eContracts Version 1.0

XML [legalxml-econ-specification-1.0.xml]
HTML [legalxml-econ-specification-1.0.html]
PDF (A4) [legalxml-econ-specification-1.0.pdf]

Copyright © 2006 OASIS Open, Inc. All Rights Reserved.

Status

Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/legalxml-econtracts.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/legalxml-econtractsipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/legalxml-econtracts [http://www.oasis-open.org/committees/legalxml-econtracts].

Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

Copyright © OASIS Open 2006. All Rights Reserved.
This is the specification for the OASIS eContracts Schema developed by the OASIS LegalXML eContracts Technical Committee. The eContracts Schema is intended to describe the generic, hierarchical structure of a wide range of contract documents. The eContracts schema defined in this specification aims to facilitate the storage, maintenance and processing of natural language precedents for contract documents and contract terms that may be used to create contract documents for electronic and paper-based transactions. Use cases covered include negotiated business contracts, ticket contracts, standard form business and consumer contracts and click-through agreements.

Purpose and Scope

Mission of the eContracts TC

The eContracts TC's mission is to promote “the efficient creation, maintenance, management, exchange and publication of contract documents and contract terms”.

The TC did not seek to focus on any particular vertical segment of the contracts domain.

The TC considered a range of scenarios covering issues in many different contract domains. A summary of these scenarios and the TC's conclusions on the scope of its specification are set out in the appendix "Contract Scenarios Considered by the eContracts TC."

Scope

This is the specification for the OASIS eContracts Schema developed by the OASIS LegalXML eContracts Technical Committee. The eContracts Schema is intended to describe the generic, hierarchical structure of a wide range of contract documents.

The eContracts schema defined in this specification aims to facilitate the storage, maintenance and processing of natural language precedents for contract documents and contract terms that may be used to create contract documents in a range of identified contract domains. The eContracts schema provides a model that can be used by persons who maintain precedent or template documents that will be used in automated document assembly, document construction
and publishing systems to create contract documents. Thus, it is expected it will be used mainly in back-end, automated processing systems, rather than by lawyers and others involved in day to day contract preparation.

The TC expects that automated processing of eContracts documents will require that eContracts XML is transformed into common word processing or other formats for use in common word processing applications.

It is possible that persons drafting contracts may work in an XML editor with the eContracts Schema. However, the TC does not expect this practice to be widespread in the forseeable future. Lawyers and others involved in the day-to-day drafting of contract documents work with common word processing software using unstructured methodologies. This is not expect to change. Even as word processing applications become capable of using arbitrary XML schemas, the TC does not identify any impetus for lawyers and other persons involved in the drafting of contracts and other legal and business documents to change current practices and adopt structured authoring methodologies. Should that change occur, the eContracts Schema is intended to be highly suitable for that purpose.

The TC expects the initial eContracts schema will be a foundation for further developments by communities with interests in specific industry domains or contracting domains.

Law firms and other enterprises who often commonly contract documents often may create other document types such as advices, correspondence and litigation documents. Many of these documents are prepared in essentially the same way as contract documents, using the same automatio tools. All the principles applicable to a schema for contract documents apply to these other documents. The TC concluded that the eContracts schema will define common content objects that can be adopted by another standards body responsible for developing a schema applicable to those other legal documents.

The eContracts Schema is not intended to overlap with the functionality provided by existing standards for electronic commerce or electronic business transactions. The eContracts Schema is intended to describe natural language contract documents, something not provided by existing standards.

**Terminology**

The key words must, must not, required, shall, shall not, should, should not, recommended, may, and optional in this document are to be interpreted as described in [RFC 2119].

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>application programmer</td>
<td>The person or person(s) developing a system that examines or parses the XML conforming to this specification, or a version customized according to the recommendations herein. This would include the developer of style sheets developed in a style sheet language such as XSLT, whether they be used for producing a rendition or other XML [XSLT].</td>
</tr>
<tr>
<td>contract</td>
<td>An agreement between parties that is intended to be legally enforceable. A contract may be oral, partly oral and partly written or wholly recorded in writing. The terms of a contract may be contained in many contract documents.</td>
</tr>
<tr>
<td>contract document</td>
<td>A document that records some or all of the draft or agreed contract terms. Contract terms are traditionally expressed in a natural language but it is assumed that some or all of the terms of a contract could be expressed in a deontic contract language (q.v.).</td>
</tr>
<tr>
<td>deontic contract language</td>
<td>Means a language that can express the rights and obligations of parties to a contract in a form that can be parsed by software applications and processed with other data to determine state information about matters governed by the contract.</td>
</tr>
</tbody>
</table>
Embedded data value

This refers to a piece of information such as a product or service description, date, name, address, quantity or monetary amount that is embedded in the natural language expression of the contract terms.

Machine readable information

This is information in the contract document that refers to information about contract rights, obligations, or states, that can be extracted from the document by a computer system. It includes information represented in deontic contract language, contract metadata and embedded data values. It does not refer to the computer readable characters in the text unless the meaning of that text can be determined by a computer system. For example, a monetary amount that can be read from the text is not machine readable information unless the system can determine useful information about the statement of that amount in the contract such as who must pay it, to whom it must be paid, and at what time is it to be paid for what purpose is it paid.

Natural language

This includes the mode of expression of contract narrative as it is commonly written by lawyers.

Precedent contract

This is a document that is used by the drafter of a new contract document as a starting point or template to assist in creating that new contract.

Rendition

The output of a transformation or styling process by which XML documents conforming to a particular schema are rendered with human-readable layout in a particular format such as RTF, PDF, HTML or displayed by a computer using a particular kind of software.

Schema customizer

The individual providing for changes, particularly additional elements or attributes to meet the needs of a set of users, e.g., a particular vertical market. As recommended in Customization, this would probably be done in the file eContracts-Reference.rnc or an equivalent file.

TC

This refers, in this document, to the Organization for the Advancement of Structured Information Standards Legal XML member Section eContracts Technical Committee.

Transaction

An instance of doing business, whether electronic or conventional.

Background and Scope

Empty Section

Overview of the eContracts Schema

Generic structural markup model

The TC considered various approaches to development of a schema for natural language contract documents, including:

a. presentation based schema such as the Microsoft Office Open XML format and OASIS OpenDocument;

b. simple web presentation schema such as XHTML 1.0;

c. generic structural markup schema such as DocBook, DITA, TEI, XHTML 2.0 and others.
The eContracts schema is a generic structural schema. It is designed to model the patterns found in a wide range of contract documents. It aims to provide maximum flexibility for long term data storage, content reuse and automation.

The eContracts Core Schema elements describe the components common to most contracts and their hierarchical relationship. The eContracts schema does not provide a vocabulary to describe the subject matter of specific contracts. Information about the subject matter of a specific contract can be provided in metadata or other markup defined by particular users.

**Reliance on existing XML standards**

The eContracts Schema relies exclusively on existing XML standards. It does not seek to define new processing models that can only be implemented if application vendors choose to implement relevant features.

Support for the simple conditional text processing model defined by the eContracts Schema can be implemented in user level applications, if desired.

**Normative schema syntax**

The eContracts Reference Schema is provided in Relax NG compact syntax, XML Schema (XSD) and as a DTD. The Relax NG compact syntax version is normative.

The Relax NG compact syntax permits a high level of flexibility for customizing the schema and provides a level of human readability similar to that provided by DTDs.

The eContracts Core Schema uses features that cannot be represented in DTD syntax. For example, the `item` element is used both recursively and as a child of the `block` element. When used as a child of `block`, `item` is re-defined to prevent it from having a child `item` element. This cannot be enforced in the DTD version.

**Name Spaces**

The eContracts Schema uses the following name spaces. A suggested prefix is also provided even though, of course, the use of any schema file may select the prefix they wish:

- http://purl.org/elements/1.1 - for the Dublin Core Meta Data set (dc)
- http://www.w3.org/2001/XMLSchema (xs)
- http://relaxng.org/ns/compatability/annotations/1.0 (a)
- http://www.w3.org/2001/XMLSchema-datayptes
- This namespace: urn:oasis:names:tc:eContracts:1:0 (ec)

**eContracts Schema files**

The eContracts Reference Schema is currently packaged as these four files:

1. eContracts-Reference.rnc

   This file defines the eContracts Reference Schema. It incorporates the other files in the schema package and sets various values for eContracts Reference Schema. These are:
a. It activates conditional text and defines the elements used for conditional text in the schema. This permits users to easily substitute their own conditional text models.

b. It activates XInclude for content reuse.

c. It activates additional features that are required to meet the Web Content Accessibility Guidelines.

References in this specification to "eContracts Reference" are to the schema defined in this file.

2. eContracts-core.rnc

This file defines all the elements and attributes in the eContracts name space, including the contract element. These definitions can be overridden in the eContracts-Reference.rnc file (or an equivalent file) to create a user customization.

References in this specification to "eContracts Core Schema" are to this file. References to "eContracts Schema" are to any schema that incorporates this file, with or without customization, provided that the customization complies with the naming conventions in Naming conventions for eContracts Schema customizations.

3. dc-metadata.rnc

This file incorporates some basic elements from Dublin Core in the metadata element at the beginning of the contract.

4. xi-include.rnc

This defines the xi:include element from World Wide Web Consortium XML Inclusions recommendation [http://www.w3.org/TR/2004/PR-xinclude-2004930].

Data exchange and normative use of the eContracts Reference Schema

The eContracts Reference Schema is intended to be the foundation upon which organization or application specific schema are built. It aims to provide the minimum definition that is likely to be common to the widest range of contract documents.

The eContracts Reference Schema is intended mainly for use with precedents in backend processing. For this reason and because many contracts will be prepared using word processing software, the TC has not identified any common business requirements for users to exchange eContracts XML between user organizations. Thus the eContracts Reference Schema is not a normative standard for the exchange of contract documents in XML. There is no requirement that parties must use eContracts Reference to exchange data or that it be capable of validating any user customization.

The eContracts Reference Schema does aim to provide the maximum level of interoperability between users that is consistent with their need to customize the eContracts Schema to their specific requirements. This is achieved in two ways:

1. Most attributes are defined as xsd:string (CDATA) in the eContracts Core Schema so that any value provided in a user customization is valid.

2. Loose content models are defined as default values in various contexts in the eContracts Core Schema so that tighter content models defined in user customizations will be valid.

The eContracts Schema can be customized and these values overridden in a user customization.

The TC expects that user communities with common interests in exchanging contract documents in XML will develop their own normative standard based on the eContracts Core Schema. The eContracts Core Schema is expected to be
the foundation for a large family of related schema that shares common core patterns and can benefit from the use of common tools.

The heart of the eContracts schema is eContracts-core.rnc. To maximize the level of interoperability between eContracts XML and processing tools, the core patterns from eContracts-core.rnc described in section "Basic building blocks in the eContracts Core Schema" are normative. As described in Basic building blocks in the eContracts Core Schema, if those patterns are changed, the resulting schema must not use a name that designates it as part of the eContracts schema family.

**Feature summary for the eContracts Schema**

Features of the eContracts Reference Schema can be summarized as follows:

1. Contract documents are composed of paragraphs and clauses that may be stored separately and reused in multiple documents. The eContracts schema defines these objects as containers that can be processed as distinct objects or content chunks for storage and retrieval in document assembly and other processing systems. The eContracts schema uses the XInclude standard to support content sharing and reuse of clauses using the item element.

2. The TC has aimed for simplicity with the eContracts Core Schema. It defines only 51 elements. Most content can be created with just a handful of elements: item, title, block and text with the item element used recursively. This is intended to make it easy to convert existing content to the eContracts schema and permit content components to be inserted without re-tagging at any desired level of the document hierarchy in document assembly applications.

3. The eContracts Core Schema defines the generic, hierarchical structure of contract documents. This provides the maximum flexibility for content reuse, reliable automated processing and transformation of eContracts XML into other formats.

4. The eContracts Core Schema provides for reuse of item elements using the XInclude specification that is supported by available processing tools.

5. The eContracts Core Schema provides for embedded data values to support variables substitution in contract preparation and the extraction of data values from XML contract documents. This is achieved through the field element. This version of the eContracts schema supports only the string type for the field element. It is anticipated that future schema versions may require richer data types, according to specific business requirements.

6. The eContracts Reference Schema provides a mechanism to support conditional processing of content at the element level and within text contexts. This model is not part of the eContracts Core Schema to allow users to adopt their own conditional text processing model.

7. The eContracts Core Schema provides a model for users to add metadata at the contract and clause level. The schema makes provision for common metadata fields required by document management, document assembly and publishing applications such as:
   
i. document identifiers, the author, version and dates;
   
ii. the legal subject matter or categorisation of distinct content objects.

8. The schema provides sufficient definition of content objects in contract documents that user applications can:
   
i. define and apply automatic numbering schemes to those objects;
   
ii. generate desired renditions, including but not limited to print, RTF, PDF, HTML and text to speech ready formats.
9. The eContracts schema allows for an entry of various kinds of values into instances of contract documents based on that schema, e.g. dates specifying start and end dates of the contracts into their respective slots. This is achieved through the field of the Schema, which can be regarded as a slot for entry of specific values into contract instances, either by the user or by a back-end system. This version of schema supports only string type for the field element and it is anticipated that future schema versions would require richer data types, according to specific business requirements.

Benefits of the eContracts Schema

The eContracts Core Schema is expected to support the widest possible range of uses in back-end contract document processing systems. It provides a standards based schema to facilitate the long term storage and maintenance of precedent contract documents and terms by law firms and other enterprises who use contract precedents to prepare contract documents for specific transactions. It will enable these users to reduce maintenance costs, provide better access to information to contract drafters and provide more reliable automation of document assembly and publishing processes. It should promote the wider availability of automated document creation systems. In particular:

a. It will enable large volumes of contract precedents to be managed without concern about changes to proprietary file formats. It will avoid the costs associated with the periodical reformatting of proprietary data with embedded formatting information.

b. It will allow off-the-shelf XML based content management and processing tools to be used for the creation, maintenance and retrieval of contract precedents. It will enable enhanced levels of automation in the assembly of contracts from precedents and from stored transaction data.

c. It will enable contract precedents to be transformed into any desired rendition such as HTML, RTF, Microsoft Office Open XML format, OpenDocument or any other format.

d. It will enable organizations to enhance precedents with metadata to facilitate improved information retrieval.

e. If supported by vendors of document assembly and other automation systems, it will minimize dependencies between contract precedents and processing systems to reduce setup and switching costs.

Basic building blocks in the eContracts Core Schema

The main document hierarchy

In contracts, the basic unit of content is often called a clause or section. Usually it has a number, a title and a block or paragraph of text.

In the eContracts Core Schema, the item is the basic building block of the document hierarchy. It is a recursive element and represents structures that may be known in contracts as "chapters," "parts," "sections," "clauses" and "subclauses." As explained in the next section, the item element is also used to represent items in a list.

The TC used the term "item" to avoid terms that may have a particular meaning to some users and not others. It can be given a citation name in automatic numbering based on its context and prevailing conventions.

It is not uncommon for contract documents to contain structures similar to the following example, often in the same document:

Example 1. Numbered document hierarchy with and without titles

1 First level
1.1 Second level
1.1.1 Third level
The eContracts schema treats each of the structures at the third level as structurally the same regardless of the presence of a title. For this reason, the main container element in the eContracts schema (item) has an optional title.

This example would be marked up as follows:

**Example 2. Markup of numbered hierarchy**

```xml
<item number="1"><title><text>First level</text></title>
<item number="1.1"><title><text>Second level</text></title>
<item number="1.1.1"><title><text>Third level</text></title>
    <block><text>Content under third level with title.</text></block>
</item>
<item number="1.2"><title><text>Second level</text></title>
    <item number="1.2.1"><block><text>Content under third level without title.</text></block>
    <item number="1.2.2"><block><text>Content under third level without title.</text></block>
</item>
</item>
```

**Use of item for clauses and lists**

In the eContracts schema, lists are created by enclosing the item element in a block. While extreme cases are obvious, it is frequently difficult to distinguish between a list and a clause or subclause in contracts. Often it is based on the numbering style the writer wishes to apply. In the previous example, some may regard the items numbered 1.2.1 and 1.2.2 as list items, others as clauses and others as subclauses. The patterns for a clause and a list item are almost identical. The TC took the view that these distinctions are often a matter of author preference and that the use of distinct elements for clauses and list items only makes it difficult to reuse content in another document or context.

In the eContracts TC, clauses, subclauses and list items all use the item element. The nature of the structure is inferred from its hierarchical location. Thus, an item enclosed by a block is a list item and should be numbered as such.

The following example shows the use of item for clause and list markup:

**Example 3. Example showing item as clause and as list items**

```xml
<item number="1"><title><text>First level</text></title>
<item number="1.1"><title><text>Second level</text></title>
<item number="1.1.1"><title><text>Third level</text></title>
```
The eContracts schema does not use a separate list container. Properties of the list, such as the number-type are controlled by an attribute on the containing block.

The concepts of ordered lists and unordered lists have not been adopted in the eContracts schema. All lists are represented in the same way. The use of bullets or sequential numbering for lists is governed by the number-type attribute on the containing block.

**Paragraph markup with block and text**

Grammatical or structural paragraphs in contract documents are represented with the eContracts block element. The block is intended to encapsulate all content that is semantically part of the paragraph, such as lists, tables, graphics etc.

The eContracts Core Schema avoids mixed content within the block. Thus, it never contains character data directly. Character data for the paragraph is contained in the text element. The text element ensures control over character data that occurs after or between other elements within the block.

For example, contracts frequently have complex list structures with character data between lists in a parent block. The text element makes it easier to reliably represent and process these structures. In other cases, a formula may be followed by the word “where:” to introduce the definitions of formula components. The text element provides a means to control whether this continues in line rather than the default position of starting a new line for text objects.

The text element may be repeated to create new lines within a block. The text element should never be used as a paragraph element.

**Examples, explanatory notes, quotations etc.**

Contracts may include examples, explanatory notes, quotations and other similar content that may occur at almost any point in the document hierarchy. The eContracts schema uses a single element called inclusion to encapsulate all such objects. The nature of the particular object is captured in the class attribute on the inclusion.

The key points about these objects is that they all may include paragraphs (block elements) or clauses (item element). The inclusion acts as a shield element to permit the content of these objects to be processed separately from the containing document hierarchy. Thus, these objects may have their own automatic numbering sequences and formatting in published renditions.
The use of a single element for all such objects simplifies the content models and core patterns in the eContracts Core Schema and makes it easier for users to customize the schema by adding new values to the class attribute.

The inclusion is also used to encapsulate graphics with titles and can be used to manage groups of objects such as tables.

The inclusion element occurs almost anywhere in the document hierarchy. It is not intended to be used as an alternative to the item to create narrative content. It is intended only to encapsulate distinct objects that require separate formatting or processing from the main content.

The inclusion element should not be confused with the xi:include element that performs a completely different function.

**Core patterns for item, block and inclusion**

Commonly, the body part of a contract will consist of numbered items. In some cases, these items may be preceded by one or more blocks representing introductory paragraphs. In content that has a poor hierarchical structure created with word processing software, authors may introduce blocks between items or after the last item. Inconsistent hierarchical structures present obvious problems for readers of contracts. At a technical level they can make it difficult to create accurate contents listings, chunk content for web publishing and control automatic numbering of items.

Often it is necessary to incorporate content created by others into an inclusion or attachment.

The eContracts Core Schema permits users to control the strictness of item structures in any part of the document. It defines three basic patterns:

1. A tight structure model permits either block or item but does not allow both at the same hierarchical level. It is represented as follows:

   \[
   \text{tight.structure.model} = \text{inclusion*},
   \left((\text{block}, \text{inclusion*})^+ \mid (\text{item*}, \text{inclusion*})\right)?
   \]

2. A standard structure model permits block elements before the first item but not otherwise at the same hierarchical level. It is represented as follows:

   \[
   \text{standard.structure.model} = \text{inclusion*}, (\text{block}, \text{inclusion*})^*, (\text{item*}, \text{inclusion*})
   \]

3. A loose structure model permits block and item to be mixed in any order at the same hierarchical level. It is represented as follows:

   \[
   \text{loose.structure.model} = (\text{block} \mid \text{inclusion} \mid \text{item})^*
   \]

Please refer to the Language Reference for complete content models.

