eContracts Version 1.0

Committee Specification
27 April 2007

Document identifier:

legalxml-econtracts-specification-1.0 (XML, HTML, PDF)

Locations:

Persistent version: http://docs.oasis-open.org/legalxml-econtracts/CS01/legalxml-econtracts-specification-1.0.html

Current version: http://docs.oasis-open.org/legalxml-econtracts/legalxml-econtracts-specification-1.0.html

Previous version: http://docs.oasis-open.org/legalxml-econtracts/CD1/legalxml-econtracts-specification-1.0.html

Technical committee:

OASIS LegalXML eContracts TC

Editors:

Laurence Leff, Individual <mflll@wiu.edu>

Peter Meyer, Individual <pmeyer@elkera.com.au>

Abstract:

This is the specification for the OASIS eContracts XML 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 TC envisages that the primary use of the eContracts Schema will be to facilitate the maintenance of precedent or template contract documents and contract terms by persons who wish to use them to create new contract documents with automated tools. Use cases covered include negotiated business contracts, ticket contracts, standard form business and consumer contracts and click-through agreements.

Status:

This document was last revised or approved by the LegalXML eContracts Technical Committee on the above date. The level of approval is also listed above. 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-econtracts/ipr.php).

The non-normative errata page for this specification is located at 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

---

Table of Contents

Appendixes

---

1. Purpose and Scope

1.1. 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."

1.2. 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 formats for use in 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.

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 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.

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.

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 such as enterprise contract management.

1.3. 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 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.
5. 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.
6. 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.
7. 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.
8. 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 busines srequirements.

1.4. 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.

2. 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].

application programmer

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].

contract

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.

contract document

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.).

deontic contract language

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.

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 sytem. 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, at what time is it to be paid and 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.

3. Overview of the eContracts Schema

3.1. 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.

3.2. 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.

3.3. 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.

3.4. 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**
- **http://www.w3.org/2001/XInclude** (xi), World Wide Web Consortium XML Inclusions Recommendation
- This namespace: **urn:oasis:names:tc:eContracts:1:0** (ec)

3.5. 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.

3.6. 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.

4. Basic building blocks in the eContracts Core Schema

4.1. 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 my 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
           Content under third level with title.
1.2     Second level
1.2.1   Content under third level without title.
1.2.2   More content under third level without title.
```

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

```
<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>
<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>
<item number="1.2.2"><block><text>Content under third level without title.</text>
</block></item></item></item>
```

4.2. 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 Schema, 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

```
<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>
<item number="1.2"><title><text>Second level</text></title>
<block><text>This is a two level list:</text>
           <item number="(a)"><block><text>First level list item.</text>
                  <item number="(i)"><block><text>Second level list item.</text>
</block></item>
                  <item number="(ii)"><block><text>Second level list item.</text>
</block></item>
                         </block></item>
```

```
          <item number="(b)"><block><text>First level list item.</text></block>
</item></block></item></item>
```

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`.

4.3. 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.

4.4. 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.

4.5. 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:

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

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:

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

```
Loose structure model
loose.structure.model = (block | inclusion | 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 the eContracts Core Schema.

5. Contract Document Structure in the eContracts Schema

5.1. 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>
```

The following exemplifies the important features in the XML produced for a contract as per this specification. On line 12, observe 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 `block`s followed by zero or more `item`s. A block contains a `text` which acts as a "text container." In additon 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.

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.

Example 5. A Simple Contract

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <contract xmlns="urn:oasis:names:tc:eContracts:1:0"
3:           xmlns:dc="http://purl.org/dc/elements/1.1/"
4:           xmlns:xi="http://www.w3.org/2001/XInclude">
5:
6:   <metadata>
7:     <dc:title>WIU Sample Contract</dc:title>
8:     <dc:creator>Julie Sasa</dc:creator>
9:   </metadata>
10:   <title> <text>Overall Example</text>
11:   </title>
12: <contract-front>
13: <date-block>Agreement dated:
14:     <field class="date" type="blank" name="contract_date" length="75mm">
15:             <?xm-replace_text {ec:field}?></field></date-block>
16: <parties><title><text>Parties</text></title>
17:     <party><person-record><name>ABC Ventures Limited</name> having
18:             its office at <address>100 Main Street, Sydney, NSW 2000</address>
19:            </person-record>, hereafter referred to as
20:                  "<term>General Partner</term>"
21:     </party>
22:     <party><person-record><name>John A. Doe</name> of
23:             <address>10 Ramrod Drive, Sydney, NSW 2000</address></person-record>
24:               and <person-record><name>John W. Smith</name> of
25:           <address>25 Pine Road, Plainsville, NSW, 0000</address>
26:             </person-record>, herafter collectively referred to as
27:                  "Limited Partners"
28:     </party>
29: </parties>
30: </contract-front>
31:
32:
```

```
33:  <body>
34:     <block>
35:       <text>A<sub>10</sub><em>Emphasized Text</em><strike>Struck out</
strike></text>
36:     </block>
37:    <block> </block>
38:    <item>  </item>
39:    <item>  </item>
40:   </body>
41:   <back>
42:
43:     <party-signature>
44:
45:       <signatory-group>
46:
47:         <block> </block>
48:
49:         <signatory-record>
50:
51:           <signatory id="T0001" xml:lang="ja">
52:             <signature-line id="I0001" xml:lang="en_US">
53:               <text>Mr. Signatory Jr.</text>
54:               <field>Field for a text.</field>
55:             </signature-line>
56:           </signatory>
57:
58:           <witness>
59:             <signature-line id="I0002" xml:lang="fr">
60:               <text>Ms. Witness Arcole</text>
61:               <field>Field for a text.</field>
62:             </signature-line>
63:           </witness>
64:
65:         </signatory-record>
66:
67:       </signatory-group>
68:
69:     </party-signature>
70:
71:   </back>
72:   <attachments>
73:     <attachment class="appendix" id="a1">
74:       <title><text>Form of notice in Schema files</text></title>
75:       <block><text>They must be sent regular mail.</text></block>
76:        <item>  </item>
77:        <block><text>Note that one can interleave block and item arbitrarily
78:            when we have a loose model container</text></block>
79:      </attachment>
80:
81:  </attachments>
82:
83: </contract>
```

