Stage three: #217 Remove @domains attribute

Remove the domains attribute, and the tokens used for the domains attribute; for specialized attributes, replace the existing parenthetical syntax with a simpler token syntax..

Champion: Robert D. Anderson

Provide information about the champion. If the proposal is submitted by a subcommittee, include the name of the point person. He or she should have prepared this proposal and thoroughly understand all of the content. The point person must be present at the TC calls when this proposal is discussed.

Tracking information

Event	**Date**	**Links**
Stage 1 proposal accepted	14 May 2019	https://lists.oasis-open.org/archives/dita/201905/msg00043.html
Stage 2 proposal submitted	14 June 2019	PDF, DITA
Stage 2 proposal discussed	18 June 2019	https://lists.oasis-open.org/archives/dita/201906/msg00068.html
Stage 2 proposal approved	2 July 2019	https://lists.oasis-open.org/archives/dita/201907/msg00013.html
Stage 3 proposal submitted to reviewers	2 December 2019	Carsten Brennecke, Eliot Kimber
Stage 3 proposal (this document) submitted to TC	9 December 2019	

Approved technical requirements

1. Remove the grammar file definition of `@domains`
2. Remove definitions for tokens currently used in `@domains`
3. Define a new attribute `@specializations` on those elements that previously allowed `@domains`
4. Define a new syntax for the attribute; currently it is defined only for attribute domains, using the syntax `@props/thing1` where "thing1" is an attribute specialization of `@props`, `@props/thing1/thing2` where "thing2" is a further specialization of `@thing1`, and so on.
5. The attribute syntax will be used for all specializations of `@props` and `@base`.

Dependencies or interrelated proposals

N/A.

Modified grammar files

`basemap.rng` (before)	**`basemap.rng` (after)**

`<a:documentation>DOMAINS ATTRIBUTE</`
`a:documentation>`
`    <define name="domains-att">`
`         <optional>`
`            <attribute name="domains"`
	`<a:documentation>SPECIALIZATIONS ATTRIBUTE</`
	`a:documentation>`
	`    <define name="specializations-att">`
	`         <optional>`

`basemap.rng` (before)

```
                              a:defaultValue="
                                (map ditavalref-d)
                                (topic hazard-d)
                                (topic hi-d)
                                (topic indexing-d)
                                (map mapgroup-d)
                                (topic ut-d)
                                a(props audience)
                                a(props
  deliveryTarget)
                                a(props platform)
                                a(props product)
                                a(props
  otherprops)"/>
            </optional>
         </define>
```

`basemap.rng` (after)

```
              <attribute
 name="specializations"
                       a:defaultValue="
                         @props/audience
                         @props/
deliveryTarget
                         @props/platform
                         @props/product
                         @props/otherprops"/
>
         </optional>
      </define>
```

`basetopic.rng` (before)

```
<a:documentation>DOMAINS ATTRIBUTE</
a:documentation>
    <define name="domains-att">
      <optional>
        <attribute name="domains"
          a:defaultValue="(topic hazard-d)
                          (topic hi-d)
                          (topic indexing-d)
                          (topic ut-d)
                          a(props audience)
                          a(props
 deliveryTarget)
                          a(props platform)
                          a(props product)
                          a(props
 otherprops)"
        />
      </optional>
    </define>
```

`basetopic.rng` (after)

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

RNG modules (before)	RNG modules (after)
Most RNG elements use the `<domainsContribution>` element as metadata. That metadata contains the token added to 1.3 level RNG shells to assemble the `@domains` attribute, and was also used to generate DTD and XSD files with 1.3. For example: `<domainsContribution>a(props audience)</domainsContribution>`	This element is not an externally defined part of our RNG grammar, but we should still update it based on the name change, so that `<domainsContribution>` becomes `<specializationsContribution>`. For example: `<specializationsContribution>@props/audience</specializationsContribution>`

Audience, deliveryTarget, platform, product, and otherprops RNG modules (before)	**Audience, deliveryTarget, platform, product, and otherprops RNG modules (after)**
Using this pattern: `<domainsContribution>a(props audience)</domainsContribution>`	Keep the XML syntax the same because this is for external tools (not part of the grammar), but change the token for the new syntax: `<specializationsContribution>@props/audience</specializationsContribution>`

`topicMod.rng` and `mapMod.rng` (before)	**`topicMod.rng` and `mapMod.rng` (after)**
`<ref name="domains-att"/>`	`<ref name="specializations-att"/>`

Each base element domain module (before)	**Each base element domain module (after)**
Find this pattern: `<domainsContribution>(topic hazard-d)</domainsContribution>`	Remove the `<domainsContribution>`

Modified terminology

N/A

Modified specification documentation

Remove topics Weak and strong constraints and Conref compatibility with constraints (without domain contributions, there is no longer a distinction and no need for processors to process differently).