These models can be applied selectively within many of the major containers in the eContracts Core Schema, including body, back, attachment, inclusion and item. The default value in eContracts Reference is the loose structure model in all contexts. This ensures that eContracts Reference will validate any customization using tight or standard models. It is not intended as a recommendation that users adopt the loose model in all contexts in their own applications.

These patterns enable users to create their own customizations with new containers and incorporate the standard patterns from eContracts Core Schema.
Contract Document Structure in the eContracts Schema

Components of contract

contract is the root tag for contracts. A contract MAY contain the following, each of which is discussed in more detail below:

1. metadata
2. title and subtitle - the contract writer MUST include a title.
3. contract-front
4. body - this MUST appear
5. back
6. attachments

The following example shows a skeleton of a contract to illustrate these parts. To illustrate the overall structure, many of the elements are empty that would normally contain text:

Example 4. A Simple Contract

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0">
  <title><text>Sample of the five elements at level 1 </text></title>
  <subtitle>This file is to test : - title - sub-title - contract-front - date-block - parties - party - body - back - attachments - attachment</subtitle>
  <contract-front>
    <date-block>
      <date><em>Long time ago</em></date>
    </date-block>
    <parties>
      <party></party>
    </parties>
  </contract-front>
  <body></body>
  <back></back>
  <attachments>
    <attachment> </attachment>
  </attachments>
</contract>
An absolute minimal contract showing that a contract MAY have only a title and an empty body:

**Example 5. A Minimal Contract**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xi="http://www.w3.org/2001/XInclude">
  <title>Minimum XML sample</title>
</contract>
```

The following exemplifies the important features in the XML produced for a contract as per this specification. On line 12, observe the The contract-front containing the dates (date-block) and a list of the parties parties (line 17) to the contract. Before this is metadata (line 6), which contains non-printing information about the contract document.

The body shows how text is prepared using the standard model. One could have customized the schema for the tight or loose model. The body has several blocks followed by zero or more items. A block contains a text which acts as a "text container." In addition to the words of the contract, it contains elements to emphasize or format text (such as em, strike, sup), as well as marked up text for name, address, party). After discussing these, the specification discusses condition elements and condition elements that are used for providing alternative text for various paragraphs. Text containers can also contain elements to include text and the field to bring in text from an application or database.

The back at the end of the example has markup illustrates the party-signature element. This supports many options, corresponding to the many way those signing documents and their witnesses.

**Example 6. A Minimal Contract**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xi="http://www.w3.org/2001/XInclude">
  <metadata>
    <dc:title>WIU Sample Contract</dc:title>
    <dc:creator>Julie Sasa</dc:creator>
    <dc:title>WIU Sample Contract</dc:title>
  </metadata>
  <title><text>Overall Example</text></title>
  <contract-front>
    <date-block>Agreement dated:
      <field class="date" type="blank" name="contract_date" length="75mm"/>
    </date-block>
    <parties><title>Parties</title>
      <party><person-record><name>ABC Ventures Limited</name> having
        its office at <address>100 Main Street, Sydney, NSW 2000</address>
      </person-record></party>
    </parties>
  </contract-front>
</contract>
```
<person-record>, hereafter referred to as "General Partner"
</party>
<party><person-record><name>John A. Doe</name> of <address>10 Ramrod Drive, Sydney, NSW 2000</address></person-record>
and <person-record><name>John W. Smith</name> of <address>25 Pine Road, Plainsville, NSW, 0000</address></person-record>, herafter collectively referred to as "Limited Partners"
</party>

<body>
<br />
<br />
<text>A<sub>10</sub><em>Emphasized Text</em><strike>Struck out</strike></text>
</body>
<back>
<party-signature>
<signatory-group>
<br />
<signatory-record>
<signatory id="T0001" xml:lang="ja">
<signature-line id="I0001" xml:lang="en_US">
<text>Mr. Signatory Jr.</text>
</signature-line>
</signatory>
</signatory-record>
</signatory-group>
<br />
<br />
<br />
<br />
<br />
<br />
<br />
<br />
<br />
</party-signature>
</back>
<attachments>
The attachment element is provided for appendices and other exhibits to the contract, e.g. blue prints in a construction contract or a detailed specification in a software development project. They contain block and item like the body, they, by default, use the loose model.

### Particular features of the eContracts schema

#### About this section

This section explains particular features of the eContracts schema that are not easily gleaned from the Language Reference. Refer to the Language Reference for the specification for each element of the schema.

#### Managing metadata

The eContracts Core Schema provides a metadata element for the contract and item elements. The content of this element is not defined in the eContracts Core Schema. Users of the eContracts Schema are free to define metadata to meet their specific needs.

However, eContracts Standard incorporates basic elements from Dublin Core from the name space http://purl.org/dc/elements/1.1.

#### Numbering of structural provisions such as item and attachment

In contracts, it is common that structural provisions such as items and attachments are numbered. The eContracts Core Schema provides the `number` attribute on the elements `item`, `attachment`, and `inclusion` to support numbering.

The eContracts Core Schema makes no assumption about the way that numbers for those provisions will be generated. Numbering must be handled in each application.

The eContracts Core Schema provides attribute placeholders on the root element and other containers so that users can add attributes to control numbering if it is implemented in the XML data.

#### List item numbering

Within a block, an item will be considered to be a list item. The block provides the `number-type` which is used to control the numbering or otherwise of list items. While expected values are defined for the numbering, the schema does not enforce this behavior in the application. It is up to user applications to enforce the specified numbering behavior. The `number-type` attribute defines the following values:

- **manual**
  
  The contract writer may enter a value for the `number` attribute on each of the items directly within this block. A application programmer MUST not alter the values found in the data.

- **none**
  
  A application programmer MUST not enter a value in the `number` attribute for the item.
The application programmer SHOULD render list items by a disc or bullet.

The application programmer SHOULD render list items by a line or dash.

The application programmer SHOULD render list items with preceding numerals ('1', '2', '3', ...)

The application programmer SHOULD render list items with preceding lower case alphabetical characters ('a', 'b', 'c', ..., 'z', 'aa', 'ab', 'ac', 'ad', 'za', 'zb', 'zc', 'za', 'zb', 'zc', 'zz')

The application programmer SHOULD render list items with preceding upper case alphabetical characters ('A', 'B', 'C', ..., 'Z', 'AA', 'AB', 'AC', 'AD', 'ZA', 'ZB', 'ZZ', 'ZZZ')

The application programmer SHOULD render list items with preceding lower case roman numerals ('I', 'ii', 'iii', 'iv', ...)

The application programmer SHOULD render upper case roman numerals ('I', 'II', 'III', 'IV', ...)

The user application MAY include formatting characters such as parenthesis around list item numbers in the attribute values or it may provide those characters during rendering.

Conditional text processing

Often, the creator of a form contract needs to provide for certain text to be included in certain circumstances. For example, certain riders in an insurance policy may be included when the customer pays for them or each jurisdiction might happen to require only certain text or disclaimers.

Conditional processing is very common in document assembly applications. However, there is no standard way for these applications to support the conditional markup or the processing logic. The TC decided it could not attempt to define a standard representation of conditional text processing in contract documents at this stage.

The eContracts Core Schema defines a simple condition attribute for use on container elements and a conditional element for use with inline elements. However, it does not activate these values in the schema. Activation is provided in eContracts Reference. Thus, users can elect to use or not use the conditional processing model provided in the eContracts Core Schema.

Condition values may be specified in the contract metadata using the conditions element. However, user applications may locate these values in any file or database that is accessible to processing applications.

If the eContracts Schema is widely adopted, it may be highly advantageous to define a more powerful model to support conditional processing functionality provided by document assembly applications.

The following example shows a simple use of conditions to control the rendering of items according to the jurisdiction that applies to the contract. An element may have multiple values for its condition attribute. If so, it will be rendered if any one of these values is true. In this example, the jurisdiction is set to jurisdiction-US. content for other jurisdictions will not be rendered.

Example 7. Showing the condition elements

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xil="http://www.w3.org/2001/XInclude">
  <metadata>
    <conditions>
```
Content reuse with XInclude

The eContracts Core Schema permits content sharing and reuse using the \texttt{xi:include} element. This is for the item element only. Implementation of the XInclude functionality is provided in the eContracts Reference Schema.

XInclude permits a user to link to any element as the target, regardless of whether it is valid at the designated location. If an invalid element is target, this will only become apparent when the document is validated after the \texttt{xi:include} links are resolved.

Party signatures

Scope of the eContracts Schema

Many contracts that are to be physically signed by the parties using a pen and ink signature or seal require elaborate structures to record the names of the party, the persons signing and to provide places for actual signatures.

The eContracts Core Schema makes provision for physical signatures on printed documents. It does not provide any specific mechanism for digital signatures to be represented in the XML markup of a contract document. The TC has not identified any business requirements for it to do so at this time. Parties can apply digital signatures for electronic transactions by applying those signatures to the electronic file that is used as the authoritative record of the transaction. Accordingly, the issue of digital signatures was considered outside the scope of the TC’s mission.

Written signatures

The eContracts Core schema provides two ways for the contract drafters to create signature information in contract documents:

1. The \texttt{party-signature} element provides a semantic markup of party signature information that can record the name of the party, persons signing and witnesses. This markup permits signature information to be laid out in horizontal or vertical alignments in renditions from the XML.

2. Signature information can be laid out in a table, if desired. The \texttt{signature-line} element is permitted in the table entry.

Users may choose either approach according to their convenience and requirement for semantic markup.
The use of the party-signature markup is described in detail in the Language Reference.

**Invocation of the eContracts Reference Schema**

**Invocation for the Relax NG schema**

Users who propose to use tools with Relax NG support will need to look at the documentation for their specific tool.

**Invocation for the XML Schema**

The XML Schema is invoked as follows:

```xml
<?xml version="1.0"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:eContracts:1:0:eContracts-Reference.xsd"
xmlns:dc="http://purl.org/dc/elements/1.1"
xmlns:xi="http://www.w3.org/2001/XInclude"
>

Note

eContracts-Reference.xsd can be a full or relative path to the schema file.

**Invocation for the DTD**

The DTD is invoked as follows:

```xml
<?xml version="1.0"?>
<!DOCTYPE contract SYSTEM
eContracts-Reference.dtd>

Note

eContracts-Reference.dtd can be a full or relative path to the DTD file.

**Customizing the eContracts Core Schema**

**Customization Introduction**

This section explains the mechanics of adding a customization layer to the eContracts Core Schema using the Relax NG files. It also provides naming conventions for user customizations to avoid confusion about what is and what is not part of the eContracts Schema family.

The TC anticipates that users will develop customizations [MIN0216]. Some of these customizations will be for vertical areas such as real estate where the PRIA and MISMA standards are in use. [MIN0119]. UBL provides ‘a set of business elements’ and one might wish to combine the standard described in this specification to provide narrative terms and use UBL, or similar standards, to document ‘business terms.’ The TC has chosen not to standardize how this might be done at this time.
The eContracts Reference is a convenient template for user customizations.

**Classes of customizations for the eContracts Schema**

There are two classes of modification to the eContracts Schema:

1. **eContracts Subset**
   
   This is a customization of eContracts Core using the `eContracts-Reference.rnc` file or an equivalent file. Contract data under a subset customization must be validated against the eContracts Reference Schema. This permits a subset customization to:
   
   a. add attribute value enumerations for `xsd:string` values and
   
   b. switch between the loose, standard and tight content models.

2. **eContracts Variant**

   This is a customization of eContracts Core that cannot be validated against the eContracts Reference Schema. There are no restrictions on this type of customization, except that the integrity of the core patterns for item and block, including the loose, standard and tight content models found in `eContracts-core.rnc` must be retained if the schema is to retain designation as part of the eContracts Schema family. This class of customization may add new attributes to existing elements, add new elements and new document types.

**Naming conventions for eContracts Schema customizations**

For an eContracts Subset customization, it is strongly recommended that the name be in the form:

`eContracts-s-application-name.rnc`

For an eContracts Variant customization, the schema customizer should use a name be in the form:

`eContracts-application-name.rnc`

If a customization is not an eContracts Subset or an eContracts Variant, it MUST not use "eContracts" as part of its name.

**Changing content models in eContracts-core.rnc**

`eContracts-core.rnc` should never be modified directly. All customizations must be undertaken in a customization layer similar to `eContracts-Reference.rnc`.

**Customizing for the loose, standard, and tight models.**

This section explains, how to customize the schema, to configure for loose, standard and tight models. Each of these is done by changing the appropriate grammar element in the main schema file (similar to `eContracts-Reference.rnc`):

**Table 1. Container Model Customization Table**

<table>
<thead>
<tr>
<th>element</th>
<th>Model to change</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>body.structure.model</td>
</tr>
<tr>
<td>back</td>
<td>back.structure.model</td>
</tr>
</tbody>
</table>
Each of these elements can use either the `tight.structure.model`, `standard.structure.model`, or `loose.structure.model`. The following example sets all five text containers to use the tight structure model:

<table>
<thead>
<tr>
<th>Element</th>
<th>Structure Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td><code>tight.structure.model</code></td>
</tr>
<tr>
<td>back</td>
<td><code>tight.structure.model</code></td>
</tr>
<tr>
<td>attachment</td>
<td><code>tight.structure.model</code></td>
</tr>
<tr>
<td>item</td>
<td><code>tight.structure.model</code></td>
</tr>
<tr>
<td>inclusion</td>
<td><code>tight.structure.model</code></td>
</tr>
</tbody>
</table>

**Customization**

The [Relax], Section 9.2, shows how one can add to an attribute list or grammar. Customizations SHOULD be added to the `eContracts-Reference.rnc` or a renamed copy of that file. Then, add the lines needed to add the modifications you wish. This is illustrated below. (Some of the material from `eContracts-Reference.rnc` were removed to save space.)

**Example 8. Adding two attributes to the item element**

```xml
default namespace ec = "urn:oasis:names:tc:eContracts:1.0"
namespace xs = "http://www.w3.org/2001/XMLSchema"
namespace xi = "http://www.w3.org/2001/XInclude"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

grammar {

    # This section includes Dublin core metadata elements into the application.
    #
    include "dc-metadata.rnc"

    # INCLUDE eCONTRACTS CORE
    include "eContracts-core.rnc" {

        metadata.content &= dc.metadata.content

        # change the structure model patterns
        body.structure.model = tight.structure.model
        back.structure.model = tight.structure.model

    }

}```
The above shows how one can change to the tight model to add two attributes, \(a\) and \(b\) to the \texttt{item} element. These are added to \texttt{block.item.attlist.extensions} Note that \texttt{item} is defined in two different places in \texttt{eContracts-core.rnc} Here is a sample contract showing the use of the \(a\) and \(b\) attributes on \texttt{item} tags as well as including \texttt{PaySomeone} in the \texttt{text} tag:

\textbf{Example 9. How one uses the \(a\) and \(b\) attributes added above}

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title>
    <text>Data validate against Loose model based on Mr. Meyer's email on Aug20.</text>
  </title>

  <body>
    <block> </block>
    <block> </block>
    </block>
</contract>
```
If the schema customizer wishes to add a customization to the item that might appear directly within a block tag, they change block.item.attlist. On the other hand, if one wishes to provide a customization to an item that might appear anywhere else, one adds it to item.attlist.extensions. In examining the definition of item in eContracts-core.rnc, one sees that one can add new elements to item by changing item.structure.module. It would have "worked" if the schema customizer added attributes by changing common.attributes, item.class.attribute, conditional.attributes, stop-contents.attribute, item.numbering.attributes, and item.attlist.extensions. However, these have special meanings and most are used elsewhere. Thus, the schema customizer SHOULD NOT make this change there. As you can see, the schema often uses the convention, element-name.attlist.extensions to add attributes to element-name. As another example, attachment.attlist.extensions is where the schema customizer MAY put additional attributes that are to appear in the attachment element. Many elements also provide a customization opportunity of the form element-name.class.list. For example, the schema defines inclusion.class.attribute. Also, the following elements have places to enter numbering options:

Table 2. Container Model Customization Table

<table>
<thead>
<tr>
<th>element</th>
<th>Label to change</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>attachment.numbering.attributes</td>
</tr>
<tr>
<td>back</td>
<td>back.numbering.attributes</td>
</tr>
<tr>
<td>background</td>
<td>background.numbering.attributes</td>
</tr>
<tr>
<td>block</td>
<td>block.numbering.attributes</td>
</tr>
<tr>
<td>contract</td>
<td>contract.numbering.attributes</td>
</tr>
<tr>
<td>entry</td>
<td>entry.numbering.attributes</td>
</tr>
<tr>
<td>inclusion</td>
<td>inclusion.numbering.attributes</td>
</tr>
<tr>
<td>item</td>
<td>item.numbering.attributes</td>
</tr>
</tbody>
</table>

The following modification file is built upon a shortened version of eContracts-Reference.rnc:

Example 10. Adds I1 and I2 attribute to inclusion and adds E attribute to all elements

default namespace ec = "urn:oasis:names:tc:eContracts:1:0"
namespace xs = "http://www.w3.org/2001/XMLSchema"
namespace xi = "http://www.w3.org/2001/XInclude"
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
It does add the attributes I1 and I2 to the inclusion element, by updating the inclusion.class.attribute and inclusion.attlist.extensions, respectively. Also, observe that it adds the attribute E to all elements. This minimized, but still valid, contract shows the effect on the possible attributes:

**Example 11. Contract document using the above**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xi="http://www.w3.org/2001/XInclude">
<title><text>Modification Test</text></title>
<body E="3">
    <item E="5">
        <inclusion I1="9"/>
        <inclusion I2="10"/>
    </item>
</body>
</contract>
```

**Creating XSD and DTD files**

After customizing the RelaxNG files, it may be necessary to create either an XML Schema (XSD) or a DTD, depending on the requirements of the user application.
Trang is available from www.thaiopensource.com/relaxng/trang.html

Trang is capable of creating an XSD file for the eContracts Schema. However, it will not create a DTD file for the eContracts Schema because of the redefinition of the eContracts item element in the block context. The DTD file must be customized manually.

The command to translate the schema to XSD using Trang is:

```
```

Note

Each of trang.jar, eContracts-Standard.rnc, eContracts-Standard.xsd can be specified as a full or relative file path to the actual location of the file. Also, the above command must be typed as one physical line.

Language Reference

Common Attributes

These attributes occur on all elements. They are summarized here once for brevity and to make the attributes that occur on many elements stand out.

