
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

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

basemap.rng (before)	basemap.rng (after)
<pre> 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> </pre>	<pre> <attribute name="specializations" a:defaultValue=" @props/audience @props/ deliveryTarget @props/platform @props/product @props/otherprops"/ > </optional> </define> </pre>

basetopic.rng (before)	basetopic.rng (after)
<pre> <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> </pre>	<pre> <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> </pre>

Audience, deliveryTarget, platform, product, and otherprops RNG modules (before)	Audience, deliveryTarget, platform, product, and otherprops RNG modules (after)
<p>Using this pattern:</p> <pre> <domainsContribution>a(props audience)</ domainsContribution> </pre>	<p>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:</p> <pre> <domainsContribution>@props/audience</ domainsContribution> </pre>

Note Question raised by Eliot during the stage 3 review - I think this probably makes sense but wanted to bring to the TC:

[Should] the RNG <domainsContribution> element should be renamed to <specializationsContribution>--anything that processes the current <domainsContribution> (i.e., the XQuery George and I wrote to generate the @domains value in RNG shells) would

need to be updated in any case, so why not correct the tag name as well? I think this is a code change that would only affect me at this point (and maybe Syncro Soft).

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

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

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
<p>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.</p> <p>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.</p> <p>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.</p> <p>The @domains attribute serves the same function when an element uses the @conref attribute to reference a more</p>	<p>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.</p>

Before	After: replace with the following
<p>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.</p> <p><i>Example of task with element domains</i></p>	
<p>Syntax and rules</p> <p>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.</p> <p>Values provided for the @domains attribute values are specified from root module (map or topic) to the provided module.</p>	<p>Syntax and rules</p> <p>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.</p>
<p>structural modules <i>[rules for this section]</i></p> <p>structural modules with dependencies <i>[rules for this section]</i></p> <p>element domains <i>[rules for this section]</i></p> <p>structural constraint modules <i>[rules for this section]</i></p> <p>domain constraint modules <i>[rules for this section]</i></p>	<p><i>Delete all of these sections from 2.0</i></p>
<p>attribute domains</p> <p>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:</p> <pre>'a(', props-or-base, (' ', atname)+, ')'</pre> <p>For example, the @mySelectAttribute specialized from @props results in the following value: a (props mySelectAttribute)</p>	<p>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:</p> <pre>'@', props-or-base, ('/', atname)+</pre> <p>For example:</p> <ul style="list-style-type: none"> • 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

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
<p>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.</p>	<p>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.</p>
<p>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 <i>domains attribute rules and syntax</i> for details on how to construct this value.</p>	<p>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 <i>@specializations attribute rules and syntax</i> for details on how to construct this value.</p>

From [DTD: Coding requirements for constraint modules](#):

Before	After
<p>Structural constraint modules have the following requirements:</p> <p>@domains contribution entity name and value</p> <p>The constraint module needs to contain a declaration for a text entity with the name <i>tagname-constraints</i>, where <i>tagname</i> 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 <i>domains attribute rules and syntax</i> for details on how to construct this value.</p> <p>For example, the following text entity provides the declaration for the strict task constraint that is shipped with the DITA standard.</p> <pre data-bbox="256 1241 802 1339" style="background-color: #f0f0f0; padding: 5px;"> <!ENTITY taskbody-constraints "(topic task strictTaskbody-c)" ></pre>	<p>[delete full<dentry> about @domains]</p>
<p>Domain constraint modules have the following requirements:</p> <p>@domains contribution entity name and value</p> <p>The constraint module needs to contain a declaration for a text entity with the name <i>domainDomain-constraints</i>, where <i>domain</i> is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming". The value of the text entity is the @domains attribute contribution for the module; see <i>domains attribute rules and syntax</i> for details on how to construct this value.</p> <p>For example, the following text entity provides the declaration for a constraint module that restricts the highlighting domain:</p> <pre data-bbox="256 1801 802 1852" style="background-color: #f0f0f0; padding: 5px;"> <!ENTITY HighlightingDomain-constraints</pre>	<p>[delete full<dentry> about @domains]</p>

Before	After
<pre>"(topic hi-d basic-HighlightingDomain- c)" ></pre>	
<p>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.</p> <p>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.</p> <p>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.</p>	<p>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.</p> <p>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.</p> <p>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.</p>

