Customizing DITA for Adobe FrameMaker

July 14, 2020

Contents

DITA Specialization in Adobe FrameMaker

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

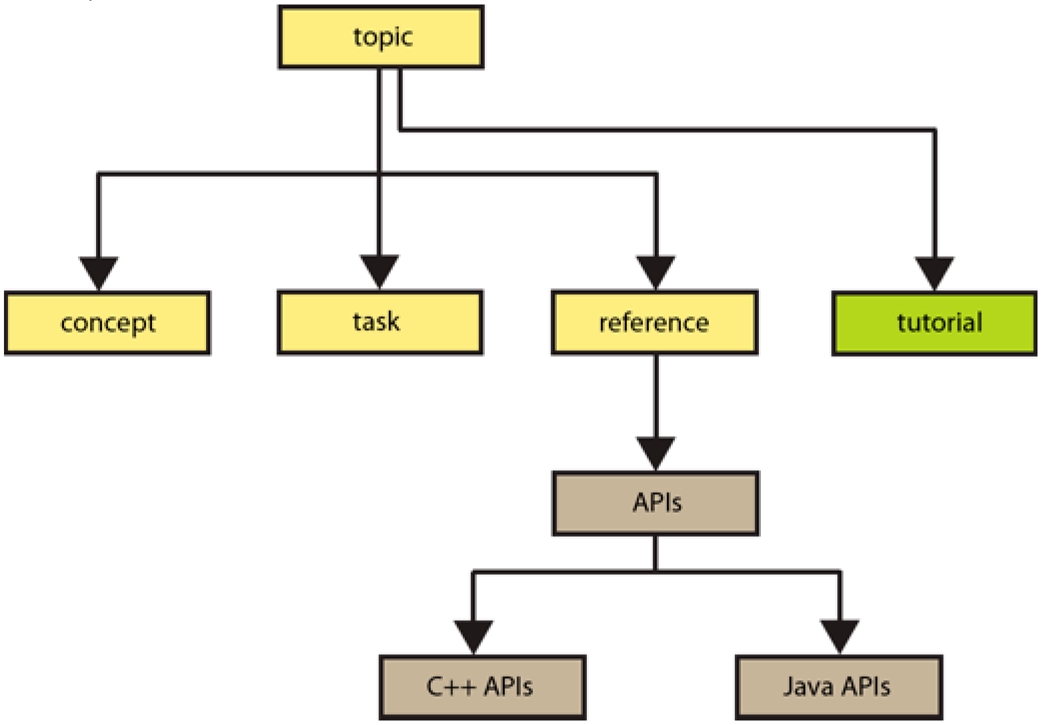
Defines markup for a specific information domain or subject area, such as programming or hardware.

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.

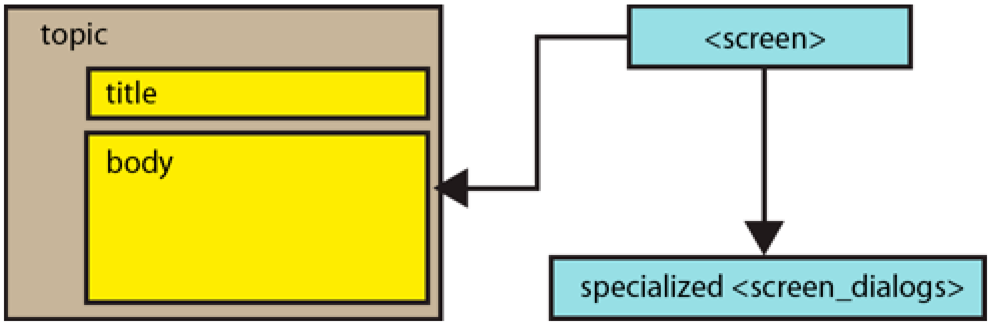


Structural specializations declare new top-level topic types and map types. You use structural specializations to define entirely new document structures. With structural specializations, you can specialize any base element type, including topic and map, as well as elements in any topic or map specialization.

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

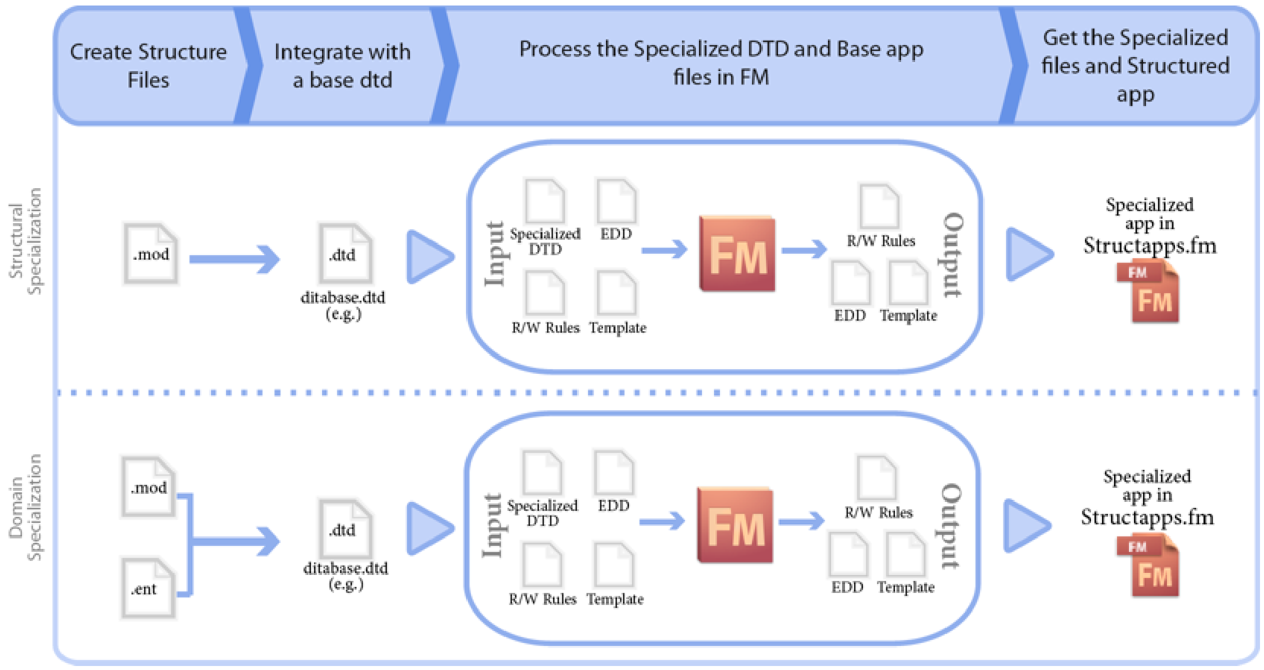
You cannot make the content models of specialized element types less restrictive than the content models of their base types.