The id attribute allows an unique identifier to be added to any element in the contract. Note that this is defined as ID while references to it are not currently defined as an IDREF. This means that upon validation, there will be no error if a reference to an ID does not have a corresponding ID. This allows one to validate fragments of contract documents, particularly in clause libraries. The schema customizer MAY change this.

```
common.attributes =
  id.attributes

id.attributes =
  attribute id { xsd:ID }?

common.attributes &= attribute xml:lang { xsd:string }?
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>xsd:ID</td>
<td>none</td>
</tr>
<tr>
<td>xml:lang</td>
<td>xsd:string</td>
<td>none</td>
</tr>
</tbody>
</table>

The xml:lang attribute is added by eContracts.standard.rnc. This semantics of this attribute are defined by the XML Specification [xml]. Please refer to that specification.

Class Attributes

This is a generic class attribute and is applied to all elements. The intention is that this attribute is redefined for specific needs on an element-by-element basis. This attribute is given here once for brevity and to make the attributes that occur on many elements stand out.
standard.class = attribute class { xsd:string }?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>xsd:string</td>
<td>none</td>
</tr>
</tbody>
</table>

**condition.attribute**

This is to enable conditional text. See the conditions element for a description of this functionality. This attribute is given here once for brevity and to make the attributes that occur on many elements stand out.

condition.attribute = attribute condition { xsd:string }?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>xsd:string</td>
<td>none</td>
</tr>
</tbody>
</table>

**orient.attribute**

This specifies whether the element contents should be rendered as portrait or landscape:

orient.attribute = attribute orient {"portrait" | "landscape" }?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>orient</td>
<td>xsd:string</td>
</tr>
</tbody>
</table>

**common.number.attribute**

This allows the user to specify the number or other designator such as "a)" for item, attachment, inclusion and note.

common.number.attribute = attribute number { xsd:string }?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>xsd:string</td>
</tr>
</tbody>
</table>

**Stop-Contents Attribute**

This stops the generation of table-of-contents entries for the elements contained within this element.

stop-contents.attribute = attribute stop-contents { "below" }?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop-contents</td>
<td>xsd:string</td>
<td>below</td>
</tr>
</tbody>
</table>
address

Synopsis

Content Model

address has a mixed content model.

address = element address {
  (inline.content | field)*,
  address.attlist
}

address.attlist =
  common.attributes,
  address.class.attribute,
  address.name.attribute,
  address.attlist.extensions

address.class.attribute = standard.class
address.name.attribute = attribute name { xsd:string }?
address.attlist.extensions = empty

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional

address ::=  
  • Zero or More of
    • text data (#PCDATA)
    • conditional
    • em
    • field
    • reference
    • statutory-em
    • strike
    • sub
    • sup
Attributes
Common Attributes and Class Attributes

Description
This can contain an entire address (see example under party). Or it may contain part of an address, with each part having an appropriate category (see example below).

Attributes
Common Attributes and Class Attributes

Parents
These elements contain address: text, person-record

Children
The following elements occur inside address:
conditional, em, field, reference, statutory-em, strike, sub, sup

See Also
name, party and person-record

Examples

Example 12. Several address elements which together define all parts of an address

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of :
    - contract-front
    - parties
    - party
    - person-record
    - name
    - address
  </text></title>
  <contract-front>
    <parties>
      <party>
        <person-record>
          <name>John Smith</name>
          <address class="street">2-198 Lamoine Avenue</address>
          <address class="city">Macomb</address><address class="state">IL</address>
        </person-record>
      </party>
    </parties>
  </contract-front>
</contract>
```
attachment

Synopsis

Content Model

attachment = element attachment {
    metadata?, title?, subtitle*,
    (attachment.doctypes | attachment.structure.model)?,
    attachment.attlist
}

attachment.structure.model = loose.structure.model

attachment.attlist =
    common.attributes,
    common.number.attribute,
    attachment.class.attribute,
    orient.attribute,
    stop-contents.attribute,
    attachment.numbering.attributes,
    attachment.attlist.extensions

attachment.numbering.attributes = empty
attachment.class.attribute = standard.class
attachment.attlist.extensions = empty
attachment.doctypes = contract

## Tight structure model

tight.structure.model = inclusion*,
    ((block, inclusion*)+ | (item.reuse.model+, inclusion*))?

## Standard structure model

standard.structure.model = inclusion*, (block, inclusion*)*,
    (item.reuse.model+, inclusion*)

## Loose structure model

loose.structure.model = (block | inclusion | item.reuse.model)*
item.reuse.model |= xi:Include

attachment ::= 
  • Sequence of
    • Zero or one metadata
    • Zero or one title
    • Zero or more subtitle
    • Zero or one of:
      • contract
        • If loose model is selected, sequence of:
          • Zero or more of:
            • block
            • inclusion
            • item
            • xi:include
        • If standard model is selected, sequence of:
          • Zero or more inclusion
          • Zero or more sequences of
            • One block
            • Zero or more inclusion
      • Sequence of
        • One or more choice of
          • item
          • xi:include
          • Zero or more inclusion
    • If tight structure model is selected, sequence of
      • Zero or more of inclusion
• Zero or one of:
  • One or more Sequences of
    • One block
    • Zero or more inclusion
  • Sequence of
    • One or more sequences of
      • item
      • xi:include
    • Zero or more inclusion

Attributes
Common Attributes, common.number.attribute, Class Attributes, orient.attribute, and Stop-Contents Attribute

Description
This element is a container for a single attachment such as a schedule, exhibit or appendix to the contract.

A key feature of the attachment is that it can contain a separate eContract as well as the standard narrative structures. This allows a contract to be included as an attachment to another contract.

By default, this element uses the Loose structure model for narrative content. The structure model used for the attachment element may be changed to one of the other structure models defined in this file as part of a customization.

Attributes
Common Attributes, common.number.attribute, Class Attributes, orient.attribute, Stop-Contents Attribute, and common.number.attribute

Parents
These elements contain attachment:attachments

Children
The following elements occur inside attachment: block, contract, inclusion, item (outside of a block), metadata, subtitle, title, and xi:include

Examples
Please see attachments
attachments

Synopsis

Content Model

attachments = element attachments {  
    attachment+,  
    attachments.attlist  
}  

attachments.attlist =  
    common.attributes,  
    attachment.numbering.attributes,  
    attachments.attlist.extensions

attachments.numbering.attributes = empty  
attachments.attlist.extensions = empty

attachments ::=  
    • One or more attachment

Attributes

Common Attributes

Description

This is the container for one or more attachment elements. This element may be used to separate or group the attached content for automatic number or layout purposes.

Attributes

Common Attributes

Parents

These elements contain attachments: contract

Children

The following elements occur inside attachments: attachment

See Also:

object and back
Examples:

Example 13. An attachments with a single attachment element

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of
    - attachments
    - attachment
    - title
    - block
    - text</text></title>
  <body></body>
  <attachments>
    <attachment class="appendix" id="a1" number="15">
      <title><text>Form of notice in Schema files</text></title>
      <block><text>They must be sent regular mail.</text></block>
      <block><text>Standard Schema</text>
        <reference href="http://www.elkera.com" print-url="true">
          can be read on the web easily!</reference>.</text></block>
    </attachment>
  </attachments>
</contract>
```

back

Synopsis

back -- contains elements that follows the body of the contract including signatures.

Content model

```xml
back = element back {
  title?, ((back.structure.model | party-signature)* & date-block?),
  back.attlist
}
back.structure.model = loose.structure.model
back.attlist =
  common.attributes,
  back.numbering.attributes,
  back.attlist.extensions
```
back ::=
  
  • Sequence of
    
    • One of title
    
    • Interleave of
      
      • Zero or more of
        
        • If loose model is selected, sequence of:
          
          • Zero or more of:
            
            • block
            
            • inclusion
            
            • item
            
            • xi:include

        
        • If standard model is selected, sequence of:
          
          • Zero or more inclusion
          
          • Zero or more sequences of
            
            • One block
            
            • Zero or more inclusion

    
    • Sequence of
      
      • One or more choice of
        
        • item
        
        • xi:include

        
        • Zero or more inclusion

    
    • If tight structure model is selected, sequence of
• Zero or more of inclusion

• Zero or one of:
  • One or more Sequences of
    • One block
    • Zero or more inclusion

• Sequence of
  • One or more sequences of
    • item
    • xi:include

• Zero or more inclusion

• party-signature

• Zero or one date-block

Attributes

Common Attributes

Description

The back element contains content that follows the main provisions of the contract, such as signatures and other material. Material found in the back is not hierarchically part of the body.

By default, this element uses the loose structure model for narrative content. The structure model used for the back element may be changed to one of the other structure modes defined in this file as part of a customization.

Attributes

Common Attributes

Parents

These elements contain back: contract

Children

The following elements occur inside back: block, inclusion, item (outside of a block) xi:include, and title
Examples

The following shows the back within a contract including one possible way that signature material could appear:

Example 14. Back Tag with one signatory and their witness

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of :
    - back
    - party-signature
    - signatory-group
    - signatory-record
    - signatory
    - signatory-line
  </text></title>
  <body></body>
  <back>
    <party-signature>
      <signatory-group>
        <block></block>
        <signatory-record>
          <signatory id="T0001" xml:lang="ja">
            <signature-line id="I0001" xml:lang="en_US">
              <text>Mr. Signatory Jr.</text>
              <field>Field for a text.</field>
            </signature-line>
            </signatory>
          <witness>
            <signature-line id="I0002" xml:lang="fr">
              <text>Ms. Witness Arcole</text>
              <field>Field for a text.</field>
            </signature-line>
          </witness>
        </signatory-record>
      </signatory-group>
    </party-signature>
  </back>
</contract>
background

Synopsis

Content Model

background = element background {
    title?, item.reuse.model*,
    background.attlist
}

item.reuse.model = (item)
    item.reuse.model |= xiInclude

background.attlist =
    common.attributes,
    background.numbering.attributes,
    background.attlist.extensions

background.numbering.attributes = empty
background.attlist.extensions = empty

background ::= 
  • Sequence of
    • Zero or one title
    • Zero or more occurences of choice of:
      • item
      • xi:include

Attributes

Common Attributes

Description

This contains recitals and other "background" information.
Attributes

Common Attributes

Parents

background appears inside: contract-front

Children

The following elements occur inside background: item title xi:include

Examples

Please see parties.

block

block -- The container for a structural or grammatical paragraph.

Synopsis

Content model

block = element block {
    (block.level.elements |
     element item {
         metadata?, title?, (block | inclusion)*,
         block.item.attlist
     })*,
     block.attlist
}

block.level.elements = text.container.element | definition | table | inclusion
text.container.element = \text

block.attlist =
    common.attributes,
    block.class.attribute,
    conditional.attributes,
    block.numbering.attributes,
    block.attlist.extensions

    conditional.attributes &= condition.attribute

block.numbering.attributes =
    block.number.type
block.class.attribute = standard.class
block.number.type = attribute number-type { ListItemNumberTypes }?

ListItemNumberTypes = "manual"
| "none"
| "disc"
| "line"
| "number"
| "loweralpha"
| "upperalpha"
| "lowerroman"
| "upperroman"

block.attlist.extensions = empty

block ::= 
  • Zero or more of:
    • text
    • definition
    • table
    • inclusion
    • party
    • person-record
    • item inside a block

Attributes

Common Attributes common.number.attribute

Additional attributes:
  • number-type (enumeration)
    • "manual"
    • "none"
    • "disc"
    • "line"
    • "number"
    • "loweralpha"
    • "upperalpha"
• "lowerroman"
• "upperroman"

Additional Constraints

If this block contains a number-type = manual, then the contract writer SHOULD manually number each list item by putting a number attribute on each of the item elements directly within this block.

Description

The block is the container element for a structural or grammatical paragraph. The actual text data of the paragraph is put in the text element.

Within a block, an item will be considered an element of a list. A block MAY contain the text element directly, an item, as well as definition, table, inclusion, party or person-record.

Attributes

Common Attributes

Additional attributes:
• number-type number-type indicates the type of numbering or markers to be used on contained list items.

Table 3. (enumeration)

<table>
<thead>
<tr>
<th>number-type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;manual&quot;</td>
<td>The contract writer SHOULD indicate list items by putting a number attribute on each of the item tags directly enclosed within this block.</td>
</tr>
<tr>
<td>&quot;none&quot;</td>
<td>The application programmer SHOULD render list items without any marker or number.</td>
</tr>
<tr>
<td>&quot;disc&quot;</td>
<td>The application program SHOULD render list items by a disc or bullet.</td>
</tr>
<tr>
<td>&quot;line&quot;</td>
<td>The application program SHOULD render list items marked by a line or dash.</td>
</tr>
<tr>
<td>&quot;number&quot;</td>
<td>The application programmer SHOULD render list items proceeded by numerals ('1', '2', '3', ...).</td>
</tr>
<tr>
<td>&quot;loweralpha&quot;</td>
<td>The application programmer should render list items proceeded by 'a', 'b', 'c', ..., 'z', 'aa', 'ab', 'ac', 'ad' ..., 'za', 'zb', 'zc'...'zz'</td>
</tr>
<tr>
<td>&quot;upperalpha&quot;</td>
<td>The application programmer should render list items proceeded by 'A', 'B', 'C', ... 'Z', 'AA', 'AB', 'AC', 'AD'... 'ZA', 'ZB', ...'ZZZ'</td>
</tr>
<tr>
<td>&quot;lowerroman&quot;</td>
<td>The application programmer should render list items as 'i', 'ii', 'iii', 'iv', ...</td>
</tr>
<tr>
<td>&quot;upperroman&quot;</td>
<td>The application programmer should render list items proceeded by upper case Roman numerals ('I', 'II', 'III', 'IV', ...)</td>
</tr>
</tbody>
</table>
Parents

These elements contain block: attachment definition inclusion item item (inside a block), note party-signature signatory signatory-group signatory-record witness

Children

The following elements in occur in block: item (inside a block), definition inclusion block.item table text

Examples

See body.

body

Synopsis

body - The body of the contract

Content model

body = element body {
    title?, body.structure.model,
    body.attlist
}

body.structure.model = loose.structure.model

body.attlist =
    common.attributes,
    body.attlist.extensions

body.attlist.extensions = empty

## Tight structure model
tight.structure.model = inclusion*,
    ((block, inclusion*)+ | (item.reuse.model+, inclusion*))?

## Standard structure model
standard.structure.model = inclusion*, (block, inclusion*)*,
    (item.reuse.model+, inclusion*)

## Loose structure model
loose.structure.model = (block | inclusion | item.reuse.model)*
item.reuse.model = (item)
    item.reuse.model |= xi:include

body ::= 
  • Zero or one of
    • title
    • • If loose model is selected, sequence of:
      • Zero or more of:
        • block
        • inclusion
        • item
        • xi:include

    • If standard model is selected, sequence of:
      • Zero or more inclusion
      • Zero or more sequences of
        • One block
        • Zero or more inclusion

    • Sequence of
      • One or more choice of
        • item
        • xi:include

    • Zero or more inclusion

    • If tight structure model is selected, sequence of
      • Zero or more of inclusion
      • Zero or one of:
        • One or more Sequences of
          • One block
          • Zero or more inclusion
• Sequence of
  • One or more sequences of
    • item
    • xi:include
  • Zero or more inclusion

Attributes

Common Attributes

Description

A body must appear in the XML for a contract document.

The body is a container to shield the main part of the contract so items within it can be automatically numbered and rendered independently of the other parts of the contract.

By default, this element uses the loose structure model for narrative content. The structure model used for the body may be changed to one of the other structure models defined in this file as part of a customization.

Parents

These elements contain body: contract

Children

The following elements occur in body: block, inclusion, item (outside of a block), title. xi:include

Examples

Example 15. A valid body under the loose model

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample Loose model data</text></title>
  <body>
    <block> </block>
    <block> </block>
    <item> </item>
    <item>
      <block></block>
    </item>
    <item> </item>
  </body>
</contract>
```
Example 16. A valid body under the standard model

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Sample Standard model data</text></title>
    <subtitle>body is valid as standard model. back is not.</subtitle>
    <body>
        <item>
            <block></block>
            <block></block>
        </item>
    </body>
    <back>
        <item>
            <block></block>
            <block></block>
        </item>
        <item>
            <block></block>
            <block></block>
        </item>
    </back>
</contract>
```

Example 17. A valid body under the tight model.

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Sample Tight model data</text></title>
    <body>
        <item>
            <block></block>
            <block></block>
        </item>
    </body>
</contract>
```
citation

Synopsis

Content Model

citation = element citation {
    inline.content,
    
    citation.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
    inline.content.inner |= conditional

citation.attlist =
    common.attributes,
    citation.attlist.extensions

citation.attlist.extensions = empty

citation ::= 

- Zero or more of
  - conditional
  - em
  - field
  - reference
  - statutory-em
  - strike
  - sub
  - sup


• text

Attributes
Common Attributes

Description
The citation is the name by which a referenced work is cited.

Processing Expectations
It is expected that this text SHOULD be rendered in-line.

Attributes
Common Attributes

Parents
These elements contain citation: reference

Children
The following elements occur inside citation:
conditional, em, field, reference, statutory-em, strike, sub, sup

Examples
See reference

colspec

Synopsis
This is used to provide formatting information and designators for columns within a tgroup of a table.

Content Model

colspec = element colspec {
    empty,

    colspec.attlist
}

colspec.attlist =
    common.attributes,
    attribute colnum    { xsd:NMToken }?,
    attribute colname   { xsd:NMToken }?,
    attribute colwidth  { xsd:string }?,
    attribute colsep    { xsd:boolean }?,
    attribute rowsep    { xsd:boolean }?,
    attribute align     { TableAlign }?,
colspec.attlist.extensions = empty

TableAlign = "left" | "right" | "center" | "justify"

colspec is empty.

Attributes

Common Attributes

Additional Attributes:
- align
- colname
- colnum
- colsep
- colwidth
- rowsep

Description

This is a column specification in a table. This element and all other elements contained by \texttt{table} are taken from OASIS Exchange Table Model. Please refer to that specification for full details of this element: [table1], [table2].

The contract writer SHOULD provide one \texttt{colspec} for each column of their table. The contract writer MAY provide a name (\texttt{colname}), a number (\texttt{colnum}) to designate their column as well as formatting information with its remaining attributes.

Attributes

Common Attributes

Additional Attributes:

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Model for details and processing semantics for these attributes.

align This is the default alignment for table cells within the column defined by the colspec:

\textbf{Table 4. (enumeration)}

<table>
<thead>
<tr>
<th>left</th>
<th>align to the left (default)</th>
</tr>
</thead>
<tbody>
<tr>
<td>right</td>
<td>align to the right</td>
</tr>
<tr>
<td>center</td>
<td>center the text</td>
</tr>
<tr>
<td>justify</td>
<td>justify the text</td>
</tr>
</tbody>
</table>

colname The name of the column
colnum The number of the column

colsep This controls the display of the column separator to the right of this cell in the table.

colwidth The width of the column

rowsep This controls the display of the row separator underneath this cell in the table.

Parents

These elements contain colspec:tgroup

Children

No elements occur inside colspec.

See Also

Some of this information may be overridden in row and entry elements.

Example

Please see table

conditional

Synopsis

Content Model

Conditional has a mixed content model

```xml
conditional = element conditional {
  inline.content,
  conditional.attlist
}
```

```xml
conditional.attlist =
  common.attributes,
  conditional.attributes,
  conditional.attlist.extensions
```

```xml
  conditional.attributes = empty
  conditional.attributes & condition.attribute
```

```xml
conditional.attlist.extensions = empty
```

```xml
inline.content = (inline.content.inner)*)
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
```
inline.content.inner |:= conditional

conditional ::= 

  • Zero or More of
  • conditional
  • em
  • field
  • reference
  • statutory-em
  • strike
  • sub
  • sup

Attributes

Common Attributes

Description

This is for text that will only be included in some versions of the document. See condition.meta and conditions.

Attributes

Common Attributes

Parents

These elements contain conditional: text address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside: conditional

conditional, em, field, reference, statutory-em, strike, sub, sup

Examples

See conditions.
**condition**

**Synopsis**

This defines one of the required strings that must match a `condition` attribute in a block or similar element in order for that content to be included in a particular instance of a contract.

**Content Model**

```xml
<div>
  # add the conditions structure to the metadata element
  metadata.content &=

  # define the conditions element and its children
  element conditions {
    element condition {
      text,
      attribute name { xsd:string },
      attribute default { xsd:boolean }?
    }+
  }?

condition is empty

**Attributes**

- "name"
- "default"

**Description**

The `condition` element is used to specify active conditions when rendering conditional text. The active condition is specified using the `name` attribute of the `condition` element. See the `conditions` element for a full description of this element.

**Attributes**

- **name**: The value of this is matched against the list of strings in a `conditional`, which is within the regular text or a `condition` attribute on an element in the regular contract text.

  - **default**: This is a boolean `true` or `false`.

**Parents**

These elements contain: `condition: conditions`
Children

No elements occur inside condition.

See Also:

conditional.

Examples

See the two examples under conditions

conditions

Synopsis

Content Model

```
metadata.content &=

    # define the conditions element and its children
    element conditions {
        element condition {
            text,
            attribute name { xsd:string },
            attribute default { xsd:boolean }?
        }+
    }?

conditions :=

• Zero or more
  • group

Attributes

none

Description

The conditions element may be used in the metadata to set the values for active conditions on conditional text in the contract.

When the conditional text functionality is activated in the eContracts schema, all eContracts elements are given a condition attribute. The condition attribute is used to specify a name for the condition or conditions that must be active before the processing application will output the element in a rendered contract.

