



IBM User Technologies

# DITA specialization by example: Defining schemas by editing documents

**Presented by**  
**Erik Hennum, IBM**  
ehenum@us.ibm.com

# Main points

- **Make the case for a new method for specializing**
- **Why use specialization and pluggability?**
  - Review of the goals
- **How to specialize by example**
  - A quick walk-through
- **Demo**
- **Summary of what's gained**

# Why use specialization and pluggability

# Controlling document structure with guidelines

## Installing a hard drive

### 1. Unscrew the cover.

The drive bay is located ...

### 2. Insert the drive ...

If you feel resistance ...

## Guidelines

### 1.6.3 Writing a procedure

1.6.3.1 Create an ordered list with an item for each action the user should take.

The list item should start with a phrase that summarizes the action.

Use a paragraph in each item to describe the result or additional information.

- **Consistency depends on editorial vigilance**
- **Mistakes will be made**
- **Processes depending on the guidelines are fragile**
- **A “ghost” markup visible only to people familiar with the guidelines**

# Controlling document structure with special markup

## Guidelines

1.6.3 Writing a procedure

1.6.3.1 Create an ordered list  
for each action the user performs

The list item should state the action  
that summarizes the action

Use a paragraph in each step to describe the  
result or additional information

## Task information type

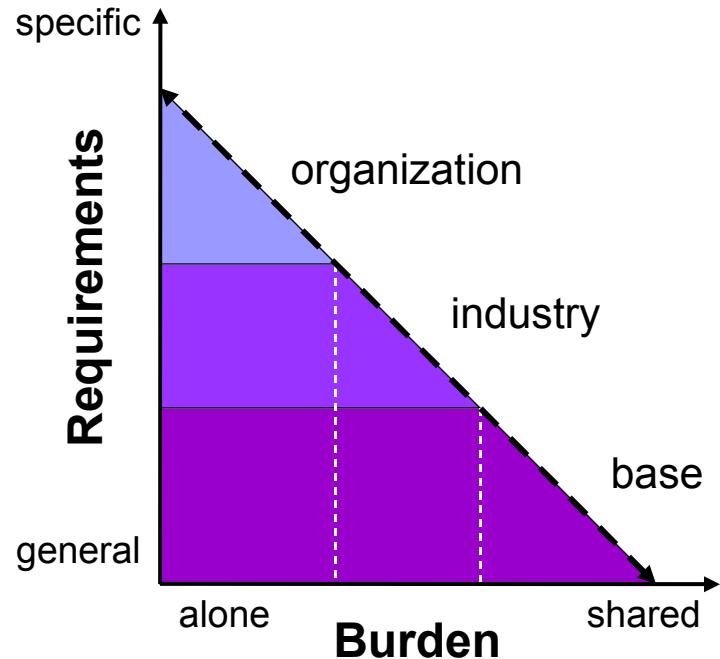
```
<task id="installstorage">
  <title>Installing a hard drive</title>
  <taskbody>
    <steps>
      <step><cmd>Unscrew the cover.</cmd>
        <stepresult>The drive...</stepresult>
      </step>
      <step><cmd>Insert the drive...</cmd>
        <info>If you feel resistance...</info>
      </step>
    </steps>
  </taskbody>
</task>
```

- Provides guidance to the writer
- Makes the content easier to understand, author, validate, and process
- Realizes your guidelines and best practices