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

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



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*.

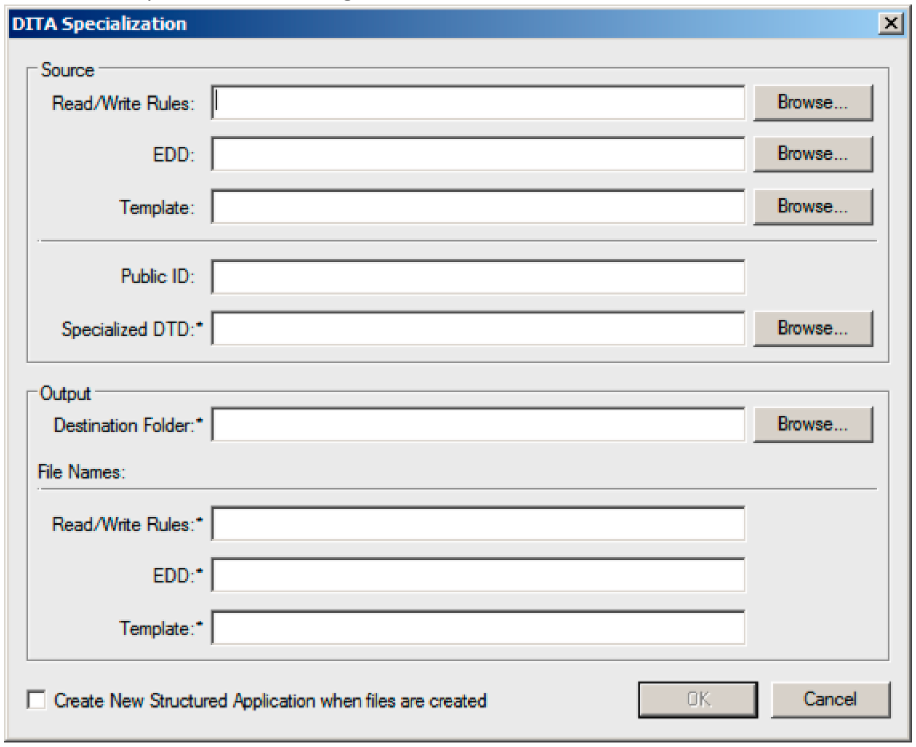
2) Integrate the .mod file with the existing DITA DTDs. For more information, see *Modify DTDs for structural specialization*.

   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 supports the viewing and authoring of subject Scheme 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*.
2) Create a .ent file with the entities for the specialized domain. For more information, see *Modify DTDs for domain specialization*.
3) Integrate the .mod and .ent files with the existing DITA DTDs in ditabase.dtd. For more information, see *Modify DTDs for domain specialization*.

   ***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.

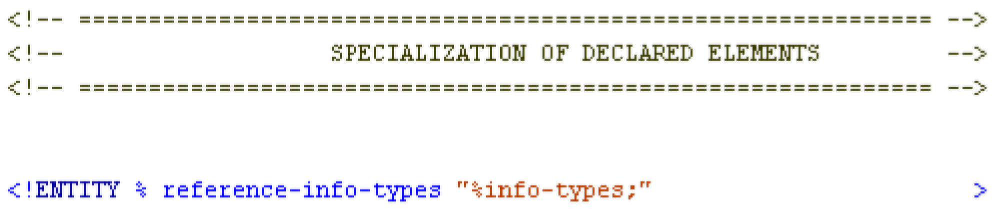
NOTE: If you have Adobe FrameMaker installed on your machine, you can access DITA DTDs or .mod files from the following location:

<installation_directory>/Adobe/AdobeFrameMaker2015/Structure/xml/DITA/app/dtd.

