Integrating DITA Specialization with ADOBE® FRAMEMAKER® 10

Adobe

Contents

DITA Specialization in Adobe® FrameMaker® 10

DITA specialization

About specialization

Specialization is the process of creating new designs based on existing designs. A specialization can reuse elements from higher-level designs. You can specialize DITA to create customized information models that meet your business requirements while retaining the benefits of the existing DITA architecture.

The DITA architecture provides for a general base topic and its three specialized variations: concept, task, and reference topic types. Each specialized topic type DTD declares elements that are specific and restricted to only that topic type. For example, the task topic DTD allows <step> and <choice> elements that are absent from other topic DTDs.

You can specialize task, concept, and reference DITA topic types when you require new structural types or new domains. For example, you can tweak your design to increase consistency or descriptiveness or to meet specific output needs.

Types of specialization

Specialization can be broadly categorized into two types:

Structural specialization Defines new topic or map structures derived from base topics and maps, such as concept, task, or reference.

Domain specialization Defines markup for a specific information domain or subject area, such as programming or hardware.

To specialize an existing component, add the specialization statements to your DITA files, clearly identifying the ancestry or evolutionary path of your specialization. This way you can retain at least a minimum level of semantic structure during information reuse or interchange.

Structural specialization
Structural specialization defines new types of structured information, such as new topic types or new map types. Structural specialization allows you to create new topic types and yet maintain compatibility with existing style sheets, transforms, and processes.
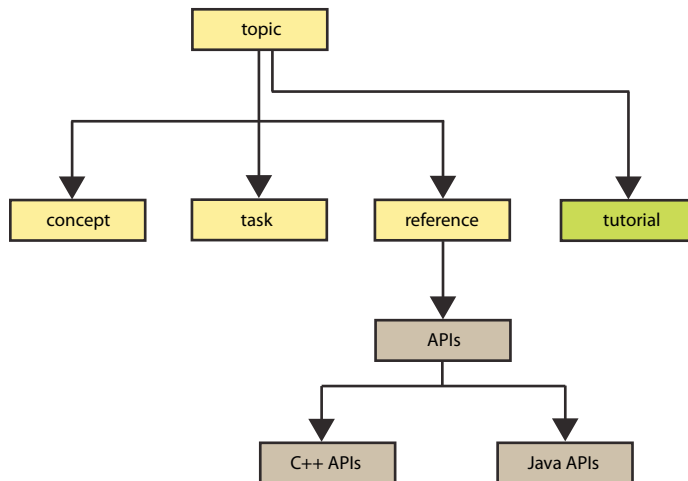
Shows the DITA base topics, concept, task, and reference, and specialized topics, tutorial, and APIs. API topic is further specialized into Java APIs and C++ APIs.

Structural specializations declare new top-level topic types and map types. You use structural specializations to define entirely new document structures. With structural specializations, you can specialize any base element type, including topic and map, as well as elements in any topic or map specialization.

Domain specialization

Domain specialization defines new types of elements, such as a new type of the <paragraph> element. These element specializations can be specific to a particular information domain or subject area, such as programming or hardware. For example, you can specialize the <screen> element from the user interface domain to catalog all dialog boxes.
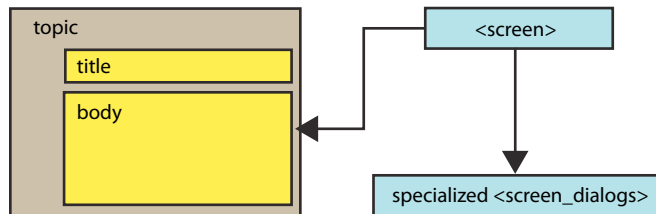
Domain specialization of the <screen> element to provide a consistent and specialized structure to dialog box documentation

When implementing domain specialization, you define new elements for use within any topic or map type. Using domain specializations, you can specialize any base DITA element that is an allowed descendant of <topic>, <map> or any element in any other domain module.

For example, map-specific elements <topicref>, <topichead>, and <topicgroup> are defined as domain elements, even though you can expect them to be part of the base <map> type. Therefore, defining specializations of <topicref> or <topichead> is domain specialization, not map specialization.

Why and when to use specialization?

