the `%otherpropsAtt-d-attribute;` is defined in the separate attribute-specialization module that defines the `@otherprops` attribute.
2. They then update the `catalog.xml` file to include the expansion module.
3. They integrate the extension module into the applicable document-type shell, **after** the declaration for the `@otherprops` attribute-specialization module:

```
<!-- ============================================================= -->
<!--              DOMAIN ATTRIBUTES DECLARATIONS                   -->
<!-- ============================================================= -->
...

<!ENTITY % otherpropsAtt-d-dec
  PUBLIC "-//OASIS//ENTITIES DITA 2.0 Otherprops Attribute Domain//EN"
         "otherpropsAttDomain.ent"
>%otherpropsAtt-d-dec;

<!ENTITY % otherprops-expansion-e-def
  PUBLIC "-//ACME//DITA 2.0 Otherprops Expansion//EN"
         "acme-otherpropsAttExpansion.mod"
  >%otherprops-expansion-e-def;

...
```

4. They remove the reference to the `@otherprops` attribute from the `props-attribute-extension` entity:

```
<!-- ============================================================= -->
<!--                    DOMAIN ATTRIBUTE EXTENSIONS                -->
<!-- ============================================================= -->

<!ENTITY % base-attribute-extensions
  ""
```

```
>

<!ENTITY % props-attribute-extensions
  "%audienceAtt-d-attribute;
   %deliveryTargetAtt-d-attribute;
   %otherpropsAtt-d-attribute;
   %platformAtt-d-attribute;
   %productAtt-d-attribute;"
>
```

5. They ensure that the `included-domains` entity contains the `@otherprops` contribution to the `@specializations` attribute:

```
<!-- ============================================================= -->
<!--                  SPECIALIZATIONS ATTRIBUTE OVERRIDE           -->
<!-- ============================================================= -->

<!ENTITY included-domains
                          "&audienceAtt-d-att;
                           &deliveryTargetAtt-d-att;
                           &otherpropsAtt-d-att;
                           &platformAtt-d-att;
                           &productAtt-d-att;"
>
```

6. After checking the test topic to ensure that the attribute lists are modified as expected, the work is done.

1.5.3.4 Example: Aggregating constraint and expansion modules using DTDs

The DITA architect wants to add some extension modules to the document-type shell for topic. The document-type shell already integrates a number of constraint modules.

The following table lists the constraints that are currently integrated into the document-type shell:

File name	What it constrains	Details
`example-TopicConstraint.mod`	`<topic>`	• Removes `<abstract>` • Makes `<shortdesc>` required • Removes `<related-links>` • Disallows topic nesting
`example-SectionConstraint.mod`	`<section>`	• Makes `<title>` required • Reduces the content model of `<section>` to a smaller subset
`example-HighlightingDomainConstraint.mod`	Highlighting domain	Reduces the highlighting domain elements to `<b>` and `<i>`

The following table lists the expansion modules that the DITA architect wants to add to the document-type shell:

File name	What it modifies	Details
`example-sectionSectionShortdescExpansion.mod`	`<section>`	Adds an optional `<sectionDesc>` element to `<section>`. The `<sectionDesc>` element can only appear directly after `<title>`.

File name	What it modifies	Details
Comment by Bill Burns Is different than the module added in 1.5.3.1 Example: Adding an element to the section element using DTDs (54)? --- Kris Eberlein, 31 August 2022 I've changed the file name to match the topic you reference. **Disposition: Completed**		
`example-dlentryModeAttExpansion.ent`	`<dlentry>`	Adds `@dlentryMode` to the attributes of `<dlentry>`.

The constraint and expansion modules that target the `<section>` element must be combined into a single element-configuration module. An element can only be targeted by a single element-configuration module.

Comment by dstevens73
Shouldn't we include an example of what the combined section modules look like? There is an example in the RNG file.

Kris Eberlein, 01 September 2022

I'm not sure. Other examples show clear examples of redefining content models (both constraint and expansion).

Yes, it would be ideal to have the DTD and RNG examples always parallel, but I don't think we can manage that for DITA 2.0.

Also, I'd have to dig into the test files I used when constructing these examples, and build that aggregated element-configuration module … I'm not sure the time would be worth it. Marking this comment deferred.