Modify DTDs for structural specialization

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

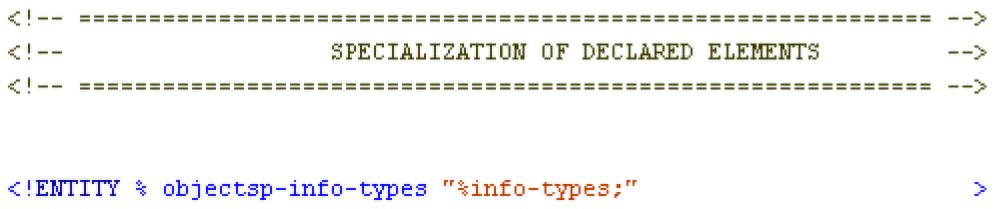
1) Copy any existing .mod file and rename it. For example, copy reference.mod and save it as objectsp.mod.
2) Open the new .mod file, objectsp.mod. In the section Specialization Of Declared Elements, change the info-type declaration to the new specialized structure type. The specialized structure type 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:

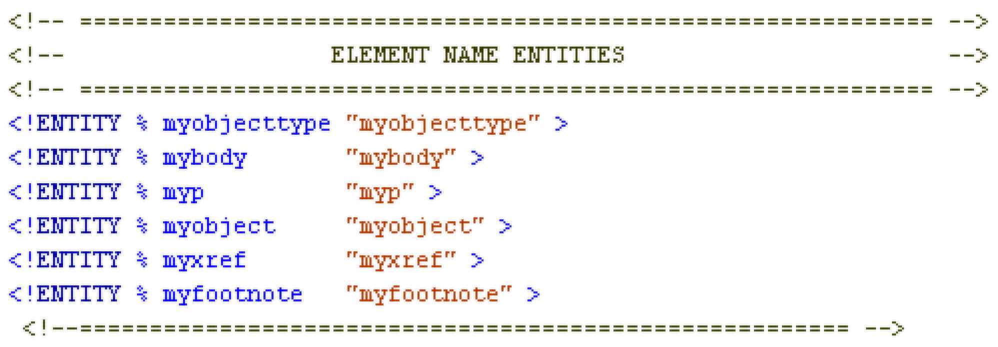


```
<!-- ============================================================== -->
<!--                  SPECIALIZATION OF DECLARED ELEMENTS           -->
<!-- ============================================================== -->


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

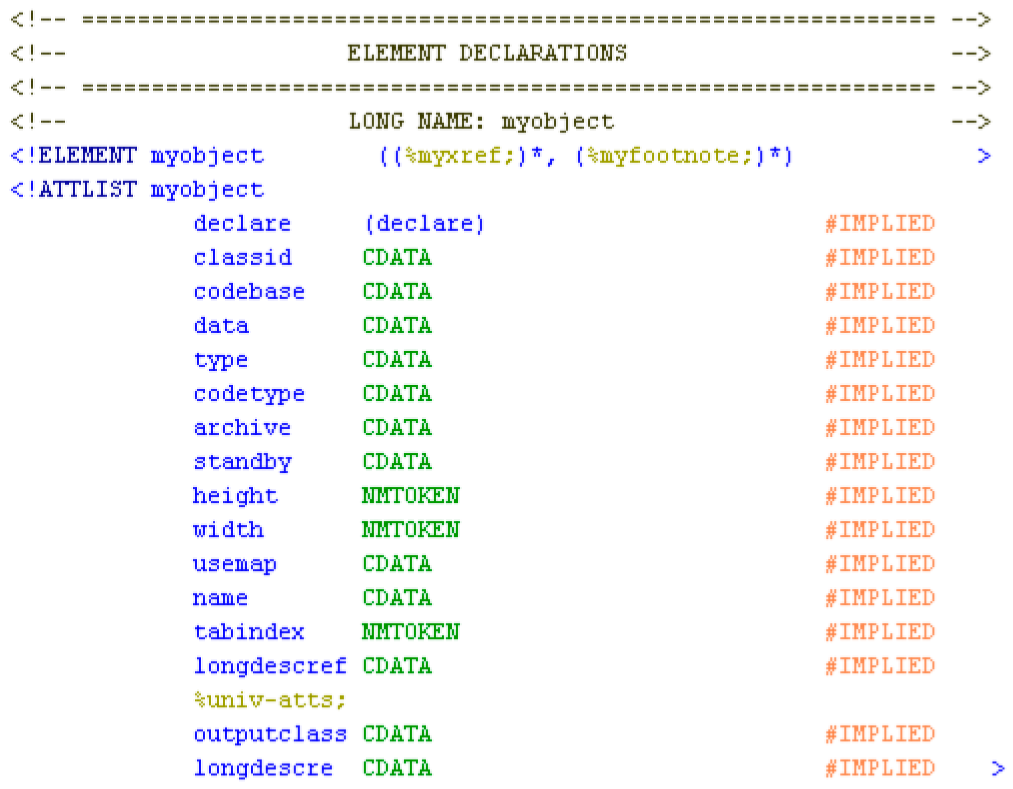
NOTE: Similarly in the following three steps, remove the existing declarations in the specified sections and replace them with the information for the new elements. This way you retain the formatted structure of the existing DTDs and map parallel information when declaring new elements.

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:

6) `- 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:

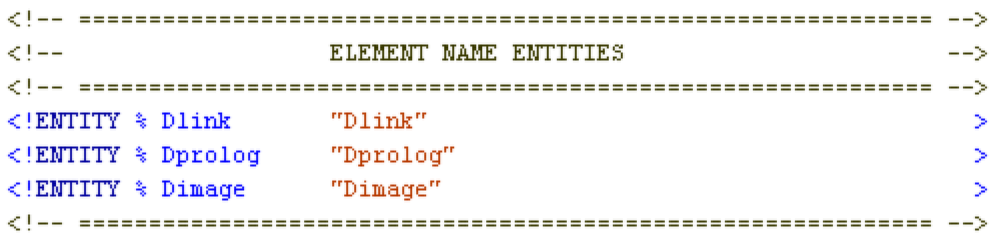
- MapGroup.mod for declaring the elements for the domain.
- MapGroup.ent for declaring the entities for the domain.
- Map.dtd or BookMap.dtd for integrating the .mod and .ent files.

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®.

DITA 1.3 Customization in Adobe FrameMaker

What's new in the DITA 1.3 implementation

DITA support in FrameMaker has been upgraded to cover DITA 1.3. The EDDs and templates for DITA 1.3 have been redesigned to allow easier customization of the document shells. Removing domains or individual elements has become much easier.

More, better organized and recognizable, variables

Many DITA elements share the same set of basic elements, and the DITA DTDs use a large number of entities to define building blocks. Most of the entities in the DITA DTDs are now mapped to FrameMaker variables. However, as variables cannot be nested, all nested entities from the DTDs had to be resolved.

Specialized elements are defined in variables, which carry the name of the base element and the domain in which the specializations are defined. Each variable starts with a pipe symbol and ends with a space. This allows easy addition and removal of variables in general rules as well as specifications for context rules.

More, easier to recognize, conditional text tags

Element definitions in domain specialization modules, as well as their occurrence in other EDD modules, are made conditional. Each domain condition contains the abbreviated name of the domain. This makes it easy to hide an entire domain. Examples are `domain-pr-d`, `domain-hi-d`, `domain-sw-d`.

All variables for specialized elements are marked with the domain condition of the domain to which they belong. This ensures that the elements are suppressed from all the general rules and specifications of context rules when their domain is excluded via the domain condition.