# Markup tradeoffs: customization vs collaboration

## The XML facts of life

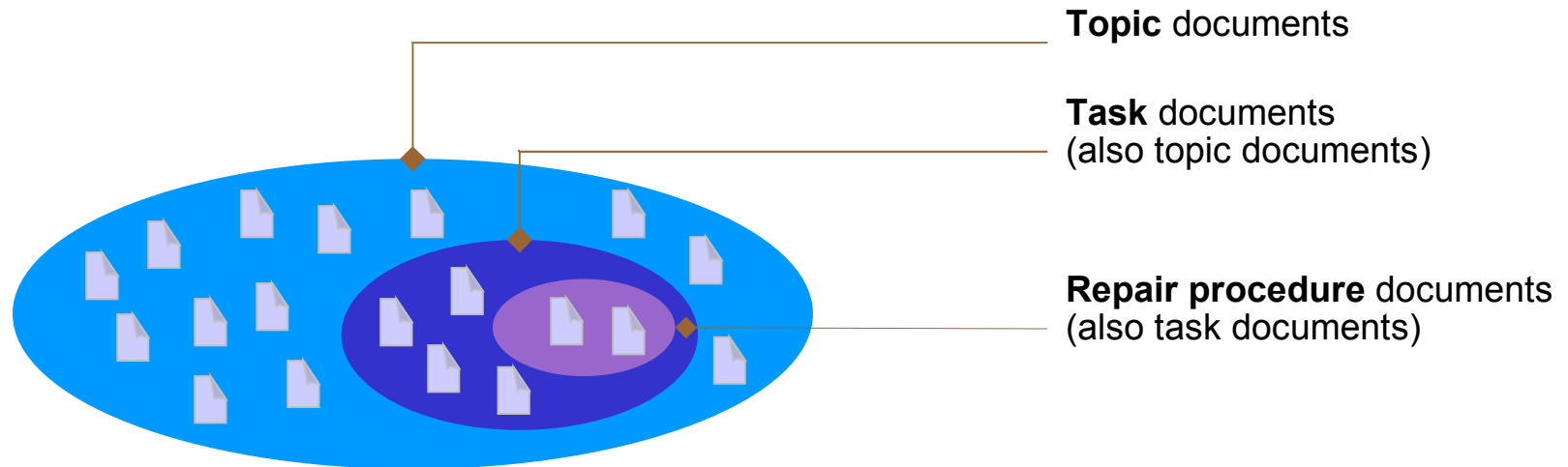
- By itself, XML alone gives you an environment
- Document design is specific to the industry, community, and even organization
- Tension between the horizontal and the vertical



## The DITA approach – graduated commonality

- Share the markup (and processing) at multiple levels by specializing
- Share general requirements broadly **AND** specific requirements narrowly

# Specialized documents remain base documents



- **Recognizing the subset of general-purpose topics that are tasks**
  - Task documents remain topic documents
  - Repair procedure documents remain task documents (and topic documents)
- **Specialization requires design discipline**
  - To get the benefits, designers accept the base and the rules of extension

# Specialization works by substitution

## General

```
<topic id="installstora
  <title>Installing a h
  <body>
    <ol>
      <li><ph>Unscrew t
        <itemgroup>The
      </li>
      <li><ph>Insert th
        <itemgroup>If y
      </li>
    </ol>
  </body>
</topic>
```

## Specialized

```
<task id="installstorage">
  <title>Installing a hard drive</title>
  <taskbody>
    <steps>
      <step><cmd>Unscrew the cover.</cmd>
        <stepresult>The drive...</stepresult>
      </step>
      <step><cmd>Insert the drive...</cmd>
        <info>If you feel resistance...</info>
      </step>
    </steps>
  </taskbody>
</task>
```