You use specialization to define new structural types or new information domains. Specialization provides a way to adapt your design for increased consistency or descriptiveness. You can also use specialization for specific output demands that the current data model does not meet.

All specializations can be processed by existing transforms and can also be transformed back to more general equivalents.

Benefits of specialization include the following:

- You can reuse the base vocabulary to define specializations and save the time it takes to create them.

- Changes to base elements automatically percolate to the specializations.

- You can easily plug in modules depending on requirements.

- You can revert specializations to their base types easily.

- Interoperability or mapping from specialized type documents to base type documents is guaranteed.

Rules of specialization

You cannot make the content models of specialized element types less restrictive than the content models of their base types.

When you specialize one element from another, the new element must obey certain rules to be valid.

- A specialized element must have a content model equivalent to or more restrictive than its base element.

- A specialized element must have attributes that are equivalent to or a subset of the attributes of its base element.

- The attributes of a specialized element must have values or value ranges that are equivalent to or a subset of the values or value ranges of the base element.

- A specialized element must have a properly formed class attribute.

- Avoid overspecialized elements and crude specializations. Don't create specialized elements when existing elements suffice.

Specializing DITA in Adobe® FrameMaker® 10

The following graphic gives an overview of the specialization process for structural and domain specialization using Adobe® FrameMaker® 10.
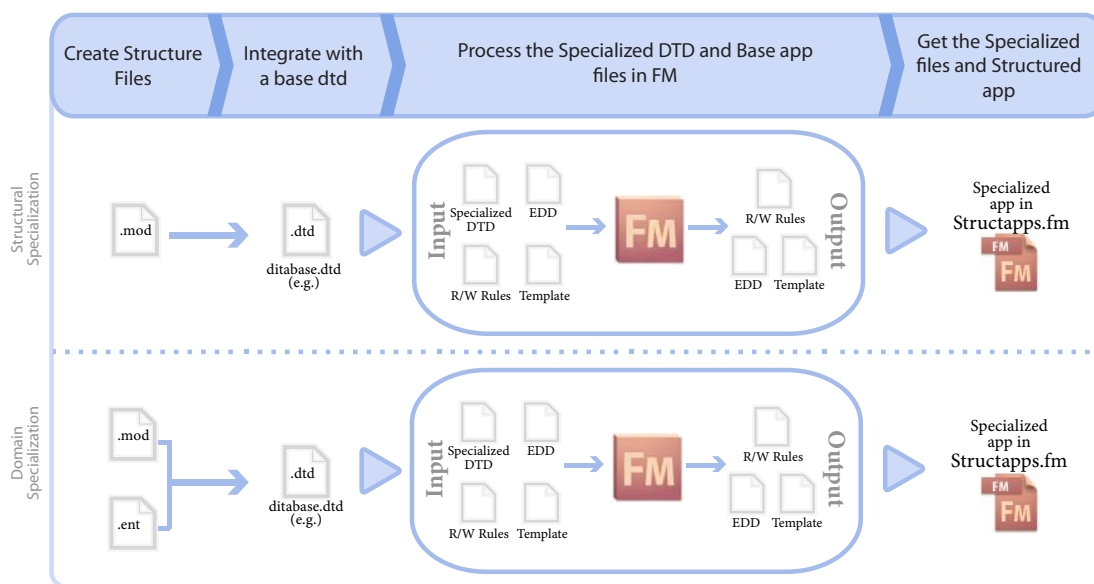
Structural specialization

Follow this workflow for structural specialization:

1 Create a .mod file with definitions of specialized elements. For more information, see "Modify DTDs for structural specialization" on page 7.

2 Integrate the .mod file with the existing DITA DTDs. For more information, see "Modify DTDs for structural specialization" on page 7.

Note: To make your specialized elements types work with the existing <topic> hierarchies, add your specialization to ditabase.dtd. Alternatively, you can create a separate DTD. For <map> specialization, modify and use map.dtd or bookmap.dtd.

