

# DRI Schema Reference

## DSpace System Documentation: DRI Schema Reference

Digital Repository Interface (DRI) is a schema that governs the structure of a Manakin DSpace page when encoded as an XML Document. It determines what elements can be present in the Document and the relationship of those elements to each other. This reference document explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. The appendix includes a complete reference for elements used in the DRI Schema, a graphical representation of the element hierarchy, and a quick reference table of elements and attributes.

### Table of Contents:

- 1 [Introduction](#)
  - 1.1 [The Purpose of DRI](#)
  - 1.2 [The Development of DRI](#)
- 2 [DRI in Manakin](#)
  - 2.1 [Themes](#)
  - 2.2 [Aspect Chains](#)
- 3 [Common Design Patterns](#)
  - 3.1 [Localization and Internationalization](#)
  - 3.2 [Standard attribute triplet](#)
  - 3.3 [Structure-oriented markup](#)
- 4 [Schema Overview](#)
- 5 [Merging of DRI Documents](#)
- 6 [Version Changes](#)
  - 6.1 [Changes from 1.0 to 1.1](#)
- 7 [Element Reference](#)
  - 7.1 [BODY](#)
  - 7.2 [cell](#)
  - 7.3 [div](#)
  - 7.4 [DOCUMENT](#)
  - 7.5 [field](#)
  - 7.6 [figure](#)
  - 7.7 [head](#)
  - 7.8 [help](#)
  - 7.9 [hi](#)
  - 7.10 [instance](#)
  - 7.11 [item](#)
  - 7.12 [label](#)
  - 7.13 [list](#)
  - 7.14 [META](#)
  - 7.15 [metadata](#)
  - 7.16 [OPTIONS](#)
  - 7.17 [p](#)
  - 7.18 [pageMeta](#)
  - 7.19 [params](#)
  - 7.20 [reference](#)
  - 7.21 [referenceSet](#)
  - 7.22 [repository](#)
  - 7.23 [repositoryMeta](#)
  - 7.24 [row](#)
  - 7.25 [table](#)
  - 7.26 [trail](#)
  - 7.27 [userMeta](#)
  - 7.28 [value](#)
  - 7.29 [xref](#)

## Introduction

This manual describes the Digital Repository Interface (DRI) as it applies to the DSpace digital repository and XMLUI Manakin based interface. DSpace XML UI is a comprehensive user interface system. It is centralized and generic, allowing it to be applied to all DSpace pages, effectively replacing the JSP-based interface system. Its ability to apply specific styles to arbitrarily large sets of DSpace pages significantly eases the task of adapting the DSpace look and feel to that of the adopting institution. This also allows for several levels of branding, lending institutional credibility to the repository and collections.

Manakin, the second version of DSpace XML UI, consists of several components, written using Java, XML, and XSL, and is implemented in [Cocoon](#). Central to the interface is the XML Document, which is a semantic representation of a DSpace page. In Manakin, the XML Document adheres to a schema called the Digital Repository Interface (DRI) Schema, which was developed in conjunction with Manakin and is the subject of this guide. For the remainder of this guide, the terms XML Document, DRI Document, and Document will be used interchangeably.

This reference document explains the purpose of DRI, provides a broad architectural overview, and explains common design patterns. The appendix includes a complete reference for elements used in the DRI Schema, a graphical representation of the element hierarchy, and a quick reference table of elements and attributes.

## The Purpose of DRI

DRI is a schema that governs the structure of the XML Document. It determines the elements that can be present in the Document and the relationship of those elements to each other. Since all Manakin components produce XML Documents that adhere to the DRI schema, The XML Document serves as the abstraction layer. Two such components, Themes and Aspects, are essential to the workings of Manakin and are described briefly in this manual.

## The Development of DRI

The DRI schema was developed for use in Manakin. The choice to develop our own schema rather than adapt an existing one came after a careful analysis of the schema's purpose as well as the lessons learned from earlier attempts at customizing the DSpace interface. Since every DSpace page in Manakin exists as an XML Document at some point in the process, the schema describing that Document had to be able to structurally represent all content, metadata and relationships between different parts of a DSpace page. It had to be precise enough to avoid losing any structural information, and yet generic enough to allow Themes a certain degree of freedom in expressing that information in a readable format.

Popular schemas such as XHTML suffer from the problem of not relating elements together explicitly. For example, if a heading precedes a paragraph, the heading is related to the paragraph not because it is encoded as such but because it happens to precede it. When these structures are attempted to be translated into formats where these types of relationships are explicit, the translation becomes tedious, and potentially problematic. More structured schemas, like TEI or DocBook, are domain specific (much like DRI itself) and therefore not suitable for our purposes.

We also decided that the schema should natively support a metadata standard for encoding artifacts. Rather than encoding artifact metadata in structural elements, like tables or lists, the schema would include artifacts as objects encoded in a particular standard. The inclusion of metadata in native format would enable the Theme to choose the best method to render the artifact for display without being tied to a particular structure.

Ultimately, we chose to develop our own schema. We have constructed the DRI schema by incorporating other standards when appropriate, such as [Cocoon's i18n schema](#) for internationalization, [DCMI's Dublin Core](#), and the [Library of Congress's METS schema](#). The design of structural elements was derived primarily from [TEI](#), with some of the design patterns borrowed from other existing standards such as DocBook and XHTML. While the structural elements were designed to be easily translated into XHTML, they preserve the semantic relationships for use in more expressive languages.

## DRI in Manakin

The general process for handling a request in DSpace XML UI consists of two parts. The first part builds the XML Document, and the second part stylizes that Document for output. In Manakin, the two parts are not discrete and instead wrapped within two processes: Content Generation, which builds an XML representation of the page, and Style Application, which stylizes the resulting Document. Content Generation is performed by Aspect chaining, while Style Application is performed by a Theme.

## Themes

A Theme is a collection of XSL stylesheets and supporting files like images, CSS styles, translations, and help documents. The XSL stylesheets are applied to the DRI Document to convert it into a readable format and give it structure and basic visual formatting in that format. The supporting files are used to provide the page with a specific look and feel, insert images and other media, translate the content, and perform other tasks. The currently used output format is XHTML and the supporting files are generally limited to CSS, images, and JavaScript. More output formats, like PDF or SVG, may be added in the future.

A DSpace installation running Manakin may have several Themes associated with it. When applied to a page, a Theme determines most of the page's look and feel. Different themes can be applied to different sets of DSpace pages allowing for both variety of styles between sets of pages and consistency within those sets. The `xmlui.xconf` configuration file determines which Themes are applied to which DSpace pages (see the Chapter 7. Manakin [XMLUI] Configuration and Customization for more information on installing and configuring themes). Themes may be configured to apply to all pages of specific type, like browse-by-title, to all pages of a one particular community or collection or sets of communities and collections, and to any mix of the two. They can also be configured to apply to a single arbitrary page or handle.

## Aspect Chains

Manakin Aspects are arrangements of Cocoon components (transformers, actions, matchers, etc) that implement a new set of coupled features for the system. These Aspects are chained together to form all the features of Manakin. Five Aspects exist in the default installation of Manakin, each handling a particular set of features of DSpace, and more can be added to implement extra features. All Aspects take a DRI Document as input and generate one as output. This allows Aspects to be linked together to form an Aspect chain. Each Aspect in the chain takes a DRI Document as input, adds its own functionality, and passes the modified Document to the next Aspect in the chain.