Apart from the domain conditions, a number of conditions have been added to remove some base elements from the rules. These conditions are named according to their purpose, e.g. `topic-generalTask` or `topic-machineryTask`.

Also, the definition of a single root element requires specific conditions, which are named accordingly, e.g. `root-topic`, `root-learningAssessment`.

New domains, elements and attributes

All new domains, elements and attributes defined in DITA 1.3 are added. Information about these additions can be found in the official DITA 1.3 specification documents.

Create customized DITA templates

DITA allows customization of document type shells. To make this work in FrameMaker, templates must be created and made available in your authoring environment.

Customizing the DITA templates may involve more or less complex procedures, depending on the customizations you want to perform, and whether the customized templates should replace existing ones or become available as extra options for your authors.

1) Copy the existing EDD.

   NOTE: It is recommended to store all customized EDDs in a single folder.

   To copy an EDD, open the original EDD in FrameMaker and use File > Save As command to create a copy. This ensures that all paths to the text inset modules are adjusted.

2) Customize the EDD as required.

   For more information, see Customizing DITA 1.3 EDDs.

3) Copy the existing template.

   Name the new template according to its usage and store all custom templates in a single folder. This allows you to keep an overview of available customizations and prevents errors when relinking the structured applications.

4) Import the new EDD into the new template.
5) Adapt formatting as required.
6) Open the structapps.fm file.

   Use the StructureTools > Edit Application Definitions command to open the user-specific file or the StructureTools > Edit Global Application Definitions command to open the system-wide file.

7) Copy an existing application definition (if required).

   If the custom template should replace an existing one, you can skip this step.

   NOTE: Ensure that you rename the copy. Also, you need to edit the DITA Mappings. For details, see *Change DITA 1.3 Mappings*.

8) Change the application definition.

   Change the path in the application definition to point to the customized template file.

Customizing DITA 1.3 EDDs

The modular design of the DITA 1.3 EDDs allows easy customization. You can switch off entire domains or remove individual elements. This makes your topic templates fit the needs of your authors.

Remove domains from your EDD

Just like the DITA DTDs, the DITA 1.3 EDDs contains separate modules for each of the included DITA domains. All element definitions in such a domain module are marked as conditional text, using a condition tag that reflects the domain.

To create a valid and consistent customized EDD, use the conditional text expression builder. This utility is available from the **Show/Hide Conditional Text** panel.

NOTE: Using the **Show** and **Hide** lists will result in an invalid EDD. Also, using the **Show All** option may not yield a valid and consistent EDD, as there may be conflicting general rules and multiple root elements in each top-level EDD.

The expression for the DITA EDDs must always be of the following form:

```
("include_1" or "include_2" or ... or "include_n")
  and not ("exclude_1" or "exclude_2" or ... or "exclude_n")
```

where `include_x` are conditions to show and `exclude_x` are conditions to hide. To remove a currently included domain, remove it from the list of included domains and add it to the list of excluded domains.

NOTE: All conditions must be mentioned in the expression, either in the include or in the exclude section.

It may be useful to name each build expression with a descriptive name, so that switching between various document shells becomes a matter of selecting the **Build Expression** from the drop-down list in the **Show/Hide Conditional Text** panel and clicking **Apply**.

NOTE: Applying a build expression may take quite a while. Have patience and do not interrupt the process, as this will result in an invalid EDD.

Remove individual domain elements from your EDD

If you want to keep a domain but exclude a particular element from that domain, you must change the value of the variable in which the element is defined. Each specialized element is only included in one single variable, which carries the name of the domain to which the element belongs plus the base element from which the specialization, was derived. Some examples:

Element	Domain	Variable	Definition
b	hi-d	hi-d-ph	`|b|i|line-through|overline|sub|sup|tt|u`
apiname	pr-d	pr-d-keyword	`|apiname|option|parmname`
hazardstatement	hazard-d	hazard-d-note	`|hazardstatement`

To remove an individual domain element, remove the element with its pipe prefix from the variable definition. If the element is the only one in the definition, change the variable definition to a single whitespace character.

NOTE: The variable change is immediate. There is no need to re-apply conditions or update the EDD.

As an example, removing the `line-through` element from your EDD, you must edit the `hi-d-ph` variable so that it reads `|b|i|overline|sub|sup|tt|u`. This makes the `line-through` element unavailable in the entire EDD.

Removing base elements from your EDD

If you want to remove an element that is not part of a domain specialization,, you need to remove it from all the variable definitions in which the element occurs. Use the variables list to determine the variables to be edited. Remove the element with its pipe prefix from the variable definitions. If the element is the only one in the definition, change the variable definition to a single whitespace character.

NOTE: In some general rules, base elements are included without using variables. To make changes to these rules, you need to convert the text inset(s) in which the element appears before being able to make changes. This is explained in a separate section below.

Removing base elements from the general rules and specifications of context rules does not remove the definition of the base elements from the EDD. When you import the changed EDD into a template, the log will show one or more defined but not referenced elements. This does not affect the validity of your EDD and the defined but not referenced elements will not appear in the element catalog, as they are not valid in any context.

Changing general rules

As all general rules are contained in modules, which are imported into the top-level EDD as text insets, you have a choice of two methods for adapting these general rules:

- Change the general rule for the current EDD only
- Change the general rule for all applicable EDDs

In the first case, you need to convert the text inset that contains the general rule to plain text. Double-click the text inset to open the **Text Inset Properties** panel, click the **Convert** button and choose the option **Only this text inset**. After this, you can edit the general rule. There is no need to update the EDD after this.

NOTE: If you choose this option, it is recommended to save the EDD under a different name in a folder that holds all your custom EDDs. This ensures that the original EDD remains available and keeps all your customized EDDs available in the same space.

In the second case, you need to open the module in which the general rule is located. Double-click the text inset to open the **Text Inset Properties** panel and click the **Open** button. In the module that opens, edit the general rule and save the module. Then return to the original EDD and update the text inset.

NOTE: The other EDDs that contain this module should also be opened and the text inset updated before changes take effect. After a text inset update, the current build expression must be re-applied.

Changing or adding context rules

Context rules may mention domain elements, which are not available when the domain is excluded. When creating additional context rules, you must take care that the specification for the new context rule is not empty and does not contain undefined elements.

If a new context rule only applies to elements from a specific domain, mark the entire context rule and apply the condition tag for that domain to it. This ensures that the entire context rule is suppressed when the domain is excluded from the EDD.

Change DITA 1.3 Mappings

When adding customized template to new applications, the DITA mappings file must be changed to make the new templates available for your technical authors.

Edit the EDDs, templates and application definitions. It is recommended to have the `structapps.fm` file open, so that you can double-check the spelling of your new structured applications.

Changing the mappings of topic types to structured applications is done in the DITA Application Mappings Manager dialog.