Disposition: Deferred

1.5.4 Examples: Expansion implemented using RNG

This section of the specification contains examples of extension modules implemented using RNG.

1.5.4.1 Example: Adding an element to the <section> element using RNG

In this scenario, a DITA architect wants to modify the content model for the `<section>` element. He wants to add an optional `<sectionDesc>` element that is specialized from `<p>`; the `<sectionDesc>` can occur once and must be directly after the section title.

To accomplish this, the DITA architect needs to create the following modules and integrate them into the document-type shells:

- An element domain module that defines the `<sectionDesc>` element
- An expansion module that adds the `<sectionDesc>` element to the content model for `<section>`

1. First, the DITA architect creates the element domain module: `sectionDescDomain.rng`. It contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <div>
    <a:documentation>DOMAIN EXTENSION PATTERNS</a:documentation>
  </div>
  <div>
    <a:documentation>ELEMENT TYPE NAME PATTERNS</a:documentation>
    <define name="sectionDesc">
      <ref name="sectionDesc.element"/>
    </define>
  </div>
  <div>
    <a:documentation>ELEMENT TYPE DECLARATIONS</a:documentation>
    <div>
      <a:documentation>LONG NAME: Section Description</a:documentation>
      <define name="sectionDesc.content">
        <zeroOrMore>
            <ref name="para.cnt"/>
          </zeroOrMore>
      </define>
      <define name="sectionDesc.attributes">
        <ref name="univ-atts"/>
      </define>
      <define name="sectionDesc.element">
        <element name="sectionDesc" dita:longName="Section Description">
          <a:documentation/>
          <ref name="sectionDesc.attlist"/>
          <ref name="sectionDesc.content"/>
        </element>
      </define>
      <define name="sectionDesc.attlist" combine="interleave">
        <ref name="sectionDesc.attributes"/>
      </define>
    </div>
  </div>
  <div>
    <a:documentation>SPECIALIZATION ATTRIBUTE DECLARATIONS</a:documentation>
    <define name="sectionDesc.attlist" combine="interleave">
      <optional>
        <attribute name="class" a:defaultValue="+ topic/p sectionDesc-d-p/sectionDesc
"/>
      </optional>
    </define>
  </div>
</grammar>
```

Comment by Bill Burns
Should `@class` be optional here?

Eliot Kimber, 29 August 2022

`@class`is optional because it has a default value.

Disposition: Closed

2. The DITA architect adds the element domain module to the `catalog.xml` file.
3. Next, the DITA architect modifies the document-type shell (in this example, the one for topic) to integrate the element domain module:

```
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    ...
```

```
    <include href="urn:example:names:tc:dita:rng:sectionDescDomain.rng:2.0"/>
  </div>
```

At this point, the new domain is integrated into the document-type shell. However, the new element is not added to the content model for <section>.

4. Next, the DITA architect created an expansion module (sectionExpansionMod.rng) that adds the <sectionDesc> element to the content model of <section>:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <div>
    <a:documentation>CONTENT MODEL AND ATTRIBUTE LIST OVERRIDES</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
      <define name="topic-info-types">
        <ref name="topic.element"/>
      </define>
      <define name="section.content" combine="interleave">
```

Comment by
Eliot Kimber

I'm not sure combine="interleave" is correct here because the intent is to redeclare the content model entirely.

Kris Eberlein, 31 August 2022

I've always been a little shaky about understanding combine="interleave". When is it needed and when it is not?

@Eliot, thanks for the additional information. I've removed combine="interleave from the <define> element.

Disposition: Completed

```
        <optional>
          <ref name="title"/>
        </optional>
        <optional>
          <ref name="sectionDesc"/>
        </optional>
        <zeroOrMore>
          <ref name="section.cnt"/>
        </zeroOrMore>
      </define>
    </include>
  </div>
</grammar>
```

Note that the expansion module directly integrates topicMod.rng.