## Common Design Patterns

There are several design patterns used consistently within the DRI schema. This section identifies the need for and describes the implementation of these patterns. Three patterns are discussed: language and internationalization issues, standard attribute triplet (*id*, *n*, and *rend*), and the use of structure-oriented markup.

## Localization and Internationalization

Internationalization is a very important component of the DRI system. It allows content to be offered in other languages based on user's locale and conditioned upon availability of translations, as well as present dates and currency in a localized manner. There are two types of translated content: content stored and displayed by DSpace itself, and content introduced by the DRI styling process in the XSL transformations. Both types are handled by Cocoon's i18n transformer without regard to their origin.

When the Content Generation process produces a DRI Document, some of the textual content may be marked up with *i18n* elements to signify that translations are available for that content. During the Style Application process, the Theme can also introduce new textual content, marking it up with *i18n* tags. As a result, after the Theme's XSL templates are applied to the DRI Document, the final output consists of a DSpace page marked up in the chosen display format (like XHTML) with *i18n* elements from both DSpace and XSL content. This final document is sent through Cocoon's *i18n* transformer that translates the marked up text.

## Standard attribute triplet

Many elements in the DRI system (all top-level containers, character classes, and many others) contain one or several of the three standard attributes: *id*, *n*, and *rend*. The *id* and *n* attributes can be required or optional based on the element's purpose, while the *rend* attribute is always optional. The first two are used for identification purposes, while the third is used as a display hint issued to the styling step.

Identification is important because it allows elements to be separated from their peers for sorting, special case rendering, and other tasks. The first attribute, *id*, is the global identifier and it is unique to the entire document. Any element that contains an *id* attribute can thus be uniquely referenced by it. The *id* attribute of an element can be either assigned explicitly, or generated from the Java Class Path of the originating object if no name is given. While all elements that can be uniquely identified can carry the *id* attribute, only those that are independent on their context are required to do so. For example, tables are required to have an *id* since they retain meaning regardless of their location in the document, while table rows and cells can omit the attribute since their meaning depends on the parent element.

The name attribute *n* is simply the name assigned to the element, and it is used to distinguish an element from its immediate peers. In the example of a particular list, all items in that list will have different names to distinguish them from each other. Other lists in the document, however, can also contain items whose names will be different from each other, but identical to those in the first list. The *n* attribute of an element is therefore unique only in the scope of that element's parent and is used mostly for sorting purposes and special rendering of a certain class of elements, like, for example, all first items in lists, or all items named "browse". The *n* attribute follows the same rules as *id* when determining whether or not it is required for a given element.

The last attribute in the standard triplet is *rend*. Unlike *id* and *n*, the *rend* attribute can consist of several space delimited values and is optional for all elements that can contain it. Its purpose is to provide a rendering hint from the middle layer component to the styling theme. How that hint is interpreted and whether it is used at all when provided, is completely up the theme. There are several cases, however, where the content of the *rend* attribute is outlined in detail and its use is encouraged. Those cases are the emphasis element *hi*, the division element *div*, and the *list* element. Please refer to the Element Reference for more detail on these elements.

## Structure-oriented markup

The final design pattern is the use of structure-oriented markup for content carried by the XML Document. Once generated by Cocoon, the Document contains two major types of information: metadata about the repository and its contents, and the actual content of the page to be displayed. A complete overview of metadata and content markup and their relationship to each other is given in the next section. An important thing to note here, however, is that the markup of the content is oriented towards explicitly stating structural relationships between the elements rather than focusing on the presentational aspects. This makes the markup used by the Document more similar to TEI or Docbook rather than HTML. For this reason, XSL templates are used by the themes to convert structural DRI markup to XHTML. Even then, an attempt is made to create XHTML as structural as possible, leaving presentation entirely to CSS. This allows the XML Document to be generic enough to represent any DSpace page without dictating how it should be rendered.

## Schema Overview

The DRI XML Document consists of the root element *document* and three top-level elements that contain two major types of elements. The three top-level containers are *meta*, *body*, and *options*. The two types of elements they contain are metadata and content, carrying metadata about the page and the contents of the page, respectively. Figure 1 depicts the relationship between these six components.

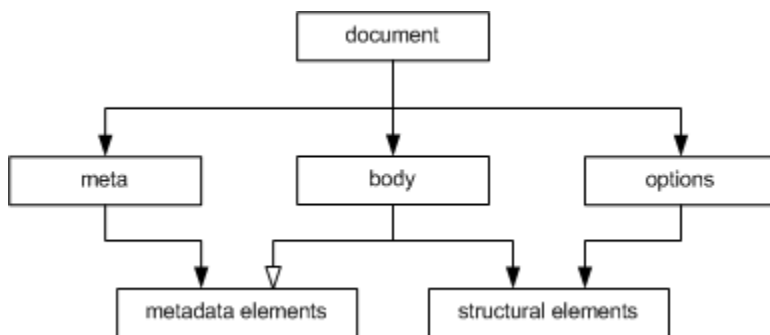


Figure 1: The two content types across three major divisions of a DRI page.

The *document* element is the root for all DRI pages and contains all other elements. It bears only one attribute, *version*, that contains the version number of the DRI system and the schema used to validate the produced document. At the time of writing the working version number is "1.1".

The *meta* element is a the top-level element under *document* and contains all metadata information about the page, the user that requested it, and the repository it is used with. It contains no structural elements, instead being the only container of metadata elements in a DRI Document. The metadata stored by the *meta* element is broken up into three major groups: *userMeta*, *pageMeta*, and *objectMeta*, each storing metadata information about their respective component. Please refer to the reference entries for more information about these elements.

The *options* element is another top-level element that contains all navigation and action options available to the user. The options are stored as items in list elements, broken up by the type of action they perform. The five types of actions are: browsing, search, language selection, actions that are always available, and actions that are context dependent. The two action types also contain sub-lists that contain actions available to users of varying degrees of access to the system. The *options* element contains no metadata elements and can only make use of a small set of structural elements, namely the *list* element and its children.

The last major top-level element is the *body* element. It contains all structural elements in a DRI Document, including the lists used by the *options* element. Structural elements are used to build a generic representation of a DSpace page. Any DSpace page can be represented with a combination of the structural elements, which will in turn be transformed by the XSL templates into another format. This is the core mechanism that allows DSpace XML UI to apply uniform templates and styling rules to all DSpace pages and is the fundamental difference from the JSP approach currently used by DSpace.

The *body* element directly contains only one type of element: *div*. The *div* element serves as a major division of content and any number of them can be contained by the *body*. Additionally, divisions are recursive, allowing *divs* to contain other *divs*. It is within these elements that all other structural elements are contained. Those elements include tables, paragraph elements *p*, and lists, as well as their various children elements. At the lower levels of this hierarchy lie the character container elements. These elements, namely paragraphs *p*, table *cells*, lists *items*, and the emphasis element *hi*, contain the textual content of a DSpace page, optionally modified with links, figures, and emphasis. If the division within which the character class is contained is tagged as interactive (via the *interactive* attribute), those elements can also contain interactive form fields. Divisions tagged as interactive must also provide *method* and *action* attributes for its fields to use.