From [DTD: Coding requirements for document-type shells](#):

Before	After
<p>Domains attribute override</p> <p>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.</p> <p>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:</p> <pre><!ENTITY included-domains "&troubleshooting-att; &task-att; &hi-d-att; &ut-d-att; &newAtt-d-att; &noBasePre-c-ph; " ></pre> <p>Note Although parameter entities (entities that begin with "%") must be defined before they are referenced, text entities (entities that begin with "&") can be referenced</p>	<p>Specializations attribute override</p> <p>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.</p> <p>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:</p> <pre><!ENTITY included-domains "&deliveryTarget-d-att; &newAtt-d-att; " ></pre> <p>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</p>

Before	After
before they are defined. This allows the <code>included-domains</code> entity to include the constraint entity, which is not defined until the constraint module is referenced later in the document type shell.	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 <code>@domains</code> 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 <code>@domains</code> attribute contribution for these modules.
<p>@domains attribute contribution</p> <p>A domain declaration entity is used to construct the effective value of the <code>@domains</code> attribute for a map or topic type.</p> <p>Text entity name</p> <p>The name of the text entity is the structural type name or the domain abbreviation, followed by a hyphen ("-") and the literal <code>att</code>.</p> <p>Text entity values</p> <p>The value of the text entity is the <code>@domains</code> attribute contribution for the current module. See <code>domains</code> attribute rules and syntax for details on how to construct this value.</p> <p>For example, the <code>@domains</code> attribute contributions for the <code>concept</code> structural module and the <code>highlighting</code> domain module are constructed as follows.</p> <ul style="list-style-type: none"> <code><!ENTITY concept-att "(topic concept)"></code> <code><!ENTITY hi-d-att "(topic hi-d)"></code>. 	[remove section]

From [DTD: Coding requirements for structural modules](#)

Before	After
<p>The topic or map element type must set the <code>@DITAArchVersion</code> attribute to the version of DITA in use, typically by referencing the <code>arch-atts</code> parameter entity. It must also set the <code>@domains</code> attribute to the <code>included-domains</code> 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.</p> <p>The following example shows how these attributes are defined for the <code><concept></code> element in DITA 1.3:</p> <pre><!ATTLIST concept %concept.attributes; %arch-atts;</pre>	<p>The topic or map element type must set the <code>@DITAArchVersion</code> attribute to the version of DITA in use, typically by referencing the <code>arch-atts</code> parameter entity. It must also set the <code>@specializations</code> attribute to the <code>included-domains</code> 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.</p> <p>The following example shows how these attributes are defined for the <code><concept></code> element in DITA 2.0:</p> <pre><!ATTLIST concept %concept.attributes; %arch-atts;</pre>

Before	After
<code>domains CDATA "&included-domains;"></code>	<code>specializations CDATA "&included-domains;"></code>

From [RELAX NG: Coding requirements for attribute domain modules](#):

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

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 equivalent 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
<p>@domains</p> <p>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.</p>	<p>@specializations</p> <p>This attribute identifies the specialized attributes that are used in a map or topic.</p>

From [Processing conrefs](#):

Before	After
<p>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 <code>@domains</code> 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.</p> <p>When pushing content with the conref mechanism, the domain checking algorithm is reversed. In this case, if the set of domains declared on the <code>@domains</code> 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</p>	<p>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 <code>@domains</code> 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.</p> <p>When pushing content with the conref mechanism, the domain checking algorithm is reversed. In this case, if the set of domains declared on the <code>@domains</code> 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</p>

Before	After
<p>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.</p> <p>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.</p> <p>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.</p>	<p>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.</p> <p>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.</p> <p>Except where allowed by weak constraints, 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.</p>

From [constraint rules](#):

Before	After
<p>Contribution to the @domains attribute</p> <p>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.</p> <p>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.</p>	<p>Contribution to the @domains attribute</p> <p>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.</p> <p>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.</p>

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
<p>A DITA document type is defined by the following:</p> <ul style="list-style-type: none"> 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 	<p>A DITA document type is defined by the following:</p> <ul style="list-style-type: none"> The set of vocabulary and constraint modules 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

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 values.

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.