User applications may implement control of conditional text differently, possibly by storing the active conditions element in a separate file, or by giving control to another application, such as the rendering application. This is left to the application programmer.
The **conditions** is provided as a simple mechanism for controlling conditional text within a single contract. Some applications may prefer to implement control of conditional text differently, possibly by storing the active **conditions** element in a separate file, or by giving control to another application, such as the rendering application. This is left to the application developer.

The **condition** element is used to specify active conditions when rendering conditional text. The active condition is specified using the name attribute of the **condition** element. For example,

```xml
<contract>
  <conditions>
    <condition name="US">United States</condition>
  </conditions>
  ...
  <block condition="US">
    <text>A jurisdiction-specific statement in the contract.</text>
  </block>
  <block condition="AU">
    <text>A jurisdiction-specific statement in the contract.</text>
  </block>
</contract>
```

In this example, the statement for the US jurisdiction would be rendered by the processing application. The statement for the AU jurisdiction would not. If the condition element's name attribute was changed so that `<condition name="AU"> .. </condition>`, only the statement for the AU jurisdiction would be rendered. If the condition element was removed, neither of the phrases would be rendered by the processing application.

An element can have multiple conditions set:

```xml
<block condition="US AU">
  <text>A statement specific to both US and AU jurisdictions</text>
</block>
```

In this example, the statement would be rendered when a **condition** element existed for either jurisdictions, i.e.,

```xml
<conditions>
  <condition name="US">United States</condition>
  <condition name="AU">Australia</condition>
</conditions>
```

In this example, the term would be rendered when a **condition** element exists for either jurisdiction.

**Attributes**

none

**Parents**

These elements contain **conditions**: *metadata*

**Children**

The following elements occur inside **conditions**: **condition**

**See Also**

conditional inside a regular inline element.
Examples

See Conditional text processing in Contract Document Structure in the eContracts Schema

contract

Synopsis

This is the root element of the eContracts schema.

Content Model

contract = element contract {
    metadata?, title, subtitle*,
    contract-front?,
    body,
    back?,
    attachments*,
    contract.attlist
}

contract.attlist =
    common.attributes,
    contract.class.attribute,
    orient.attribute,
    contract.numbering.attributes,
    contract.attlist.extensions

contract.numbering.attributes = empty
contract.class.attribute = standard.class
contract.attlist.extensions = empty

contract ::= 
    • Sequence of
      • Zero or one metadata
      • one title
      • Zero or more subtitle
      • Zero or one contract-front
      • One body
      • Zero or one back
      • Zero or more attachments
Attributes

Common Attributes Class Attributes orient.attribute

Description

This is the root element for any contract created using the eContracts schema. This element can also appear inside an attachment element to allow a contract to be added as an attachment to the contract.

Parents

These elements may also contain contract: attachment

Children

The following elements occur inside contract: attachments back body metadata subtitle title

Examples

This illustrates the basic structure of a contract:

Example 18. Basic Structure of a Contract

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0">
  <title><text>Sample of the five elements at level 1</text></title>
  <subtitle>This file is to test: - title - sub-title - contract-front -
    date-block - parties - party - body - back -
    attachments - attachment
  </subtitle>
  <contract-front>
    <date-block>
      <date><em>Long time ago</em></date>
    </date-block>
    <parties>
      <party></party>
    </parties>
  </contract-front>
  <body></body>
  <back></back>
  <attachments>
    <attachment></attachment>
  </attachments>
</contract>
```
Here is an example showing a minimal contract which would consist have only a title and a body:

**Example 19. A Minimal Contract**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Minimum XML sample</text></title>
  <body>
    Minimum XML sample
  </body>
</contract>
```

**contract-front**

**Synopsis**

**Content Model**

```
contract-front = element contract-front {
    ((date-block?, parties) | block+), background?),
    contract-front.attlist
}
```

```
contract-front.attlist =
    common.attributes,
    contract-front.class.attribute,
    contract-front.attlist.extensions
```

```
contract-front.class.attribute = standard.class
contract-front.attlist.extensions = empty
```

```
contract-front::=
  • Sequence of
    • Choice of
      • Sequence of
        • Zero or more date-block
      • One parties
      • One or more block
      • Zero or one background
```
Attributes

Common Attributes Class Attributes

Description

This contains the information that is traditionally found at the beginning of a contract that includes date-block, parties and background.

Attributes

Common Attributes Class Attributes

Parents

These elements contain contract-front:contract

Children

The following elements occur inside contract-front:background block date-block parties

Examples

Example 20. A contract-front with a date-block and parties.

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0">
  <title><text>Sample of the five elements at level 1 </text></title>
  <subtitle>This file is to test : - title - sub-title - contract-front - date-block - parties - party - body - back - attachments - attachment</subtitle>
  <contract-front>
    <date-block>
      <date><em>Long time ago</em></date>
    </date-block>
    <parties>
      <party></party>
    </parties>
  </contract-front>
  <body/>
  <back/>
  <attachments>
    <attachment> </attachment>
  </attachments>
</contract>
```
Example 21. A contract front with a block of text

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title> <text>Sample of : 
        - contract-front 
        - block 
        - text 
        - citation 
        - background 
    </text>
    </title>
    <contract-front>
        <block>
            <text>Elkera Pty Limited is the original developer of the XML schema and accompanying documents described as the <reference><citation>BNML Schema</citation></reference>. The BNML Schema is designed to be adapted and extended for use in a wide variety of applications.</text>
        </block>
        <background> </background>
    </contract-front>

    <body> </body>
</contract>

data

Synopsis

Content Model

data = element data { 
    data.content,
    data.attlist
} 
data.content = empty

data.attlist =
    common.attributes,
    data.src.attribute,
    data.height.attribute,
    data.width.attribute,
    data.attlist.extensions

data.src.attribute =    attribute src { xsd:string }?
data.height.attribute = attribute height   { xsd:integer }?
data.width.attribute =  attribute width   { xsd:integer }?
data.attlist.extensions = empty

Attributes

Common Attributes

• height
• src
• width

Description

This is used to reference the source file and specify dimensions for a multimedia object, particularly images.

Attributes

Common Attributes

height This integer gives the height of the picture. (Units are not defined by this specification.)
src This gives the file path or URI for the multimedia information
width This integer gives the width of the picture. (Units are not defined by this specification.)

Parents

data appears inside: object

Children

No elements occur inside data.

See Also

fallback

Examples

Please see fallback.XML under fallback
date

Synopsis

Content Model

date has a mixed content model.

date = element date {
    (inline.content )*,
    date.attlist
}
date.attlist =
    common.attributes,
    date.class.attribute,
    date.name.attribute,
    date.attlist.extensions
date.class.attribute = standard.class
date.name.attribute = attribute name { xsd:string }?
date.attlist.extensions = empty

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
    inline.content.inner |= conditional
date ::= 
    • Zero or More of
      • conditional
      • em
      • field
      • reference
      • statutory-em
      • strike
      • sub
      • sup
      • text
Attributes
Common Attributes and Class Attributes

Description
This is used to indicate a date. This schema does not specify the date format. The application programmer or contract writer may use a format that is appropriate to the contract.

Attributes
Common Attributes Class Attributes

Parents
These elements contain date: date-block and text

Children
The following elements occur inside date:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also
date-block name field party person-record

Examples
See the first example inside contract.

date-block

Synopsis
Content Model
date-block has a mixed content model.

date-block = element date-block {
  (text.content.inner)*,
  date-block.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

  inline.content.inner |= conditional

text.content = (text.content.inner)*
text.content.inner = inline.content.inner | object | term | phrase | field
  | note | note-in-line | name | address | date | party | person-record
date-block.attlist =
  common.attributes,
  date-block.attlist.extensions

date-block.attlist.extensions = empty

date-block ::= 
  • Zero or More of 
    • em
    • field
    • party
    • person-record
    • reference
    • statutory-em
    • strike
    • sub
    • sup
    • text

Attributes
Common Attributes

Description
This is used to format a date on a separate line.

Attributes
Common Attributes

Parents
These elements contain date-block: contract-front and back

Children
The following elements occur inside date-block: em, field, party, person-record, reference, statutory-em, strike, sub, sup, and text

See Also
parties
Example:
Please see the first example in contract.

dc:contributor

Synopsis
A person, organization, service or other entity which made a contribution to the content.

Content Model

\[
\text{element } \text{dc:contributor} ( \text{xsd:string} )* \\
\text{dc:contributor ::= string}
\]

Attributes
This element has no attributes.

Description
A person, organization, service or other entity which made a contribution to the content. This element is taken from the Dublin Core Metadata Element Set dc. Please refer to that specification for a description of the semantics of this element.

Attributes
This element has no attributes.

Parents
These elements contain dc:contributor:metadata

Children
No elements occur inside dc:contributor.

See Also
dc:creator dc:publisher

Examples
Please see metadata

dc:creator

Synopsis
This is the name of the "entity primarily responsible for making the content" of this contract.
Content Model

    element dc:creator { xsd:string }* &

dc:creator := string

Attributes

This element has no attributes.

Description

This is the name of the "entity primarily responsible for making the content" of this contract. It may be a person, organization or service. This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.

Attributes

This element has no attributes.

Parents

dc:creator appears inside metadata

Children

No elements occur inside dc:creator.

Examples

Please see metadata

dc:date

Synopsis

A date associated with the "lifecycle" of the resource.

Content Model

    element dc:date { xsd:date
    ( pattern = "[0-9]{4}-[0-9]{2}-[0-9]{2}" )
}?

Attributes

This element has no attributes.

Description

This is formally described to be any date associated with the "lifecycle" of the resource but typically it will be the date of creation or availability. Its format will consist of a four-digit year, a dash, a two-digit month, and a two-digit day of the month. This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.
Attributes
This element has no attributes.

Parents
These elements contain dc:date: metadata

Children
No elements occur inside dc:date.

Examples
Please see metadata
dc:description

Synopsis
A description of the contract.

Content Model

```
  element dc:description { xsd:string } ? &
```

dc:description ::= string

Attributes
This element has no attributes.

Description
This is a description of the contract.

This element appears within the metadata. This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.

Attributes
This element has no attributes.

Parents
These elements contain dc:description: metadata

Children
No elements occur inside dc:description.

See Also
dc:subject
Examples
Please see metadata

dc:publisher

Synopsis
The "entity responsible for making the resource available."

Content Model

    element dc:publisher { xsd:string }? &

dc:publisher ::= string

Attributes
This element has no attributes.

Description
This element appears within the metadata. This is the entity responsible for making the resource available. This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.

Attributes
This element has no attributes.

Parents
These elements contain dc:publisher:metadata

Children
No elements occur inside dc:publisher.

See Also
dc:creator

dc:rights

Synopsis
Intellectual Property rights, including copyright, for the text of this contract.
Content Model

```
  element dc:rights { xsd:string }?
```

dc:rights ::= string

Attributes

This element has no attributes.

Description

Intellectual Property rights, including copyright, for the text of this contract. This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.

Attributes

This element has no attributes.

Parents

dc:right appears inside metadata

Children

No elements occur inside dc:rights.

Examples

Please see metadata

dc:subject

Synopsis

The subject matter of the contract.

Content Model

```
  element dc:subject { xsd:string }? &
```

Attributes

This element has no attributes.

Description

This is a description of the subject of the contract. It may often be a list of key words or selected from a "controlled vocabulary." This element is taken from the Dublin Core Metadata Element set dc. Please refer to that specification for a description of the semantics of this element.

Attributes

This element has no attributes.
Parents
The elements contain dc:subject:metadata

Children
No elements occur inside dc:subject.

See Also
dc:description

Examples
Please see metadata

dc:title

Synopsis
The title or name of the contract.

Content Model

  element dc:title { xsd:string }? &

dc:title := string

Attributes
This element has no attributes.

Description
The name given to the contract and will be the name by which it will be formally known. This element is taken from the Dublin Core Metadata Element set dc. Please do not confuse this with the title element that is used throughout the rest of the contract.

Attributes
This element has no attributes.

Parents
These elements contain dc:title:metadata

Children
No elements occur inside dc:title.

Examples
Please see metadata
**definition**

**Synopsis**

**Content Model**

\[
\text{definition} = \text{element definition} \{ \\
\quad (\text{term} \mid \text{terms}), \text{block+}, \\
\quad \text{definition.attlist} \\
\}
\]

\[
\text{definition.attlist} = \\
\quad \text{common.attributes}, \\
\quad \text{definition.class.attribute}, \\
\quad \text{definition.attlist.extensions}
\]

\[
\text{definition.class.attribute} = \text{standard.class}
\]

\[
\text{definition.attlist.extensions} = \text{empty}
\]

\[
\text{definition ::=}
\]

- Sequence of
  - Zero or More Of
    - term
    - terms

**Attributes**

Common Attributes Class Attributes

**Description**

This is for a formal definition structure containing terms and the text which defines their meaning. One uses the terms when there are multiple terms associated with one meaning.

**Processing Expectations**

Each definition SHOULD appear on a separate line and MAY appear with a hanging indentation.

**Attributes**

Common Attributes Class Attributes

**Parents**

These elements contain definition:block.
Children
The following elements occur inside definition: block, term and terms

Examples
See field and term.

em

Synopsis

Content Model

em has a mixed content model.

em = element em {
    inline.content,
    em.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional
em.attlist =
    common.attributes,
    em.class.attribute,
    em.attlist.extensions

em.class.attribute = standard.class
em.attlist.extensions = empty

d :=

- Zero or More of
  - conditional
  - em
  - field
  - reference
  - statutory-em
  - strike
  - sub
  - sup

Children

The following elements occur inside definition: block, term and terms

Examples
See field and term.

em

Synopsis

Content Model

em has a mixed content model.

em = element em {
    inline.content,
    em.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional
em.attlist =
    common.attributes,
    em.class.attribute,
    em.attlist.extensions

em.class.attribute = standard.class
em.attlist.extensions = empty

d :=

- Zero or More of
  - conditional
  - em
  - field
  - reference
  - statutory-em
  - strike
  - sub
  - sup
Attributes
Common Attributes and Class Attributes

Description
The content is emphasized in some way. This specification does not specify how. Typically, the class attribute would be redefined in a specific application to provide enumerated emphasis types.

Attributes
Common Attributes and Class Attributes

Parents
These elements contain em: text

Children
The following elements occur inside em:
conditional, em, field, reference, statutory-em, strike, sub, sup

See Also
statutory-em strike

Examples
Example 22. Example of some in-line elements including em

<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:xi="http://www.w3.org/2001/XInclude">

    <title><text>Sample of inline elements</text></title>

    <body>
        <block>
            <text>The location to deliver <em>the item</em>
            <strike>has to be one of <statutory-em>designated terminals.</statutory-em></strike> will be any places.</text>
        </block>
    </body>

</contract>

entry

Synopsis
This is a table cell.
Content Model

entry = element entry {
  entry.model,
  entry.attlist
}

entry.elements = block | item.reuse.model | inclusion | signature-line
entry.model = (entry.elements)*

item.reuse.model = (item)
  item.reuse.model |= xiInclude

entry.attlist =
  common.attributes,
  attribute colname { xsd:NMTOKEN }?,
  attribute namest { xsd:NMTOKEN }?,
  attribute nameend { xsd:NMTOKEN }?,
  [a:defaultValue = "0"]
  attribute morerows { xsd:NMTOKEN }?,
  attribute colsep { xsd:boolean }?,
  attribute rowsep { xsd:boolean }?,
  attribute align { TableAlign }?,
  attribute valign { TableValign }?,
  entry.numbering.attributes,
  entry.attlist.extensions

TableValign = "top" | "middle" | "bottom"

TableAlign = "left" | "right" | "center" | "justify"

entry.numbering.attributes = empty
entry.attlist.extensions = empty
  entry.attlist &| attribute is-row-header {xsd:boolean}?
  entry.attlist &| attribute abbreviation { xsd:string }?

entry ::= 
  • Zero or more of
    • block
    • item (outside of a block)
    • xi:include
    • inclusion
    • signature-line

70
**Attributes**

Common Attributes

Additional Attributes

- align
- colname
- colsep
- morerows
- nameend
- namest
- rowsep
- valign

**Description**

A cell in a table. This element and all other elements contained by `table` are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2]

**Attributes**

Common Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes.

Additional Attributes:

- **abbreviation**  This attribute is provided to expand an abbreviation found in the `entry`. This attribute is provided to support the WAI Web Content Accessibility Guidelines.
- **align**  The horizontal alignment for content contained in the cell.

**Table 5. (enumeration)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>align to the left (default)</td>
</tr>
<tr>
<td>right</td>
<td>align to the right</td>
</tr>
<tr>
<td>center</td>
<td>center the text</td>
</tr>
<tr>
<td>justify</td>
<td>justify the text</td>
</tr>
</tbody>
</table>

- **colname**  The column name of the entry.
- **colsep**  This controls the display of the column separator to the right of this cell in the table.
- **namest**  The name of the first column in a column span.
- **nameend**  The name of the last column in a column span.
is-row-header  This attribute indicates the entry is a header for the row. This attribute is provided to support the WAI Web Content Accessibility Guidelines.

morerows  The number of rows in a row span.

rowsep  This controls the display of the row separator underneath this cell in the table.

valign  The vertical alignment of the table cell.

**Table 6. (enumeration)**

<table>
<thead>
<tr>
<th></th>
<th>Align content at the top of the cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
</tr>
<tr>
<td>middle</td>
<td>Center content within the cell</td>
</tr>
<tr>
<td>bottom</td>
<td>Align content at the bottom of the cell.</td>
</tr>
</tbody>
</table>

**Parents**

These elements contain **entry: row**

**Children**

The following elements occur inside **entry: block item (outside of a block) xi:include inclusion signature-line**

**fallback**

**Synopsis**

**Content Model**

fallback has a mixed content model.