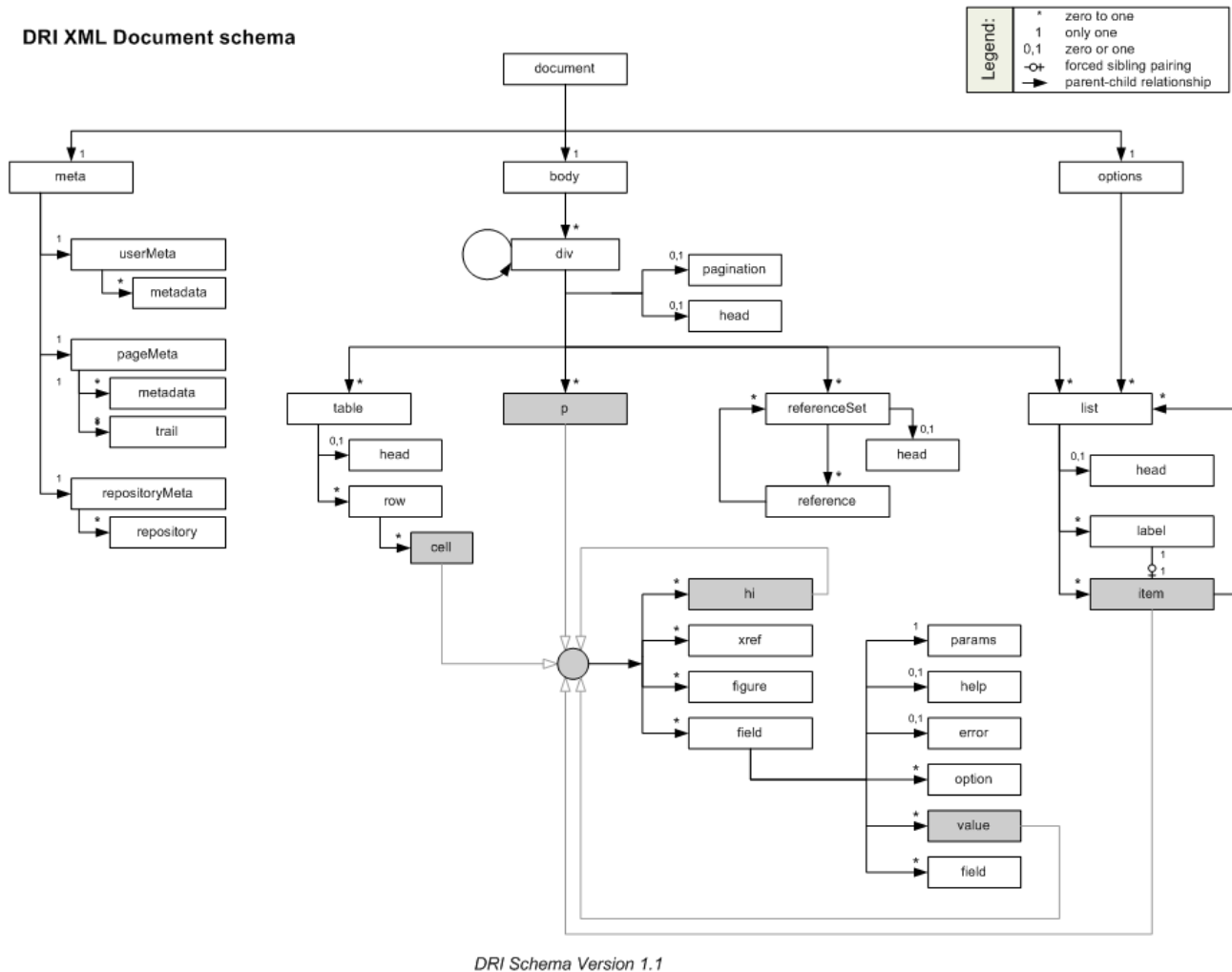


Figure 2: All the elements in the DRI schema (version 1.1).

## Merging of DRI Documents

Having described the structure of the DRI Document, as well as its function in Manakin's Aspect chains, we now turn our attention to the one last detail of their use: merging two Documents into one. There are several situations where the need to merge two documents arises. In Manakin, for example, every Aspect is responsible for adding different functionality to a DSpace page. Since every instance of a page has to be a complete DRI Document, each Aspect is faced with the task of merging the Document it generated with the ones generated (and merged into one Document) by previously executed Aspects. For this reason rules exist that describe which elements can be merged together and what happens to their data and child elements in the process.

When merging two DRI Documents, one is considered to be the main document, and the other a feeder document that is added in. The three top level containers (*meta*, *body* and *options*) of both documents are then individually analyzed and merged. In the case of the *options* and *meta* elements, the children tags are taken individually as well and treated differently from their siblings.

The *body* elements are the easiest to merge: their respective *div* children are preserved along with their ordering and are grouped together under one element. Thus, the new *body* tag will contain all the *divs* of the main document followed by all the *divs* of the feeder. However, if two *divs* have the same *n* and *rend* attributes (and in case of an interactive *div* the same *action* and *method* attributes as well), those *divs* will be merged into one. The resulting *div* will bear the *id*, *n*, and *rend* attributes of the main document's *div* and contain all the *divs* of the main document followed by all the *divs* of the feeder. This process continues recursively until all the *divs* have been merged. It should be noted that two divisions with separate pagination rules cannot be merged together.

Merging the *options* elements is somewhat different. First, *list* elements under *options* of both documents are compared with each other. Those unique to either document are simply added under the new *options* element, just like *divs* under *body*. In case of duplicates, that is *list* elements that belong to both documents and have the same *n* attribute, the two *lists* will be merged into one. The new *list* element will consist of the main document's head element, followed *label-item* pairs from the main document, and then finally the *label-item* pairs of the feeder, provided they are different from those of the main.

Finally, the *meta* elements are merged much like the elements under *body*. The three children of *meta* - *userMeta*, *pageMeta*, and *objectMeta* - are individually merged, adding the contents of the feeder after the contents of the main.

## Version Changes

The DRI schema will continue to evolve overtime as the needs of interface design require. The version attribute on the document will indicate which version of the schema the document conforms to. At the time Manakin was incorporated into the standard distribution of DSPac the current version was "1.1", however earlier versions of the Manakin interface may use "1.0".

### Changes from 1.0 to 1.1

There were major structural changes between these two version numbers. Several elements were removed from the schema: *includeSet*, *include*, *objectMeta*, and *object*. Originally all metadata for objects were included in-line with the DRI document, this proved to have several problems and has been removed in version 1.1 of the DRI schema. Instead of including metadata in-line, external references to the metadata is included. Thus, a *reference* element has been added along with *referenceSet*. These new elements operate like their counterparts in the previous version except referencing metadata contained on the *objectMeta* element they reference metadata in external files. The *repository* and *repositoryMeta* elements were also modified in a similar manner removing in-line metadata and referencing external metadata documents.