3 Copy the mod file at *$StructDir\xml\DITA_1.2\app\base\dtd*.

4 In FrameMaker®, click DITA > DITA Specialization.... The DITA specialization dialog box appears.

DITA Specialization dialog box

5 Specify the path of the source files and a public ID:

Field	Value
Read/Write rules	$StructDir\xml\DITA_1.2\app\technicalContent\rules\<topic.rules.txt>
EDD	$StructDir\xml\DITA_1.2\app\technicalContent\edd\<topic.edd.fm>
Template	$StructDir\xml\DITA_1.2\app\technicalContent\template\<topic.template.fm>
Public ID	A public ID for the generated files. *Note: If you do not enter a Public ID, the XML files created from these specialized files are not portable to other systems. The XML files contain the absolute path of the DTD that is not available on other systems.*
Specialized DTD	$StructDir\xml\DITA_1.2\app\technicalContent\dtd\<ditabase.dtd> (or the base dtd file you have integrated the .mod file with)

6 Specify the output destination folder and filenames:

Field	Value
Destination Folder	Path of the folder containing the output files for specialization
Read\Write Rules	Name of the read\write rules file
EDD	Name of the EDD file
Template	Name of the FrameMaker template

Note: Enter file names with the appropriate filename extensions. Adobe® FrameMaker® creates the output files with the filenames and extensions you specify. Entering incorrect file extensions leads to file association issues.

7 Select Create New Structured Application when files are created check box and click **OK**.

Note: The ditafm.ini file specifies the default DITA version as 1.2 (DitaVersion=1.2). If you want to map your specialization DITA application to DITA 1.1, change DITA version after selecting DITA > DITA options... and restart FrameMaker® .

8 You are prompted to select a structured application. Select an existing structured application from which you want to derive the Doctypes and Entity Locations (Public IDs) and click **Continue**. The Structured Application Designer dialog box appears.

9 Edit the structured application name and, if necessary, other application settings and click **Save**. Your structured application is now created.

10 Select StructureTools > Edit Application Definitions.

11 Select StructureTools > Read Application Definitions.

Start authoring.

Note: Adobe® FrameMaker® 10 supports the viewing and authoring of subjectScheme specialized files but not the processing.

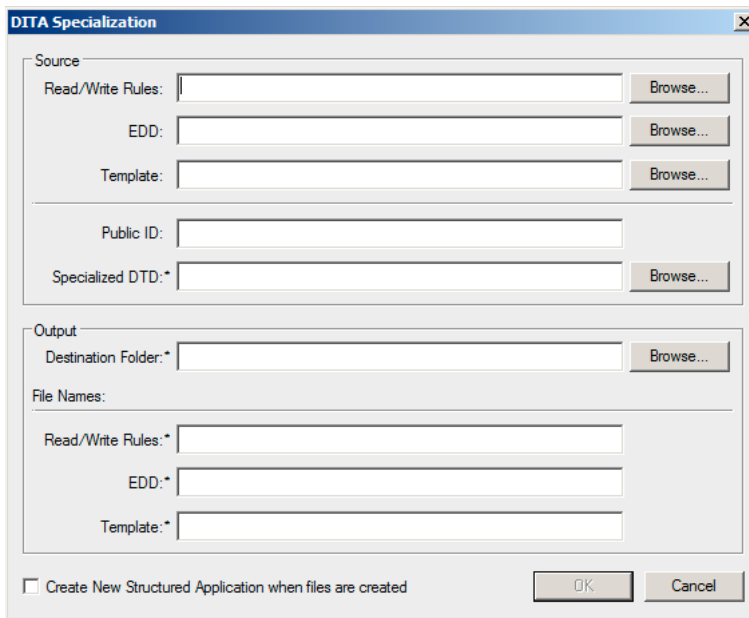## Domain specialization

Follow this workflow for domain specialization:

1 Create a .mod file with definitions of specialized elements. For more information, see "Modify DTDs for domain specialization" on page 9

2 Create a .ent file with the entities for the specialized domain. For more information, see "Modify DTDs for domain specialization" on page 9.

3 Integrate the .mod and .ent files with the existing DITA DTDs in ditabase.dtd. For more information, see "Modify DTDs for domain specialization" on page 9

Note: To make your specialized elements types work with the existing <topic> hierarchies, add your specialization to ditabase.dtd. Alternatively, you can create a separate DTD. For <map> specialization, modify and use map.dtd or bookmap.dtd.