5.2. Particular features of the eContracts Schema

5.2.1. 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.

5.2.2. 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.

5.2.3. 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.

5.2.4. 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 `item` elements 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`.

disc

The application programmer SHOULD render list items by a disc or bullet.

line

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

number

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

loweralpha

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

upperalpha

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

lowerroman

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

upperroman

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.

5.2.5. 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 in 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 multple values for its `condition` attribute. If so, it will be rendered if any one of these values is true. In this example, the jursidiction is set to `jurisdiction-US`. content for other jurisdictions will not be rendered.

Example 6. Showing the condition 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">

  <metadata>
    <conditions>
       <condition name="jurisdiction-US">Jurisdiction United States</condition>
       </conditions>
  </metadata>
  <title><text>Sample of Conditions</text></title>

  <body>
    <block condition="jurisdiction-US">
      <text>A United States - specific term  in the contract</text>
    </block>
    <block condition="jurisdiction-AU">
      <text>An Australia-specific statement in the contract</text>
    </block>
    <block condition="jurisdiction-AU jurisdiction-US">
      <text> This block will be included if either value is true.</text>
    </block>
  </body>

</contract>
```

5.2.6. Content reuse with XInclude

The eContracts Core Schema permits content sharing and reuse using the `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 `xi:include` links are resolved.

5.2.7. Party signatures

5.2.7.1. 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.

5.2.7.2. Written signatures

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

1. The `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 `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.

6. Invocation of the eContracts Reference Schema

6.1. 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.

6.2. Invocation for the XML Schema

The XML Schema is invoked as follows:

```
<?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.

6.3. Invocation for the DTD

The DTD is invoked as follows:

```
<?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.

7. Customizing the eContracts Core Schema

7.1. 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.

7.2. 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.

7.3. 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.

7.4. 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**.

7.5. 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

element	Model to change
body	body.structure.model
back	back.structure.model
attachment	attachment.structure.model
item	item.structure.model
inclusion	inclusion.structure.model

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:

```
   body.structure.model        = tight.structure.model
   back.structure.model        = tight.structure.model
   attachment.structure.model  = tight.structure.model
   item.structure.model        = tight.structure.model
   inclusion.structure.model   = tight.structure.model
```

7.6. 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 7. Adding attributes and elements to the item element

The above shows how one can change to the tight model to add two attributes, `a` and `b` to the `item` element. These are added to `block.item.attlist.extensions`. Note that `item` is defined in two different places in **eContracts-core.rnc**

In addition, we added the `I1` and `I2` to the `inclusion` element, by updating the `inclusion.class.attribute` and `inclusion.attlist.extensions`, respectively. It also adds the attribute `E` to all elements.

Example 8. Illustrates possible contract document after schema is customized.

```
<?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 agaist Loose model based on Mr. Meyer's email on Aug20.</text>
 </title>

 <body>
   <block> </block>
   <block> </block>
   <block>
   <item b="4"></item>
   <item a="3">
     <block><text><PaySomeone>3212</PaySomeone></text></block>
     <block></block>
   </item>
   <item E="5">
      <inclusion I1="9"></inclusion>
      <inclusion I2="10"></inclusion>
   </item>
    </block>
   <block> </block>
 </body>

</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

element	Label to change
attachment	attachment.numbering.attributes
back	back.numbering.attributes
background	background.numbering.attributes
block	block.numbering.attributes
contract	contract.numbering.attributes
entry	entry.numbering.attributes
inclusion	inclusion.numbering.attributes
item	item.numbering.attributes

7.7. 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:

```
java -jar trang.jar -I rnc -O xsd -o any-process-contents=lax -o
any-attribute-process-contents=lax -o disable-abstract-elements
eContracts-Standard.rnc
eContracts-Standard.xsd
```

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.

A. 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.

1. 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.

2. 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.

3. 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. It also inhibits automated document creation, information reuse, information extraction and change traceability.

4. 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.

5. 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.

6. 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.

7. 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.

8. Scope analysis

8.1. 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.

8.2. Natural language contract precedents domain

8.2.1. 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.

8.2.2. 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. 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.

8.2.3. 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.

8.3. Contract drafting and negotiation domain

8.3.1. 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.

8.3.2. 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.

8.3.3. 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.

8.4. Form contracts domain

8.4.1. 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.

8.4.2. 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.