## Element Reference

Element	Attributes	Required?
<b>BODY</b>		
cell		
	cols	
	id	
	n	
	rend	
	role	
	rows	
div		
	action	required for interactive behavior
	behaviorSensitivityFields	
	currentPage	
	firstItemIndex	
	id	required
	interactive	
	itemsTotal	
	lastItemIndex	
	method	required for interactive
	n	required
	nextPage	
	pagesTotal	

	pageURLMask	
	pagination	
	previousPage	
	rend	
<b>DOCUMENT</b>	version	required
field		
	disabled	
	id	required
	n	required
	rend	
	required	
	type	required
figure		
	rend	
	source	
	target	
head		
	id	
	n	
	rend	
help		
hi	rend	required
instance		
item		
	id	
	n	
	rend	
label		
	id	
	n	
	rend	
list		
	id	required
	n	required
	rend	
	type	
META		
metadata		
	element	required
	language	
	qualifier	
OPTIONS		
p		
	id	
	n	
	rend	
pageMeta		

params		
	cols	
	maxlength	
	multiple	
	operations	
	rows	
	size	
reference		
	url	required
	repositoryID	required
	type	
referenceSet		
	id	required
	n	required
	orderBy	
	rend	
	type	required
repository		
	repositoryID	required
	url	required
repositoryMeta		
row		
	id	
	n	
	rend	
	role	required
table		
	cols	required
	id	required
	n	required
	rend	
	rows	required
trail		
	rend	
	target	
userMeta	authenticated	required
value		
	optionSelected	
	optionValue	
	type	required
xref	target	required

## BODY

Top-Level Container

The *body* element is the main container for all content displayed to the user. It contains any number of *div* elements that group content into interactive and display blocks.

Parent

- document

Children

- div (any)

Attributes

- None

```
<document version=1.0>
  <meta> ... </meta>
  <body>
    <div n="division-example1"
      id="XMLExample.div.division-example1">
      ...
    </div>
    <div n="division-example2" id="XMLExample.div.division-example2"
      interactive="yes" action="www.DRItest.com"
      method="post">
      ...
    </div>
    ...
  </body>
  <options> ... </options>
</document>
```

## cell

Rich Text Container

Structural Element

The *cell* element contained in a *row* of a *table* carries content for that table. It is a character container, just like *p*, *item*, and *hi*, and its primary purpose is to display textual data, possibly enhanced with hyperlinks, emphasized blocks of text, images and form fields. Every *cell* can be annotated with a *role* (the most common being "header" and "data") and can stretch across any number of rows and columns. Since cells cannot exist outside their container, *row*, their *id* attribute is optional.

Parent

- row

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

Attributes

- **cols**: (optional) The number of columns the cell spans.
- **id**: (optional) A unique identifier of the element.
- **n**: (optional) A local identifier used to differentiate the element from its siblings.
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **role**: (optional) An optional attribute to override the containing row's role settings.
- **rows**: (optional) The number of rows the cell spans.



```

<table n="table-example" id="XMLExample.table.table-example" rows="2"
  cols="3">
  <row role="head">
    <cell cols="2">Data Label One and Two</cell> <cell>Data Label
      Three</cell>
    ...
  </row>
  <row>
    <cell> Value One </cell> <cell> Value Two </cell> <cell> Value
      Three </cell>
    ...
  </row>
  ...
</table>

```

## div

### Structural Element

The *div* element represents a major section of content and can contain a wide variety of structural elements to present that content to the user. It can contain paragraphs, tables, and lists, as well as references to artifact information stored in *artifactMeta*, *repositoryMeta*, *collections*, and *communities*. The *div* element is also recursive, allowing it to be further divided into other divs. Divs can be of two types: interactive and static. The two types are set by the use of the *interactive* attribute and differ in their ability to contain interactive content. Children elements of divs tagged as interactive can contain form fields, with the *action* and *method* attributes of the *div* serving to resolve those fields.

### Parent

- body
- div

### Children

- head (zero or one)
- pagination (zero or one)
- table (any)
- p (any)
- referenceSet (any)
- list (any)
- div (any)

### Attributes

- **action**: (required for interactive) The form action attribute determines where the form information should be sent for processing.
- **behavior**: (optional for interactive) The acceptable behavior options that may be used on this form. The only possible value defined at this time is "ajax" which means that the form may be submitted multiple times for each individual field in this form. Note that if the form is submitted multiple times it is best for the behaviorSensitiveFields to be updated as well.
- **behaviorSensitiveFields**: (optional for interactive) A space separated list of field names that are sensitive to behavior. These fields must be updated each time a form is submitted with out a complete refresh of the page (i.e. ajax).
- **currentPage**: (optional) For paginated divs, the currentPage attribute indicates the index of the page currently displayed for this div.
- **firstItemIndex**: (optional) For paginated divs, the firstItemIndex attribute indicates the index of the first item included in this div.
- **id**: (required) A unique identifier of the element.
- **interactive**: (optional) Accepted values are "yes", "no". This attribute determines whether the div is interactive or static. Interactive divs must provide action and method and can contain field elements.
- **itemsTotal**: (optional) For paginated divs, the itemsTotal attribute indicates how many items exist across all paginated divs.
- **lastItemIndex**: (optional) For paginated divs, the lastItemIndex attribute indicates the index of the last item included in this div.
- **method**: (required for interactive) Accepted values are "get", "post", and "multipart". Determines the method used to pass gathered field values to the handler specified by the action attribute. The multipart method should be used for uploading files.
- **n**: (required) A local identifier used to differentiate the element from its siblings.
- **nextPage**: (optional) For paginated divs the nextPage attribute points to the URL of the next page of the div, if it exists.
- **pagesTotal**: (optional) For paginated divs, the pagesTotal attribute indicates how many pages the paginated divs spans.
- **pageURLMask**: (optional) For paginated divs, the pageURLMask attribute contains the mask of a url to a particular page within the paginated set. The destination page's number should replace the {pageNum} string in the URL mask to generate a full URL to that page.
- **pagination**: (optional) Accepted values are "simple", "masked". This attribute determines whether the div is spread over several pages. Simple paginated divs must provide previousPage, nextPage, itemsTotal, firstItemIndex, lastItemIndex attributes. Masked paginated divs must provide currentPage, pagesTotal, pageURLMask, itemsTotal, firstItemIndex, lastItemIndex attributes.
- **previousPage**: (optional) For paginated divs the previousPage attribute points to the URL of the previous page of the div, if it exists.
- **rend**: (optional) A rendering hint used to override the default display of the element. In the case of the div tag, it is also encouraged to label it as either "primary" or "secondary". Divs marked as primary contain content, while secondary divs contain auxiliary information or supporting fields.

```

<body>
  <div n="division-example"
    id="XMLExample.div.division-example">
    <head> Example Division </head>
    <p> This example shows the use of divisions. </p>
    <table ...>
      ...
    </table>
    <referenceSet ...>
      ...
    </referenceSet>
    <list ...>
      ...
    </list>
    <div n="sub-division-example"
      id="XMLExample.div.sub-division-example">
      <p> Divisions may be nested </p>
      ...
    </div>
    ...
  </div>
  ...
</body>

```

## DOCUMENT

### Document Root

The document element is the root container of an XML UI document. All other elements are contained within it either directly or indirectly. The only attribute it carries is the version of the Schema to which it conforms.