```
fallback = element fallback {
    (inline.content.inner | object)*,
    fallback.attlist
}
```

```
inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
```

```
fallback.attlist =
    common.attributes,
    fallback.attlist.extensions
```

```
fallback.attlist.extensions = empty
```

```
fallback::=
```
• Zero or More of
  • conditional
  • em
  • field
  • reference
  • statutory-em
  • strike
  • sub
  • sup

Attributes

Common Attributes

Description

This contains the fallback content for an object. The fallback element is used as the fallback position if an object cannot be rendered by a particular application. If the application cannot retrieve, find or render that object, it would go to the fallback element.

The fallback element can contain text data as well as other objects. This provides for nested object elements allowing for a series of fallback positions. The intention is that the rendering application will fallback until it finds an object it can render. To this end, the inner most fallback should always be only text data.

Attributes

Common Attributes

Parents

These elements contain fallback: object.

Children

The following elements occur inside fallback:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also

data

Examples

Example 23. an object with several recursive fallback elements

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
         xmlns:dc="http://purl.org/dc/elements/1.1/
         xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Sample of:
    - object
    - fallback
    - data
</text></title>
    <body>
        <block>
            <text>
                <object type="video/mpeg">
                    <data src="mympeg.mpeg"/>
                </object>
                <fallback>
                    <object type="image/png">
                        <data src="myimage.png"/>
                        <fallback>The alternate text that should be displayed if both
                        the video and image cannot be displayed.
                    </fallback>
                </fallback>
            </text>
        </block>
    </body>
</contract>

field

Synopsis

Content Model

field = element field {
    text,
    field.attlist
}

field.attlist =
    common.attributes,
    field.class.attribute,
    field.label.attribute,
    field.name.attribute,
    field.type.attribute,
field.source.attribute,
field.action.attribute,
field.length.attribute,
field.attlist.extensions

field.class.attribute = standard.class
field.label.attribute = attribute label { xsd:string }?
field.name.attribute = attribute name { xsd:string }?
field.type.attribute = attribute type { xsd:string }?
field.source.attribute = attribute source { xsd:string }?
field.action.attribute = attribute action { xsd:string }?
field.length.attribute = attribute length { xsd:string }?

field.attlist.extensions = empty

field ::= text (#PCDATA)

Attributes

Common Attributes and Class Attributes

Additional attributes:

• action
• label
• length
• name
• source

Description

A field is a generic element that can be used to mark up a unit of information in the contract that is either captured
from the user, inserted from a database, generated by a processing application such as a cross reference tool, or extracted
from the contract for other uses, such as to populate a contract-management database.

Specific functionality is not provided for the field element. The design of the element provides hooks to allow the ap-
lication developer to implement functionality according to the requirements of a specific application.

Typically, this element would be used for generating space on a printed contract page where information, such as a
date or signature, may be written. This may be rendered differently depending on the output format. For example, in
a printed contract, a space may be rendered. In a web forms-based rendition, the field may be rendered as a text input
box, allowing the user to type in a value.

An application will read the XML representation of the contract document to extract information. An example scenario
would be a firm that leases products from many different firms. These firms provide the lease document in XML using
this specification. The company’s management information systems department writes an application to read each one
and extracts the amount to be paid each month. This is used to update the accounts receivable and liabilities section
of the general ledger program.
Another scenario would be a firm generating form contracts from database information. An apartment complex has a table containing for each apartment, the legal description of the apartment and the monthly rent. The contract XML would have field elements for each of these inputs. Again, their MIS department would write an application. In this case, it would look for the appropriate tuple in the database. It would replace the field elements by the corresponding fields or columns from that tuple.

**Attributes**

**Common Attributes and Class Attributes**

**Additional attributes:**

- **action** This is used to specify a statement relating to the type of processing action that may be applied to the field by processing applications.

- **label** This is for display purposes. For example, a form-based graphical-user interface program could display a text box for each field element it finds. The label attribute MAY be used to label this box so the user knows what to put there.

- **length** This is used to specify the length of the field when the field is used for data entry. This specification does not define the units of measurement. The units of measurement used are left to the application developer.

- **name** This is used to assign a name to the field for data extraction and processing purposes. For example, if this application is extracting data from the contract and putting it in a database, the name may contain the name of the database column where the content of the field is extracted from or written to.

- **source** This is used to identify the source of the field information when the field is to be populated by a processing application. This could be a SQL or XPATH query.

- **type** This is used to indicate the data type of the field. This would typically be enumerated for a particular application. This specification does not define field data types. These are left to the application developer.

**Parents**

These elements contain field: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

**Children**

field can only contain text data (#PCDATA)

**See Also**

note, note-in-line, reference, party, phrase, and term.

**Examples**

**Example 24. The Payment Amount as a field.**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
```
Sample of field

In this licence:

BNML Standard Schema

means the XML Schema called Pay <field source="ida" name="PaymentAmount"></field>.

inclusion

Synopsis

Content Model

inclusion = element inclusion {
    metadata?, title?, inclusion.structure.model,
    inclusion.attlist
}

inclusion.structure.model = loose.structure.model

## Tight structure model
tight.structure.model = inclusion*,
    ((block, inclusion*)+ | (item.reuse.model+, inclusion*))?

## Standard structure model
standard.structure.model = inclusion*, (block, inclusion*)*,
    (item.reuse.model+, inclusion*)

## Loose structure model
loose.structure.model = (block | inclusion | item.reuse.model)*
item.reuse.model |= xiInclude

inclusion.attlist =
   common.attributes,
   inclusion.class.attribute,
   common.number.attribute,
   inclusion.align.attribute,
   orient.attribute,
   inclusion.numbering.attributes,
   inclusion.attlist.extensions

inclusion.numbering.attributes = empty
inclusion.class.attribute = standard.class
inclusion.align.attribute = attribute align { ShortAlignment }?
inclusion.clear.attribute =
   [a:defaultValue = "both"]
   attribute clear { ClearEnumValues }?
ClearEnumValues = "left" | "right" | "both"
inclusion.width.attribute = attribute width { xsd:string }?
inclusion.attlist.extensions = empty

inclusion ::=  
  
  Sequence of  
  
  Zero or one metadata  
  
  title.  
  
  If loose model is selected, sequence of:  
  
    Zero or more of:  
    
    block  
    
    inclusion  
    
    item  
    
    xi:include  
  
  If standard model is selected, sequence of:  
  
    Zero or more inclusion  
  
    Zero or more sequences of  
    
      One block
• Zero or more inclusion

• Sequence of
  • One or more choice of
    • item
    • xi:include

• Zero or more inclusion

• If tight structure model is selected, sequence of
  • Zero or more of inclusion
  • Zero or one of:
    • One or more Sequences of
      • One block
      • Zero or more inclusion

• Sequence of
  • One or more sequences of
    • item
    • xi:include

  • Zero or more inclusion

**Attributes**

Common Attributes Class Attributes orient.attribute common.number.attribute

Additional Attributes:

• align

**Description**

The inclusion element is a generic container element for content that is distinct from the narrative, such as quotations, annotations, notes and examples. It is also used to provide a title and number on graphical objects and tables.
Typically, the inclusion element contains content that is separate from the main contract provisions for automatic number of layout purposes.

By default, the inclusion uses the loose structure model. This model is very flexible and is able to mark up a variety of content that may not conform to regular structures and where a more strict structure is generally not necessary. The structure model used for the inclusion element may be changed to one of the other structure models as part of a customization.

**Processing Expectations**

Hierarchical content inside the inclusion should use a separate number sequence from the main contract hierarchy.

**Attributes**

Common Attributes Class Attributes `orient.attribute`, `common.number.attribute`

Additional Attributes:

- `align`: This is the horizontal alignment for content within the inclusion as given by `ShortAlignment` pattern:

  **Table 7. (enumeration)**

<table>
<thead>
<tr>
<th>align</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>The content of the inclusion is left aligned.</td>
</tr>
<tr>
<td>center</td>
<td>The content of the inclusion is centered.</td>
</tr>
<tr>
<td>right</td>
<td>The content of the inclusion is right-aligned.</td>
</tr>
</tbody>
</table>

**Parents**

These elements contain: `inclusion: block`, `entry`, `attachment`, `back`, `block`, `inclusion` and `item`

**Children**

The following elements occur inside: `inclusion`, `inclusion`, `item`, `metadata`, `item`, `xi:include`

**See Also:**

- `object`

**Examples:**

**Example 25. Two inclusion elements**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">

  <title><text>Sample of</text>
    - item
    - num
    - block
    - inclusion
    - phrase
  </text></title>
</contract>
```
BNML Standard consists of 6 BNML specific files and 2 files that incorporate external features into the application. The bnml-standard is the main file which includes all of the other files:

```
/inclusion class="code-listing"
    <block number-type="none">
        <text>/BNML Standard</text>
        <item>
            <block number-type="none">
                <text>/RelaxNG</text>
                <item>
                    <block>
                        <text>bnml-standard.rnc</text>
                        <text>bnml-document.rnc</text>
                        <text>bnml-contract.rnc</text>
                        <text>bnml-correspondence.rnc</text>
                        <text>bnml-core.rnc</text>
                        <text>bnml-structure.rnc</text>
                        <text>dc-metadata.rnc</text>
                        <text>xi-include.rnc</text>
                    </block>
                </item>
            </block>
        </item>
    </block>
</inclusion>
```

```
/inclusion class="example">
    <block>
        <text>
            <phrase class="code">bnml-s-application name.rnc</phrase>
        </text>
    </block>
</inclusion>
```
item (outside of a block)

Synopsis

The item is the basic building block of the document hierarchy and represents structures that may have a title and a number. These may be known in documents as "chapters," "parts," "sections," "clauses" and "subclauses."

Content Model

```xml
item = element item {
    metadata?,
    title?,
    item.structure.model,
    item.attlist
}
```

```xml
item.structure.model = loose.structure.model
item.reuse.model = (item)
```

## Tight structure model

```xml
tight.structure.model = inclusion*,
    ((block, inclusion*)+ | (item.reuse.model+, inclusion*))?
```

## Standard structure model

```xml
standard.structure.model = inclusion*, (block, inclusion*)*,
    (item.reuse.model+, inclusion*)
```

## Loose structure model

```xml
loose.structure.model = (block | inclusion | item.reuse.model)*
```

```xml
    item.reuse.model |= xiInclude
```

```xml
item.attlist =
    common.attributes,
    item.class.attribute,
    common.number.attribute,
    conditional.attributes,
    stop-contents.attribute,
    item.numbering.attributes,
    item.attlist.extensions
```

```xml
item.numbering.attributes = empty
item.class.attribute = standard.class
item.attlist.extensions = empty
```

item ::= 

• Sequence of:
• Zero or one metadata

Zero or one title

Attributes

Common Attributes Class Attributes condition.attribute Stop-Contents Attribute

Description

The item element is the basic building block of the contract hierarchy. It represents structures that may have a title or a number. Commonly, such structures are known as provisions, sections, clauses and subclauses.

It also intended to represent items in a list. The TC was careful not to use a name for elements that matches any of that list so as to avoid confusion or biasing the reader's idea of how they might be used: [meyer03]. Please see item (inside a block) as item when it appears directly inside a block as a list item.

The item is intended to be the basic re-usable element that can be inserted almost anywhere in the document hierarchy.

The item element is typically added to an element's content models by using the item.reuse.model pattern, defined above. This pattern allows the content reuse features of the schema to be used in all content models where item occurs.

Processing expectations

If the stop-contents attribute is not used on a parent or ancestor element, the item's number and title SHOULD be presented in the table of contents. The item may be suppressed based upon its condition attribute.

Attributes

Common Attributes Class Attributes condition.attribute Stop-Contents Attribute

Parents

These elements contain item: attachment, back, block, inclusion and item

Children

The following elements occur inside item: inclusion item, metadata, item xi:include

item (inside a block)

Synopsis

This is the list item element.

Content Model

\[
\text{block} = \text{element block} \{
\text{block.level.elements} |
\text{element item} \{
\text{metadata?}, \text{title?}, \text{block} | \text{inclusion} \},
\}
\]
block.item.attlist
)
)*,

block.attlist
}

block.level.elements = text.container.element | definition | table | inclusion
text.container.element = \text

block.attlist =
  common.attributes,
  block.class.attribute,
  conditional.attributes,
  block.numbering.attributes,
  block.attlist.extensions

block.numbering.attributes =
  block.number.type

block.class.attribute = standard.class
block.number.type =    attribute number-type { ListItemNumberTypes }?

ListItemNumberTypes =   "manual"
  | "none"
  | "disc"
  | "line"
  | "number"
  | "loweralpha"
  | "upperalpha"
  | "lowerroman"
  | "upperroman"

block.attlist.extensions = empty

item ::=  
  • Sequence of
    • Zero or one metadata
    • Zero or more of
      • block
      • inclusion
Attributes
Common Attributes, Class Attributes and condition.attribute common.number.attribute

Description
Relax NG supports the ability to define an element with a different content model when it is directly under some other element. That is, it creates a context-sensitive grammar. See [sperberg] and [Relax]. This feature is used only once in the grammar for this specification, an item has a different definition inside a block than when it appears in other elements.

This is the list item element. The list element is redefined in the schema within the block context. This constrains the content model to allow only structures that would appear in list items.

Process Expectations
The list item should be numbered or marked according to the number-type attribute on the enclosing block element.

Attributes
Common Attributes, Class Attributes and condition.attribute

Parents
These elements contain a list item: block

Children
The following elements occur inside a list item: metadata, title, block and inclusion

See Also
item (outside of a block)

Examples
See body.

metadata

Synopsis

Content Model

metadata = element metadata {
    metadata.content,
    metadata.attlist
}
metadata.content = empty
metadata.attlist =
    common.attributes,
    metadata.attlist.extensions
metadata.attlist.extensions = empty

div {
  # add the condition attribute to all eContract elements.
  conditional.attributes &= condition.attribute
  inline.content.inner |= conditional

  # add the conditions structure to the metadata element
  metadata.content &=

    # define the conditions element and its children
    element conditions {
      element condition {
        text,

        attribute name { xsd:string },
        attribute default { xsd:boolean }?
      }+
    }?
}

metadata ::=  
  · Zero or more instances of
    · conditions
    · dc:contributor
    · dc:creator
    · dc:date
    · dc:description
    · dc:publisher
    · dc:rights
    · dc:subject
    · dc:title

Attributes

Common Attributes
**Description**

This is a container for metadata. Metadata is currently defined to include a number of basic elements from the Dublin Core standard defined in `dc-metadata.rnc` [dc] and the `conditions` element.

It is intended that applications of the eContract schema will customize the metadata element to provide values that are applicable to that application.

**Processing Expectations**

The information in metadata tags SHOULD NOT be rendered in the normal narrative of the contract document. However, metadata values MAY be used in presentation, such as for page headers and footers.

**Attributes**

Common Attributes

**Parents**

These elements contain metadata: attachment, item (inside a block), contract, inclusion, and item

**Children**

The following elements occur inside metadata: conditions dc:contributor dc:creator dc:date dc:description dc:publisher dc:_rights dc:subject dc:title

**See Also:**

title

**Example**

**Example 26. A metadata element showing several elements from the Dublin Core**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <metadata>
        <dc:title>WIU CS Department notice 001</dc:title>
        <dc:creator>Julie Sasa</dc:creator>
        <dc:contributor>Keith Wallen</dc:contributor>
        <dc:subject>Scope of licence</dc:subject>
        <dc:publisher>WIU</dc:publisher>
        <!-- <dc:date>2005-09-20</dc:date> -->
        <dc:rights>Copyright, Elkera Pty Limited, 2005</dc:rights>
    </metadata>
    
    <title><text>Sample of metadata and its child elements</text></title>
    
    <body> </body>
</contract>
```
name

Synopsis

Content Model

name has a mixed content model.

```
name = element name {
    (inline.content | field)*,
    name.attlist
}
```

```
inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
    inline.content.inner |= conditional
```

```
name.attlist =
    common.attributes,
    name.class.attribute,
    name.name.attribute,
    name.attlist.extensions
```

```
name.class.attribute = standard.class
name.name.attribute = attribute name { xsd:string }?
name.attlist.extensions = empty
```

name :=

- Zero or More of
  - conditional
  - em
  - field
  - reference
  - statutory-em
  - strike
  - sub
  - sup
• text

Attributes

Common Attributes Class Attributes

Description

This provides a text content for a name of a person, organization or company. It is often used in a party-record to mark up the name of the party to a contract.

Attributes

Common Attributes Class Attributes

Parents

These elements contain name: date-block, person-record, and text

Children

The following elements occur inside name:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also:

address name

Examples

See address and parties

note

Synopsis

This is a footnote or endnote.

Content Model

```
note = element note {
   block+,
   note.attlist
 }

note.attlist =
   common.attributes,
   note.class.attribute,
   common.number.attribute,
   note.attlist.extensions
```
note.class.attribute = standard.class
note.attlist.extensions = empty

note ::= 
  • Sequence of
    •
    • One or more block

Attributes
Common Attributes, common.number.attribute and Class Attributes

Description
This is used for a foot note, end note or other type of note. These would be notes that are NOT rendered in line with its surrounding text.

Attributes
Common Attributes and Class Attributes

Parents
These elements contain note: date-block and text

Children
The following elements occur inside note: block

See Also
note-in-line

Example

Example 27. Two notes and a note-in-line

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of note and note-in-line.</text></title>
  <body>
    <block>
      <text>Send me a shirt.</text>
      <note-in-line>standard size.</note-in-line>
    </block>
  </body>
</contract>
Send me desks as listed below.

<note number="1"><block><item></item></block></note>
<note number="2"><block><item></item></block></note>

note-in-line

Synopsis

This is for notes that are kept inline with surrounding text.

Content Model

note-in-line has a mixed content model.

```plaintext
note-in-line = element note-in-line { 
  inline.content,

  note-in-line.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

inline.content.inner |= conditional
note-in-line.attlist =
  common.attributes,
  note-in-line.class.attribute,
  note-in-line.attlist.extensions

note-in-line.class.attribute = standard.class
note-in-line.attlist.extensions = empty
```

note-in-line ::= 

- Zero or More of 
  - conditional 
  - em 
  - field 
  - reference
Attributes

Common Attributes and Class Attributes

Description

This is for notes that are kept inline with surrounding text.

Processing Expectations

This SHOULD be rendered so the text comprising the note-in-line can be easily distinguished from the other text surrounding it.

Attributes

Common Attributes Class Attributes

Parents

These elements contain note-in-line: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside: note-in-line:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also

note phrase reference

Examples

Example 28. Two notes and a note-in-line

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of note and note-in-line.</text></title>
  <body>
    <block>
```

<text>Send me a shirt.<br>
    <note-in-line>standard size.</note-in-line></text>
</block>

<block>
    <text>Send me desks as listed below.<br>
        <note number="1">item</note>
        <note number="2">item</note>
</text>
</block>

<contract>

<object>

Synopsis

Content Model

object = element object {
    data, fallback,
    object.attlist
}

object.attlist =
    common.attributes,
    object.type.attribute,
    object.scale.attribute,
    object.rotate.attribute,
    object.attlist.extensions

object.type.attribute = attribute type { xsd:string }?
object.scale.attribute = attribute scale   { xsd:string }?
object.rotate.attribute = attribute rotate  { xsd:string }?

object.attlist.extensions = empty

object ::=  
    • Sequence of
        • data
        • object
Attributes

Common Attributes and Class Attributes

Additional Attributes:

- type
- scale
- rotate

Description

The object element is a container element for external objects such as images and other multimedia files.

An object can contain one data and one fallback element. The data element indicates the source of the object and the fallback element contains the fallback position if a particular application cannot render the object.

The type attribute MUST be a MIME content type (e.g. image/png), and the data element MUST point to a valid location from which the object may be accessed.

Processing Expectations

The data element indicates the source of the object, e.g. a file name or URL. It is possible that the application may not be able to locate the object or may not be able to render it once reached. For example, the application may not know how to deal with a format or the network may fail or the object data might be corrupted. If so, the fallback contains the item or text to be displayed. As discussed under fallback, this may be a cascading or recursive set of fallbacks.

Attributes

Common Attributes and Class Attributes

Additional Attributes:

- rotate: This is used to rotate the object. This attribute typically contains the degrees of anti-clockwise rotation applied to the object.
- scale: This is used to scale the object. This specification does not state how this should be written.
- type: This specifies the MIME type of the object, e.g. Image/Jpeg or Audio/Basic [Grand93]

Parents

These elements contain object: address, date, em, fallback, field, name, note, note-in-line, object, party, person-record, phrase, reference, statutory-em, strike, sub, sup, term

Children

The following elements occur inside object: data and fallback

Example

Please see fallback.
parties

Synopsis

The parties element contains the parties to the contract.

Content Model

parties = element parties {
    title?, party+,
    parties.attlist
}

parties.attlist =
    common.attributes,
    parties.class.attribute,
    parties.attlist.extensions

parties.class.attribute =  standard.class
parties.attlist.extensions = empty

parties ::= 
    • Sequence of 
        • Zero or one title 
        • One or more party

Attributes

Common Attributes and Class Attributes

Description

The parties contains the parties to the contract.

In some jurisdictions, such as Australia, New Zealand and the UK, it is common to layout the parties to a contract on separate lines under a Parties title. For example:

This agreement is made on the ___ day of ___

between the

first party (short-name)

and

second party (short-name)
In US contracts, this is not so common. In the US cases, the parties element may not be required. For example:

This contract is made and entered into between first party, hereinafter referred to as "short-name" and second party, hereinafter referred to as "short-name"

Or taking the form from the American Institute of Architects standard contract:

AGREEMENT made as of the day of in the year

BETWEEN the short-name:

date

and the short-name:

date

the short-name is then used throughout the contract. For example, one might write that "Acme Business Systems, Inc." would be referred to as "the Company" hereinafter.

The party element may be used directly inside block for US contracts.

**Processing Expectations**

The title SHOULD appear at the top with the information for each party on a separate line.

Below is an example of how the front might look for such a contract. Please note that this does NOT match any of the xml examples provided:

---

**Example party-signature markup**

Dated: 2006

**Parties**

XYZ Limited of Suite 101, 12 Main Street, Sometown, State 00000. (XYZ)

John W. Doe of 12 Long Street, Suburbia State 00000. (JWD)

**Party Attributes**

Common Attributes and Class Attributes

**Parents**

These elements contain parties:contract-front

**Children**

The following elements occur inside parties:title and party
See Also

party

Examples

Example 29. Parties Example

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>
  Sample of :
  - contract-front
  - parties
  - party
  - person-record
  - name
  - address
  - background
  </text></title>
  <contract-front>
  <date-block>Agreement dated:
    <field class="date" type="blank" name="contract_date" length="75mm">
    <?xm-replace_text {ec:field}?></field></date-block>
  <parties><title><text>Parties</text></title>
    <party><person-record><name>ABC Ventures Limited</name> having
      its office at <address>100 Main Street, Sydney, NSW 2000</address>
      </person-record>, hereafter referred to as
      "<term>General Partner</term>"
    </party>
    <party><person-record><name>John A. Doe</name> of
      <address>10 Ramrod Drive, Sydney, NSW 2000</address>
      and <person-record><name>John W. Smith</name> of
      <address>25 Pine Road, Plainsville, NSW, 0000</address>
      </person-record>, herafter collectively referred to as
      "Limited Partners"
    </party>
  </parties>
  <background><title><text>Background</text></title>
    <item number="A">This is the text of the first recital</item>
    <item number="B">Text of the second recital</item>
  </background>
</contract-front>
party

Synopsis

This is a party to the contract.

Content Model

party has a mixed content model.

\[
\text{party} = \text{element party} \ ( \\
(\text{text} \mid \text{person-record} \mid \text{term})^*, \\
\text{party.attlist}
) \\
\text{party.attlist} = \\
\text{common.attributes}, \\
\text{party.class.attribute}, \\
\text{party.attlist.extensions}
\]

\text{party.class.attribute} = \text{standard.class} \\
\text{party.attlist.extensions} = \text{empty}

party ::= 

\begin{itemize}
\item Zero or More of 
\begin{itemize}
\item text 
\item person-record 
\item term 
\end{itemize}
\end{itemize}

Attributes

Common Attributes and Class Attributes

Description

The party element is used to contain the details of each party to the contract.
A party may occur inside the parties element or it may occur inside a text element.

Within the party element, the details such as name and address of each person or entity in the party are captured using the person-record element.

**Attributes**

Common Attributes Class Attributes

**Parents**

These elements contain party: date-block parties text

**Children**

The following elements occur inside party: text, person-record and term

**See Also**

name address

**Example**

See parties

**party-signature**

**Synopsis**

party-signature -- Place where a signature or seal will be applied to the contract document.

**Content model**