4 Copy the .mod and .ent files at *$StructDir\xml\DITA_1.2\app\base\dtd*.

5 In FrameMaker®, click DITA > DITA Specialization.... The DITA specialization dialog box appears.

6 Specify the path of the source files and a public ID:

Field	Value
Read/Write rules	$StructDir\xml\DITA_1.2\app\technicalContent\rules\<topic.rules.txt>
EDD	$StructDir\xml\DITA_1.2\app\technicalContent\edd\<topic.edd.fm>
Template	$StructDir\xml\DITA_1.2\app\technicalContent\template\<topic.template.fm>
Public ID	A public ID for the generated files. *Note: If you do not enter a Public ID, the XML files created from these specialized files are not portable to other systems. The XML files contain the absolute path of the DTD that is not available on other systems.*
Specialized DTD	$StructDir\xml\DITA_1.2\app\technicalContent\dtd\<ditabase.dtd> (or the base dtd file you have integrated the .mod file with)

7 Specify the output destination folder and filenames:

Field	Value
Destination Folder	Path of the folder containing the output files for specialization
Read\Write Rules	Name of the read\write rules file
EDD	Name of the EDD file
Template	Name of the FrameMaker template

Note: Enter file names with the appropriate filename extensions. Adobe® FrameMaker® creates the output files with the filenames and extensions you specify. Entering incorrect file extensions leads to file association issues.

8 Select Create New Structured Application when files are created check box and click **OK**.

Note: The ditafm.ini file specifies the default DITA version as 1.2 (DitaVersion=1.2). If you want to map your specialization DITA application to DITA 1.1, change DITA version after selecting DITA > DITA options....

9 You are prompted to select a structured application. Select an existing structured application from which you want to derive the Doctypes and Entity Locations (Public IDs) and click **Continue**. The Structured Application Designer dialog box appears.

10 Edit the structured application name and, if necessary, other application settings and click **Save**. Your structured application is now created.

11 Select StructureTools > Edit Application Definitions.

12 Select StructureTools > Read Application Definitions.

Start authoring.

Create DITA DTDs for specialization

DITA DTDs are divided into smaller modules. These modules reflect the base elements hierarchy (Topic and Map) and their respective domain and structural specializations such as Task, Concept, Bookmap, UIDomain, and Programming Domain. The fixed set of changes that you make in DTDs is defined below.

Note: If you have Adobe® FrameMaker® 10 installed on your machine, you can access DITA DTDs or .mod files from: <installation_directory>/Adobe/FrameMaker10/Structure/xml/DITA/app/dtd You can also download all the sample files used in the steps for structural specialization from: https://share.acrobat.com/DITA_Specialization.

Modify DTDs for structural specialization

Create the .mod file for structural specialization for <topic>

1 Copy any existing .mod file and rename it. For example, copy reference.mod and save it as objectsp.mod.

2 Open the new .mod file, objectsp.mod. In the section Specialization Of Declared Elements, change the info-type declaration to the new specialized structure type. The specialized structure type is required for integrating the specialized modules with existing ones. For example, replace the line:

```
<!-- ============================================================ -->
<!--                 SPECIALIZATION OF DECLARED ELEMENTS         -->
<!-- ============================================================ -->


<!ENTITY % reference-info-types "%info-types;"                      >
```

with this one:

```
<!-- ============================================================ -->
<!--                 SPECIALIZATION OF DECLARED ELEMENTS         -->
<!-- ============================================================ -->


<!ENTITY % objectsp-info-types "%info-types;"                      >
```

Note: Similarly in the following three steps, remove the existing declarations in the specified sections and replace them with the information for the new elements. This way you retain the formatted structure of the existing DTDs and map parallel information when declaring new elements.

3 Declare the new entities for the specialized elements required, up to 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, and so on, for other elements.

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

5 In the Specialization Attribute Declarations section, declare the element from which the specialized element is derived. You must declare the hierarchy down to the base <topic> or <map> type (starting with a "-" for structural specialization). For example, if the specialized element is derived from a reference element, include the complete hierarchy:

```
- topic/reference/refbody specialtopic/specialbody
```

Add the following lines to the Specialization Attribute Declarations section:

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

Update ditabase.dtd

Integrate the new .mod file with the existing ones by modifying ditabase.dtd.

Note: To avoid overwriting the original ditabase.dtd, you can rename it to ditabaseObjectsp.dtd for this example.