- A specialization makes substitutions for base elements

<topic> turns into <task>

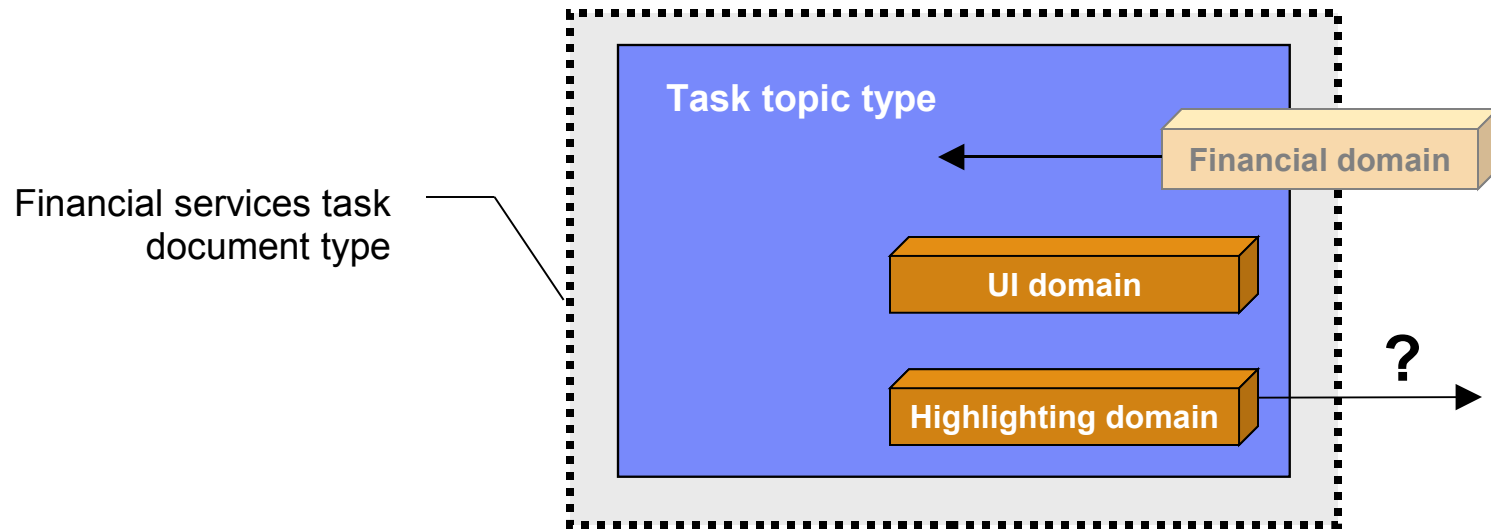
<ol> turns into <steps>

<ph> turns into <cmd>

...

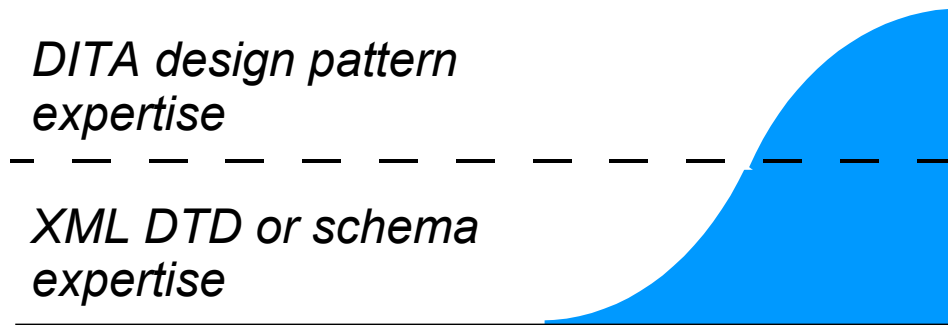


# Pluggability: combining specialization modules



- **To create a document type, you plug in specialization modules**  
Like blades in a Swiss Army knife
- **For financial services, the document type might include**  
The task type and UI vocabulary domain from core DITA  
A new vocabulary domain that you created for the financial industry  
The core highlighting domain – if you are a pragmatist

# The learning curve for implementors



## 1. Define the domain specialization entities.

A domain specialization entity lists the specialized elements as well as base elements.

In the scenario, the domain defines a domain specialization element (which is the base element for `apiname`):

```
<!ENTITY % c1-d-apiname "classname | file"
<!ENTITY % c1-d-keyword "classname | file"
```

## 2. Define the domain identification entity.

The domain identification entity lists the topic type dependencies. Each domain is identified by its domain identifier and `-att`.

In the scenario, the class library domain has a dependency:

```
<!ENTITY c1-d-att "(topic pr-d c1-d)">
```