1) Open the DITA Applications Mappings Manager.

   Choose DITA > DITA Options.

2) Click Application Mappings.
3) Select the Mapping Tag.

   The application mapping that is selected in the DITA Options dialog determines which doctypes are available in the DITAmenu.

4) Select the doctype that you want to remap.

   Click on the doctype you want to change. Then click on Edit in the top of the dialog. This shows the mapping tag and its type. Click OK to load the doctype definition in the editable fields of the dialog.

5) Make changes as required.

   Change the Application name to point to the new structured application.

6) Click Add/Edit to save changes.
7) After making all required changes, close the dialogs.

After Completing This Task:

Restart FrameMaker. This allows the application to reload the mappings, recreate the menus and load the changed structured applications.

Customizing DITA using the plugin

The DITA customization plugin in FrameMaker presents an easy-to-use mechanism to customize your DITA templates. The plugin is based on Regular Language for XML Next Generation (RELAX NG). RELAX NG is a simplified schema language for XML. It defines patterns for the structure as well as content of an XML document. The DITA customization plugin uses the RELAX NG schema of DITA 1.3 and Lightweight DITA.

The plugin takes the RELAX NG files of a DITA document, flattens it out by resolving dependencies, then it start converting them in XML files and finally it creates an EDD. In this process, it also takes the content styling information from a CSS and apply it on the template. The final output is in the form of a structured application that can then be used to create documents based on the customized DITA.

This topic describes the DITA customization process using the plugin, taking a backup of your customizations, and restoring customization from backed up configuration files.

Use the Customize DITA plugin

Perform the following steps to use the Customize DITA plugin to change the structure and formatting of your DITA templates:

1. Launch FrameMaker.
2. In the main menu, click **Structure > DITA > Customize DITA > Customize DITA shell.**

   NOTE: If the **Initialize plugin** command is shown, this needs to be executed first. A restart is required to create the new menu.

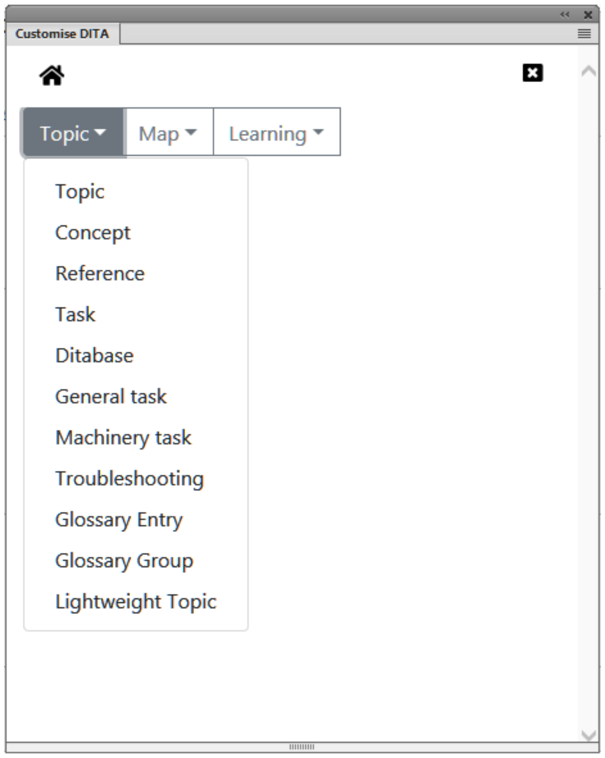
   The plugin initializes the code that is required for customization. Once this is done, the dialog shows the list of currently active customized DITA shells.

   NOTE: *Initializing the plugin for the first time in FrameMaker takes a while, as the required XSLT processor is loaded.*
3. Change an existing customization or add new one. The following table describes the user interface elements available in the Customize DITA pod.

⊕	Start a new customization. The dialog shows a menu with drop-down lists, from which you can select the topic or map to be customized.
🔧	Change an existing customization.
🗑	Remove a customization. The original shell for this topic or map is restored.

4. Click the plus icon to start a new customization.

   A list of DITA topics is shown. The topics are categorized under three branches of Topic, Map, and Learning.



5. Click on a DITA topic that you want to customize.

6. Select the modules, domains, and elements to include

   The dialog shows a tree with checkboxes. The checkboxes have three possible states:

☑	Component (all of its possible child components) selected.	Click to unselects component (and all children).
☐	Component (and all of its possible child components) unselected.	Click to selects component (and all children).
▣	Some but not all child components selected.	Click to selects component (and all children).

   NOTE: Checkboxes for the required domains are disabled.

7. Select the attributes to show.

   The attributes that you choose not to include in the custom DITA are marked as hidden, they are not removed. This implies that they can still be used by FrameMaker and possible plugins, but you will not be able to see them or change their values via the Attributes pod.

   NOTE: The attribute selection is stored and automatically re-applied in other shells that have the same basic attribute list. There are separate lists for topics, maps, and lightweight shells.

8. Click ➔ to generate the EDD

   This takes a while, as multiple transformations need to be performed.

9. Select the CSS.

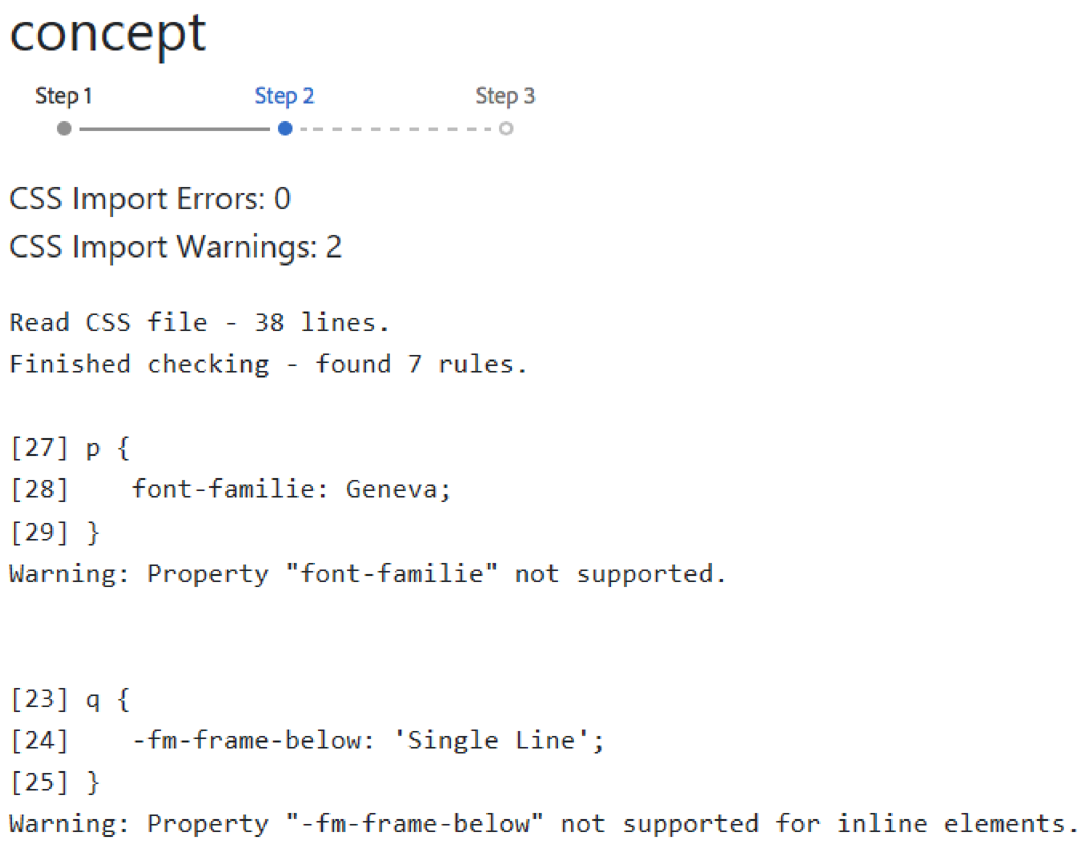
   The currently selected CSS is shown. If you want to use another CSS, click 🔧 to open the Choose CSS File dialog and select another CSS. If an alternative CSS is currently selected, click 🗑 to reset the selection to the default CSS.

10. Click ➔ to import the selected CSS into the EDD.

    If warnings or errors occur, an import report is shown. The report lists the lines of the CSS that caused the warning or error. For more information about the supported CSS properties, see *CSS import*. In addition to the supported CSS properties, there are some restrictions on using the CSS files. See *Restrictions on CSS file import* for more information.

    NOTE: When the import report is shown, you can either click ➔ to ignore the warnings or errors and continue to the next step or click ← to edit or change the selected CSS and try again.

    FOR EXAMPLE: The following image shows warnings for CSS properties that cannot be mapped to the EDD. The lines in the CSS for which the error or warning is generated are listed and the error or warning explains what the problem is.



CSS Import Errors: 0

CSS Import Warnings: 2