5. Finally, the DITA architect integrates the expansion module into the document-type shell and removes the inclusion statement for topicMod.rng:

```
  <div>
    <a:documentation>ELEMENT-TYPE CONFIGURATION INTEGRATION</a:documentation>
    <include href="sectionExpansionMod.rng"/>
  </div>
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
      <define name="topic-info-types">
        <ref name="topic.element"/>
      </define>
    </include>
    ...
```

```
    <include href="urn:example:names:tc:dita:rng:sectionDescDomain.rng:2.0"/>
  </div>
```

6. After updating the `catalog.xml` file to include the expansion module and testing, the work is done.

1.5.4.2 Example: Adding an attribute to certain table elements using RNG

In this scenario, a company makes extensive use of complex tables to present product listings. They occasionally highlight individual cells, rows, or columns for various purposes. The DITA architect wants to implement a semantically meaningful way to identify the purpose of various table elements.

The DITA architect decides to create a new attribute (`@cell-purpose`) and add it to the content model of the following elements:

- `<entry>`
- `<row>`
- `<colspec>`
- `<stentry>`
- `<strow>`

The new attribute will be specialized from `@base`, and it will take a small set of tokens as values.

The DITA architect decides to integrate the attribute declaration and its assignment to elements into a single expansion module. An alternate approach would be to put each `<!ATTLIST` declaration in its own separate expansion module, thus allowing DITA architects who construct document-type shells to decide the elements to which to apply the attribute.

Comment by Eliot Kimber
Replace "`<!ATTLIST` declaration" with "attribute list pattern"

Kris Eberlein, 31 August 2022

Done.

Disposition: Completed

1. The DITA architect creates an expansion module: `cellPurposeAtt.rng`. It contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                          schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

  <!-- DEFINE THE ATTRIBUTE SPECIALIZATION -->
  <define name="cellPurposeAtt">
    <optional>
      <attribute name="cellPurpose">
        <a:documentation>Specifies the purpose of the table cell. This is a specialized
          attribute for Acme Corporation.
        </a:documentation>
        <choice>
          <value>sale</value>
          <value>out-of-stock</value>
          <value>new</value>
          <value>last-chance</value>
          <value>inherit</value>
```

```
          <value>none</value>
        </choice>
      </attribute>
    </optional>
  </define>

  <!-- ASSIGN THE ATTRIBUTE TO CERTAIN ELEMENTS -->
  <define name="entry.attributes" combine="interleave">
    <ref name="cellPurposeAtt"/>
  </define>
  <define name="stentry.attributes" combine="interleave">
    <ref name="cellPurposeAtt"/>
  </define>
  <define name="row.attributes" combine="interleave">
    <ref name="cellPurposeAtt"/>
  </define>
  <define name="strow.attributes" combine="interleave">
    <ref name="cellPurposeAtt"/>
  </define>
  <define name="colspec.attributes" combine="interleave">
    <ref name="cellPurposeAtt"/>
  </define>
</grammar>
```

2. They then update the `catalog.xml` file to include the expansion module.
3. They integrate the expansion module into the document-type shell:

```
<div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    ...
    <include href="urn:example:names:tc:dita:rng:cellPurposeAtt.rng:2.0"/>
  </div>
```

4. They specify the value that the `@cellPurpose` attribute contributes to the `@specializations` attribute:

```
  <div>
    <a:documentation>SPECIALIZATIONS ATTRIBUTE</a:documentation>
    <define name="specializations-att">
      <optional>
        <attribute name="specializations" a:defaultValue="
                         @props/audience
                         @props/deliveryTarget
                         @props/otherprops
                         @props/platform
                         @props/product
                         @base/cellPurpose"/>
      </optional>
    </define>
  </div>
```

5. After checking the test file to ensure that the attribute lists are modified as expected, the work is done.

1.5.4.3 Example: Adding an existing domain attribute to certain elements using RNG

In this scenario, a company wants to use the `@otherprops` attribute specialization. However, they want to make the attribute available **only** on certain elements: `<p>`, `<div>`, and `<section>`.

The DITA architect will need to create an extension module and integrate it into the appropriate document-type shells.

1. The DITA architect creates an expansion module that adds the `@otherprops` attribute to the selected elements: `acme-otherpropsAttExpansion.mod`. The expansion module contains the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                        schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <div>
    <a:documentation>CONTENT MODEL AND ATTRIBUTE LIST OVERRIDES</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
      <define name="topic-info-types">
        <ref name="topic.element"/>
      </define>
      <define name="p.attributes" combine="interleave">
        <optional>
          <attribute name="otherprops"/>
        </optional>
      </define>
      <define name="div.attributes" combine="interleave">
        <optional>
          <attribute name="otherprops"/>
        </optional>
      </define>
      <define name="section.attributes" combine="interleave">
        <optional>
          <attribute name="otherprops"/>
        </optional>
      </define>
    </include>
  </div>
</grammar>
```