- **Creating pluggable specializations requires expertise**

Understanding either DTD or XML Schema languages

Understanding the design pattern for DITA specialization modules

- **An obstacle to many people who understand the documents**

# How to specialize by example

# Working by example

- **A template that looks like an example but matches many cases**

Query by Example – a form with wildcards that matches many records

- **Examplotron from Eric van der Vlist**

Generates a RelaxNG schema from a well-formed XML document

Uses attributes in a special namespace for wildcards control

<http://examplotron.org/>

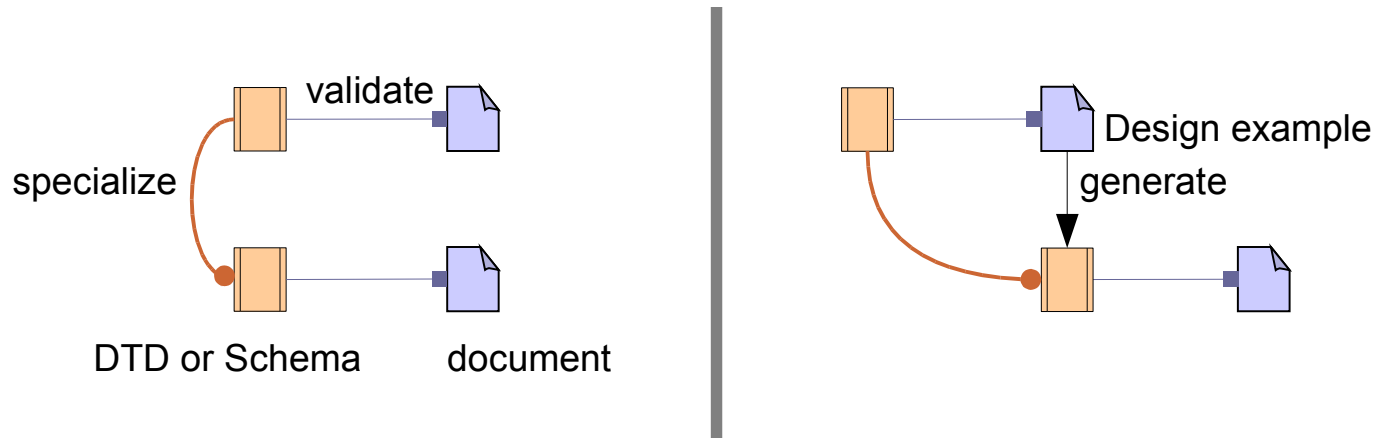
- **What if we generated the DTD or Schema from an example?**

A mock up in the base type is already a good practice for specialization

Can validation of a base document prevent some specialization errors?

Working from a base document instead of a well-formed XML document –  
simplify the design task by leveraging the base definition?

# What's different in the by-example approach



- **Today – manually create a new DTD or XML Schema**  
Responsible for specializing the base DTD or XML Schema
- **Proposed – generate a new DTD or XML Schema**  
Work in a special document validated by the base DTD or XML Schema  
Difference in design approach invisible to other tools

# Specializing an element (case 1 of 8)

```
<section outputclass="background" />
```

**Design  
example**

**Document**

```
<background>  
  <title>Key principles of Object Orientation</title>  
  Object orientation ...  
</background>
```

- **Identify the specialized name with the outputclass attribute**

The example element is the base

Inherits base attributes and subelements unless specified  
(similar to processing)

## Specifying a sequence of subelements (2 of 8)

```
<section outputclass="background">
  <title outputclass="?" /> <!-- or -optional -->
  <p outputclass="+" /> <!-- or -some -->
  <note outputclass="*" /> <!-- or -any -->
</section> <!-- also . or -required -->
```

**Design  
example**

**Document**

```
<background>
  <title>Key principles of Object Orientation</title>
  <p>Object orientation ... </p>
</background>
```

- **Identify the number of occurrences**

Same wildcards as EBNF, regular expression, DTD, RelaxNG Compact  
Option name alternative (-optional, -some, ...)

- **Validation of the subelements prevents many specialization errors**

Can't introduce invalid subelements

## Specifying a choice of subelements (3 of 8)

```
<section outputclass="background">
  <title outputclass="?" />
  <p outputclass="-choice +"/>
  <ol>...</ol>
  <ul>...</ul>
  <note outputclass="*" />
</section>
```

**Design  
example**

### **Document**

```
<background>
  <p>Object orientation has these key principles:</p>
  <ul>
    <li>Inheritance of ... </li>
    ...
  </ul>
</background>
```