### Parent

- none

### Children

- meta (one)
- body (one)
- options (one)

### Attributes

- **version:** (required) Version number of the schema this document adheres to. At the time of writing the only valid version numbers are "1.0" or "1.1". Future iterations of this schema may increment the version number.

```

<document
  version="1.1">
  <meta>
    ...
  </meta>
  <body>
    ...
  </body>
  <options>
    ...
  </options>
</document>

```

## field

### Text Container

### Structural Element

The *field* element is a container for all information necessary to create a form field. The required *type* attribute determines the type of the field, while the children tags carry the information on how to build it. Fields can only occur in divisions tagged as "interactive".

#### Parent

- cell
- p
- hi
- item

#### Children

- params (one)
- help (zero or one)
- error (any)
- option (any - only with the select type)
- value (any - only available on fields of type: select, checkbox, or radio)
- field (one or more - only with the composite type)
- valueSet (any)

#### Attributes

- **disabled**: (optional) Accepted values are "yes", "no". Determines whether the field allows user input. Rendering of disabled fields may vary with implementation and display media.
- **id**: (required) A unique identifier for a field element.
- **n**: (required) A non-unique local identifier used to differentiate the element from its siblings within an interactive division. This is the name of the field use when data is submitted back to the server.
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **required**: (optional) Accepted values are "yes", "no". Determines whether the field is a required component of the form and thus cannot be left blank.
- **type**: (required) A required attribute to specify the type of value. Accepted types are:
  - **button**: A button input control that when activated by the user will submit the form, including all the fields, back to the server for processing.
  - **checkbox**: A boolean input control which may be toggled by the user. A checkbox may have several fields which share the same name and each of those fields may be toggled independently. This is distinct from a radio button where only one field may be toggled.
  - **file**: An input control that allows the user to select files to be submitted with the form. Note that a form which uses a file field must use the multipart method.
  - **hidden**: An input control that is not rendered on the screen and hidden from the user.
  - **password**: A single-line text input control where the input text is rendered in such a way as to hide the characters from the user.
  - **radio**: A boolean input control which may be toggled by the user. Multiple radio button fields may share the same name. When this occurs only one field may be selected to be true. This is distinct from a checkbox where multiple fields may be toggled.
  - **select**: A menu input control which allows the user to select from a list of available options.
  - **text**: A single-line text input control.
  - **textarea**: A multi-line text input control.
  - **composite**: A composite input control combines several input controls into a single field. The only fields that may be combined together are: checkbox, password, select, text, and textarea. When fields are combined together they can possess multiple combined values.

```
<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  ...
  <field id="XMLEExample.field.name" n="name" type="text"
    required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
      content</i18n>.</help>
    <value type="raw">Default value goes
      here</value>
  </field>
</p>
```

## figure

#### Text Container

#### Structural Element

The *figure* element is used to embed a reference to an image or a graphic element. It can be mixed freely with text, and any text within the tag itself will be used as an alternative descriptor or a caption.

#### Parent

- cell
- p

- hi
- item

Children

- none

Attributes

- **rend:** (optional) A rendering hint used to override the default display of the element.
- **source:** (optional) The source for the image, using either a URL or a pre-defined XML entity.
- **target:** (optional) A target for an image used as a link, using either a URL or an id of an existing element as a destination.

```
<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <field> ... </field>
  ...
  <figure source="www.example.com/fig1"> This is a static image.
    </figure> <figure source="www.example.com/fig1"
      target="www.example.net">
      This image is also a link.
    </figure>
  ...
</p>
```

## head

Text Container

Structural Element

The *head* element is primarily used as a label associated with its parent element. The rendering is determined by its parent tag, but can be overridden by the *rend* attribute. Since there can only be one *head* element associated with a particular tag, the *n* attribute is not needed, and the *id* attribute is optional.

Parent

- div
- table
- list
- referenceSet

Children

- none

Attributes

- **id:** (optional) A unique identifier of the element
- **n:** (optional) A local identifier used to differentiate the element from its siblings
- **rend:** (optional) A rendering hint used to override the default display of the element.

```

<div ...>
  <head> This is a simple header associated with its div element.
  </head>
  <div ...>
    <head rend="green"> This header will be green.
    </head>
    <p>
      <head> A header with <i18n>localized content</i18n>.
      </head>
      ...
    </p>
  </div>
  <table ...>
    <head> ...
    </head>
    ...
  </table>
  <list ...>
    <head> ...
    </head>
    ...
  </list>
  ...
</body>

```

## help

Text Container

Structural Element

The optional *help* element is used to supply help instructions in plain text and is normally contained by the *field* element. The method used to render the help text in the target markup is up to the theme.

Parent

- field

Children

- none

Attributes

- None

```

<p>
  <hi> ... </hi>
  ...
  <xref> ... </xref>
  ...
  <figure> ... </figure>
  ...
  <field id="XMLExample.field.name" n="name" type="text"
    required="yes">
    <params size="16" maxlength="32" />
    <help>Some help text with <i18n>localized
      content</i18n>.</help>
  </field>
  ...
</p>

```

## hi

Rich Text Container

## Structural Element

The *hi* element is used for emphasis of text and occurs inside character containers like *p* and *list* item. It can be mixed freely with text, and any text within the tag itself will be emphasized in a manner specified by the required *rend* attribute. Additionally, *hi* element is the only text container component that is a rich text container itself, meaning it can contain other tags in addition to plain text. This allows it to contain other text containers, including other *hi* tags.

### Parent

- cell
- p
- item
- hi

### Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

### Attributes

- **rend:** (required) A required attribute used to specify the exact type of emphasis to apply to the contained text. Common values include but are not limited to "bold", "italic", "underline", and "emph".

```
<p>
  This text is normal, while <hi rend="bold">this text is bold and
    this text is <hi rend="italic">bold and
      italic.</hi></hi>
</p>
```

## instance

### Structural Element

The *instance* element contains the value associated with a form field's multiple instances. Fields encoded as an instance should also include the values of each instance as a hidden field. The hidden field should be appended with the index number for the instance. Thus if the field is "firstName" each instance would be named "firstName\_1", "firstName\_2", "firstName\_3", etc...

### Parent

- field

### Children

- value

### Attributes

- None listed yet.

Example needed.

## item

### Rich Text Container

### Structural Element

The *item* element is a rich text container used to display textual data in a list. As a rich text container it can contain hyperlinks, emphasized blocks of text, images and form fields in addition to plain text.

The *item* element can be associated with a label that directly precedes it. The Schema requires that if one *item* in a *list* has an associated *label*, then all other items must have one as well. This mitigates the problem of loose connections between elements that is commonly encountered in XHTML, since every item in particular list has the same structure.

### Parent

- list

### Children

- `hi` (any)
- `xref` (any)
- `figure` (any)
- `field` (any)
- `list` (any)

#### Attributes

- **id**: (optional) A unique identifier of the element
- **n**: (optional) A non-unique local identifier used to differentiate the element from its siblings
- **rend**: (optional) A rendering hint used to override the default display of the element.

```
<list n="list-example"
      id="XMLExample.list.list-example">
  <head> Example List </head>
  <item> This is the first item
    </item> <item> This is the second item with <hi ...>highlighted text</hi>,
    <xref ...> a link</xref> and an <figure
    ...>image</figure>.</item>
  ...
</list>
<list n="list-example2"
      id="XMLExample.list.list-example2">
  <head> Example List </head>
  <label>ITEM ONE:</label>
  <item> This is the first item
    </item>
  <label>ITEM TWO:</label>
  <item> This is the second item with <hi ...>highlighted
    text</hi>, <xref ...> a link</xref> and an <figure
    ...>image</figure>.</item>
  <label>ITEM THREE:</label>
  <item> This is the third item with a <field ...> ... </field>
    </item>
  ...
</list>
<item> This is the third item in the list
  </item>
...
</list>
```

## label

Text Container

Structural Element

The *label* element is associated with an item and annotates that item with a number, a textual description of some sort, or a simple bullet.