```
Read CSS file - 38 lines.
Finished checking - found 7 rules.

[27] p {
[28]     font-familie: Geneva;
[29] }
Warning: Property "font-familie" not supported.


[23] q {
[24]     -fm-frame-below: 'Single Line';
[25] }
Warning: Property "-fm-frame-below" not supported for inline elements.
```

11. Select the template.

    The EDD must be imported into an template before it can be used to create new DITA topics or maps. The currently selected (empty) template is shown. If you want to use another template, click 🔧 to open the Choose Template dialog and select a new template. If an alternative template is currently selected, click 🗑 to reset the selection to the default template.

12. Click ➔ to import the EDD into the selected template.

    If no errors occur, the newly customized shell is now available. The dialog shows the list of customized shells and their modification dates.

13. Start using the customized shell.

    The new shell is listed in the *Customize DITA* dialog. A new file using the customized shell can be created via the **File > New > DITA** submenu. When (re)opening an existing DITA file of the customized type, the new template is used.

Backup DITA customization

Perform the following steps to create a backup of your customize DITA template:

1. In the main menu, click **Structure > DITA > Customize DITA > Backup Customizations.**
2. In the Choose Backup Location dialog, click **Select** to browse to a location where you want to store the custom DITA configurations.
3. Click **OK** to create a backup of your customized DITA templates.

Restore DITA customization

Perform the following steps to restore your customize DITA templates from an existing backup:

1. In the main menu, click **Structure > DITA > Customize DITA > Restore Customizations.**
2. In the Restore Options dialog, select the location where your custom DITA configurations are available. You can choose from the Program Folder or any Other Folder location.
3. Click **OK** to restore your customized DITA templates from the backed up location.

CSS import

The DITA Customization plugin allows you to associate a CSS to define the styling information for the DITA elements. This topic lists all supported CSS properties plus FrameMaker-specific extensions. You will also find information about the selectors and pseudo classes that are supported.

Supported selectors

Not all CSS selectors, pseudo classes and combinations are currently supported. Some are impossible to map to EDD contexts, others are simply not programmed yet.

Selector	Example	How this is handled
[name]	p	AllContextRule
[whitespace]	section p	ContextRule on child element - context is * < followed by a parent
>	section > title	ContextRule on child element - context is section before the selector
:before	note:before { content: ‘NOTE:’ }	Prefix rule - text between quotes in content format rule is entered as prefix string
:after	note:after { content: ‘NOTE:’ }	Suffix rule - text between quotes in content format rule is entered as suffix string
[attr=value]	p[audience=”novice”]	ContextRule on element - context is attribute value (only exact matches)
[name]:first-child	ol > li:first-child	ContextRule on child element - context is {first} inside the parent
[name:last-child	section > p:last-child	ContextRule on child element - context is {last} inside the parent
*:first-child	section > *:first-child	AllContextRule on FirstParagraphRules of the parent element
*:last-child	section > *:last-child	AllContextRule on LastParagraphRules of the parent element

Supported CSS properties

Currently supported CSS properties and values.

Property	Value(s)	FrameMaker EDD mapping
display	‘inline’	TextRangeFormatting > TextRange
color	color	PropertiesFont > Color
background-color	color	BackgroundColor
margin-bottom	size	PropertiesBasic > ParagraphSpacing > SpaceBelow
margin-left	size	PropertiesBasic > Indents > LeftIndent
margin-right	size	PropertiesBasic > Indents > RightIndent
margin-top	size	PropertiesBasic > ParagraphSpacing > SpaceAbove
text-align	‘left’	PropertiesBasic > PgfAlignment > Left
	‘right’	PropertiesBasic > PgfAlignment > Right
	‘center’	PropertiesBasic > PgfAlignment > Center
	‘justify’	PropertiesBasic > PgfAlignment > Justified
text-indent	size	PropertiesBasic > Indents > FirstIndent
text-transform	‘uppercase’	PropertiesFont > Case > UpperCase
	‘lowercase’	PropertiesFont > Case > LowerCase
text-decoration	‘underline’	PropertiesFont > Underline > Single
	‘overline’	PropertiesFont > Overline
	‘line-through’	PropertiesFont > Strikethrough
font-family	string	PropertiesFont > Family

Property	Value(s)	FrameMaker EDD mapping
font-size	size	PropertiesFont > Size
font-stretch	‘ultra- condensed’	PropertiesFont > Stretch > 50%
	‘extra- condensed’	PropertiesFont > Stretch > 60%
	‘condensed”	PropertiesFont > Stretch > 72%
	‘semi- condensed’	PropertiesFont > Stretch > 86%
	‘normal’	PropertiesFont > Stretch > 100%
	‘semi- expanded’	PropertiesFont > Stretch > 120%
	‘expanded’	PropertiesFont > Stretch > 144%
	‘extra- expanded’	PropertiesFont > Stretch > 173%
	‘ultra- expanded’	PropertiesFont > Stretch > 207%
font-style	‘normal’	PropertiesFont > Angle > Regular