- **Specify the -choice option on the first alternative**

Specify the number of occurrences of elements from the choice

List the other alternative elements without occurrence wildcards

Choice ends at the next subelement with an occurrence wildcard



## Specifying textual content (4 of 8)


```
<p outputclass="introduction">
  -text -choice *
  <keyword/>
  <term/>
</p>
```

**Design  
example**

**Document** `<introduction><term>Object Orientation</term> is a  
...  
</introduction>`

- **Specify -text as the first content item**
  - Required -choice \* for now because of DTD and Schema limitations
  - List the subelements that can be mixed with the text
- **Reliable validation for mixed content**
  - Can't introduce subelements that won't be a valid specialization

## Specifying a specialized subelement (5 of 8)



```
<section outputclass="background">
  <title outputclass="?" />
  <p outputclass="introduction ?">
    -text -choice *
    <term/>
  </p>
  <p outputclass="*" />
</section>
```

**Design  
example**

**Document**

```
<background>
  <introduction><term>Object Orientation</term> is ...
  <p>Object orientation ...
</background>
```

- **Supply the specialized name before the occurrence wildcard**  
Specify the content of the subelement as usual

## Reusing an element in many containers (6 of 8)

```

<p outputclass="-declare introduction">
  ...
</p>
<section outputclass="background">
  <p      outputclass="$introduction ."/>
  <p      outputclass="*/>
</section>
<section outputclass="summary">
  <p      outputclass="$introduction ?"/>
  <p      outputclass="*/>
</section>

```

**Design  
example**

### Document

```

<background>
  <introduction><term>Object Orientation</term> is ...
</background>
<summary>
  <introduction>You have learned ...
</summary>

```

- **Define once, reference multiple times with different wildcards**
  - Use -declare to avoid adding the new element to the content of its parent
  - The -declare can appear on a parent of multiple new specialized elements

## Specifying a reusable set of elements (7 of 8)

```
<section outputclass="-set Block">
  <p/>
  <ol>...</ol>
  <ul>...</ul>
</section>
<section outputclass="background">
  <title outputclass="?" />
  <p outputclass="$Block +" />
  <note outputclass="*" />
</section>
```

**Design  
example**

### Document

```
<background>
  <p>Object orientation has these key principles:</p>
  <ul>
    <li>Inheritance of ... </li>
    ...
  </ul>
</background>
```

- **The set expands to its members – for instance, in a choice**  
Members can include elements, specialized elements, and other sets  
Provide a reference to the set on a member element

## Specifying attribute values (8 of 8)

```
<sl outputclass="features"  
  audience="a-choice administrator developer user"  
  platform="a-default linux"  
  product="a-required">  
  ...  
</sl>
```

**Design  
example**

**Document**

```
<features product="Widgets!" audience="administrator">  
  ...  
</features>
```

### ■ Options for providing detail about attribute values

- Turn an attribute from optional into required
- Specify a default or required value (fixed)
- Restricting to a choice of values (enumeration)
- A combination of the above

# Limitations

- **Still have to learn a design notation (though simpler)**  
Useful to get familiar with sets defined by base vocabularies
- **Base element must have an outputclass attribute**  
A best practice anyway
- **No support for a choice of sequences or a sequence of sequences**  
A dubious practice – better to specialize a container for the sequence
- **A choice for a single base position is awkward to define**  
Possible and not all that common
- **Additional restrictions on a restricted attribute are awkward**  
Possible and even less common

# Comparison of concept DTD with design example

## DTD

```
<!ELEMENT concept (
  (%title;),
  (%titlealts;)?,
  (%shortdesc; | %abstract;)?,
  (%prolog;)?,
  (%conbody;)?,
  (%related-links;)?,
  (%concept-info-types;)*
)>
<!ATTLIST concept
  ...
  %arch-atts;
  class CDATA "- topic/topic co
  domains CDATA "&included-doma
>
```

## Design Example

```
<topic id="concept" outputclass="concept">
  <title outputclass="."/>
  <titlealts outputclass="?"/>
  <shortdesc outputclass="$TopicDesc ?"/>
  <prolog outputclass="?"/>
  <body outputclass="conbody ?">
    <p outputclass="$Flow *"/>
    <section outputclass="-choice *"/>
    <example/>
  </conbody>
  <related-links outputclass="?"/>
  <topic id="subtopic" outputclass="*">
    <title/>
  </topic>
</concept>
```