Parent

- item

Children

- none

Attributes

- **id**: (optional) A unique identifier of the element
- **n**: (optional) A local identifier used to differentiate the element from its siblings
- **rend**: (optional) An optional rend attribute provides a hint on how the label should be rendered, independent of its type.

```

<list n="list-example"
      id="XMLExample.list.list-example">
  <head>Example List</head>
  <label>1</label>
  <item> This is the first item  </item>
  <label>2</label>
  <item> This is the second item with <hi ...>highlighted text</hi>,
    <xref ...> a link</xref> and an <figure
    ...>image</figure>.</item>
  ...
<list n="list-example2"
      id="XMLExample.list.list-example2">
  <head>Example Sublist</head>
  <label>ITEM
    ONE:</label>
  <item> This is the first item  </item>
  <label>ITEM
    TWO:</label>
  <item> This is the second item with <hi ...>highlighted
    text</hi>, <xref ...> a link</xref> and an <figure
    ...>image</figure>.</item>
  <label>ITEM
    THREE:</label>
  <item> This is the third item with a <field ...> ... </field>
    </item>
  ...
</list>
<item> This is the third item in the list </item>
...
</list>

```

## list

### Structural Element

The *list* element is used to display sets of sequential data. It contains an optional *head* element, as well as any number of *item* and *list* elements. *Items* contain textual information, while sublists contain other *item* or *list* elements. An *item* can also be associated with a *label* element that annotates an item with a number, a textual description of some sort, or a simple bullet. The list type (ordered, bulleted, gloss, etc.) is then determined either by the content of *labels* on *items* or by an explicit value of the *type* attribute. Note that if *labels* are used in conjunction with any *items* in a list, all of the *items* in that list must have a *label*. It is also recommended to avoid mixing *label* styles unless an explicit type is specified.

### Parent

- div
- list

### Children

- head (zero or one)
- label (any)
- item (any)
- list (any)

### Attributes

- **id**: (required) A unique identifier of the element
- **n**: (required) A local identifier used to differentiate the element from its siblings
- **rend**: (optional) An optional rend attribute provides a hint on how the list should be rendered, independent of its type. Common values are but not limited to:
  - **alphabet**: The list should be rendered as an alphabetical index
  - **columns**: The list should be rendered in equal length columns as determined by the theme.
  - **columns2**: The list should be rendered in two equal columns.
  - **columns3**: The list should be rendered in three equal columns.
  - **horizontal**: The list should be rendered horizontally.
  - **numeric**: The list should be rendered as a numeric index.
  - **vertical**: The list should be rendered vertically.
- **type**: (optional) An optional attribute to explicitly specify the type of list. In the absence of this attribute, the type of a list will be inferred from the presence and content of labels on its items. Accepted values are:
  - **form**: Used for form lists that consist of a series of fields.
  - **bulleted**: Used for lists with bullet-marked items.



- **gloss:** Used for lists consisting of a set of technical terms, each marked with a *label* element and accompanied by the definition marked as an *item* element.
- **ordered:** Used for lists with numbered or lettered items.
- **progress:** Used for lists consisting of a set of steps currently being performed to accomplish a task. For this type to apply, each *item* in the list should represent a step and be accompanied by a *label* that contains the displayable name for the step. The *item* contains an *xref* that references the step. Also the *rend* attribute on the *item* element should be: "available" (meaning the user may jump to the step using the provided *xref*), "unavailable" (the user has not meet the requirements to jump to the step), or "current" (the user is currently on the step)
- **simple:** Used for lists with items not marked with numbers or bullets.

```
<div ...>
...
<list n="list-example"
      id="XMLExample.list.list-example">
  <head>Example List</head>
  <item> ... </item>
  <item> ... </item>
...
<list n="list-example2"
      id="XMLExample.list.list-example2">
  <head>Example Sublist</head>
  <label> ... </label>
  <item> ... </item>
  <label> ... </label>
  <item> ... </item>
  <label> ... </label>
  <item> ... </item>
...
</list>
<label> ... </label>
<item> ... </item>
...
</list>
</div>
```

## META

### Top-Level Container

The *meta* element is a top level element and exists directly inside the *document* element. It serves as a container element for all metadata associated with a document broken up into categories according to the type of metadata they carry.

### Parent

- document

### Children

- userMeta (one)
- pageMeta (one)
- repositoryMeta (one)

### Attributes

- None

```
<document version=1.0>
  <meta>
    <userMeta> ... </userMeta>
    <pageMeta> ... </pageMeta>
    <repositoryMeta> ... </repositoryMeta>
  </meta>
  <body> ... </body>
  <options> ... </options>
</document>
```

## metadata

Text Container

Structural Element

The *metadata* element carries generic metadata information in the form of an attribute-value pair. The type of information it contains is determined by two attributes: *element*, which specifies the general type of metadata stored, and an optional *qualifier* attribute that narrows the type down. The standard representation for this pairing is `element.qualifier`. The actual metadata is contained in the text of the tag itself. Additionally, a *language* attribute can be used to specify the language used for the metadata entry.

Parent

- `userMeta`
- `pageMeta`

Children

- none

Attributes

- **element:** (required) The name of a metadata field.
- **language:** (optional) An optional attribute to specify the language used in the metadata tag.
- **qualifier:** (optional) An optional postfix to the field name used to further differentiate the names.

```
<meta>
  <userMeta>
    <metadata element="identifier" qualifier="firstName"> Bob
    </metadata> <metadata element="identifier" qualifier="lastName"> Jones
    </metadata> <metadata ...> ...
    ...
  </userMeta>
  <pageMeta>
    <metadata element="rights"
      qualifier="accessRights">user</metadata> <metadata ...> ...
    </metadata>
    ...
  </pageMeta>
</meta>
```

## OPTIONS

Top-Level Container

The *options* element is the main container for all actions and navigation options available to the user. It consists of any number of *list* elements whose items contain navigation information and actions. While any list of navigational options may be contained in this element, it is suggested that at least the following 5 lists be included.

Parent

- `document`

Children

- `list (any)`

Attributes

- `None`

```
<document version=1.0>

  <meta> ... </meta>

  <body> ... </body>

  <options>

    <list n="navigation-example1"
      id="XMLExample.list.navigation-example1">
```

```

        <head>Example Navigation List 1</head>

        <item><xref target="/link/to/option">Option
One</xref></item>

        <item><xref target="/link/to/option">Option
two</xref></item>

        ...

    </list>

    <list n="navigation-example2"
id="XMLExample.list.navigation-example2">

        <head>Example Navigation List 2</head>

        <item><xref target="/link/to/option">Option
One</xref></item>

        <item><xref target="/link/to/option">Option
two</xref></item>

        ...

    </list>

    ...

</options>

</document>

```

## p

Rich Text Container

Structural Element

The *p* element is a rich text container used by *divs* to display textual data in a paragraph format. As a rich text container it can contain hyperlinks, emphasized blocks of text, images and form fields in addition to plain text.

