

How to make changes in DITA DTDs for specialization

DITA dtds are divided into smaller modules, based on the base elements hierarchy (Topic and Map) and their respective domain and structural specializations like Task, Concept, BookMap, UIDomain, Programing Domain etc. There are fixed set of changes which need to be done in DTDs which are defined below.

Making changes in DTDs for structural specialization

For structural specialization, we need to create a new DTD file with specialized elements and then we have to integrate it with the existing DITA DTDs , the way we want our final output to work. If we want to make our specialized elements types work with the existing topic/map hierarchies, we can add our specialization to ditabase.dtd or we can create a separate dtd. We will take an example where user want to define a new specialized object element with only specialized xref and footnote elements as its content model. The Steps to perform are

TASK

1. Copy any existing MOD file and rename it. e.g. refrence.mod to objectsp.mod.
2. Open the new mod file e.g. objectsp.mod and in section "SPECIALIZATION OF DECLARED ELEMENTS", change infotype declaration to the newly declared structured type, which will be required for integrating the specialized modules with the existing ones.

```
<!------->
<!-- SPECIALIZATION OF DECLARED ELEMENTS -->
<!------->
<ENTITY % objectsp-info-types "%info-types;" >
```

ADDITIONAL INFORMATION: In next 3 steps, remove the existing stuff in the respected heads and add the new elements related information. We copy existing DTds mainly for the purpose that we get a formatted structure defined for declaring our new elements.

3. Declare the new entities for the specialized elements required till the top of the hierarchy.

```
<!------->
<!-- ELEMENT NAME ENTITIES -->
<!------->
```

```
<!ENTITY % myobjecttype "myobjecttype" >
<!ENTITY % mybody      "mybody" >
<!ENTITY % myp         "myp" >
<!ENTITY % myobject    "myobject" >
<!ENTITY % myxref      "myxref" >
<!ENTITY % myfootnote  "myfootnote" >
<!--===== -->
```

4. Declare the new specialized elements.

```
<!--===== -->
<!-- ELEMENT DECLARATIONS -->
<!--===== -->
<!--                LONG NAME: myobjecttype -->
<!ELEMENT myobject ((%myxref;)*, (%myfootnote;)*)>
<!ATTLIST myobject
declare      (declare) #IMPLIED
classid     CDATA #IMPLIED
codebase    CDATA #IMPLIED
data        CDATA #IMPLIED
type        CDATA #IMPLIED
codetype    CDATA #IMPLIED
archive     CDATA #IMPLIED
standby     CDATA #IMPLIED
height      NMTOKEN #IMPLIED
width       NMTOKEN #IMPLIED
usemap      CDATA #IMPLIED
name        CDATA #IMPLIED
tabindex    NMTOKEN #IMPLIED
longdescref CDATA #IMPLIED
%univ-atts;
outputclass CDATA #IMPLIED
longdescre  CDATA #IMPLIED >
```

----- and so on for other elements

5. In "SPECIALIZATION ATTRIBUTE DECLARATIONS" section, declare the element from which the specialized element is derived from. We have to declared the hierarchy till the base topic/Map type (starting with a "--" for structural specialization) e.g. if the specialized element is derived from any reference element, we have to define the complete hierarchy

from specialized element to reference to topic (as reference is again specialized from topic).

```
<!--===== -->
<!-- SPECIALIZATION ATTRIBUTE DECLARATIONS -->
<!--===== -->

<!ATTLIST myobjecttype %global-atts; class CDATA "- topic/topic
myobjecttype/myobjecttype "      >

<!ATTLIST mybody      %global-atts; class CDATA "- topic/body
myobjecttype/mybody "      >

<!ATTLIST myp      %global-atts; class CDATA "- topic/p
myobjecttype/myp "      >

<!ATTLIST myxref      %global-atts; class CDATA "- topic/xref
myobjecttype/myxref "      >

<!ATTLIST myobject %global-atts; class CDATA "- topic/object
myobjecttype/myobject "      >

<!ATTLIST myfootnote %global-atts; class CDATA "- topic/fn
myobjecttype/myfootnote "      >
```

6. Integrate the new mod file with the existing ones by modifying database.dtd. For the specialization in the example stated above, add entry in the "TOPIC NESTING OVERRIDE" section for declaring the new type with the base types and in "TOPIC ELEMENT INTEGRATION" section for importing the mod file

```
<!--===== -->
<!-- TOPIC NESTING OVERRIDE -->

<!ENTITY % info-types "topic | concept | task | reference | ?
myobjecttype | glossentry" >

<!--===== -->
<!-- TOPIC ELEMENT INTEGRATION -->
<!--===== -->

<!-- Embed topic to get generic elements -->

<!ENTITY % topic-type PUBLIC "-//OASIS//ELEMENTS DITA
Topic//EN" ?"topic.mod" >

%topic-type;

<!ENTITY % objectsp-type PUBLIC "-//OASIS//ELEMENTS DITA
Topic//EN" ?"objectsp.mod"
>

%objectsp-type;
```

----- and the other existing ones

ADDITIONAL INFORMATION: If we want to restrict multiple topic types in a single topic type we can create a new integration file and not pull in all the topic types together like we did in the current example.