- **Clear relation to base**

Makes the continuity more obvious

- **Hides the complex architectural attributes and design pattern**

# Comparison of XML Schema with design example

## XML Schema

```

<xs:complexType name="concept.c
  <xs:sequence>
    <xs:group ref="title"/>
    <xs:group ref="titlealts" r
    <xs:choice minOccurs="0">
      <xs:group ref="shortdesc"
      <xs:group ref="abstract"
    </xs:choice>
    <xs:group ref="prolog" minc
    <xs:group ref="conbody" mir
    <xs:group ref="related-link
    <xs:group ref="concept-info
      maxOccurs="unbounded"/>
    </xs:sequence>
    ... attributes ...
  </xs:complexType>
<xs:complexType name="conbody.c
  ... conbody definition ...
</xs:complexType>

```

## Design Example

```

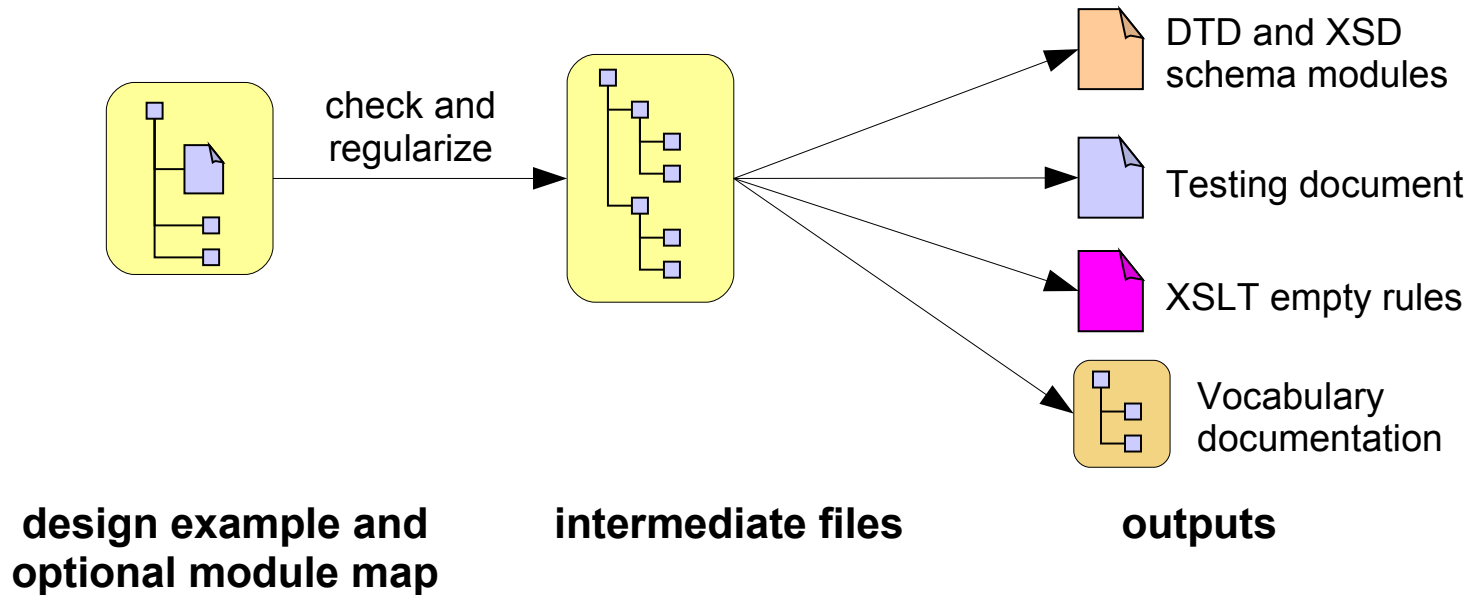
<topic id="concept" outputclass="concept">
  <title outputclass="."/>
  <titlealts outputclass="?"/>
  <shortdesc outputclass="$TopicDesc ?"/>
  <prolog outputclass="?"/>
  <body outputclass="conbody ?">
    <p outputclass="$Flow *"/>
    <section outputclass="-choice *"/>
    <example/>
  </conbody>
  <related-links outputclass="?"/>
  <topic id="subtopic" outputclass="*">
    <title/>
  </topic>
</concept>

