Eberlein
Consulting

DITA training, information architecture, and strategy

Constraints

Contents

DTD: Coding requirements for constraint modules

A structural constraint module defines the constraints for a map or topic element type. A domain constraint module defines the constraints for a domain module.

Structural constraint modules

Structural constraint modules have the following requirements:

File names

Structural constraint modules *SHOULD* be named using the following format:

```
qualifierTagnameConstraint.mod
```

where:

- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "strict" or "requiredTitle" or "myCompany-".
- *Tagname* is the element type name with an initial capital, for example, "Taskbody."

For example, the file name for the constraint that is applied to the general task to create the strict task type is `strictTaskbodyConstraint.mod`.

Parameter entity name and value

The constraint module *MUST* contain a declaration for a general text entity with the following name:

```
"tagname-constraints"
```

where *tagname* is the name of the element type to which the constraints apply.

It also *MUST* contain the replacement text for the entity in the following format:

```
"(inheritance-hierachy qualifierTagname-c)"
```

where:

- *inheritance-hierachy* is the specialization hierarchy, for example, `topic task`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "strict" or "requiredTitle" or "myCompany-".
- *Tagname* is the element type name with an initial capital, for example, "Taskbody"."
- The literal "-c" indicates that the name is the name of a constraint.

For example, the following code provides the declaration for the strict task constraint that is shipped with the DITA standard.

```
<!ENTITY taskbody-constraints
  "(topic task strictTaskbody-c)"
>
```

Declaration of the `%tagname.content;` parameter entity

There also *MUST* be a declaration of the `%tagname.content;` parameter entity that defines the constrained content model.

The following parameter entity defines a more restricted content model for `<topic>`, in which either the `<abstract>` or `<shortdesc>` element is required.

```
<!ENTITY  topic-constraints  "(topic strictTopic-c)">

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

<!ENTITY % topic.content

  "((%title;),
    (%titlealts;)?,
    (%shortdesc;|
     %abstract;),
    (%prolog;)?,
    (%body;)?,
    (%topic-info-types;)*)"
>
```

Domain constraint modules

Domain constraint modules have the following requirements:

File names

Domain constraint modules *SHOULD* be named using the following format:

```
qualifierdomainDomainConstraint.mod
```

where:

- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "noSyntaxDiagram" or "myCompany-".
- *domain* is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming".

For example, the file name for a constraint module that removes the syntax diagram from the programming domain might be `noSyntaxDiagramProgrammingDomainConstraint.mod`.

Parameter entity name and value

The constraint module *MUST* contain a declaration for a general text entity with the following name:

```
 "DomainDomain-constraints"
```

where *domain* is the name of the domain to which the constraints apply, for example "Highlighting" or "Programming".

It also *MUST* contain the replacement text for the entity in the following format:

```
"(inheritance-hierachy qualifierdomainDomain-c)"
```

where:

- *inheritance-hierachy* is the specialization hierarchy, for example, `topic hi-d`.
- *qualifier* is a string that is specific to the constraints module and characterizes it, for example, "noSyntaxDiagram" or "myCompany-".
- *domain* is the name of the domain to which the constraints apply, for example, "Highlighting" or "Programming".

- The literal "-c" indicates that the name is the name of a constraint.

Declaration of the `%tagname.content;` parameter entity

There also *MUST* be a declaration of the `%tagname.content;` parameter entity that defines the constrained content model.

Requirements for document type shells

Information on how to integrate a constraint module into a DTD-based, document-type shell can be found in *DTD document-type shell: Coding requirements*.

Examples: Constraints

This section of the specification contains examples and scenarios. They illustrate a variety of ways that constraints can be used; they also provide examples of the DTD coding requirements for constraints and how constraints are integrated into document-type shells.

Example: Redefine the content model for a topic type

In this scenario, an information architect for Acme, Incorporated wants to redefine the content model for the topic document type. She wants to omit the `<abstract>` element and make the `<shortdesc>` element required; she also wants to omit the `<related-links>` element and disallow topic nesting.

1. She creates a .mod file using the following naming conventions: `*qualiferTagname*Constraint.mod`, where *qualifer* is a string the describes the constraint, and *Tagname* is the element type name with an initial capital. Her contraint module is named `acme-TopicConstraint.mod`.
2. She adds the following content to `acme-TopicConstraint.mod`:

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

<!-- Declares the entity for the constraint module and defines     -->
<!-- its contribution to the @domains attribute.                   -->

<!ENTITY topic-constraints
  "(topic basic-Topic-c)"
>

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

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

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

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

3. She then integrates the constraint module into her document-type shell for topic by adding the following section above the "TOPIC ELEMENT INTEGRATION" comment:

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

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

4. She then adds the constraint to the list of domains and constraints that need to be included in the value of the `@domains` attribute for `<topic>`:

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

<!ENTITY included-domains
                          "&hi-d-att;
                           &ut-d-att;
                           &indexing-d-att;
                           &topic-constraints;
  "
>
```

5. After updating the `catalog.xml` file to include the new constraints file, her work is done.

Example: Constrain attributes for the `<section>` element

In this scenario, an information architect wants to modify the content model for the `<section>` element. He wants to make the `@id` attribute required and omit the `@spectitle` attribute.

1. He creates a .mod file named *qualifer*`SectionContraint.mod`, where *qualifer* is a string that characterizes the constraint.
2. He adds the following content to *qualifer*`SectionContraint.mod`:

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

<!ENTITY section-constraints
  "(topic idRequired-section-c)"
>

<!-- Declares the entities referenced in the constrained content  -->
<!-- model.                                                       -->
<!ENTITY % conref-atts
             'conref    CDATA #IMPLIED
              conrefend CDATA #IMPLIED
              conaction (mark|pushafter|pushbefore|pushreplace|-dita-use-conref-target) #IMPLIED
              conkeyref CDATA #IMPLIED' >
<!ENTITY % filter-atts
             'props      CDATA #IMPLIED
              platform   CDATA #IMPLIED
              product    CDATA #IMPLIED
              audience   CDATA #IMPLIED
              otherprops 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|unchanged|-dita-use-conref-target) #IMPLIED' >
<!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' >

<!-- Declares the constrained content model. Original definition   -->
<!-- included %univ-atts;, spectitle, and outputclass; now includes-->
<!-- individual pieces of univ-atts, to make ID required.          -->

<!ENTITY % section.attributes
          "id          CDATA   #REQUIRED
```

```
            %conref-atts;
            %select-atts;
            %localization-atts;
            outputclass CDATA   #IMPLIED">
```

Note: The information architect had to declare all the parameter entities that are referenced in his constrained content model for `<section>`. If he did not do so, none of the attributes declared in the `%conref-atts;`, `%select-atts;`, or `%localization-atts;` 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 `%filter-atts;` parameter entity.

3. He then integrates the constraint module into the applicable document-type shells and adds it to his `catalog.xml` file.

Example: Constrain a domain module

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

1. She creates an .ent file named *`qualifer`*`HighlightingDomainContraint.ent`, where *qualifer* is a string that characterizes the constraint.
2. She adds the following content to *`qualifer`*`HighlightingDomainContraint.ent`:

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

<!ENTITY HighlightingDomain-constraints
  "(topic hi-d basic-HighlightingDomain-c)"
>

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

3. She then integrates the constraint module into her company-specific, document-type shell for the task topic by adding the following section directly before the "DOMAIN ENTITY DECLARATIONS" comment:

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

<!ENTITY % HighlightingDomain-c-dec
  PUBLIC "-//ACME//ENTITIES DITA Highlighting Domain Constraint//EN"
  "acme-HighlightingDomainConstraint.ent"
>%basic-HighlightingDomain-c-dec;
```

4. In the "DOMAIN EXTENSIONS" section, she replaces the parameter entity for the highlighting domain with the parameter entity for the constrained highlighting domain:

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

5. She then adds the constraint to the list of domains and constraints that need to be included in the value of the `@domains` attribute for `<task>`:

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

```
<!ENTITY included-domains
                          "&task-att;
                           &hi-d-att;
                           &indexing-d-att;
                           &pr-d-att;
                           &sw-d-att;
                           &ui-d-att;
                           &taskbody-constraints;
                           &HighlightingDomain-constraints;
  "
>
```

6. After updating the `catalog.xml` file to include the new constraints file, her work is done.

Example: Replace a base element with domain extensions

In this scenario, an information 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. The information architect creates an entities file named *`qualifer`*`PhContraint.ent`, where *qualifer* is a string that characterizes the constraint.
2. The information architect adds the following content to *`qualifer`*`PhContraint.ent`:

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

<!ENTITY ph-constraints
  "(topic noPh-ph-c)"
>
```

Note: Because the highlighting and programming domains cannot be generalized without the `<ph>` element, this entity must be defined so that there is a separate parenthetical expression that can be included in the `@domains` attribute for the topic.

3. The information architect then integrates the constraint module into a document-type shell for concept by adding the following section above the "TOPIC ELEMENT INTEGRATION" comment:

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

<!ENTITY % noPh-ph-c-def
  PUBLIC "-//ACME//ELEMENTS DITA Ph Constraint//EN"
  "acme-PhConstraint-constraints">
%noPh-ph-c-def;
```

4. In the "DOMAIN EXTENSIONS" section, the information 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;
                        ">
```

5. She then adds the constraint to the list of domains and constraints that need to be included in the value of the `@domains` attribute:

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

```
<!ENTITY included-domains
                              "&concept-att;
                               &hi-d-att;
                               &indexing-d-att;
                               &pr-d-att;
                               &sw-d-att;
                               &ui-d-att;
                               &ph-constraint;
  "
>
```

6. After updating the `catalog.xml` file to include the new constraints file, the information architect's work is done.

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

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

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

File name	What it constrains	Details	Contribution to the `@domains` attribute
`example-TopicConstraint.mod`	`<topic>`	• Removes `<abstract>` • Makes `<shortdesc>` required • Removes `<related-links>` • Disallows topic nesting	`(topic basic-Topic-c)`
`example-SectionConstraint.mod`	`<section>`	• Makes `@id` required • Removes `@spectitle` attribute	`(topic idRequired-section-c)`
`example-HighlightingDomainConstraint.mod`	Highlighting domain	Reduces the highlighting domain elements to `<b>` and `<i>`	`(topic hi-d basic-HighlightingDomain-c)`
`example-PhConstraint.mod`	`<ph>`	Removes the `<ph >`element	(topic noPh-ph-c)

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 three constraints are listed does not matter.

Each constraint module provides a unique contribution to the `@domains` attribute. When integrated into the document-type shell for `<topic>`, the effective value of the domains attribute will include the following values, as well as values for any other modules that are integrated into the document-type shell:

```
(topic basic-Topic-c) (topic idRequired-section-c) (topic hi-d basic-HighlightingDomain-c) (topic
noPh-ph-c)
```

Index