Remove the topic Processing documents with different values of the domains attribute which no longer applies.

For example topics Specializing to include non-DITA content, Example: Redefine the content model for the topic element, Example: Constrain a domain module, Example: Replace a base element with the domain extensions, and Example: Apply multiple constraints to a single document-type shell, remove the example sections that define `@domains`.

From domains attribute rules and syntax:

Before	**After: replace with the following**
The `@domains` attribute enables processors to determine whether two elements or two documents use compatible domains. The attribute is declared on the root element for each topic or map type. Each structural, domain, and constraint module defines its ancestry as a parenthesized sequence of space-separated module names; the effective value of the `@domains` attribute is composed of these parenthesized sequences. Document type shells collect the values that are provided by each module to construct the effective value of the `@domains` attribute. Processors can examine the collected values when content from one document is used in another, in order to determine whether the content is compatible.	The `@specializations` attribute enables processors to determine what attribute extensions 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; the effective value of the `@specializations` attribute is composed of these tokens.

Before	**After: replace with the following**
For example, when an author pastes content from one topic into another topic within an XML editor, the application can use the `@domains` attribute to determine if the two topics use compatible domains. If not, copied content from the first topic might need to be generalized before it can be placed in the other topic. The `@domains` attribute serves the same function when an element uses the `@conref` attribute to reference a more specialized version of the element. For example, a `<note>` element in a concept topic conrefs a `<hazardstatement>` element in a reference document. If the hazard statement domain is not available in the concept topic, the `<hazardstatement>` element is generalized to a `<note>` element when the content reference is resolved. *Example of task with element domains*	
Syntax and rules Each domain and constraint module MUST provide a value for use by the `@domains` attribute. Each structural vocabulary module SHOULD provide a value for use by the `@domains` attribute, and it MUST do so when it has a dependency on elements from any module that is not part of its specialization ancestry. Values provided for the `@domains` attribute values are specified from root module (map or topic) to the provided module.	**Syntax and rules** The `@props` and `@base` attributes are the only two core attributes available for specialization. Each specialization of one of these attributes *MUST* provide a token for use by the `@specializations` attribute.
structural modules *[rules for this section]* **structural modules with dependencies** *[rules for this section]* **element domains** *[rules for this section]* **structural constraint modules** *[rules for this section]* **domain constraint modules** *[rules for this section]*	*Delete all of these sections from 2.0*
attribute domains The value uses an "a" before the initial parenthesis to indicate an attribute domain. Within the parenthesis, the value includes the attribute specialization hierarchy, starting with `@props` or `@base`: `'a(', props-or-base, (' ', attname)+, ')'` For example, the `@mySelectAttribute` specialized from `@props` results in the following value: `a(props mySelectAttribute)`	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`

Before	**After: replace with the following**
	• If that specialized attribute `@myFirstBase` is further specialized to create `@mySecondBase`, this results in the following token: `@base/myFirstBase/mySecondBase`

From Element generalization, remove the section on "Generalization and conref", which no longer applies.

From Processor expectations when generalizing elements, remove the middle table which involves checking `@domains`.

From DTD: Coding requirements for attribute domain modules:

Before	**After**
The vocabulary modules that define attribute domains have additional coding requirements. The module must include a parameter entity for the new attribute, which can be referenced in document-type shells, as well as a text entity that specifies the contribution to the `@domains` attribute for the attribute domain.	The vocabulary modules that define attribute domains have additional coding requirements. The module must include a parameter entity for the new attribute, which can be referenced in document-type shells, as well as a text entity that specifies the contribution to the `@specializations` attribute for the attribute domain.
The text entity name is the attribute domain name, followed by the literal `-d-Att`. The value of the text entity is the `@domains` attribute contribution for the module; see *domains attribute rules and syntax* for details on how to construct this value.	The text entity name is the attribute domain name, followed by the literal `-d-Att`. The value of the text entity is the `@specializations` attribute contribution for the module; see *@specializations attribute rules and syntax* for details on how to construct this value.

From DTD: Coding requirements for constraint modules:

Before	**After**

Structural constraint modules have the following requirements:
 	
@domains contribution entity name and value
The constraint module needs to contain a declaration for a text entity with the name *`tagname`*`-constraints`, where *tagname* is the name of the element type to which the constraints apply. The value of the text entity is the `@domains` attribute contribution for the module; see *domains attribute rules and syntax* for details on how to construct this value.
 	