8.4.3. 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.

8.5. Contract management domain

8.5.1. 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.

8.5.2. 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.

An XML deontic contract language could provide the means to communicate contract terms to a contract management system. 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.

8.5.3. 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 decided it could not develop a deontic contracts langauge at this time due to the absence of involvement of commercial interests in such a language.

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 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.

B. Language Reference

1. 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 }?
```

Name	Type	Default
id	xsd:ID	none
xml:lang	xsd:string	none

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.

2. 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 }?
```

Name	Type	Default
class	xsd:string	none

3. 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 }?
```

Name	Type	Default
condition	xsd:string	none

4. orient.attribute

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

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

Name	Type	Default
orient	xsd:string	

5. 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 }?
```

Name	Type	Default
number	xsd:string	

6. 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" }?
```

Name	Type	Default
stop-contents	xsd:string	below

7. address

7.1. Synopsis

7.1.1. 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

7.1.2. Attributes

Common Attributes and Class Attributes

7.2. 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).

7.2.1. Attributes

Common Attributes and Class Attributes

7.2.2. Parents

These elements contain address: text, person-record

7.2.3. Children

The following elements occur inside address:

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

7.2.4. See Also

name, party and person-record

7.2.5. Examples

Example B.1. Several address elements which together define all parts of an address

```
<?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>

  <body></body>

</contract>
```

8. attachment

8.1. Synopsis

8.1.1. 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 |= xiInclude
```

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`

8.1.2. Attributes

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

8.2. 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 attachemnt element may be changed to one of the other structure models defined in this file as part of a customization.

8.2.1. Attributes

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

8.2.2. Parents

These elements contain `attachment`: `attachments`

8.2.3. Children

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

8.2.4. Examples

Please see `attachments`

9. attachments

9.1. Synopsis

9.1.1. 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`

9.1.2. Attributes

Common Attributes

9.2. 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.

9.2.1. Attributes

Common Attributes

9.2.2. Parents

These elements contain `attachments`: `contract`

9.2.3. Children

The following elements occur inside `attachments`: `attachment`

9.2.4. See Also:

`object` and `back`

9.2.5. Examples:

Example B.2. An attachments with a single attachment element

```
<?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
          <reference href="http://www.elkera.com" print-url="true">
can be read on the web easily!</reference>.</text></block>
    </attachment>
 </attachments>

</contract>
```

10. back

10.1. Synopsis

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

10.1.1. Content model

```
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.numbering.attributes =     empty
back.attlist.extensions =       empty
```

back ::=

- Sequence of
  - One or more 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

10.1.2. Attributes

Common Attributes

10.2. 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 chagned to one of the other structure modes defined in this file as part of a customization.

10.2.1. Attributes

Common Attributes

10.2.2. Parents

These elements contains back: contract

10.2.3. Children

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

10.2.4. Examples

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

Example B.3. 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>
```

11. background

11.1. Synopsis

11.1.1. Content Model

```
background = element background {

    title?, item.reuse.model*,

    background.attlist
}

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


background.attlist =
```

```
    common.attributes,
    backgound.numbering.attributes,
    background.attlist.extensions


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

background ::=

- Sequence of
  - Zero or one `title`
  - Zero or more occurences of choice of:
    - `item`
    - `xi:include`

11.1.2. Attributes

Common Attributes

11.2. Description

This contains recitals and other "background" information.

11.2.1. Attributes

Common Attributes

11.2.2. Parents

`background` appears inside: `contract-front`

11.2.3. Children

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

11.2.4. Examples

Please see `parties`.

12. block

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

12.1. Synopsis

12.1.1. 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
  - `item` inside a `block`

12.1.2. Attributes

Common Attributes common.number.attribute

Additional attributes:

- `number-type`(enumeration)
  - "manual"
  - "none"
  - "disc"
  - "line"
  - "number"
  - "loweralpha"
  - "upperalpha"
  - "lowerroman"
  - "upperroman"

12.1.3. Additional Constraints

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

12.2. 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`.

12.2.1. Attributes

Common Attributes

Additional attributes:

- `number-type` `number-type` indicates the type of numbering or markers to be used on contained list items.

Table B.1. (enumeration)

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

12.2.2. Parents

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

12.2.3. Children

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

12.2.4. Examples

See body.

13. body

13.1. Synopsis

body - The body of the contract

13.1.1. 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 |= xiInclude
```

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

13.1.2. Attributes

Common Attributes

13.2. 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.

13.2.1. Parents

These elements contain body: contract

13.2.2. Children

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

13.2.3. Examples

Example B.4. A valid body under the loose model

```
<?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>
     <block></block>
     <item> </item>
   </item>
   <block> </block>
   <item>  </item>
 </body>

</contract>
```

Example B.5. A valid body under the standard model

```
<?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>
   <block> </block>
   <block> </block>
   <item>  </item>
   <item>  </item>
 </body>

 <back>
   <item>  </item>
   <item>  </item>
   <block> </block>
   <block> </block>
 </back>

</contract>
```

Example B.6. A valid body under the tight model.