Making changes in DTDs for domain specialization

For domain specialization, we need to create 2 DTD files. In first file we declare the specialized elements and in the second dtd we declare the entities for integration related information as domain specialized elements should be available where ever their base element is. We will take an example where user want to define 3 new domain specialized elements for image, prolog and link respectively. The Steps to perform are

TASK

1. Copy any existing MOD file and rename it. e.g. utilitiesDomain.mod to domainsp.mod.

ADDITIONAL INFORMATION: In next 3 steps, remove the existing stuff in the respected heads and add the new elements related information. We copy existing DTDs mainly for the purpose that we get a formatted structure defined for declaring our new elements

2. Open the new mod file e.g. domainsp.mod and in section "ELEMENT NAME ENTITIES", declare the new entities for the specialized elements.

```
<!--===== -->
<!-- ELEMENT NAME ENTITIES -->
<!--===== -->

<!ENTITY % Dlink          "Dlink" >
<!ENTITY % Dprolog        "Dprolog" >
<!ENTITY % Dimage         "Dimage" >

<!--===== -->
```

3. Declare the new specialized elements.

```
<!--===== -->
<!-- ELEMENT DECLARATIONS -->
<!--===== -->
<!--          LONG NAME: Dimage -->

<!ELEMENT Dimage          (%alt;) >

<ATTLIST Dimage href       CDATA          #REQUIRED
keyref      NMTOKEN       #IMPLIED
alt         CDATA         #IMPLIED
longdescref CDATA         #IMPLIED
```

```

height      NMTOKEN      #IMPLIED
width       NMTOKEN      #IMPLIED
align       CDATA        #IMPLIED
scale       NMTOKEN      #IMPLIED
placement   (inline | break | -dita-use-coneref-target)
"inline"

%univ-atts; outputclass CDATA #IMPLIED  >

```

----- and so on for other elements.

4. In "SPECIALIZATION ATTRIBUTE DECLARATIONS" section, declare the element from which the specialized element is derived from. We have to declared the hierarchy till the base topic/Map type (starting with a "+" for domain specialization) e.g. if the specialized element is derived from any other utility domain element, we have to define the complete hierarchy from specialized element to utilities domain to topic (as utilities domain is again specialized from topic).

```

<!------->
<!-- SPECIALIZATION ATTRIBUTE DECLARATIONS -->
<!------->
<!ATTLIST Dprolog      %global-atts; class CDATA "+ topic/prolog
domainsp-d/Dprolog "  >
<!ATTLIST Dlink        %global-atts; class CDATA "+ topic/link
domainsp-d/Dlink "    >
<!ATTLIST Dimage       %global-atts; class CDATA "+ topic/image
domainsp-d/Dimage "   >

```

5. Next we need to define the ENT file which allows the elements to be substituted instead of being aggregated i.e. wherever the parent element is allowed, the specialized one should be allowed as well. Create a new ENT file (or copy any existing one).
6. Open the ENT file and declare the entities for integration of new elements with the existing one (using domain extensions)

```

<!------->
<!-- ELEMENT EXTENSION ENTITY DECLARATIONS -->
<!------->
<!ENTITY % domainsp-d-image      "Dimage" >
<!ENTITY % domainsp-d-link       "Dlink" >
<!ENTITY % domainsp-d-prolog     "Dprolog" >

```

7. In same ENT file, declare the domain attribute entity to define the ancestry till the root from which the elements are derived. If you are

specializing any element from some domain extension, then you need to declare till the top.

```
<!--===== -->  
<!-- DOMAIN ENTITY DECLARATION -->  
<!--===== -->  
<!ENTITY domainsp-d-att "(topic ank-d)">
```

8. Integrate the new mod file with the existing ones by modifying ditabase.dtd. For domain specialization we need to integrate both element definition dtd and the ENT file we created so we need to modify at 4 different places

a First we need to do the vocabulary declaration and define the new domain.

```
<!--===== -->  
<!-- DOMAIN ENTITY DECLARATIONS -->  
<!--===== -->  
<!ENTITY % domainsp-d-dec PUBLIC "-//domainsp//ENTITIES DITA  
domainsp Domain//EN" "domainsp.ent" >  
%domainsp-d-dec;
```

----- and the other existing ones

.

b Do the vocabulary substitution and define the elements from the which the domain specialized elements are extending from.

```
<!--===== -->  
<!-- DOMAIN DOMAIN EXTENSIONS-->  
<!--===== -->  
<!ENTITY % image "image | %domainsp-d-image;" >  
<!ENTITY % prolog "prolog | %domainsp-d-prolog;" >  
<!ENTITY % link "link | %domainsp-d-link;" >
```

----- and the other existing ones

c Do the vocabulary attribute declaration.

```
<!--===== -->  
<!-- DOMAINS ATTRIBUTE OVERRIDE-->  
<!--===== -->  
<!ENTITY included-domains "&ui-d-att; &hi-d-att; &pr-d-att;  
&sw-d-att; &ut-d-att; &indexing-d-att; &domainsp-d-att;" >
```

d Finally put the vocabulary definition and pull in the mod file for domain element integration. It will pull in all the specialized elements declared in the mod file.

```
<!--=====-->  
<!-- DOMAIN ELEMENT INTEGRATION -->  
<!--=====-->  
<!ENTITY % ank-d-def PUBLIC ?"-//domainsp//ELEMENTS DITA User  
Interface Domain//EN" "domainsp.mod" >  
%domainsp-d-def;
```

----- and the other existing ones