	‘italic’	PropertiesFont > Angle > Italic
	‘oblique’	PropertiesFont > Angle > Italic
font-variant	‘normal’	PropertiesFont > Case > Normal
	‘small-caps’	PropertiesFont > Case > SmallCaps
	‘all-caps’	PropertiesFont > Case > AllCaps
font-weight	‘normal’	PropertiesFont > Weight > Bold
	‘bold’	PropertiesFont > Weight > Bold
	‘100’	PropertiesFont > Weight > Regular
	‘200’	PropertiesFont > Weight > Regular
	‘300’	PropertiesFont > Weight > Regular
	‘400’	PropertiesFont > Weight > Regular
	‘500’	PropertiesFont > Weight > Bold
	‘600’	PropertiesFont > Weight > Bold

Property	Value(s)	FrameMaker EDD mapping
	‘700’	PropertiesFont > Weight > Bold
	‘800’	PropertiesFont > Weight > Bold
	‘900’	PropertiesFont > Weight > Bold

FrameMaker-specific properties

Property	Value(s)	FrameMaker EDD mapping
-fm-pgf-box-color	color	PropertiesAdvanced > PgfBoxColor
-fm-text-decoration	‘double underline’	PropertiesFont > Underline > Double
	‘numeric- underline’	PropertiesFont > Underline > Numeric
-fm-font-variant	‘subscript’	PropertiesFont > SuperscriptSubscript > Subscript
	‘superscript’	PropertiesFont > SuperscriptSubscript > Superscript
-fm-banner-text	string	BannerText (only on element)
-fm-descriptive-tag	string	DescriptiveTag (only on element)
-fm-frame-above	string	PropertiesAdvanced > FrameAbove
-fm-frame-below	string	PropertiesAdvanced > FrameBelow
-fm-pgf-format-tag	string	ParagraphFormatTag
-fm-char-format-tag	string	CharacterFormatTag
-fm-autonum-string	string	PropertiesNumbering > AutonumberFormat > [string]
-fm-autonum-position	‘start’	PropertiesNumbering > Position > StartOfParagraph

Property	Value(s)	FrameMaker EDD mapping
	'end'	PropertiesNumbering > Position > EndOfParagraph
-fm-hyphenate	'yes'	PropertiesAdvanced > Hyphenation > Hyphenate > Yes
	'no'	PropertiesAdvanced > Hyphenation > Hyphenate > No
-fm-hyphenate- max-adjacent	integer	PropertiesAdvanced > Hyphenation > MaxAdjacent
-fm-hyphenate- shortest-prefix	integer	PropertiesAdvanced > Hyphenation > ShortestPrefix
-fm-hyphenate- shortest-suffix	integer	PropertiesAdvanced > Hyphenation > ShortestSuffix
-fm-hyphenate- shortest-word	integer	PropertiesAdvanced > Hyphenation > ShortestWord

Restrictions on CSS file import

The Customize DITA plugin imports a CSS file to determine the look and feel of your customized DITA topics. There are some important restrictions on the CSS files that you need to be aware of.

Only one CSS file

The plugin only imports one single CSS file. This implies that all the rules have to be listed in that single file. If you want to create your custom CSS file, it is suggested to save a copy of the default CSS file, rename it, and make modification to it.

NOTE: A planned redesign will enable using multiple CSS files. For example, put your overrides in a single CSS file and use the default CSS for all standard configurations.

Required CSS rules

There is no way in RELAX NG (or DTD) to differentiate between a block level or inline element. This has to be done in the CSS, by setting a `display: inline` rule. This rule is translated into the **`<TextRange>`** element in the EDD. Without this rule, the element will be shown as a block element.

The `display: inline` rule is listed at the top of the default CSS file.

Order of rules

The rules in the CSS files are processed top to bottom. This implies that the order of the formatting rules in the EDD is the same as the order in which the rules are found in the CSS file. If you have multiple context rules, only the first one that matches the context specification will be applied. This means that the ordering of rules in your CSS file should be from most specific to most generic, to prevent unwanted effects in the EDD.

Example: If you want to make the font size and/or style of a **`<p>`** dependent on its context. The order of the CSS rules determines whether you will get the desired result.

```
body p {
    font-size: 14pt;
}
section p {
    font-size: 12pt;
    font-style: italic;
}
```

The above CSS rules will make every **`<p>`** show up with font size 14 pt, as the first context (the **`<body>`** ancestor) listed in the EDD always applies. Reversing the order of these rules in the CSS corrects this problem. Another option is making the first rule more specific:

```
body > p {
    font-size: 14pt;
}
```

This specifies that only a **`<p>`** that is a direct child of **`<body>`** gets the 14 pt font size. This change may have other unwanted side effects, such as a **`<p>`** inside an **`<li>`** not getting any specific styling.

FrameMaker-specific properties

A number of formatting-related EDD properties are not available in CSS. These are added to the CSS mapping by using the vendor extension pattern of CSS. Such properties have the prefix '-fm-'. This makes the CSS valid against any CSS syntax checking tool.

```
section > *:first-child {
-fm-frame-above: SingleLine;
}
```

Some properties that do allow mapping from CSS have extra values in FrameMaker which are not valid in CSS. In these cases, the extra FrameMaker values are mapped to a property that has the '-fm-' prefix added.

```
keyword {
-fm-text-decoration: double-underline;
}
```

Legal notices

For legal notices, visit the Legal Notices page.