```
<?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>
   <item>
     <item> </item>
     <item> </item>
   </item>
 </body>

</contract>
```

14. citation

14.1. Synopsis

14.1.1. 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

14.1.2. Attributes

Common Attributes

14.2. Description

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

14.2.1. Processing Expectations

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

14.2.2. Attributes

Common Attributes

14.2.3. Parents

These elements contain `citation`: `reference`

14.2.4. Children

The following elements occur inside `citation`:

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

14.2.5. Examples

See `reference`

15. colspec

15.1. Synopsis

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

15.1.1. 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

colspec.attlist.extensions = empty

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

colspec is empty.

15.1.2. Attributes

Common Attributes

Additional Attributes:

- align
- colname
- colnum
- colsep
- colwidth
- rowsep

15.2. Description

This is a column specification in a table. This element and all other elements contained by `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 `colspec` for each column of their table. The contract writer MAY provide a name (`colname`), a number (`colnum`) to designate their column as well as formatting information with its remaining attributes.

15.2.1. 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:

Table B.2. (enumeration)

`left`	align to the left (default)
`right`	align to the right
`center`	center the text
`justify`	justify the text

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.

15.2.2. Parents

These elements contain `colspec`: `tgroup`

15.2.3. Children

No elements occur inside `colspec`.

15.2.4. See Also

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

15.2.5. Example

Please see `table`

16. conditional

16.1. Synopsis

16.1.1. Content Model

Conditional has a mixed content model

```
conditional = element conditional {
    inline.content,

    conditional.attlist
}


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


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

conditional.attlist.extensions = empty



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`

16.1.2. Attributes

Common Attributes

16.2. Description

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

16.2.1. Attributes

Common Attributes

16.2.2. 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`

16.2.3. Children

The following elements occur inside: `conditional`

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

16.2.4. Examples

See conditions.

17. condition

17.1. 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.

17.1.1. Content Model

condition has a mixed content model.

```
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

17.1.2. Attributes

- "name"
- "default"

17.2. 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.

17.2.1. 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.

17.2.2. Parents

These elements contain: condition: conditions

17.2.3. Children

No elements occur inside condition.

17.2.4. See Also:

conditional.

17.2.5. Examples

See the two examples under `conditions`

18. conditions

18.1. Synopsis

18.1.1. 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`

18.1.2. Attributes

none

18.2. 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 elment 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 condtional 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 condtions when rendering conditional text. The active condition is specified using the name attribute of the `condition` element. For example,

```
<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 jurisidiction 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:

```
<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.,

18.2.1. Attributes

none

18.2.2. Parents

These elements contain `conditions`: `metadata`

18.2.3. Children

The following elements occur inside `conditions`: condition

18.2.4. See Also

conditional inside a regular inline element.

18.2.5. Examples

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

19. contract

19.1. Synopsis

This is the root element of the eContracts schema.

19.1.1. 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`

19.1.2. Attributes

Common Attributes Class Attributes orient.attribute

19.2. 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.

19.2.1. Parents

These elements may also contain contract: `attachment`

19.2.2. Children

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

19.2.3. Examples

This illustrates the basic structure of a `contract`:

Example B.7. Basic Structure of a 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>
```

Here is an example showing a minimal contract which would consist have only a `title` and a `body`:

Example B.8. A Minimal Contract

```
<?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>
```

20. contract-front

20.1. Synopsis

20.1.1. 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`

20.1.2. Attributes

Common Attributes Class Attributes

20.2. Description

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

20.2.1. Attributes

Common Attributes Class Attributes

20.2.2. Parents

These elements contain `contract-front`: `contract`

20.2.3. Children

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

20.2.4. Examples

Example B.9. A contract-front with a date-block and parties.

```
<?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>
```

Example B.10. 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>
```

21. data

21.1. Synopsis

21.1.1. 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
```

21.1.2. Attributes

Common Attributes

- `height`
- `src`
- `width`

21.2. Description

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

21.2.1. 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.)

21.2.2. Parents

`data` appears inside: `object`

21.2.3. Children

No elements occur inside `data`.

21.2.4. See Also

fallback

21.2.5. Examples

Please see **fallback.XML** under fallback

22. date

22.1. Synopsis

22.1.1. 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

22.1.2. Attributes

Common Attributes and Class Attributes

22.2. 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.

22.2.1. Attributes

Common Attributes Class Attributes

22.2.2. Parents

These elements contain date: date-block and text

22.2.3. Children

The following elements occur inside `date`:

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

22.2.4. See Also

`date-block` `name` `field` `party` `person-record`

22.2.5. Examples

See the first example inside `contract`.

23. date-block

23.1. Synopsis

23.1.1. 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
  - address
  - `date`
  - `em`
  - `field`
  - `name`
  - `note`
  - `note-in-line`
  - `object`
  - `party`
  - `person-record`
  - `phrase`
  - `reference`
  - `statutory-em`
  - `strike`

- sub
- sup
- term

23.1.2. Attributes

Common Attributes

23.2. Description

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

23.2.1. Attributes

Common Attributes

23.2.2. Parents

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

23.2.3. Children

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

23.2.4. See Also

`parties`

23.2.5. Example:

Please see the first example in `contract`.

24. dc:contributor

24.1. Synopsis

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

24.1.1. Content Model

```
            element dc:contributor { xsd:string }* &
```

dc:contributor ::= string

24.1.2. Attributes

This element has no attributes.

24.2. Description

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

24.2.1. Attributes

This element has no attributes.