Parent

- div

Children

- hi (any)
- xref (any)
- figure (any)
- field (any)

Attributes

- **id**: (optional) A unique identifier of the element.
- **n**: (optional) A local identifier used to differentiate the element from its siblings.
- **rend**: (optional) A rendering hint used to override the default display of the element.

```

<div n="division-example"
id="XMLExample.div.division-example">

    <p> This is a regular paragraph.
    </p> <p> This text is normal, while <hi rend="bold">this text is bold
and this text is <hi rend="italic">bold and italic.</hi></hi>
    </p> <p> This paragraph contains a <xref
target="/link/target">link</xref>, a static <figure
source="/image.jpg">image</figure>, and a <figure target=
"/link/target" source="/image.jpg">image link.</figure>
    </p>

```

</div>

## pageMeta

### Metadata Element

The *pageMeta* element contains metadata associated with the document itself. It contains generic *metadata* elements to carry the content, and any number of *trail* elements to provide information on the user's current location in the system. Required and suggested values for *metadata* elements contained in *pageMeta* include but are not limited to:

- browser (suggested): The user's browsing agent as reported to server in the HTTP request.
- browser.type (suggested): The general browser family as derived from the browser metadata field. Possible values may include "MSIE" (for Microsoft Internet Explorer), "Opera" (for the Opera browser), "Apple" (for Apple web kit based browsers), "Gecko" (for Netscape, Mozilla, and Firefox based browsers), or "Lynx" (for text based browsers).
- browser.version (suggested): The browser version as reported by HTTP Request.
- contextPath (required): The base URL of the Digital Repository system.
- redirect.time (suggested): The time that must elapse before the page is redirected to an address specified by the redirect.url *metadata* element.
- redirect.url (suggested): The URL destination of a redirect page
- title (required): The title of the document/page that the user currently browsing.

See the *metadata* and *trail* tag entries for more information on their structure.

### Parent

- meta

### Children

- metadata (any)
- trail (any)

### Attributes

- None

```
<meta>

  <userMeta> ... </userMeta>

  <pageMeta>

    <metadata element="title">Example DRI
    page</metadata>

    <metadata
    element="contextPath">/xmlui/</metadata>

    <metadata ...> ... </metadata>

    ...

    <trail source="123456789/6"> A bread crumb item
    </trail>

    <trail ...> ... </trail>

    ...

  </pageMeta>

</meta>
```

## params

### Structural Component

The *params* element identifies extra parameters used to build a form field. There are several attributes that may be available for this element depending on the field type.

#### Parent

- field

#### Children

- none

#### Attributes

- **cols:** (optional) The default number of columns that the text area should span. This applies only to textarea field types.
- **maxlength:** (optional) The maximum length that the theme should accept for form input. This applies to text and password field types.
- **multiple:** (optional) yes/no value. Determine if the field can accept multiple values for the field. This applies only to select lists.
- **operations:** (optional) The possible operations that may be performed on this field. The possible values are "add" and/or "delete". If both operations are possible then they should be provided as a space separated list. The "add" operations indicates that there may be multiple values for this field and the user may add to the set one at a time. The front-end should render a button that enables the user to add more fields to the set. The button must be named the field name appended with the string "\_add", thus if the field's name is "firstName" the button must be called "firstName\_add". The "delete" operation indicates that there may be multiple values for this field each of which may be removed from the set. The front-end should render a checkbox by each field value, except for the first. The checkbox must be named the field name appended with the string "\_selected", thus if the field's name is "firstName" the checkbox must be called "firstName\_selected" and the value of each successive checkbox should be the field name. The front-end must also render a delete button. The delete button name must be the field's name appended with the string "\_delete".
- **rows:** (optional) The default number of rows that the text area should span. This applies only to textarea field types.
- **size:** (optional) The default size for a field. This applies to text, password, and select field types.

```
<p>

<field id="XMLExample.field.name" n="name" type="text"
  required="yes">

  <params size="16"
    maxlength="32" />

  <help>Some help text with <i18n>localized
    content</i18n>.</help>

  <default>Default value goes here</default>

</field>

</p>
```

## reference

#### Metadata Reference Element

*reference* is a reference element used to access information stored in an external metadata file. The *url* attribute is used to locate the external metadata file. The *type* attribute provides a short limited description of the referenced object's type.

*reference* elements can be both contained by *includeSet* elements and contain *includeSets* themselves, making the structure recursive.

#### Parent

- referenceSet

#### Children

- referenceSet (zero or more)

#### Attributes

- **url:** (required) A url to the external metadata file.
- **repositoryIdentifier:** (required) A reference to the repositoryIdentifier of the repository.
- **type:** (optional) Description of the reference object's type.

```
<includeSet n="browse-list"
id="XMLTest.includeSet.browse-list">
  <reference url="/metadata/handle/123/4/mets.xml"
repositoryID="123" type="DSpace
Item"/> <reference url="/metadata/handle/123/5/mets.xml"
repositoryID="123" />
```

```
...
</includeSet>
```

## referenceSet

Metadata Reference Element

The *referenceSet* element is a container of artifact or repository references.

Parent

- div
- reference

Children

- head (zero or one)
- reference (any)

Attributes

- **id**: (required) A unique identifier of the element
- **n**: (required) Local identifier used to differentiate the element from its siblings
- **orderBy**: (optional) A reference to the metadata field that determines the ordering of artifacts or repository objects within the set. When the Dublin Core metadata scheme is used this attribute should be the element.qualified value that the set is sorted by. As an example, for a browse by title list, the value should be sortedBy=title, while for browse by date list it should be sortedBy=date.created
- **rend**: (optional) A rendering hint used to override the default display of the element.
- **type**: (required) Determines the level of detail for the given metadata. Accepted values are:
  - **summaryList**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that is suitable for quick scanning.
  - **summaryView**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a partial view of the referenced object or objects.
  - **detailList**: Indicates that the metadata from referenced artifacts or repository objects should be used to build a list representation that provides a complete, or near complete, view of the referenced objects. Whether such a view is possible or different from summaryView depends largely on the repository at hand and the implementing theme.
  - **detailView**: Indicates that the metadata from referenced artifacts or repository objects should be used to display complete information about the referenced object. Rendering of several references included under this type is up to the theme.

```
<div ...>
<head> Example Division </head>
<p> ... </p>
<table> ... </table>
<list>
...
</list>
  <referenceSet n="browse-list"
id="XMLTest.referenceSet.browse-list" type="summaryView"
informationModel="DSpace">
  <head>A header for the includeset</head>
  <reference
url="/metadata/handle/123/34/mets.xml"/>
  <reference
url="/metadata/handle/123/34/mets.xml/>
  </referenceSet>
  ...
</p>
```

## repository

Metadata Element

The *repository* element is used to describe the repository. Its principal component is a set of structural metadata that carrier information on how the repository's objects under *objectMeta* are related to each other. The principal method of encoding these relationships at the time of this writing is a METS document, although other formats, like RDF, may be employed in the future.