1 In the Topic Nesting Override section, add the declaration for the specialized topic type. Ensure you include the base type information in the declaration.

```
<!-- ============================================================ -->
<!--                  TOPIC NESTING OVERRIDE                      -->
<!-- ============================================================ -->
<!ENTITY % info-types   "topic | concept | task | reference |
                         glossentry | myobjecttype"              >
```

2   To import the new .mod file, add an entry in the Topic Element Integration of the ditabase.dtd.

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

Note: *To restrict multiple topic types in a single topic type, create an integration file but don't integrate topic types together as shown in this example.*

Modify DTDs for domain specialization

DITA domains are implemented with two files:

*   A **.mod** file that declares the elements for the domain.

*   A **.ent** file that declares the entities for the domain.

For domain specialization, create both files. In the .mod file, declare the specialized elements; in the .ent file, declare the entities for integration-related information. The .ent file is required because domain specialized elements must be available wherever their base elements are.

After creating these files, update the ditabase.dtd for implementing domain specialization for <topic>. The steps in the following sections define three new domain specialized elements for <image>, <prolog>, and <link> for <topic>.

For implementing domain specialization for <map>, follow the same procedure but edit the following files:

*   MapGroup.mod for declaring the elements for the domain.

*   MapGroup.ent for declaring the entities for the domain.

*   Map.dtd or BookMap.dtd for integrating the .mod and .ent files.

Create the .mod file

1   Copy any existing .mod file and rename it. For example, copy utilitiesDomain.mod and save it as domainsp.mod.

Note: *In the following three steps, remove the existing declarations in the specified sections and replace them with the information for the new elements. This way you retain the formatted structure of the existing DTDs and map parallel information when declaring new elements.*

2   Open the new mod file, domainsp.mod. In the 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. Copy the following lines for the specialized element, <Dimage>.

```
<!-- =========================================================== -->
<!--                    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-conref-target) "inline"
               %univ-atts;
               outputclass     CDATA                      #IMPLIED  >
<!-- =========================================================== -->
```

4 In the Specialization Attribute Declarations section, declare the element from which the specialized element is derived. Declare the hierarchy down to the base <topic> or <map> type (starting with a "+" for domain specialization). For example, if the specialized element is derived from another utility domain element, define the complete hierarchy from specialized element to utilities domain to topic. (The utilities domain is 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 " >
```

Create the .ent file

1 Create the .ent file with the filename domainsp.ent.

The information in this file allows the elements to be substituted instead of aggregated. That is, wherever the base element is allowed, its specialized element is also allowed.

Note: As with the .mod files, you can rename an existing .ent file and replace the declaration statements as required.

2 Open the .ent file and declare the entities for integration of new elements with the existing ones (using domain extensions).

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

3 Declare the domain attribute entity to define the ancestry down to the root from which the elements are derived. If you are specializing any element from some domain extension, then you need to declare up to the top.

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

Update ditabase.dtd

Integrate the specialized .mod file with the existing ones by modifying ditabase.dtd. For domain specialization, specify both the .mod and .ent files in the ditabase.dtd as follows:

1 Define the new domain in the vocabulary section in the Domain Entity Declarations section.

```
<!-- ============================================================ -->
<!--                   DOMAIN ENTITY DECLARATION                  -->
<!-- ============================================================ -->
<!ENTITY % domainsp-d-dec PUBLIC "-//domainsp//ENTITIES DITA domainsp Domain//EN" "domainsp.ent" >
%domainsp-d-dec;
<!---------- and the other existing ones --------------------------->
```

2 Define the vocabulary substitution for the specialized elements. Include the elements from which the domain specialized elements extend.

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

3 Add the vocabulary attribute declaration statements.

```
<!-- ============================================================ -->
<!--                   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;" >
```

4 Specify the vocabulary definition and include the .mod file for domain element integration. This entry includes all the specialized elements declared in the .mod file.

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

Publishing specialized topics

In DITA specialization you have the advantage of processing specialized content with unspecialized, general tools. However, these tools process the elements according to the general content model from which the specialization is derived. For example, specialized forms of <paragraph> are still formatted as paragraphs.

To fine-tune or deviate from the base formats, you can modify EDDs, XSLT stylesheets, templates, and read/write rules within the FrameMaker® environment. After making these changes, you can publish Adobe PDFs with the new format definitions from FrameMaker®.