24.2.2. Parents

These elements contain `dc:contributor`: `metadata`

24.2.3. Children

No elements occur inside `dc:contributor`.

24.2.4. See Also

`dc:creator` `dc:publisher`

24.2.5. Examples

Please see `metadata`

25. dc:creator

25.1. Synopsis

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

25.1.1. Content Model

```
            element dc:creator { xsd:string }* &
```

dc:creator := string

25.1.2. Attributes

This element has no attributes.

25.2. 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.

25.2.1. Attributes

This element has no attributes.

25.2.2. Parents

`dc:creator` appears inside `metadata`

25.2.3. Children

No elements occur inside `dc:creator` .

25.2.4. Examples

Please see `metadata`

26. dc:date

26.1. Synopsis

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

26.1.1. Content Model

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

26.1.2. Attributes

This element has no attributes.

26.2. 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.

26.2.1. Attributes

This element has no attributes.

26.2.2. Parents

These elements contain `dc:date`: `metadata`

26.2.3. Children

No elements occur inside `dc:date` .

26.2.4. Examples

Please see `metadata`

27. dc:description

27.1. Synopsis

A description of the contract.

27.1.1. Content Model

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

dc:description ::= string

27.1.2. Attributes

This element has no attributes.

27.2. 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.

27.2.1. Attributes

This element has no attributes.

27.2.2. Parents

These elements contain `dc:description`: `metadata`

27.2.3. Children

No elements occur inside `dc:description`.

27.2.4. See Also

`dc:subject`

27.2.5. Examples

Please see `metadata`

28. dc:publisher

28.1. Synopsis

The "entity responsible for making the resource available."

28.1.1. Content Model

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

dc:publisher ::= string

28.1.2. Attributes

This element has no attributes.

28.2. 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.

28.2.1. Attributes

This element has no attributes.

28.2.2. Parents

These elements contain `dc:publisher`: `metadata`

28.2.3. Children

No elements occur inside `dc:publisher`.

28.2.4. See Also

`dc:creator`

28.2.5. Examples

Please see `metadata`

29. dc:rights

29.1. Synopsis

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

29.1.1. Content Model

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

dc:rights ::= string

29.1.2. Attributes

This element has no attributes.

29.2. 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.

29.2.1. Attributes

This element has no attributes.

29.2.2. Parents

`dc:rights` appears inside `metadata`

29.2.3. Children

No elements occur inside `dc:rights`.

29.2.4. Examples

Please see `metadata`

30. dc:subject

30.1. Synopsis

The subject matter of the contract.

30.1.1. Content Model

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

30.1.2. Attributes

This element has no attributes.

30.2. 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.

30.2.1. Attributes

This element has no attributes.

30.2.2. Parents

The elements contain `dc:subject`: `metadata`

30.2.3. Children

No elements occur inside `dc:subject`.

30.2.4. See Also

`dc:description`

30.2.5. Examples

Please see `metadata`

31. dc:title

31.1. Synopsis

The title or name of the contract.

31.1.1. Content Model

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

dc:title := string

31.1.2. Attributes

This element has no attributes.

31.2. 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.

31.2.1. Attributes

This element has no attributes.

31.2.2. Parents

These elements contain `dc:title`: `metadata`

31.2.3. Children

No elements occur inside `dc:title` .

31.2.4. Examples

Please see `metadata`

32. definition

32.1. Synopsis

32.1.1. Content Model

```
definition = element definition {
    (term | terms), block+,

    definition.attlist
}

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

definition.class.attribute =    standard.class
definition.attlist.extensions = empty
```

definition ::=

- Sequence of
  - Choice of
    - `term`
    - `terms`
  - One or more `block`

32.1.2. Attributes

Common Attributes Class Attributes

32.2. 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.

32.2.1. Processing Expectations

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

32.2.2. Attributes

Common Attributes Class Attributes

32.2.3. Parents

These elements contain `definition`: `block`.

32.2.4. Children

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

32.2.5. Examples

See `field` and `term`.

33. em

33.1. Synopsis

33.1.1. 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
```

em ::=

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

33.1.2. Attributes

Common Attributes and Class Attributes

33.2. 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.

33.2.1. Attributes

Common Attributes and Class Attributes

33.2.2. Parents

These elements contain `em`: `text`

33.2.3. Children

The following elements occur inside `em`:

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

33.2.4. See Also

`statutory-em` `strike`

33.2.5. Examples

Example B.11. 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>
```

34. entry

34.1. Synopsis

This is a table cell.

34.1.1. 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`

34.1.2. Attributes

Common Attributes

Additional Attributes

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

34.2. 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]

34.2.1. Attributes

Common Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchagne 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 B.3. (enumeration)

`left`	align to the left (default)
`right`	align to the right
`center`	center the text
`justify`	justify the text

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 B.4. (enumeration)

`top`	Align content at the top of the cell
`middle`	Center content within the cell
`bottom`	Align content at the bottom of the cell.

34.2.2. Parents

These elemetns contain `entry`: `row`

34.2.3. Children

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

35. fallback

35.1. Synopsis

35.1.1. 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

    inline.content.inner |= conditional
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

35.1.2. Attributes

Common Attributes

35.2. 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 innermost `fallback` should always be only text data.

35.2.1. Attributes

Common Attributes

35.2.2. Parents

These elements contain `fallback`: `object`.