For example, the following text entity provides the declaration for the strict task constraint that is shipped with the DITA standard.
 	
`<!ENTITY taskbody-constraints`
`  "(topic task strictTaskbody-c)"`
`>`
	[delete full`<dlentry>` about `@domains`]

Domain constraint modules have the following requirements: **@domains contribution entity name and value** The constraint module needs to contain a declaration for a text entity with the name *`domain`*`Domain-constraints`, where *domain* is the name of the domain to which the constraints apply, for example,	[delete full`<dlentry>` about `@domains`]

Before	After

"Highlighting" or "Programming". The value of the text entity is the `@domains` attribute contribution for the module; see *domains attribute rules and syntax* for details on how to construct this value.
 	
For example, the following text entity provides the declaration for a constraint module that restricts the highlighting domain:
 	
`<!ENTITY HighlightingDomain-constraints`
 	
`  "(topic hi-d basic-HighlightingDomain-c)"`
`>`	

When element domains are used to extend a base element, those extensions can be used to replace the base element. This form of constraint is done inside the document-type shell. Within a document-type shell, domain extensions are implemented by declaring an entity for a base element. The value of the entity can omit any base element types from which the other element types that are listed are specialized. Omitting a base type constitutes a form of constraint; as with any other constraint, this form of constraint must contribute a token to the `@domains` attribute. That token can be defined in a module file (which does not define any other entities or values), or the token can be placed directly into the document-type shell definition for the `included-domains` entity. In the following example, the `<pre>` base type is removed from the entity declaration, effectively allowing only specializations of `<pre>` but not `<pre>` itself. This omission would require the use of a `@domains` contribution token within the `included-domains` entity.	When element domains are used to extend a base element, those extensions can be used to replace the base element. This form of constraint is done inside the document-type shell. Within a document-type shell, domain extensions are implemented by declaring an entity for a base element. The value of the entity can omit any base element types from which the other element types that are listed are specialized. Omitting a base type constitutes a form of constraint~~; as with any other constraint, this form of constraint must contribute a token to the `@domains` attribute. That token can be defined in a module file (which does not define any other entities or values), or the token can be placed directly into the document-type shell definition for the `included-domains` entity~~. In the following example, the `<pre>` base type is removed from the entity declaration, effectively allowing only specializations of `<pre>` but not `<pre>` itself. ~~This omission would require the use of a `@domains` contribution token within the `included-domains` entity.~~

From DTD: Coding requirements for document-type shells:

Before	After

Domains attribute override
 	
This section sets the effective value of the `@domains` attribute for the top-level document type that is configured by the document type shell. It redefines the `included-domains` entity to include the text entity for each domain, constraint, and structural specialization that is either included or reused in the document type shell.
 	
In the following example, entities are included for both the troubleshooting specialization and the task specialization on which the troubleshooting specialization depends; for the highlighting and utilities element domains; for the `newAtt-d` attribute domain, and for the `noBasePre-c` constraint module:
 	
`<!ENTITY included-domains`
`    "&troubleshooting-att;`
	Specializations attribute override
	
	This section sets the effective value of the `@specializations` attribute for the top-level document type that is configured by the document type shell. It redefines the `included-domains` entity to include the text entity for each attribute domain ~~for each domain, constraint, and structural specialization~~ that is either included or reused in the document type shell.
	
	In the following example, entities are included ~~for both the troubleshooting specialization and the task specialization on which the troubleshooting specialization depends; for the highlighting and utilities element domains;~~ for the `newAtt-d` and `deliveryTarget-d` attribute domains:
	
	`<!ENTITY included-domains`
	`    "&deliveryTarget-d-att;`

Before	After

```
        &task-att;
        &hi-d-att;
        &ut-d-att;
        &newAtt-d-att;
        &noBasePre-c-ph;
      "
  >
```
 Note Although parameter entities (entities that begin with "%") must be defined before they are referenced, text entities (entities that begin with "&") can be referenced before they are defined. This allows the `included-domains` entity to include the constraint entity, which is not defined until the constraint module is referenced later in the document type shell.
	```
	        &newAtt-d-att;
	      "
	  >
	```
	 Note ~~Although parameter entities (entities that begin with "%") must be defined before they are referenced, text entities (entities that begin with "&") can be referenced before they are defined. This allows the `included-domains` entity to include the constraint entity, which is not defined until the constraint module is referenced later in the document type shell.~~