```
party-signature = element party-signature {
    (block*, ( signatory-group | signatory-record ), block*),
    party-signature.attlist
}
```

```
party-signature.attlist =
    common.attributes,
    party-signature.layout.attribute,
    party-signature.party-id.attribute,
    party-signature.attlist.extensions
```

```
party-signature.layout.attribute =
    {a:defaultValue = "from-left"}
attribute layout { party-signature.layout.values }?
```

```
party-signature.layout.values = "right-column-only" | "from-left"
```
The party-signature markup provides a semantic representation of the components needed to render complex signature provisions in contracts. It also includes some presentational characteristics. If this complexity is not required, signature provisions may be created by using tables and the signature-line element.

This is the place for markup for what would be considered a signature. In the case, where this is rendered to a physical document, this is where a person would apply a "pen-and-ink" signature and/or affix a seal. There will be sets of signatures, for example, a signatory and witness or two partners or officers of the same company.

Processing Expectations

If align-records is horizontal, then the signatures will be rendered in multiple columns, usually two. If set to vertical, then they will be rendered vertically.

Attributes

Common Attributes

<table>
<thead>
<tr>
<th>layout</th>
<th>The page is divided into two columns. Render the signature in the right column.</th>
</tr>
</thead>
<tbody>
<tr>
<td>right-column-only</td>
<td>render signatures from the left page margin.</td>
</tr>
<tr>
<td>from-left</td>
<td></td>
</tr>
</tbody>
</table>
Parents

These elements contain `party-signature: back`

Children

The following elements occur in `party-signature: block, signatory-group, signatory-record`

Examples

The following example consists of a single `signatory-record` containing both a `signatory-record` and a `witness` within the `party-signature` record:

**Example 30. A single signer plus a witness**

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of:
    - back
    - party-signature
    - signatory-group
    - signatory-record
    - signatory
    - signatory-line</text></title>
  <body></body>
  <back>
    <party-signature>
      <signatory-group>
        <block> </block>
      </signatory-group>
      <signatory-record>
        <signatory id="T0001" xml:lang="ja">
          <signature-line id="I0001" xml:lang="en_US">
            <text>Mr. Signatory Jr.</text>
            <field>Field for a text.</field>
          </signature-line>
        </signatory>
        <witness>
          <signature-line id="I0002" xml:lang="fr">
            <text>Ms. Witness Arcole</text>
            <field>Field for a text.</field>
          </signature-line>
        </witness>
      </signatory-record>
    </party-signature>
  </back>
</contract>
```
A simple party signature by John Doe is here. Observe the initial block to give the text "Signed by John Doe" A signatory-record, signatory and signature-line

Signed by John W. Doe

John W. Doe

Example 31. A single person signing

<party-signature> <block><text>Signed by <name class="person">John W. Doe</name></text></block><signatory-record><signatory><signature-line><field type="blank" length="50"/> <text>John W. Doe</text></signature-line></signatory> </signatory-record></party-signature>

When there are two persons signing for one party, there are two signature-line tags within one signature-group:

Signed by the Limited Partners

John A Doe Partner
John W. Smith Partner

Example 32. Two Partners or officers of the same company

<party-signature> <block><text>Signed by the Limited Partners</text></block><signatory-group align-records="horizontal"> <signatory-record><signatory><signature-line><field type="blank" length="50"/> <text>John A Doe</text></signature-line></signatory> </signatory-record></signatory-group></party-signature>
Observe the align-record=horizontal which forces both Mr. Doe and Mr. Smith's signature lines to be side by side.

Similarly, one can include a witness element in a signatory-record:

Signed by John W. Smith in the presence of:

\[ \begin{array}{cc}
\text{Witness} & \text{John W. Smith} \\
\end{array} \]

Example 33. A signatory and a witness

Signed by John W. Smith in the presence of:

\[ \begin{array}{cc}
\text{Witness} & \text{John W. Smith} \\
\end{array} \]

Observe the align-signatory-witness attribute which appears on the signatory-record. Here is a similar example, but where the align-signatory-witness attribute is specified to be vertical:

Signed by John W. Smith:

\[ \begin{array}{cc}
\text{John W. Smith} & \\
\text{Witness} & \\
\end{array} \]
Example 34. Illustration of align-signatory-witness


The user may put the attribute, layout="right-column-only" on the party-signature to have several lines of text as well as the place where the user will sign lined up on the right side of the page:

Accepted and agreed:
Signed on behalf of XYZ Corporation Limited, by John W. Smith, its authorized director:

.........................................................
John W. Smith
Director

Example 35. Right-column-only illustration

<party-signature layout="right-column-only"><block><text>Accepted and agreed:</text></block><text>Signed on behalf of XYZ Corporation Limited, by John W. Smith, its authorized director:</text><signatory-record><signature-line><field length="50" type="dotleader"/>John W. Smith</signature-line><text>Director</text></signatory-record></party-signature>

This example shows the use of the brace attribute on the signatory-record:
Signed on behalf of XYZ Corporation Limited by John W. Smith, its managing director in the presence of:

Signature of witness

Name of witness

Example 36. The brace attribute

signatory-record's can be nested to show delegation. This is rendered by a small indentation for the information within the nested signatory-record.
"Lender"
Roadway Finance Company,
a Delaware limited company
By: XYZ Limited Partnership,
a Delaware limited partnership, its sole member
   By: XYZ Office Properties Trust,
a Delaware real estate investment trust, its general partner

   By: /s/ John A. Doe
   Name: John A. Doe
   Title: Senior Vice President

   1000 North Riverside Plaza
   Chicago, IL 60606
   Attention: General Counsel

Example 37. Nested signatory-records to show delegation.

  <block><text>This is a signature block with a nested record:</text></block>
  <party-signature layout="right-column-only">
    <block><text>"Lender"</text></block>
    <block><text>Roadway Finance Company,</text></block>
    <text>a Delaware limited company</text>
  </signatory-record>
  <signatory-record>
    <text>By: XYZ Limited Partnership,</text>
    <text>a Delaware limited partnership, its sole member</text>
  </signatory-record>
  <signatory-record>
    <text>By: XYZ Office Properties Trust,</text>
    <text>a Delaware real estate investment trust, its general partner</text>
  </signatory-record>
  <signature-line>
    <text>By: /s/ John A. Doe</text>
    <field length="50" type="blank"/>
  </signature-line>
  <block><text>Name: John A. Doe</text></block>
Synopsis

This is used to contain information about a person or other entity.

Content Model

person-record has a mixed content model.

\[
\text{person-record} = \text{element person-record} \{
\begin{align*}
\text{text} & | \text{name} | \text{address} | \text{field} | \text{term}^*, \\
\text{person-record.attlist} &
\end{align*}
\}
\]

\[
\text{person-record.attlist} = \\
\quad \text{common.attributes,} \\
\quad \text{person-record.class.attribute,} \\
\quad \text{person-record.party-id.attribute,} \\
\quad \text{person-record.attlist.extensions}
\]

\[
\begin{align*}
\text{person-record.class.attribute} & = \text{standard.class} \\
\text{person-record.party-id.attribute} & = \text{attribute party-id \{ xsd:string \}?} \\
\text{person-record.attlist.extensions} & = \text{empty}
\end{align*}
\]

\[
\text{person-record} ::= \\
\text{Zero or more} \\
\text{address} \\
\text{field} \\
\text{name} \\
\text{term} \\
\text{text}
\]
Attributes

Common Attributes Class Attributes

Additional attributes:
- party-id

Description

The `person-record` element is used to capture the details of a person or other entity that is a party to the contract. It may also be used to describe any person who is referenced in the contract, such as an attorney under a power of attorney.

Normally, a `person-record` would be contained within a `party` element. However, in some US-style contracts, the person's details may be broken up so it is not possible to capture all details within a single `party` element and still maintain semantic integrity of the markup and grammatical integrity of the sentence. For situations where the person's details cannot be captured within a single party element, the `person-record` element is allowed to occur inside the `text` element.

When a `person-record` element is not used within a `party` element, the `person-record`'s `party-id` attribute is used to reference the `party` to which the record relates.

Attributes

Common Attributes Common Attributes

Additional Attributes:

- party-id This is used to associate a `person-record` with a `party` element when the `person-record` needs to be created outside the `party` element.

Parents

These elements contain `person-record`: `date-block`, `party`, and `text`.

Children

The following elements occur inside `person-record`: `address`, `field`, `name` and `term`.

See Also

- party

Examples

See `parties`

phrase

Synopsis

Content Model

`phrase` has a mixed content model.
phrase = element phrase {
    inline.content,
    phrase.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional
phrase.attlist =
    common.attributes,
    phrase.class.attribute,
    phrase.attlist.extensions

phrase.class.attribute = standard.class
phrase.attlist.extensions = empty

phrase ::= 

• Zero or more of
  • conditional
  • em
  • field
  • reference
  • statutory-em
  • strike
  • sub
  • sup

Attributes

Common Attributes and Class Attributes

Description

This might be used on something for which emphasis is placed in rendering. It may be used to explain something outside the normal flow of text. It may be a technical or foreign word.

Processing Expectations

This is rendered inline. Typically, it would be rendered differently so as to give emphasis, possibly as indicated by class attribute.
Attributes

Common Attributes and Class Attributes

Parents

The following elements occur inside phrase: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside phrase:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also

note and note-in-line

reference

Synopsis

Content Model

reference = element reference {
  (inline.content | citation)*,
  reference.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

  inline.content.inner |= conditional
reference.attlist =
  common.attributes,
  reference.class.attribute,
  reference.href.attribute,
  reference.print-url.attribute,
  reference.attlist.extensions

reference.class.attribute = standard.class
reference.href.attribute = attribute href { xsd:anyURI }?
reference.print-url.attribute = attribute print-url { xsd:boolean }?

reference.attlist.extensions = empty

  reference.attlist &= attribute destination-type { xsd:string }?
  reference.attlist &= attribute destination-lang { xsd:string }?
reference ::= 
  • Zero or more of 
    • conditional 
    • citation 
    • em 
    • field 
    • reference 
    • statutory-em 
    • strike 
    • sub 
    • sup 

Attributes
Common Attributes and Class Attributes

Additional attributes:
• destination-lang 
• destination-type 
• href 
• print-url 

Description
A reference is a word or phrase which refers to another resource, whether internal or external to the contract. Typically, it is used for cross references and citations to other works.

Processing Expectations
The content of this element is rendered inline.

Attributes
Common Attributes

Additional Attributes:

destination-type This attribute indicates the type of resource that is referenced. This attribute is provided to support the WAI Web Content Accessibility Guidelines.

destination-lang This attribute indicates the language of the resource that is referenced. This attribute is provided to support the WAI Web Content Accessibility Guidelines.
href

This specifies the URL or other way of finding the object, as one would write it in HTML.

print-url

This is used to control rendering of the references to external web sites in print outputs. For example: <reference href="http://www.oasis-open.org/committees/documents.php?wg_abbrev=legalxml-econtracts">This specification</reference> In print output, it might be desirable to be able to output this as:

This specification (www.oasis-open.org/committees/documents.php?wg_abbrev=legalxml-econtracts)

Parents

These elements contain reference: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside reference:

citation em field reference statutory-em strike sub sup

See Also

term

Examples

Example 38. Shows one reference element with print-url true

<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">

<title><text>Sample of
  - attachments
  - attachment
  - title
  - block
  - text</text></title>

<body></body>

<attachments>
  <attachment class="appendix" id="a1" number="15">
    <title><text>Form of notice in Schema files</text></title>
    <block><text>They must be sent regular mail.</text></block>
    <block><text>Standard Schema</text>
    <reference href="http://www.elkera.com" print-url="true">
    </reference>
  </attachment>
</attachments>

</contract>
can be read on the web easily!"</reference>.</text></block>
</attachment>
</attachments>

</contract>

**row**

**Synopsis**

This is a table row.

**Content Model**

```xml
row = element row {  
  entry+,  
  row.attlist  
}  
row.attlist =  
  common.attributes,  
  attribute rowsep { xsd:boolean }?,  
  attribute valign { TableValign }?,  
row.attlist.extensions  
TableValign = "top" | "middle" | "bottom"  
row.attlist.extensions = empty  
row ::=  
  • One or more  
  • entry  
```

**Attributes**

Common Attributes

Additional Attributes:

• rowsep
• valign

**Description**

A row in a table. This element and all other elements contained by `table` are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2].
Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes: Common Attributes

Additional Attributes:

rowsep  This controls the display of the row separator underneath this cell in the table.
valign  The vertical alignment of the table row.

Table 9. (enumeration)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>Align the content at the top of the row.</td>
</tr>
<tr>
<td>middle</td>
<td>Align the content within the row.</td>
</tr>
<tr>
<td>bottom</td>
<td>Align the content at the bottom of the row.</td>
</tr>
</tbody>
</table>

Parents

These elements contain row: thead and tbody.

Children

The following element occur inside row: entry

Example:

Please see table.

signatory

Synopsis

A signatory is a person who signs a contract or document.

Content Model

```xml
signatory = element signatory {
  (signature-line | block)*,
  signatory.attlist
}
```

```xml
signatory.attlist =
  common.attributes,
  signatory.person-record-id.attribute,
  signatory.attlist.extensions
```

```xml
signatory.person-record-id.attribute = attribute person-record-id { xsd:string }?
```

```xml
signatory.attlist.extensions = empty
```
signatory ::=  
  • Zero or more of
    • signature-line
    • block

Attributes
Common Attributes
Additional Attributes
  • person-record-id

Description
This provides the blank line and printed information for a person who signs a contract.

Attributes
Common Attributes
Additional Attributes:
  person-record-id  This allows the signatory to be associated with a person-record element that contains more information about the signatory.

Parents
The elements contain signatory: signatory-record

Children
The following elements occur inside: signatory
signature-line and block

See Also
witness

Examples
Please see party-signature for extensive examples.

Example 39. Shows use of person-record-id

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:xi="http://www.w3.org/2001/XInclude">
<title><text>Showing person-record-id</text></title>
<contract-front>
```
<parties>
  <party>
    <person-record id="AA2">
      <name>John Smith</name>
    </person-record>
  </party>
  and <person-record id="AA3">
    <name>Jaime Velazquez</name>
  </person-record>
</parties>
<body>
  <block><text>Empty Body</text></block>
</body>
<back>
  <party-signature>
    <signatory-group>
      <signatory-record>
        <signatory id="T001" xml:lang="en" person-record-id="AA2">
          <signature-line><text>Mr. John Smith</text><field>___</field></signature-line>
        </signatory>
      </signatory-record>
      <signatory-record>
        <signatory id="T002" xml:lang="en" person-record-id="AA3">
          <signature-line><text>Ms. Jaime Velazquez</text><field>___</field></signature-line>
        </signatory>
      </signatory-record>
    </signatory-group>
  </party-signature>
</back>
</contract>

**signatory-group**

**Synopsis**

**Content Model**

```xml
signatory-group = element signatory-group { (block*, signatory-record+),
  signatory-group.attlist
}

signatory-group.attlist =
  common.attributes,
  signatory-group.align-records.attribute,
  signatory-group.brace.attribute,
  signatory-group.attlist.extensions

signatory-group.align-records.attribute =
  attribute align-records { signatory-group.align-records.values }?

signatory-group.align-records.values = "horizontal" | "vertical"

signatory-group.brace.attribute =
  attribute brace { signatory-group.brace.values }?
```
signatory-group.brace.values = "block-only"

signatory-group.attlist.extensions = empty

signatory-group ::= 
    • Sequence of
      • block
      • One or more signatory-record

Attributes

Common Attributes

Additional attributes:

• align-records (enumeration)
  • "horizontal"
  • "vertical"

Description

This represents a group of related signatures, e.g., several officers for the same corporation or a witness and a signature. These are represented as the signature and witness tags, respectively.

A signatory-group is used where several persons must sign on behalf of a party, such as to attest the fixing of a common seal or for partners in a partnership.

Processing Expectations

The signature information will be aligned across the page if align-records is horizontal or vertically if specified as vertical. A brace or thick line will be to the right of the signatures if the brace is specified.

Attributes

Common Attributes Additional Attributes:

• align-records. This specifies how signatory-records are aligned with the signatory-group. This attribute can have the following values:

  Table 10. (enumeration)

  | "horizontal" | The information for this signature group SHOULD be layed out across the page. |
  | "vertical"   | The information for this signature group SHOULD be layed out underneath each other. |

• brace This specifies that a brace (or thick column separator) is rendered to the right of the content in the left column. This is ignored when the layout attribute for party-signature is right-column-only.
Table 11. (enumeration)

| "block-only" | A brace or thick column separator is rendered to the right of the content in the left column. |

Parents

These elements contain `signatory-group:party-signature`.

Children

The following elements occur inside `signatory-group:block` and `signatory-record`

See Also

`signatory-record`

Examples

See `party-signature`

`signatory-record`

Synopsis

Content Model

```
signatory-record = element signatory-record { 
  (block*, 
   (signatory-record | 
    ((signatory+, witness*) | (witness+, signatory)) ), 
   block*), 
  signatory-record.attlist }

signatory-record.attlist = 
  common.attributes, 
  signatory-record.align-signatory-witness.attribute, 
  signatory-record.brace.attribute, 
  signatory-record.attlist.extensions

signatory-record.align-signatory-witness.attribute = 
  attribute align-signatory-witness { align-signatory-witness.values }?