2. They then update the `catalog.xml` file to include the expansion module.
3. They integrate the extension module into the applicable document-type shell, and remove the `<include>` element for `topicMod.rng`:

```
  <div>
    <a:documentation>ELEMENT-TYPE CONFIGURATION INTEGRATION</a:documentation>
      <include href="acme-otherpropsAttExpansion.rng"/>
  </div>
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.x"/>
    ...
    <include href="urn:oasis:names:tc:dita:rng:otherpropsAttDomain.rng:2.0">
    </include>
  </div>
```

4. They remove the reference to the `@otherprops` attribute from the `props-attribute-extension` pattern:

```
  <div>
    <a:documentation>MODULE INCLUSIONS</a:documentation>
      ...
    <include href="urn:oasis:names:tc:dita:rng:otherpropsAttDomain.rng:2.0">
      <define name="props-attribute-extensions" combine="interleave">
      <empty/>
      </define>
    </include>
```

5. They ensure that the `included-domains` entity contains the `@otherprops` contribution to the `@specializations` attribute:

```
<div>
  <a:documentation>SPECIALIZATIONS ATTRIBUTE</a:documentation>
  <define name="specializations-att">
    <optional>
      <attribute name="specializations" a:defaultValue="
                       @props/audience
                       @props/deliveryTarget
                       @props/otherprops
                       @props/platform
                       @props/product"/>
    </optional>
  </define>
</div>
```

6. After checking the test topic to ensure that the attribute lists are modified as expected, the work is done.

1.5.4.4 Example: Aggregating constraint and expansion modules using RNG

The DITA architect wants to add some extension modules to the document-type shell for topic. The document-type shell already integrates a constraint module.

The following table lists the constraint module and the extension modules that the DITA architect wants to integrate into the document-type shell for topic.

Type of element configuration	File name	What it does
Constraint	`topicSectionConstraint.rng`	Constrains `<topic>`: • Removes `<abstract>` • Makes `<shortdesc>` required • Removes `<related-links>` • Disallows topic nesting Constrains `<section>`: • Makes `@id` required
Expansion	`sectionExpansionMod.rng`	Adds `<sectionDesc>` to the content model of `<section>`
Expansion	`tableCellAttExpansion.rng`	Adds `@cellPurpose` to the attribute lists for certain table elements

Because all of these element-configuration modules target elements declared in `topicMod.rng`, the DITA architect needs to combine them into a single element-configuration module like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="urn:oasis:names:tc:dita:rng:vocabularyModuleDesc.rng"
                        schematypens="http://relaxng.org/ns/structure/1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:dita="http://dita.oasis-open.org/architecture/2005/"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <div>
    <a:documentation>CONTENT MODEL AND ATTRIBUTE LIST OVERRIDES</a:documentation>
    <include href="urn:oasis:names:tc:dita:rng:topicMod.rng:2.0">
      <!-- Redefines attribute list for section: Makes @id required -->
      <define name="section.attributes">
        <attribute name="id">
          <data type="ID"/>
```

```
        </attribute>
        <ref name="conref-atts"/>
        <ref name="select-atts"/>
        <ref name="localization-atts"/>
        <optional>
          <attribute name="outputclass"/>
        </optional>
      </define>
      <!-- Adds sectionDesc to the content model of section -->
      <define name="section.content" combine="interleave">
```

Comment by
Eliot Kimber

combine="interleave" may not be correct here.

Kris Eberlein, 01 September 2022

I've removed combine="interleave".

Disposition: Completed