35.2.3. Children

The following elements occur inside `fallback`:

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

35.2.4. See Also

`data`

35.2.5. Examples

Example B.12. 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" />
          <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>
            </object>
          </fallback>
        </object>

      </text>

    </block>
  </body>
</contract>
```

36. field

36.1. Synopsis

36.1.1. 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)

36.1.2. Attributes

Common Attributes and Class Attributes

Additional attributes:

- `action`
- `label`
- `length`
- `name`
- `source`
- `type`

36.2. 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 application 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.

36.2.1. 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 processnig 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.

36.2.2. 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`

36.2.3. Children

`field` can only contain text data (#PCDATA)

36.2.4. See Also

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

36.2.5. Examples

Example B.13. The Payment Amount as a field.

```
<?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 field</text></title>

  <body>
    <block>
      <text>In this licence:</text>
      <definition>
        <term>BNML Standard Schema</term>
        <block>
          <text>means the XML Schema called
Pay <field source="ida" name="PaymentAmount"></field>.
                .
          </text>
        </block>
      </definition>
    </block>
  </body>
</contract>
```

37. inclusion

37.1. Synopsis

37.1.1. 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`
  - Zero or more `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`

37.1.2. Attributes

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

Additional Attributes:

- align

37.2. 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.

37.2.1. Processing Expectations

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

37.2.2. Attributes

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

Additional Attributes:

align

This is the horizontal alighment for content within the inclusion as given by `ShortAlignment` pattern:

Table B.5. (enumeration)

left	The content of the `inclusion` is left aligned.
center	The content of the `inclusion` is centered.
right	The content of the `inclusion` is right-aligned.

37.2.3. Parents

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

37.2.4. Children

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

37.2.5. See Also:

object

37.2.6. Examples:

Example B.14. Two inclusion 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
               - item
               - num
               - block
               - inclusion
               - phrase
         </text></title>

  <body>

    <item number="2.1">

      <block number-type="none">

        <text>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:</text>

        <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>

      </block>

    </item>

  </body>

</contract>
```

38. item (outside of a block)

38.1. 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."

38.1.1. Content Model

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


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


## 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

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


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

item ::=

- Sequence of:
  - Zero or one `metadata`
  - 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`

Zero or one `title`

38.1.2. Attributes

Common Attributes Class Attributes condition.attribute Stop-Contents Attribute

38.2. Description

The `item` element is the basic building block of the contract hierarchy. It represents structures that may have a titile 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 maches any of that list so as to avoid confusion or biassing 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.

38.2.1. 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.

38.2.2. Attributes

Common Attributes Class Attributes condition.attribute Stop-Contents Attribute

38.2.3. Parents

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

38.2.4. Children

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

39. item (inside a block)

39.1. Synopsis

This is the list item element.

39.1.1. 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


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`

39.1.2. Attributes

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

39.2. 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.

39.2.1. Process Expectations

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

39.2.2. Attributes

Common Attributes, Class Attributes and condition.attribute

39.2.3. Parents

These elements contain a list `item`: `block`

39.2.4. Children

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

39.2.5. See Also

item (outside of a block)

39.2.6. Examples

See `body`.

40. metadata

40.1. Synopsis

40.1.1. 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 }?
```

```
            }+
        }?

}


    # 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

40.1.2. Attributes

Common Attributes

40.2. 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.

40.2.1. 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.

40.2.2. Attributes

Common Attributes

40.2.3. Parents

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

40.2.4. Children

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

40.2.5. See Also:

title

40.2.6. Example

Example B.15. A metadata element showing several elements from the Dublin Core

```
<?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>
```

41. name

41.1. Synopsis

41.1.1. 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

41.1.2. Attributes

Common Attributes Class Attributes

41.2. 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.

41.2.1. Attributes

Common Attributes Class Attributes

41.2.2. Parents

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

41.2.3. Children

The following elements occur inside name:

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

41.2.4. See Also:

address name

41.2.5. Examples

See address and parties

42. note

42.1. Synopsis

This is a footnote or endnote.

42.1.1. 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

42.1.2. Attributes

Common Attributes , common.number.attribute and Class Attributes

42.2. 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.

42.2.1. Attributes

Common Attributes and Class Attributes

42.2.2. Parents

These elements contain note: date-block and text

42.2.3. Children

The following elements occur inside note: block

42.2.4. See Also

note-in-line

42.2.5. Example

Example B.16. 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.
             <note-in-line>standard size.</note-in-line></text>
    </block>

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

  </body>

</contract>
```

43. note-in-line

43.1. Synopsis

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

43.1.1. Content Model

`note-in-line` has a mixed content model.