```

- Easier to see the structure of the document



# Generating the outputs from the design example



## ■ Principal pipeline for design-evaluate-change cycle

The DTD and Schemas to validate specialized documents

A document for testing and evaluating the design

Maintainable linking within the reference documentation for vocabularies

# Demo

# Opportunity for vendors of XML editors

- **Lightweight extension on an existing validating editor**  
Initialize a design example from a selected base design module
- **A GUI form for the outputclass options**  
A pick list for sets
- **Highlighting for the specialized elements**  
Maybe show the name of the new element instead of the base element
- **A GUI for launching the output transforms**  
Cycle through design-evaluate-change

# Summary of notation

<b>Occurrence</b>	? <i>or</i> -optional . <i>or</i> -required * <i>or</i> -any + <i>or</i> -some # <i>or</i> -ignore
<b>Content</b>	-choice -declare -empty -member -set -text
<b>Reference</b>	<i>\$name</i>
<b>Attribute</b>	a-choice a-default a-drop a-name a-required a-type
<b>Assembly</b>	<vocabularyModule> and <documentType> maps

## Work in progress – more to do

- **Add outputclass on all DITA elements (titlealts, prolog, ...)**
- **More robust implementation – proof-of-concept now**
  - Type definition for base DITA and core specializations
  - Transform to interlinked documentation
- **Loose ends**
  - Verify specializing maps by example
  - DITA 1.1 attribute addition – a global a-name?
  - Define subsets by excluding members of a base set?
  - Transform for downcasting from base to specialized types?

# Summary of potential benefits

- **Concrete design**

  - The design artifact looks like the intended document

- **Reduced barriers**

  - People who understand the document can contribute to the design

- **Specialization orientation**

  - Visibly extends an existing design

  - Prevents many specialization errors through base validation

- **Higher productivity**

  - Handles the architectural attributes and design pattern for you

  - Generates DTD and XSD from a single source for broad use of plugins

- **Low risk**

  - Adopters can always switch to the generated DTD or XSD

# Questions? Comments?

# DITA resources

- **Learn more**

- OASIS DITA Technical Committee –  
<http://www.oasis-open.org/committees/dita/>

- Cover Pages – <http://xml.coverpages.org>

- DeveloperWorks –  
<http://dita-ot.sourceforge.net/SourceForgeFiles/doc/DITA-dWarticles.html>

- **Download the tools**

- DITA Open Toolkit – <http://dita-ot.sourceforge.net/>

- **Participate in the community**

- DITA Focus Area – <http://dita.xml.org>

- dita-users discussion group – <http://groups.yahoo.com/group/dita-users/>

Erik Hennum – [ehennum@us.ibm.com](mailto:ehennum@us.ibm.com)



# Backup

# Domain alternatives for base elements

```

<topic id="backgroundDomain">
  <title/>
  <body>
    <section outputclass="background">
      <title outputclass="?"/>
      <p outputclass="introduction ?">
        -text -choice *
        <term/>
      </p>
      <p outputclass="+"/>
      <note outputclass="*"/>
    </section>
  </body>
</topic>

```

**Design example**

padding elements

alternative element

specialized subelement

- **A new element without a specialized parent becomes an alternative**

Can appear anywhere the base element can appear

For instance, **<background>** becomes an alternative to **<section>**, but the **<introduction>** subelement can only appear within **<background>**

A new topic element becomes a topic type

# Declaring the vocabulary module

```
<vocabularyModule>
  <vocabularyProperties>
    <vocabularyPublicID> ... </vocabularyPublicID>
    ... other properties of the vocabulary ...
  </vocabularyProperties>
  <vocabularyBase href=" ../topic/topic.ditamap"/>
  <vocabularyExample href="backgroundDomain.dita"/>
  <vocabularyItem href="background.dita"/>
  ... other topics documenting an element or set ...
</vocabularyModule>
```