align-signatory-witness.values = "horizontal" | "vertical"

signatory-record.brace.attribute = 
  attribute brace { brace.values }?
```
brace.values = "block-only"

signatory-record.attlist.extensions = empty

signatory-record ::= 
  • Sequence of
    • block
  • Choice of
    • signatory-record
  • Choice of
    • Sequence of
      • One or more signatory
      • Zero or more witness
  • Sequence of
    • One or more witness
    • One signatory
  • Zero or more block

Attributes

Common Attributes

Additional attributes:
  • align-signatory-witness (enumeration)
    • "horizontal"
    • "vertical"
  • brace (enumeration)
    • "block-only"

Additional Constraints

brace is ignored when contained the enclosing party-signature specified layout as right-column-only or if enclosed within a signatory-group element.
Description

This is used to bind one or more signatories as well as a possible witness. Its content can be in three forms:

- A single signatory-record. This is used for sub-delegations.
- One or more signatory elements. These may be followed by witness. The multiple signatory records reflect several people signing on behalf of one party.
- One or more witness records followed by a signatory record. (See the third example under party-signature.)

The signatory-record can be used recursively or as a flat model. The recursive model allows nested signature-record elements for complex signatures with sub-delegations. This is used in US contracts.

Processing Expectations

If one has signatory-record nested, then the inner one, representing a subdelegation, would be nested. See the last screen shot in party-signature.

If a align-signatory-witness is horizontal the information in the signatory-record is laid out horizontally. If vertical, then the information is laid out one under another. Please look at the second and third example under party-signature.

If brace is block-only, then there will be a brace or thick column separator to the right of the left column’s content. See the second to last example under party-signature.

Attributes

Common Attributes

Additional Attributes:

align-signatory-witness

This indicates whether the witness and signatories arranged by row or vertically within each defined by the signatory-record element. This attribute can have the following values:

Table 12. (enumeration)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal</td>
<td>The signature information is laid out across the page.</td>
</tr>
<tr>
<td>vertical</td>
<td>The signature information is laid out vertically, with one item underneath another</td>
</tr>
</tbody>
</table>

brace

This specifies that a brace (or thick column separator) is rendered to the right of the content in the left column. This attribute is ignored when the layout attribute for a party-signature is right-column-only.

Table 13. (enumeration)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;horizontal&quot;</td>
<td>The signature information is laid out across the page.</td>
</tr>
<tr>
<td>&quot;vertical&quot;</td>
<td>The signature information is laid out with one item underneath another.</td>
</tr>
</tbody>
</table>
Parents
These elements contain `signatory-record`, `party-signature`, `signatory-group`, and `signatory-record`.

Children
The following elements occur inside `signatory-record`: `block`, `signatory`, `signatory-record` and `witness`.

See Also
`signatory`

Examples
See `party-signature`

signature-line

signature-line creates the line on which a pen-and-ink signature is applied.

Synopsis

Content Model

```plaintext
signature-line = element signature-line {
    (\text* & field )*,
    signature-line.attlist
}
```

```plaintext
signature-line.attlist =
    common.attributes,
    signature-line.attlist.extensions
```

```plaintext
signature-line.attlist.extensions = empty
```

signature-line ::

- Zero or more interleave of
  - `text`
  - `field`

Attributes

Common Attributes
Description
This creates the line on which a pen-and-ink signature may be written on a printed document.

Attributes
Common Attributes

Parents
These elements contain signature-line: entry signatory and witness

Children
The following elements occur inside signature-line: text and field

Examples
Please see party-signature for extensive examples.

statutory-em

Synopsis

Content Model
statutory-em has a mixed content model.

statutory-em = element statutory-em {
inline.content,
      statutory-em.attlist
}
inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
inline.content.inner |= conditional
statutory-em.attlist =
      common.attributes,
      statutory-em.class.attribute,
      statutory-em.attlist.extensions
statutory-em.class.attribute = standard.class
statutory-em.attlist.extensions = empty

statutory-em ::= 
  • Zero or more of
    • conditional
    • em
Attributes

Common Attributes and Class Attributes

Description

This is used to mark up content that must be emphasized as per a particular statute. Presumably, the application program will render the contract emphasizing the text as required by that statute. The TC recognized this need in [Min0216].

Attributes

Common Attributes and Class Attributes

Parents

These elements contain statutory-em: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi: fallback

Children

The following elements occur inside: statutory-em:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also

em strike

Examples

Example 40. Statutory-em Example

<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:xi="http://www.w3.org/2001/XInclude">
   <title><text>Sample of inline elements</text></title>
   <body>
   <block>
The location to deliver the item has to be one of designated terminals. will be any places.

strike

Synopsis

Content Model

strike has a mixed content model.

strike = element strike {
    inline.content,
    strike.attlist
}

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional

strike.attlist =
    common.attributes,
    strike.attlist.extensions

strike.attlist.extensions = empty

strike ::= 
    Zero or more of
    conditional
    em
    field
    reference
    statutory-em
    strike
    sub
    sup
Attributes

Common Attributes and Class Attributes

Description
This is used to markup text that is to be struck through. This is commonly used to indicate changes between versions of documents.

Attributes
Common Attributes and Class Attributes

Parents
These elements contain strike: text

Children
The following elements occur inside strike:
conditional, em, field, reference, statutory-em, strike, sub, sup

See Also
sub sup

Examples

Example 41. Example of the strike element

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Sample of inline elements</text></title>
    <body>
        <block>
            <text>The location to deliver <em>the item</em> <strike>has to be one of <statutory-em>designated terminals.</statutory-em></strike> will be any places.</text>
        </block>
    </body>
</contract>
```
sub

Synopsis

Content Model

sub has a mixed content model:

sub = element sub {  
    inline.content,  
    sub.attlist  
}

inline.content = (inline.content.inner)*  
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional  
sub.attlist =  
    common.attributes,  
    sub.attlist.extensions

sub.attlist.extensions = empty

sub ::=  
• Zero or more of  
  • conditional  
  • em  
  • field  
  • reference  
  • statutory-em  
  • strike  
  • sub  
  • sup

Attributes

Common Attributes and Class Attributes

Description

This is used to mark up content to be subscripted.
Attributes

Common Attributes and Class Attributes

Parents

These elements contain sub: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside sub:

conditional, em, field, reference, statutory-em, strike, sub, sup

See Also

sup strike

Examples

Example 42. Example of sub and sup elements

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:xi="http://www.w3.org/2001/XInclude">

  <title><text>Sample of sub and sup</text></title>

  <body>
    <block>
      <text>The Elkera<sup>3</sup> Business Narrative Markup Language (BNML<sup>1</sup>)</text>
    </block>
    <block>
      <text>A<sub>10</sub></text>
    </block>
  </body>
</contract>
```

subtitle

Synopsis

Content Model

subtitle has a mixed content model.

subtitle = element subtitle {
inline.content,
subtitle.attlist
}
inline.content = (inline.content.inner) *
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

inline.content.inner |= conditional
subtitle.attlist =
  common.attributes,
  subtitle.attlist.extensions
subtitle.attlist.extensions = empty
subtitle ::= 
  • Zero or More of
    • conditional
    • citation
    • em
    • field
    • reference
    • statutory-em
    • strike
    • sub
    • sup

Attributes
Common Attributes

Description
This contains a secondary title for the parent structure. The subtitle is usually intended to further explain the title.

Attributes
Common Attributes

Parents
These elements contain subtitle:attachment and contract

Children
The following elements occur inside subtitle:text
See Also:

title

Examples

Example 43. Example of a subtitle element

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Subtitle Example </text></title>
    <subtitle>Here is a <em>subtitle</em> for this subcontract
eContracts specification draft</subtitle>
</contract>
```

sup

Synopsis

Content Model

sup has a mixed content model.

```xml
sup = element sup {
    inline.content,
    sup.attlist
}
```
inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

    inline.content.inner |= conditional

sup.attlist =
    common.attributes,
    sup.attlist.extensions

sup.attlist.extensions = empty

sup ::= 
    • Zero or More of
        • conditional
        • em
        • field
        • reference
        • statutory-em
        • strike
        • sub
        • sup

Attributes

Common Attributes and Class Attributes

Description

This is used to markup text that is to be superscripted.

Attributes

Common Attributes and Class Attributes

Parents

These elements contain sup: address, citation, conditional, date, date-block, em, fallback, name, note-in-line, phrase, reference, statutory-em, strike, sub, subtitle, sup, term, terms, text and xi:fallback

Children

The following elements occur inside sup:

conditional, em, field, reference, statutory-em, strike, sub, sup
See Also
sub strike

Examples

Example 44. Example showing the sub and sup elements

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/">
  <title><text>Sample of sub and sup</text></title>
  <body>
    <block>
      <text>The Elkera<sup>3</sup> Business Narrative Markup Language (BNML<sup>1</sup>)</text>
    </block>
    <block>
      <text>A<sub>10</sub></text>
    </block>
  </body>
</contract>
```

table

Synopsis

This is a table. This element and all other elements contained by table are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1], [table2]

Content Model

```xml
table = element table {
  title?, tgroup+,
  table.attlist
}
```

table.attlist =
  common.attributes,
  orient.attribute,
  [a:defaultValue = "all"]
  attribute frame { "top" | "bottom" | "topbot" | "all" | "sides" | "none" }?,
  attribute colsep { xsd:boolean }?,
  attribute rowsep { xsd:boolean }?,
  attribute pgwide { xsd:boolean }?,
  table.attlist.extensions
```
**table.attlist.extensions = empty**

table ::=  
  • Sequence of  
    • title  
    • One or more tgroup

**Attributes**

Common Attributes and orient.attribute

Additional Attributes
  • frame  
    • "top"  
    • "bottom"  
    • "topbot"  
    • "all"  
    • "sides"  
    • "none"  
  • colsep  
  • rowsep  
  • pgwide  
  • summary  
  • title

**Description**

This is a table. This element and all other elements contained by `table` are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2].

**Attributes**

Common Attributes and orient.attribute

Additional Attributes: This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes.

frame  
This is to control the table frame, that is the border rendered around the table:
Table 14. (enumeration)

<table>
<thead>
<tr>
<th>top</th>
<th>below title</th>
</tr>
</thead>
<tbody>
<tr>
<td>sides</td>
<td>(left and right sides)</td>
</tr>
<tr>
<td>bottom</td>
<td>after information for last row</td>
</tr>
<tr>
<td>topbot</td>
<td>both top and bottom</td>
</tr>
<tr>
<td>none</td>
<td>none of the above</td>
</tr>
<tr>
<td>all</td>
<td>all of the above</td>
</tr>
</tbody>
</table>

colsep This controls the display of the column separator to the right of the cells in the table.

colsep This controls the display of the row separator underneath the cell in the table.

pgwide This controls whether the table spans the entire page.

summary This is a summary of the table content. This attribute is provided to support the WAI Web Content Accessibility Guidelines.

title This is the table title. This attribute should be used when the title element is not specified (i.e. when the table title is not required in the rendered text of the table). This attribute is provided to support the WAI Web Content Accessibility Guidelines.

Parents

These elements contain table: block

Children

The following elements occur inside table: tgroup and title

See also:

The TC noted the possibility of using markup for table for signatures. However, the TC thought that it would be more preferable to use markup specifically for signatures such as that provided by party-signature and signatory-group.

Example

Example 45. Example showing a table

```xml
c<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">

    <title><text>Sample of
        - table
        - tgroup
    </text></title>

    <body>
        <block>
```
<table>
<group cols="1">
<colspec colnum="1" colname="Cl" colwidth="10" colsep="true"
     rowsep="false" align="justify"/>
<thead>
    <row><entry><block></block></entry></row>
</thead>
<tbody>
    <row><entry><block></block></entry></row>
</tbody>
</group>
</table>
</contract>

tbody

Synopsis

Content Model

tbody = element tbody {
    row+,
    tbody.attlist
}

tbody.attlist =
    common.attributes,
    [a:defaultValue = "middle"]
    attribute valign ( TableValign )?,
    tbody.attlist.extensions

TableValign = "top" | "middle" | "bottom"

tbody.attlist.extensions = empty

tbody ::= 
    • One or more
    • row

Attributes

Common Attributes

Additional Attributes
    • valign
Description

This is the table body. This element and all other elements contained by `table` are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2].

Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes: Common Attributes

`valign` This is the default for all `row` and `entry` within the `tbody` element.

Table 15. (enumeration)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>Align content at the top of the row.</td>
</tr>
<tr>
<td>middle</td>
<td>Center content within the row (default)</td>
</tr>
<tr>
<td>bottom</td>
<td>Align content at the bottom of the row.</td>
</tr>
</tbody>
</table>

Parents

These elements contain `tbody`: `tgroup`

Children

The following elements occur inside `tbody`: `row`

Example:

Please see `table`.

term

Synopsis

Content Model

term has a mixed content model.

term = element term {
    inline.content,
    term.attlist
}

term.attlist =
    common.attributes,
    term.class.attribute,
    term.attlist.extensions

term.class.attribute =     standard.class

term.attlist.extensions =   empty
term ::=  
  • Zero or more of  
    • conditional  
    • em  
    • field  
    • reference  
    • statutory-em  
    • strike  
    • sub  
    • sup

**Attributes**

Common Attributes and Class Attributes  

Additional Attributes  
  • "abbreviation"

**Description**

The term element contains a single term, either at its place of definition, or its place of reference.

**Attributes**

Common Attributes and Class Attributes  

Additional Attributes:

abbreviation This is to provide an abbreviation for a defined term. This attribute is provided to support the WAI Web Content Accessibility Guidelines.

**Parents**

These elements contain date: date-block party person-record terms text

**Children**

The following elements occur inside date:  
conditional, em, field, reference, statutory-em, strike, sub, sup

**See Also**

phrase and field
Examples
Please see terms

terms

Synopsis
Content Model
terms has a mixed content model.

terms = element terms ( (inline.content | term)*,
                   terms.attlist
)

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field

   inline.content.inner |= conditional
terms.attlist =
                   common.attributes,
                   terms.attlist.extensions

terms.attlist.extensions = empty
terms ::= 

• Zero or More of
  • conditional
  • em
  • field
  • reference
  • statutory-em
  • strike
  • sub
  • sup

Attributes
Common Attributes
Description

This is for definitions where the meaning is related to more than a single term. The terms element may also contain text data (#PCDATA) between term elements to allow for punctuation.

Attributes

Common Attributes

Parents

These element contain terms: definition

Children

The following elements occur inside terms:

conditional, em, field, reference, statutory-em, strike, sub, sup term

Examples

Example 46. Examples of term and terms elements

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of
    - terms and -term </text></title>
  <body>
    <block>
      <definition>
        <terms><term>You</term> or <term>Your</term></terms>
        <block>
          <text>means an individual or entity exercising rights under this Licence.</text>
        </block>
      </definition>
    </block>
  </body>
</contract>
```

text

text - contains text and elements that markup parts of text, particularly for formatting purposes.
Synopsis

Content Model

text has a mixed content model.

\text = element text {
  (text.content.inner)*,
  text.attlist
}

text.attlist =
  common.attributes,
  text.class.attribute,
  text.textflow.attribute,
  text.xmlspace.attribute,
  text.attlist.extensions

  text.class.attribute =  standard.class
  text.textflow.attribute =  attribute textflow { "runon" }?

  text.xmlspace.attribute =
    [ a:defaultValue = "default" ]
    attribute xml:space { "default" | "preserve" }?

  text.attlist.extensions =  empty

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
  inline.content.inner |= conditional

text.content = (text.content.inner)*
text.content.inner = inline.content.inner | object | term | phrase | field
  | note | note-in-line | name | address | date | party | person-record

text ::= 
  • Choice of:
    • address
    • conditional
    • date
    • em
    • field
• note
• note-in-line
• object
• party
• person-record
• phrase
• reference
• statutory-em
• strike
• sub
• sup
• term

Attributes
Common Attributes and Class Attributes

Additional attributes:

• textflow (enumeration)
  • "runon"

• xml:space (enumeration)
  • "default"
  • "preserve"

Description

This is a container element for text data (#PCDATA) and other inline elements such as those for emphasis. It is a semantic line element that enables formatting control #PCDATA that occurs between and after block-formatted content.

Attributes
Common Attributes and Class Attributes

Additional Attributes:

• textflow This indicates whether this text element should be rendered in the same line as the previous element. Normally, if a text element occurs after a structure that is formatted as a block, there will be a line break. If this attribute is runon, then it will be on the same line.
### Table 16. (enumeration)

<table>
<thead>
<tr>
<th>&quot;runon&quot;</th>
<th>If this text follows a structure that is formatted as a block, the enclosed characters will appear on the same line as the text from the information from block.</th>
</tr>
</thead>
</table>

- **xml:space** This attribute is used to preserve white-space within an element. This attribute is defined by the XML specification. Please refer to that specification for processing semantics.

### Table 17. (enumeration)

<table>
<thead>
<tr>
<th>&quot;default&quot;</th>
<th>Use application default</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;preserve&quot;</td>
<td>Preserve white space</td>
</tr>
</tbody>
</table>

### Parents

These elements contain text: address, block, citation Conditional date date-block, em, fallback name note-in-line party, person-record, reference sub, signature-line, statutory-em sup, sub, sup, subtitle, strike, term terms, and title

### Children

The following elements occur inside text: address, date, em, field, note, note-in-line, object, party, person-record, phrase, reference, statutory-em, strike, sub, sup, and term

### See Also

term, note-in-line

### Examples

The two examples below show the text in its most typical home, a block element as well as some of the in-line content such as em and sub that it often contains:

#### Example 47. An example showing text element

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of sub and sup</text></title>
  <body>
    <block>
      <text>The Elkera<sup>3</sup> Business Narrative Markup Language (BNML<sup>1</sup>)</text>
    </block>
  </body>
</contract>
```
Example 48. Another example showing text element including inline elements

```xml
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Sample of inline elements</text></title>
    <body>
        <block>
            <text>The location to deliver <em>the item</em> has to be one of <statutory-em>designated terminals.</statutory-em> will be any places.</text>
        </block>
    </body>
</contract>
```

tgroup

Synopsis

This is a set of rows and possibly a header from a table.

Content Model

tgroup = element tgroup {
    colspec*, thead?, tbody,
    tgroup.attlist
}

tgroup.attlist =
    common.attributes,
    attribute cols     { xsd:NMTOKEN },
    [a:defaultValue = "1"]
    attribute colsep   { xsd:boolean }?,
    [a:defaultValue = "1"]
    attribute rowsep   { xsd:boolean }?,
    [a:defaultValue = "left"]
    attribute align    { TableAlign }?,
tgroup.attlist.extensions

TableAlign = "left" | "right" | "center" | "justify"
tgroup.attlist.extensions = empty

tgroup ::= 
  - Sequence of
    - Zero or more colspec
    - Zero or one thead
    - Exactly one tbody

Attributes
Common Attributes

Additional Attributes:
- cols
- colsep
- rowsep
- align

Description
This is the table group. This element and all other elements contained by table are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2].

Attributes
Common Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes.

Additional Attributes:
align The horizontal alignment for content contained in the cell.

Table 18. (enumeration)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>align to the left (default)</td>
</tr>
<tr>
<td>right</td>
<td>align to the right (default)</td>
</tr>
<tr>
<td>center</td>
<td>center the text</td>
</tr>
<tr>
<td>justify</td>
<td>justify the text</td>
</tr>
</tbody>
</table>

cols Number of columns in the tgroup
colsep This controls the display of the column separator to the right of the cells within the tgroup.
rowsep  This controls the display of this row separator underneath the cells in this tgroup.

Parents
These elements occur inside tgroup: table

Children
The following elements occur inside tgroup: colspec, thead and tbody

Example:
Please see table.

thead

Synopsis
This is the table header.

Content Model

thead = element thead {
    row+,
    thead.attlist
}

thead.attlist =
    common.attributes,
    [a:defaultValue = "middle"]
    attribute valign { TableValign }?,
    thead.attlist.extensions

TableValign = "top" | "middle" | "bottom"

thead.attlist.extensions = empty

thead ::= 

- One or more
  - row

Attributes
Common Attributes

Additional Attributes

- valign
Description

This is the table header. This element and all other elements contained by table are taken from the OASIS Exchange Table Model. Please refer to that specification for full details of this element. [table1],[table2].

Attributes

Common Attributes

This element only provides brief description of each attribute's purpose. Please refer to the OASIS Exchange Table Model for details and processing semantics of these attributes.

valign The vertical alignment for all row and entry within the thead element.