```
        <optional>
          <ref name="title"/>
        </optional>
        <optional>
          <ref name="sectionDesc"/>
        </optional>
        <zeroOrMore>
          <ref name="section.cnt"/>
        </zeroOrMore>
      </define>
      <!-- Adds @cellPurpose to certain table and simple table elements -->
      <define name="colspec.attributes" combine="interleave">
        <optional>
          <attribute name="cellPurpose"/>
        </optional>
      </define>
      <define name="entry.attributes" combine="interleave">
        <optional>
          <attribute name="cellPurpose"/>
        </optional>
      </define>
      <define name="row.attributes" combine="interleave">
        <optional>
          <attribute name="cellPurpose"/>
        </optional>
      </define>
      <define name="stentry.attributes" combine="interleave">
        <optional>
          <attribute name="cellPurpose"/>
        </optional>
      </define>
      <define name="strow.attributes" combine="interleave">
        <optional>
          <attribute name="cellPurpose"/>
        </optional>
      </define>
      <!-- Redefines topic: removes abstract and related-links; makes shortdesc -->
      <!--                  required; disallows topic nesting                   -->
      <define name="topic.content">
        <ref name="title"/>
        <ref name="shortdesc"/>
        <optional>
          <ref name="prolog"/>
        </optional>
        <optional>
          <ref name="body"/>
        </optional>
      </define>
    </include>
```

```
   </div>
 </grammar>
```

When the DITA architect edits the document-type shell to integrate the element configuration module, they also need to do the following:

- Remove the include statement for `topicMod.rng`
- Add `<section>` to the "ID-DEFINING ELEMENT OVERRIDES" division

A Aggregated RFC-2119 statements

This appendix contains all the normative statements from the DITA 2.0 specification. They are aggregated here for convenience in this non-normative appendix.

Item	Conformance statement
001 (9)	While the DITA specification only defines coding requirements for DTD and RELAX NG, conforming DITA documents **MAY** use other document-type constraint languages, such as Schematron.
002 (9)	With two exceptions, a document-type shell **MUST NOT** directly define element or attribute types; it only includes vocabulary and element-configuration modules (constraint and expansion). The exceptions to this rule are the following: • The ditabase document-type shell directly defines the `<dita>` element. • RNG-based document-type shells directly specify values for the `@specializations` attribute. These values reflect the details of the attribute domains that are integrated by the document-type shell.
003 (9)	Document-type shells that are not provided by OASIS **MUST** have a unique public identifier, if public identifiers are used.
004 (9)	Document-type shells that are not provided by OASIS **MUST NOT** indicate OASIS as the owner. The public identifier or URN for such document-type shells **SHOULD** reflect the owner or creator of the document-type shell.
005 (18)	Structural modules based on topic **MAY** define additional topic types that are then allowed to occur as subordinate topics within the top-level topic. However, such subordinate topic types **MAY NOT** be used as the root elements of conforming DITA documents.
006 (18)	Domain elements intended for use in topics **MUST** ultimately be specialized from elements that are defined in the topic module. Domain elements intended for use in maps **MUST** ultimately be specialized from elements defined by or used in the map module. Maps share some element types with topics but no map-specific elements can be used within topics.
007 (23)	Every DITA element (except the `<dita>` element that is used as the root of a ditabase document) **MUST** declare a `@class` attribute. **Comment by Eliot Kimber** I think "exhibit" is more correct than "declare" here in that "declare" implies "declared in a grammar", which is not required. But I'm not sure any reader would understand what "exhibit" means so I'm OK with keeping "declare". --- Kris Eberlein, 31 August 2022 I agree with both your points. Marking this comment as "Closed." **Disposition: Closed**
008 (24)	When the `@class` attribute is declared in an XML grammar, it **MUST** be declared with a default value. In order to support generalization round-tripping (generalizing specialized content into a generic form and then returning it to the specialized form) the default value **MUST NOT** be fixed. This allows a generalization process to overwrite the default values that are defined by a general document type with specialized values taken from the document being generalized.
009 (24)	A vocabulary module **MUST NOT** change the `@class` attribute for elements that it does not specialize, but simply reuses by reference from more generic levels.

Item	Conformance statement
010 (24)	Authors **SHOULD NOT** modify the `@class` attribute.
011 (26)	Each specialization of the `@props` and `@base` attributes **MUST** provide a token for use by the `@specializations` attribute.

Index

T