- **A specialized map for vocabulary properties and documentation**
  - Captures the metadata for generating a DTD or XML Schema module
  - Identifies the base module and the design example
  - May document the elements and sets in the design example

# Specifying a choice for a single base position

```
<topic id="topicDescSet1" outputclass="-set TopicDesc">
  <title outputclass="#"/>
  <titlealts outputclass="#"/>
  <shortdesc/>
</topic>
<topic id="topicDescSet2" outputclass="-member $TopicDesc">
  <title outputclass="#"/>
  <titlealts outputclass="#"/>
  <abstract/>
</topic>
<topic id="conceptTemplate" outputclass="concept">
  <title outputclass="."/>
  <titlealts outputclass="?"/>
  <shortdesc outputclass="$TopicDesc ?"/>
  ...
</topic>
```

## Design template

- **Specify -member to add an alternative to a set**

Special case – awkward but possible

Note that # (or **-ignore**) skips over an element in the template (for instance, to avoid adding <title> or <titlealts> to the TopicDesc set)

# Specifying attributes for a set

```
<section outputclass="-set Audience">
  <p outputclass="-attributes"
    audience="a-choice administrator developer user"/>
  <p outputclass="introduction"/>
  <note outputclass="advisory"/>
</section>
```

**Design  
template**

- **Before the first member, use -attribute to specify**

Must be the same element as a member of the set

# Adding restriction to a restricted attribute

```
<topic outputclass="defectReport
  a-name importance
    a-choice low normal high urgent
    a-required
    a-default"
  importance="normal">
  ...
</topic>
```

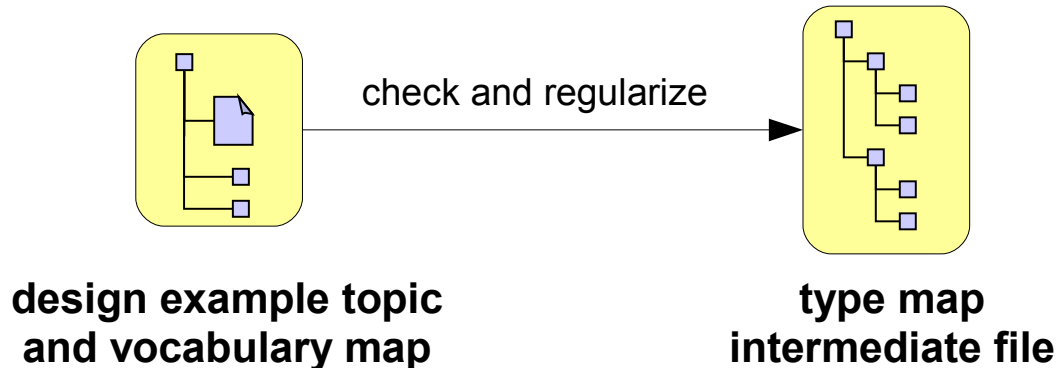
**Design  
template**

- **Used for attributes restricted to an enumeration or datatype**

Specify the options after a-name with outputclass

Specify the default value in the attribute to ensure a valid value

# Generating the processable intermediate format



- **First transform in the processing pipeline**

- Check for specialization errors not prevented by base validation

- Read element relationships from design example topic

- Represent containment and inheritance relationships in type map

- **Type map is easy to process**

- Same principle as taxonomy map – relationships between definitional topics

- Could be generated from other sources

# Pluggability via a specialized map

```
<documentTypeShell id="concept">
  <documentTypeProperties>
    <documentPublicID> ... </documentPublicID>
    ... other properties of the document type ...
  </documentTypeProperties>
  <structure>
    <topicModule href="concept/concept.ditamap">
      <topicModule href="concept/concept.ditamap"/>
    </topicModule>
  </structure>
  <vocabulary>
    <domainModule href="highlight/highlight.ditamap"/>
    ...
  </vocabulary>
</documentType>
```

- **Assembles a document type**

Captures metadata for generating a complex document type

Plugs in topic types and vocabulary domain modules

Controls nesting of topics