Parent

- repositoryMeta

Children

- none

#### Attributes

- **repositoryID:** requiredA unique identifier assigned to a repository. It is referenced by the *object* element to signify the repository that assigned its identifier.
- **url:** requiredA url to the external METS metadata file for the repository.

```
<repositoryMeta>

  <repository repositoryID="123456789"
    url="/metadata/handle/1234/4/mets.xml" />

</repositoryMeta>
```

## repositoryMeta

### Metadata Element

The *repositoryMeta* element contains metadata references about the repositories used in the used or referenced in the document. It can contain any number of *repository* elements.

See the *repository* tag entry for more information on the structure of *repository* elements.

#### Parent

- Meta

#### Children

- repository (any)

#### Attributes

- None

```
<meta>

  <userMeta> ... </userMeta>

  <pageMeta> ... </pageMeta>

  <repositoryMeta>

    <repository repositoryIID="..." url="..."
      />

  </repositoryMeta>

</meta>
```

## row

### Structural Element

The row element is contained inside a *table* and serves as a container of *cell* elements. A required *role* attribute determines how the row and its cells are rendered.

#### Parent

- table

#### Children

- cell (any)

#### Attributes

- **id:** (optional) A unique identifier of the element

- **n:** (optional) A local identifier used to differentiate the element from its siblings
- **rend:** (optional) A rendering hint used to override the default display of the element.
- **role:** (required) Indicates what kind of information the row carries. Possible values include "header" and "data".

```
<table n="table-example" id="XMLExample.table.table-example" rows="2"
      cols="3">

  <row
    role="head">

    <cell cols="2">Data Label One and
      Two</cell>

    <cell>Data Label Three</cell>

    ...

  </row> <row>

    <cell> Value One </cell>

    <cell> Value Two </cell>

    <cell> Value Three </cell>

    ...

  </row>

  ...

</table>
```

## table

### Structural Element

The *table* element is a container for information presented in tabular format. It consists of a set of *row* elements and an optional *header*.

### Parent

- div

### Children

- head (zero or one)
- row (any)

### Attributes

- **cols:** (required) The number of columns in the table.
- **id:** (required) A unique identifier of the element
- **n:** (required) A local identifier used to differentiate the element from its siblings
- **rend:** (optional) A rendering hint used to override the default display of the element.
- **rows:** (required) The number of rows in the table.

```
<div n="division-example"
      id="XMLExample.div.division-example">

  <table n="table1" id="XMLExample.table.table1" rows="2"
        cols="3">

    <row role="head">

      <cell cols="2">Data Label One and
        Two</cell>

      <cell>Data Label Three</cell>
```



```

        ...

</row>

<row>

    <cell> Value One </cell>

    <cell> Value Two </cell>

    <cell> Value Three </cell>

    ...

</row>

...

</table>
...
</div>

```

## trail

Text Container

Metadata Element

The *trail* element carries information about the user's current location in the system relative of the repository's root page. Each instance of the element serves as one link in the path from the root to the current page.

Parent

- pageMeta

Children

- none

Attributes

- **rend:** (optional) A rendering hint used to override the default display of the element.
- **target:** (optional) An optional attribute to specify a target URL for a trail element serving as a hyperlink. The text inside the element will be used as the text of the link.

```

<pageMeta>

    <metadata element="title">Example DRI
        page</metadata>

    <metadata
        element="contextPath">/xmlui</metadata>

    <metadata ...> ... </metadata>

    ...

    <trail target="/myDSpace"> A bread crumb item pointing to a
        page. </trail> <trail ...> ... </trail>

    ...

</pageMeta>

```

## userMeta

Metadata Element

The *userMeta* element contains metadata associated with the user that requested the document. It contains generic *metadata* elements, which in turn carry the information. Required and suggested values for *metadata* elements contained in *userMeta* include but not limited to:

- identifier (suggested): A unique identifier associated with the user.
- identifier.email (suggested): The requesting user's email address.
- identifier.firstName (suggested): The requesting user's first name.
- identifier.lastName (suggested): The requesting user's last name.
- identifier.logoutURL (suggested): The URL that a user will be taken to when logging out.
- identifier.url (suggested): A url reference to the user's page within the repository.
- language.RFC3066 (suggested): The requesting user's preferred language selection code as describe by RFC3066
- rights.accessRights (required): Determines the scope of actions that a user can perform in the system. Accepted values are:
  - none: The user is either not authenticated or does not have a valid account on the system
  - user: The user is authenticated and has a valid account on the system
  - admin: The user is authenticated and belongs to the system's administrative group

See the *metadata* tag entry for more information on the structure of *metadata* elements.

Parent

- meta

Children

- metadata (any)

Attributes

- **authenticated**: (required) Accepted values are "yes", "no". Determines whether the user has been authenticated by the system.

```
<meta>

  <userMeta>

    <metadata element="identifier" qualifier="email">bobJones@tamu.edu</metadata>

    <metadata element="identifier" qualifier="firstName">Bob</metadata>

    <metadata element="identifier" qualifier="lastName">Jones</metadata>

    <metadata element="rights" qualifier="accessRights">user</metadata>

    <metadata ...> ... </metadata>

    ...

    <trail source="123456789/6">A bread crumb item</trail>

    <trail ...> ... </trail>

    ...

  </userMeta>

  <pageMeta> ... </pageMeta>

</meta>
```

## value

Rich Text Container

Structural Element

The value element contains the value associated with a form field and can serve a different purpose for various field types. The value element is comprised of two subelements: the raw element which stores the unprocessed value directly from the user or other source, and the interpreted element which stores the value in a format appropriate for display to the user, possibly including rich text markup.

Parent

- field

Children

- `hi` (any)
- `xref` (any)
- `figure` (any)

#### Attributes

- **optionSelected**: (optional) An optional attribute for select, checkbox, and radio fields to determine if the value is to be selected or not.
- **optionValue**: (optional) An optional attribute for select, checkbox, and radio fields to determine the value that should be returned when this value is selected.
- **type**: (required) A required attribute to specify the type of value. Accepted types are:
  - **raw**: The raw type stores the unprocessed value directly from the user or other source.
  - **interpreted**: The interpreted type stores the value in a format appropriate for display to the user, possibly including rich text markup.
  - **default**: The default type stores a value supplied by the system, used when no other values are provided.

```
<p>
  <hi> ... </hi>
  <xref> ... </xref>
  <figure> ... </figure>
  <field id="XMLExample.field.name" n="name" type="text"
    required="yes">
    <params size="16" maxlength="32"/>
    <help>Some help text with <i18n>localized
      content</i18n>.</help>
    <value type="default">Author,
      John</value>
  </field>
</p>
```

## xref

#### Text Container

#### Structural Element

The *xref* element is a reference to an external document. It can be mixed freely with text, and any text within the tag itself will be used as part of the link's visual body.

#### Parent

- `cell`
- `p`
- `item`
- `hi`

#### Children

- `none`

#### Attributes

- **target**: (required) A target for the reference, using either a URL or an id of an existing element as a destination for the *xref*.

```
<p>
  <xref target="/url/link/target">This text is shown as a link.</xref>
</p>
```