From DTD: Coding requirements for element type declarations:

Before	After
This topic covers general coding requirements for defining element types in both structural and element-domain vocabulary modules. In addition, it covers how to create the `@domains` attribute contribution for these modules.	This topic covers general coding requirements for defining element types in both structural and element-domain vocabulary modules. ~~In addition, it covers how to create the `@domains` attribute contribution for these modules.~~
`@domains` attribute contribution A domain declaration entity is used to construct the effective value of the `@domains` attribute for a map or topic type. **Text entity name** The name of the text entity is the structural type name or the domain abbreviation, followed by a hyphen ("-") and the literal `att`. **Text entity values** The value of the text entity is the `@domains` attribute contribution for the current module. See domains attribute rules and syntax for details on how to construct this value. For example, the `@domains` attribute contributions for the concept structural module and the highlighting domain module are are constructed as follows. • `<!ENTITY concept-att "(topic concept)">` • `<!ENTITY hi-d-att "(topic hi-d)">`.	[remove section]

From DTD: Coding requirements for structural modules

Before	After
The topic or map element type must set the `@DITAArchVersion` attribute to the version of DITA in use, typically by referencing the `arch-atts` parameter entity. It must also set the `@domains` attribute to the	The topic or map element type must set the `@DITAArchVersion` attribute to the version of DITA in use, typically by referencing the `arch-atts` parameter entity. It must also set the `@specializations` attribute

Before	After

`included-domains` entity. These attributes give processors a reliable way to check the architecture version and look up the list of domains available in the document type.
 	
The following example shows how these attributes are defined for the `<concept>` element in DITA 1.3:
 	
`<!ATTLIST concept`
`  %concept.attributes;`
`  %arch-atts;`
`  domains  CDATA  "&included-domains;">`
	to the `included-domains` entity. These attributes give processors a reliable way to check the architecture version and look up the list of specialized attributes available in the document type.
	
	The following example shows how these attributes are defined for the `<concept>` element in DITA 2.0:
	
	`<!ATTLIST concept`
	`  %concept.attributes;`
	`  %arch-atts;`
	`  specializations  CDATA  "&included-`
	`domains;">`

From RELAX NG: Coding requirements for attribute domain modules:

Before	After
All vocabulary and constraint modules must document their `@domains` attribute contribution. The value of the contribution is constructed according to the rules found in *domains attribute rules and syntax*.	All vocabulary and constraint modules must document their `@specializations` attribute contribution. The value of the contribution is constructed according to the rules found in *@specializations attribute rules and syntax*.
Domains attribute contribution The `@domains` contribution must be documented in the module. The value is constructed according to the rules found in *domains attribute rules and syntax*.	**Specializations attribute contribution** The `@specializations` contribution must be documented in the module. The value is constructed according to the rules found in *@specializations attribute rules and syntax*.

From RELAX NG: Coding requirements for document-type shells: change equivalent to matching DTD topic above.

From DTD: Coding requirements for element type declarations: change equiavlent to matching DTD topic above.

From RELAX NG: Coding requirements for structural modules: change equivalent to matching DTD topic above.

From RELAX NG: Coding requirements for element domain modules: change equivalent to matching DTD topic above.

From RELAX NG: Coding requirements for constraint modules: change equivalent to matching DTD topic above.

From architectural attributes:

Before	After
@domains This attribute identifies the domain modules (and optionally the structural modules) that are used in a map or topic. Each module also declares its module dependencies.	**@specializations** This attribute identifies the specialized attributes that are used in a map or topic.

From Processing conrefs:

Before	After
When pulling content with the conref mechanism, if the referenced element is the same type as the referencing element, and the set of domains declared on the `@domains`	~~When pulling content with the conref mechanism, if the referenced element is the same type as the referencing element, and the set of domains declared on the `@domains`~~