Table 19. (enumeration)

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>Align content at the top of cell.</td>
</tr>
<tr>
<td>middle</td>
<td>Centered content within the cell.</td>
</tr>
<tr>
<td>bottom</td>
<td>Align content at the bottom of the cell.</td>
</tr>
</tbody>
</table>

Parents

These elements contain thead: tgroup

Children

The following elements occur inside thead: row

Example:

Please see table.

title

Synopsis

Content Model

title = element title {  
  \text+,  
  title.attlist  
}  

title.attlist =  
  common.attributes,  
  title.attlist.extensions  

title.attlist.extensions = empty  
title ::=  
  • Zero or more occurrences of
Attributes

Common Attributes

Description

This is used to indicate the title for the contract itself, attachments, elements in lists, and many other elements, as listed below.

Parents

These elements contain title: attachment background back.body contract inclusion item block.item parties table

Children

The following elements occur inside title: text

See Also:

subtitle

Examples

Example 49. To illustrate a title in an otherwise minimal contract.

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <title><text>Minimum XML sample</text></title>
    <body>
    </body>
</contract>
```

witness

Synopsis

witness (a person who witnesses the signing of a contract or document)

Content Model

```xml
code
witness = element witness {  
    (signature-line | block)*,  
    witness.attlist
}  
```
witness.attlist =
    common.attributes,
    witness.attlist.extensions

witness.attlist.extensions = empty

witness ::= 
  • Zero or more of
    • signature-line
    • block

Attributes

Common Attributes

Description

This is the person who witnesses the signing of the document.

Attributes

Common Attributes

Parents

These elements contain witness: signatory-record

Children

The following elements appear inside witness: signature-line and block.

See Also

signatory

Examples

Please see party-signature

xi:fallback

Synopsis

Content Model

xi:fallback has a mixed content model.

    xiInclude = element xi:include {
        element xi:fallback {
            (xiInclude.fallback.model)*
        }
    }
xiInclude.fallback.model = xiInclude | inline.content.inner
xiInclude.extensions = empty

inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
      inline.content.inner |= conditional

xi:fallback ::= 
  • Zero or more of
    • Conditional
    • em
    • field
    • reference
    • statutory-em
    • strike
    • sub
    • sup
    • text
    • xi:include

Attributes
None.

Description
This is xi:include fallback text. This element and all other elements contained by xi:include are taken from the World Wide ConsortiumXML Inclusions recommendation. Please refer to that specification for full details of this element.

Parents
These elements contain xi:fallback:xi:include
Children

The following elements occur inside xi:fallback:

conditional, em, field, reference, statutory-em, strike, sub, sup xi:include

See Also:

Please do not confuse this with fallback
fallback xi:fallback can only be used inside xi:include

Example

Please see the example for xi:include

xi:include

Synopsis

This allow including or bringing in content from other files. It uses the World Wide Web Consortium XML Inclusions recommendation:


Content Model

```xml
xiInclude = element xi:include {
    element xi:fallback {
        (xiInclude.fallback.model)*
    }?,
    attribute href { xsd:anyURI }?,
    xiInclude.extensions
}
```

```xml
inline.content = (inline.content.inner)*
inline.content.inner = text | reference | em | statutory-em | strike | sub | sup | field
```

```xml
inline.content.inner |= conditional
xiInclude.fallback.model = xiInclude | inline.content.inner
xiInclude.extensions = empty
```

```xml
xi:include::=
```

- Zero or one xi:fallback
Attributes
href

Description
As mentioned above, this is taken from other files. It uses the World Wide Web Consortium XML Inclusions recommendation:

http://www.w3.org/TR/2004/PR-xinclude-20040930

It specifies an external reference from which to load the content.

A sample use might be for a law firm that has certain standard boilerplate that is put in every lease. At the appropriate point in the xml text for the lease one would have:

```xml
<xi:include href="boilerplate.xml"/>
```

Currently, only the href attribute is implemented in the schemas. As per the X-Include standard, we provide for an xi:fallback. This gives the information to be included when the indicated file is not located.

Attributes
href This specifies the URI (possibly after escaping characters) from which to load the content.

Parents
xi:include appears inside: xi:fallback attachment back background body entry inclusion item regular

Children
The following elements occur inside xi:include: xi:fallback

Example:

This will bring in Inventory01.xml. If that is not available, it will bring in Acct.xml. If both of these files are not available, it will bring in the text:

This is the fallback of Acct.xml

It will then include Terminal.xml. Should this file not be available, this will be a fatal error.

Example 50. Example of xInclude element as well as fallback element

```xml
<?xml version="1.0" encoding="utf-8"?>
<contract xmlns="urn:oasis:names:tc:eContracts:1:0"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:xi="http://www.w3.org/2001/XInclude">
  <title><text>Sample of x:include.</text></title>
  <body>
```

150
Contract Scenarios Considered by the eContracts TC

A range of scenarios were presented by TC members to represent their interests in the TC’s work. The main scenarios and the problems sought to be addressed are summarized in these Case summaries.

Case 1 – On-line click through transactions

Buyers of many goods and services must accept contract terms shown online before they can complete their purchase. Many of these contracts have ongoing effect.

The stated problems:

a. Party’s transacting online may not know what contracts they have entered into.

b. They will have great difficulty determining their obligations under those contracts.

c. Consumers cannot easily compare terms offered by different providers.

Case 2 – Extraction of contract management data

Many contracts, such as construction contracts require parties to meet obligations and exercise rights at specified intervals or on the occurrence of events over a long period.

The stated problems:

a. There are frequent disputes over change authorisation in construction contracts and similar transactions where frequent variations occur.

b. It is difficult to ensure all parties have reliable information about upcoming obligations under the contract.

c. It is difficult to extract terms and embedded data values from the contract into content management systems.

d. It is difficult to access the content of external documents that are incorporated into the contract.

e. There is no reliable way to determine the state of contract events, obligations and processes.

f. It is difficult to monitor and analyse performance of parties over extended time periods.
Case 3 – Contract precedent management and contract drafting

Lawyers rely heavily on precedent documents and documents from previous transactions when preparing new documents. It is common for lawyers to use document assembly and variables substitution systems when creating new documents.

The stated problems:

a. Contract documents are created using word processing applications. These documents can’t easily be processed at convenient levels of granularity by automated systems. Content components are not self describing.

b. In addition to legal maintenance, precedent documents must be revised to deal with changing file formats and proprietary processing systems. This adds to the cost of maintenance.

c. The absence of standard storage formats for contract narrative terms and documents means that it is difficult to process these documents outside the creating application. This creates proprietary ties between data and software applications. It frustrates the use of off-the-shelf tools, inhibits automated document creation, information reuse, information extraction and change traceability.

Case 4 – Represent contract semantics for machine processing of contract terms

This case was really another approach to Case 2. Various models have been developed to define contract rights and obligations using formal languages that can be interpreted by computer systems (deontic contract languages). Contracts using deontic contract language could be useful in various contract management contexts from performance monitoring to dispute resolution.

The stated problems:

a. There is no standard deontic contract language that can be used for a wide range of contracts.

b. There is no way to manage the relationship between deontic contract language and the natural language terms.

Case 5 – Define contract terms that would apply to a computer negotiated contract

Contract negotiation is slow and expensive. The inefficiencies inherent in human contract negotiation limit the value of the transaction, particularly where rich parameter sets are involved. A program was developed by a member for computer negotiation of contracts with defined parameters.

The stated problem: There is no standard way to map a given set of negotiated contract parameters to a unique set of contract terms.

Case 6 – Develop a taxonomy for contract terms

Contract users at various stages in the contract life cycle and in various transactions wish to be able to identify contract terms by their legal subject matter.

The stated problem: There is no standard framework for the description of contract terms using a taxonomy or other controlled vocabulary to support contract negotiation, document assembly and contract management.
Case 7 – Management of eCommerce contract terms

Electronic commerce transactions are governed by a master contract. Current standards do not deal with the formation and management of these contract terms.

The stated problems:

a. In electronic commerce transactions there is currently no way to map or validate electronic transactions against their master agreement.

b. There may be no way to automatically determine if there is a master agreement.

c. Human negotiation of bilateral master agreements is too time-consuming.

Scope analysis

Four domains of contract documentation

So as to understand the characteristics of contract documents that are common to the widest range of contract transactions, the TC divided contract documentation into four domains.

Natural language contract precedents domain

Description

In law firms and many other environments, natural language contract terms are prepared and stored as a library of terms or draft contracts that may be incorporated into or used as a starting point for new contracts in many future transactions. These stored terms are commonly called precedents or templates. Often, the incorporation of these terms into new contract documents will involve the use of information retrieval and automated document assembly or document creation systems.

The natural language precedents domain covers a variety of cases considered by the TC:

a. Precedents may provide the starting point for contracts in which the natural language terms are negotiated in detail between the parties.

b. Precedents may be used to generate contract documents with highly standardized transactions where there is little or no negotiation of natural language terms.

c. Precedents may be used to define standard terms for contracts created in on-line business to consumer transactions.

d. Precedents may be used to generate contract documents that result from computer negotiated contracts.

e. Precedents may be required to provide human readable documents for contracts also represented in a deontic contract language.

Characteristics

Key characteristics of documents in the natural language precedents domain include:

a. Contract precedents are usually maintained in back-end systems for long periods of time. During that time, precedent terms often must be updated in response to changes in the law and other factors which processing. In addition, software versions and file formats for processing systems may change. Current models based on word processing
formats impose high long term maintenance costs and are inflexible for use with processing systems other than those used to create them.

b. Persons wanting to use the contract terms need subject information (metadata) about individual terms to facilitate access.

c. It may be necessary to render natural language terms in a wide variety of output formats, including print, word processing formats and HTML.

d. Natural language terms may be stored and processed as whole contract documents or as components that may be assembled into whole contract documents.

e. Enterprises that maintain databases of contract terms and documents usually expend a great deal of effort to provide for the accuracy and consistency of the stored terms because of their central importance in effective service delivery. They represent a major investment by their creators.

**Scope assessment**

Natural language precedents can be maintained and processed into contract documents in all the contract domains identified by the TC. The TC concluded that an XML schema that models natural language contract documents and terms would benefit a wide range of enterprises who maintain contract precedents. The use of a purpose designed XML model to facilitate better long term storage and automated processing of precedent contract documents could be highly advantageous to many of these enterprises.

**Contract drafting and negotiation domain**

**Description**

The contract drafting and negotiation domain involves the day to day preparation of original, contract narrative terms for specific transactions between identified parties, as undertaken by lawyers and others. Once the contract is formed, transactions under these contracts may be completed quickly or over many months or years. In the later case, the contract may govern the rights and obligations of the parties during a complex set of events over a long period.

**Characteristics**

Currently, lawyers and other persons who draft contracts do so with desktop word processing software. The TC concluded, that for the foreseeable future, there is little likelihood that these people will materially change the tools they use. It is likely that word processing tools will begin to use XML file formats such as the Microsoft Office Open XML format (WordprocessingML) or OASIS OpenDocument. These word processing XML formats are presentation based XML formats in which the semantics of the data is mainly represented in styles. Such semantics cannot be validated.

Lawyers and others who draft contract documents rely heavily on precedent documents and terms to provide complex wording and know how.

During the contract negotiation process, the parties may exchange drafts in electronic form. Currently this is done by exchanging word processing documents or PDFs. Based on the TC’s expectation of continued use of word processing software by contract drafters, the TC does not expect this to change.

**Scope assessment**

The TC considered whether its schema should facilitate the exchange of contract data between parties to negotiations. The TC could not identify any basis for the foreseeable future in which the parties are likely to work with and exchange XML documents other than the word processing XML formats, as the use of those formats becomes more prevalent. The TC proposes that the eContracts schema may be used for this purpose but this use is not expected to be commercially significant.
The TC considered whether parties to a contract may use an XML document as the formal record or artifact of the contract terms. The TC concluded that the parties to negotiated contracts are likely to continue to use printed documents and other electronic formats such as PDF for formal records of contract terms. The TC was not given any convincing use case for the contracting parties to use an XML document as the formal record of contract terms outside of electronic business transactions already governed by other standards. Those transactions are outside the scope of the TC’s charter.

The TC considered whether it could develop a standard that would involve adding additional semantics to either or both of the common word processing XML formats. It concluded that the value of this is unclear at this time. Lawyers and others involved in contract drafting have little time or interest in adding markup to their documents. There is no reason to expect them to do so unless it is required by their clients. Even if in particular contexts they will do so, the TC could not decide whether it should work with the Microsoft Office Open XML format or OpenDocument. In the market place, there is likely to be ongoing competition between these XML formats. The TC concluded that the use of a generic structural XML schema for use in the stored precedents domain would provide the greatest flexibility and enable users to transform precedents into any word processing XML format.

The TC considered whether it should attempt to develop a metadata or embedded markup model for use with one or both the word processing XML formats. The TC could not, at this time, identify commercially practicable requirements for a specification covering the contract drafting and negotiation domain.

The TC concluded that this domain is best supported by a specification to facilitate the preparation, maintenance and use of natural language precedents to be used in the preparation of negotiated contracts.

Form contracts domain

Description

This domain covers standard form contract documents in which there is little or no negotiation of contract terms. If negotiation occurs it is only to determine whether particular terms are included. It does not affect the natural language of any particular term. In on-line, business to consumer transactions assent may be manifested via a “clickwrap” mechanism. In cases where the contract document is printed, assent may occur by conduct or by signing a printed document as in many consumer finance and sale of goods transactions.

Characteristics

Form contracts involve the use of standard natural language terms. The only variables are matters such as product description, quantity, price, delivery etc. In either an on-line environment or off-line, the consumer or an agent of the service provider enters transaction variables into a form to provide the additional information required for the complete contract.

Many form contracts are likely to be generated from natural language precedents.

Scope assessment

The TC did not identify business requirements specific to this domain. It concluded that this domain is best supported by a specification to facilitate the management of natural language precedents that can be used to generate form contracts in appropriate cases where there is potential to improve the presentation and management of contract terms for specific contracts.
Contract management domain

Description

This domain covers the use of contract documentation after assent. Information may be derived from the contract documents to support contract management activities and dispute resolution. In this domain, contract documentation could exist in natural language, deontic contract language or in both forms.

Characteristics

Currently, information required by contract management systems must be manually extracted from the contract documentation and entered into a database system. Alternatively, it is common in many standardized transactions, that transaction data is held in a database and used to generate standard form contract documents in the form contract domain. In those cases, there is no need to extract this data from a contract document.

A deontic contract language could be XML-based. Such an XML document would be as the means to communicate contract terms to a contract management system. It is likely that only highly standardized contracts would be prepared in a deontic contract language. The TC concluded that it is likely that the natural language contract documents and deontic contract language documents would be separate.

Scope assessment

The TC considered that contract management information could be extracted from information suitably defined by XML markup in natural language contract documents or from contracts expressed in deontic contract language. However, for this to be effective, authoritative contract documents must be prepared in XML. The TC could not identify any meaningful use case for the use XML in the formal natural language contract document. An XML document may be used to generate the human readable rendition (paper, PDF etc) that is adopted as the formal record of contract terms.

The TC concluded that it could only address the contract management domain if it is likely that natural language contract documents will be available in a suitable XML format. For the reasons considered in connection with the contract drafting and negotiation domain, the TC considered that XML documents are not likely to be used as the formal record of contract terms, except in very specific situations.

In negotiated contracts the TC did not identify any change from current practice where a print rendition is likely to be the formal contract artifact. The XML document from which that contract is generated may not contain all contract terms. Some terms may be altered by hand on the printed document before signature.

Even in highly standardized transactions, the TC was not satisfied on the information currently available that embedding markup for contract management purposes in natural language contract documents was likely to be of practical benefit. It is just as likely that contract management information will be maintained separately from the contract document. This might be achieved using a separate XML representation to facilitate communication between contract management database systems.

The TC decided it could not develop a deontic contracts language at this time due to the absence of involvement of commercial interests in such a language.

The TC concluded that a specification covering stored precedents could provide benefits for the contract management domain for use with natural language contracts. It could provide the semantics to permit extraction of data values and other semantic information defined by the parties, if they so choose.

Feature summary for the eContracts Reference Schema

Features of the eContracts Reference Schema can be summarized as follows:
a. Contract documents are composed of paragraphs and clauses that may be stored separately and reused in multiple
documents. The eContracts schema defines these objects as containers that can be processed as distinct objects
or content chunks for storage and retrieval in document assembly and other processing systems. The eContracts
schema uses the XInclude standard to support content sharing and reuse of these components.

b. The TC has aimed for simplicity with the eContracts Core Schema. It defines only 51 elements. Most content can
be created with just a handful of elements: item, title, block and text with the item element used recursively. This is intended to make it easy to convert existing content to the eContracts schema and permit content components to be inserted without re-tagging at any desired level of the document hierarchy in document assembly applications.

c. The eContracts Core Schema defines the generic, hierarchical structure of contract documents. This provides the
maximum flexibility for content reuse, reliable automated processing and transformation of eContracts XML into
other formats.

d. The eContracts Core Schema provides for reuse of item elements using the XInclude specification that is supported
by available processing tools.

e. The eContracts Core Schema provides for embedded data values to support variables substitution in contract
preparation and the extraction of data values from XML contract documents.

f. The eContracts Reference Schema provides a mechanism to support conditional processing of content at the element
level and within text contexts. This model is not part of the eContracts Core Schema to allow users to adopt their
own conditional text processing model.

g. The eContracts Core Schema provides a model for users to add metadata at the contract and clause level. The
schema makes provision for common metadata fields required by document management, document assembly
and publishing applications such as:
    i. document identifiers, the author, version and dates;
    ii. the legal subject matter or categorisation of distinct content objects.

h. The schema provides sufficient definition of content objects in contract documents that user applications can:
    i. define and apply automatic numbering schemes to those objects;
    ii. generate desired renditions, including but not limited to print, RTF, PDF, HTML and text to speech ready
        formats.

i. The eContracts schema allows for an entry of various kinds of values into instances of contract documents based
on that schema, e.g. dates specifying start and end dates of the contracts into their respective slots. This is achieved
through the field of the Schema, which can be regarded as a slot for entry of specific values into contract in-
stances, either by the user or by a back-end system. This version of schema supports only string type for the field
element and it is anticipated that future schema versions would require richer data types, according to specific
business requirements.

Law firms and other enterprises who commonly create contract documents commonly create other document types
such as advices, correspondence and litigation documents. Many of these documents are prepared in essentially the
same way as contract documents, using the same automation tools. All the principles applicable to a schema for contract
documents apply to these other documents. The TC concluded that the eContracts schema will define common content
objects that can be adopted by another standards body responsible for developing a schema applicable to those other
legal documents.
Acknowledgments

The following members of the committee have participated in the creation of this specification:

- Rolly Chambers, Smith, Curry, and Hancock
- Daniel Greenwood, Electronic Commerce Architecture Project, Massachusetts Institute of Technology
- Dr. Laurence Leff, Western Illinois University
- Mr. Dave Marvit, Fujitsu Corporation
- Mr. Peter Meyer, Elkera Pty Limited
- Dr. Zoran Milosevic, Deontik Pty Ltd.

The following persons assisted in the writing and review of this specification. Their contributions are gratefully acknowledged:

- Mr. Nobuhiro Sasagawa, Western Illinois University
- Mr. Andrew Squire, Elkera Pty Limited.
- Dr. Hoylen Sue

Contributions by the following persons who were formerly members of the Technical Committee are gratefully acknowledged:

- Mr. Jason Harrop, Exari Systems
- Mr. John McClure, Hypergrove Engineering

The eContracts Schema is derived from an instance of the BNML Schema contributed to OASIS by Elkera Pty Limited (http://www.elkera.com). The contribution of Elkera Pty Limited is gratefully acknowledged.

References

Normative


[XInclude] XML Inclusions (XInclude) Version 1.0 W3C Proposed Recommendation 30 September 2004


Non-normative


eContracts Version 1.0

brrev = legalxml-econtracts


[table1] CALS "XML Exchange Table Model" www.oasis-open.org/specs/soextblx.dtd