Before	After
attribute in the referenced topic or map instance is the same as or a subset of the domains declared in the referencing document, the element set allowed in the referenced element is guaranteed to be the same as, or a subset of, the element set allowed in the referencing element. When pushing content with the conref mechanism, the domain checking algorithm is reversed. In this case, if the set of domains declared on the `@domains` attribute in the referencing topic or map instance is the same as or a subset of the domains declared in the referenced document, the element set allowed in the pushed content is guaranteed to be the same as, or a subset of, the element set allowed in the new location. When both pulling or pushing content with the conref mechanism, processors resolving conrefs **SHOULD** tolerate specializations of valid elements and generalize elements in the pushed or pulled content fragment as needed for the resolving context. Except where allowed by weak constraints, a conref processor **MUST NOT** permit resolution of a reuse relationship that could be rendered invalid under the rules of either the reused or reusing content.	~~attribute in the referenced topic or map instance is the same as or a subset of the domains declared in the referencing document, the element set allowed in the referenced element is guaranteed to be the same as, or a subset of, the element set allowed in the referencing element.~~ ~~When pushing content with the conref mechanism, the domain checking algorithm is reversed. In this case, if the set of domains declared on the `@domains` attribute in the referencing topic or map instance is the same as or a subset of the domains declared in the referenced document, the element set allowed in the pushed content is guaranteed to be the same as, or a subset of, the element set allowed in the new location.~~ When content is reused between two documents with different domains or constraints, it is possible for the reused content to include domain extensions that are not defined for the new context, or to include elements that would be constrained out of the new context. When ~~both~~ pulling or pushing content with the conref mechanism, processors resolving conrefs **SHOULD** tolerate specializations of valid elements. Processors **MAY** generalize elements in the pushed or pulled content fragment as needed for the resolving context. ~~Except where allowed by weak constraints, a~~A conref processor **MUST NOT** permit resolution of a reuse relationship that ~~could be rendered~~ is known to be invalid under the rules of either the reused or reusing content.

From constraint rules:

Before	After
Contribution to the @domains attribute Each constraint that is integrated into a DITA document type **MUST** be declared in the `@domains` attribute for each structural type that is integrated into the document type. For DTDs, the contribution for the `@domains` attribute is specified in the constraint module file; for XSD and RELAX NG, the contribution to the `@domains` attribute is specified directly in the document type shell.	~~**Contribution to the @domains attribute**~~ ~~Each constraint that is integrated into a DITA document type **MUST** be declared in the `@domains` attribute for each structural type that is integrated into the document type.~~ ~~For DTDs, the contribution for the `@domains` attribute is specified in the constraint module file; for XSD and RELAX NG, the contribution to the `@domains` attribute is specified directly in the document type shell.~~

Remove the topic Conref compatibility with constraints, which explains how to resolve conref with weak vs strong constraints; this process is obsolete with the removal of the domain tokens.

Remove the topic Weak and strong constraints, which explains weak vs strong constraints; this distinction is obsolete with the removal of the domain tokens.

From Equivalence of document-type shells:

Before	After
A DITA document type is defined by the following:	A DITA document type is defined by the following: • The set of vocabulary and constraint modules that are integrated by the document type shell

Before	After
• The set of modules that are declared in the `@domains` attribute on the root element of the document • The values of the `@class` attributes of all the elements in the document • Rules for topic nesting	• The values of the `@class` attributes of all the elements in the document • Rules for topic nesting

From Overview of document type shells:

Before	After
A DITA document must either have an associated document-type definition or all required attributes must be made explicit in the document instances. Most DITA documents have an associated document-type shell. DITA documents that reference a document-type shell can be validated using standard XML processors. Such validation enables processors to read the XML grammar files and determine default values for the `@domains` and `@class` attributes.	A DITA document must either have an associated document-type definition or all required attributes must be made explicit in the document instances. Most DITA documents have an associated document-type shell. DITA documents that reference a document-type shell can be validated using standard XML processors. Such validation enables processors to read the XML grammar files and determine default values for the ~~`@domains`~~`@specializations` and `@class` attributes.

From Vocabulary modules:

Before	After
The name (or short name) of an element domain module is used to identify the module in `@class` and `@domains` attribute values.	The name (or short name) of an element domain module is used to identify the module in `@class` ~~and `@domains`~~ attribute value~~s~~.

Migration plans for backwards incompatibilities

Processors will need to be updated to understand the new attribute name and syntax.

The specification already recommends against specifying `@domains` in source files (it can instead be read from the grammar files). However, it is legal to include it in source. If included, a search/replace expression that finds the `@domains` attribute in source files can be used to rename it to `@specializations`.

Remaining modifications affect grammar files rather than source, which cannot easily be automated. The following items in grammar files will need to be migrated manually:

- Specialized topics and maps must change the declaration of `@domains` on the topic or map element to be named `@specializations`
- Attribute domains will need to be updated to use the new `@domains` token syntax.
- Configured document type shells will remove all domain tokens (apart from attribute domains) from the `included-domains` entity.
- Configured document type shells will remove references to ENT files that do nothing but declare an entity for structural domain contributions to `@domains`
- All modules (except attribute domain modules) will remove declarations of existing tokens.