```
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`
  - `statutory-em`
  - `strike`
  - `sub`
  - `sup`

43.1.2. Attributes

Common Attributes and Class Attributes

43.2. Description

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

43.2.1. Processing Expectations

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

43.2.2. Attributes

Common Attributes Class Attributes

43.2.3. 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`

43.2.4. Children

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

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

43.2.5. See Also

`note` `phrase` `reference`

43.2.6. Examples

Example B.17. 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.
             <note-in-line>standard size.</note-in-line></text>
    </block>

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

  </body>

</contract>
```

44. object

44.1. Synopsis

44.1.1. 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

44.1.2. Attributes

Common Attributes and Class Attributes

Additional Attributes:

- type
- scale
- rotate

44.2. 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.

44.2.1. 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.

44.2.2. 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]

44.2.3. 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

44.2.4. Children

The following elements occur inside object: data and fallback

44.2.5. Example

Please see fallback.

45. parties

45.1. Synopsis

The parties element contains the parties to the contract.

45.1.1. 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`

45.1.2. Attributes

Common Attributes and Class Attributes

45.2. 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)

[crc], [harrop03], [oxford].

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*:

name and address

and the *short-name*:

name and address[aia]

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.

45.2.1. 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

45.2.2. Attributes

Common Attributes and Class Attributes

45.2.3. Parents

These elements contain `parties`: `contract-front`

45.2.4. Children

The following elements occur inside `parties`: `title` and `party`

45.2.5. See Also

`party`

45.2.6. Examples

Example B.18. 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></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>
</parties>
<background><title><text>Background</text></title>
    <item number="A">
      <block><text>This is the text of the first recital</text></block>
    </item>
    <item number="B">
      <block><text>Text of the second recital</text></block>
    </item>
</background>

</contract-front>

<body></body>

</contract>
```

Remember that this does NOT match the picture above.

46. party

46.1. Synopsis

This is a party to the contract.

46.1.1. Content Model

`party` has a mixed content model.

```
party = element party {

    (text | person-record | term)*,

    party.attlist
}


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


party.class.attribute =    standard.class
party.attlist.extensions = empty
```

party ::=

- Zero or More of
  - `text`
  - `person-record`
  - `term`

46.1.2. Attributes

Common Attributes and Class Attributes

46.2. 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.

46.2.1. Attributes

Common Attributes Class Attributes

46.2.2. Parents

These elements contain `party`: `date-block` `parties` `text`

46.2.3. Children

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

46.2.4. See Also

`name` `address`

46.2.5. Example

See `parties`

47. party-signature

47.1. Synopsis

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

47.1.1. 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"

party-signature.party-id.attribute = attribute party-id { xsd:string }?
party-signature.attlist.extensions = empty
```

party-signature ::=

- Sequence of
  - Zero or more `block`
  - One of
    - signatory-group
    - signatory-record
  - Zero or more `block`

47.1.2. Attributes

Common Attributes

Additional attributes:

- `layout` (enumeration)
  - "from-left"
  - "right-column-only"
- `party-id`

47.2. Description

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

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.

47.2.1. 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.

47.2.2. Attributes

Common Attributes

layout

Table B.6. (enumeration)

right-column-only	The page is divided into two columns. Render the signature in the right column.
from-left	render signatures from the left page margin.
party-id	

47.2.3. Parents

These elements contain `party-signature`: `back`

47.2.4. Children

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

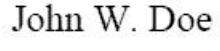
47.2.5. Examples

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

Example B.19. A single signer plus a 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>
```

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`



Example B.20. 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 B.21. 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>
   <text>Partner</text></signature-line></signatory>
</signatory-record><signatory-record><signatory><signature-line>
<field type="blank" length="50"/>
<text>John W.  Smith</text><text>Partner</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:

.....
Witness

.....
John W. Smith

Example B.22. A signatory and a witness

```
<party-signature><block><text>Signed by John W. Smith in the presence
of:</text></block>
<signatory-record align-signatory-witness="horizontal">
<witness><signature-line><field type="dotleader" length="50"/>
<text>Witness</text></signature-line></witness><signatory>
<signature-line><field type="dotleader" length="50"/><text>John W.
Smith</text></signature-line></signatory></signatory-record></party-signature>
```

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:

.....
John W. Smith

.....
Witness

Example B.23. Illustration of align-signatory-witness

```
<party-signature><block><text>Signed by John W.
Smith:</text></block><signatory-record align-signatory-witness="vertical">
<signatory><signature-line><field type="dotleader" length="50"/>
<text>John W.  Smith</text></signature-line></signatory>
<witness><signature-line><field type="dotleader" length="50"/>
   <text>Witness</text></signature-line></witness>
</signatory-record></party-signature>
```

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:

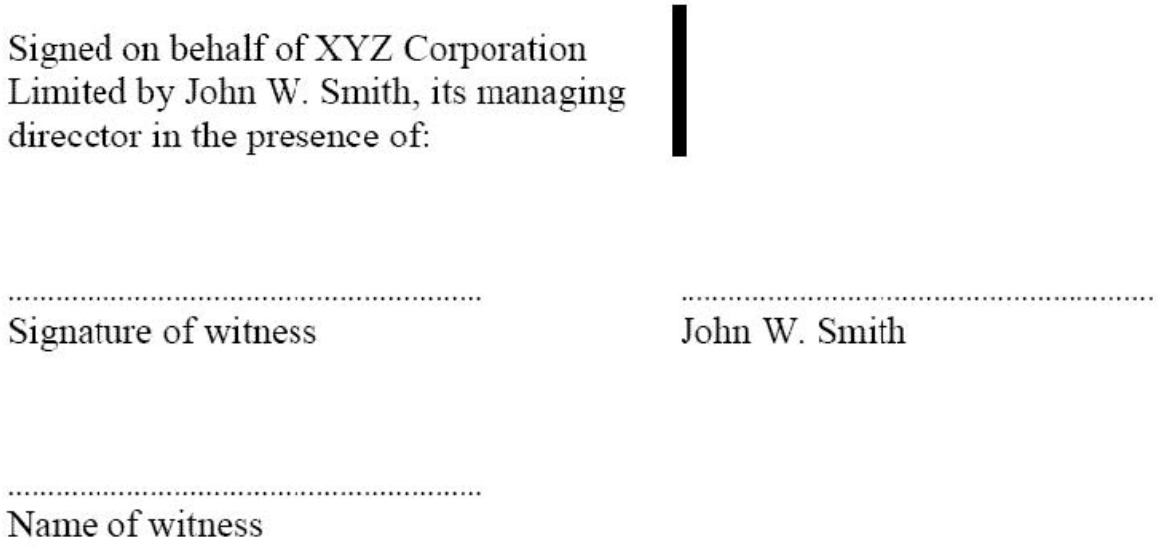
Signed on behalf of XYZ Corporation Limited, by John W. Smith, its authorized director:

.....
John W. Smith
Director

Example B.24. Right-column-only illustration

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

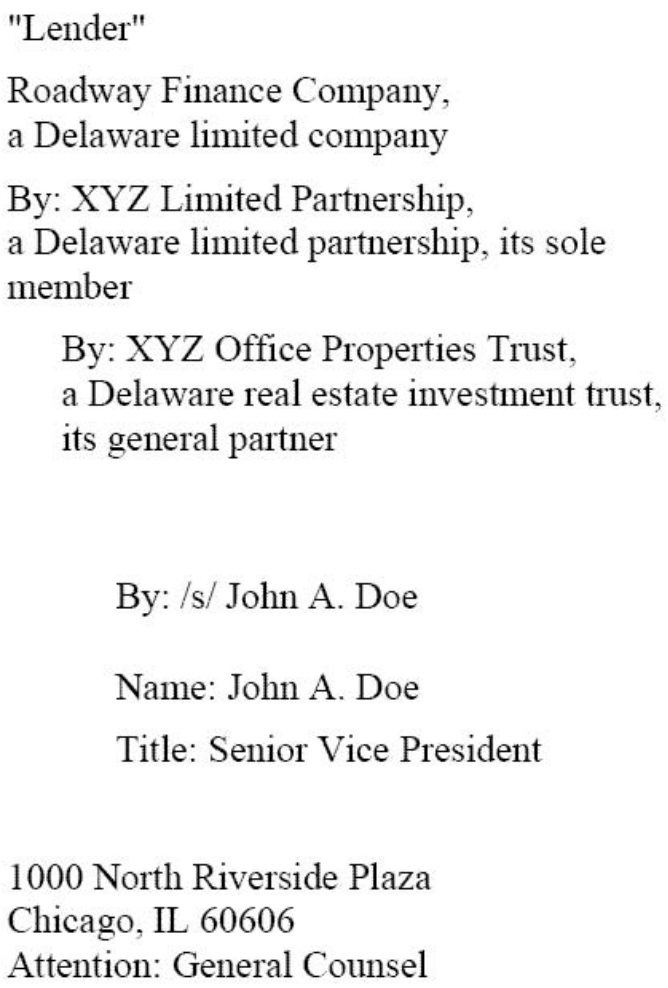
This example shows the use of the `brace` attribute on the `signatory-record`:



Example B.25. The brace attribute

```
<party-signature> <signatory-group align-records="vertical" brace="block-only">
<block><text>Signed, sealed and delivered on behalf of XYZ Corporation Limited
by two of its directors:
</text></block><signatory-record align-signatory-witness="horizontal">
<witness> <signature-line><field type="dotleader" length="50"/>
<text>Signature of
witness</text></signature-line><signature-line><field type="dotleader"
length="50"/><text>Name of witness</text></signature-line>
</witness><signatory><signature-line><field type="dotleader" length="50"/>
    <text>John W.  Smith</text></signature-line></signatory></signatory-record>
<signatory-record align-signatory-witness="horizontal">
<witness> <signature-line><field type="dotleader" length="50"/>
<text>Signature of witness</text></signature-line>
<signature-line>
   <field type="dotleader" length="50"/><text>Name of witness</text>
</signature-line></witness>
<signatory><signature-line><field type="dotleader" length="50"/>
    <text>John A.  Doe</text></signature-line></signatory>
</signatory-record></signatory-group>
</party-signature>
```

`signatory-record`'s can be nested to show delegation. This is rendered by a small indentation for the information within the nested `signatory-record`.



Example B.26. 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>
       <text>a Delaware limited company</text></block>
<signatory-record><block>
   <text>By: XYZ Limited Partnership,</text>
   <text>a Delaware limited partnership, its sole member</text></block>
<signatory-record> <block><text>By: XYZ Office Properties Trust,</text>
       <text>a Delaware real estate investment trust,
             its general partner</text></block>
<signatory-record><signatory>
   <signature-line><text>By: /s/ John A. Doe</text>
         <field length="50" type="blank"/>
   </signature-line>
   <block><text>Name: John A. Doe</text></block>
   <block><text>Title: Senior Vice President</text></block>
</signatory></signatory-record></signatory-record></signatory-record>
<block><text></text></block><block>
       <text>1000 North Riverside Plaza</text>
       <text>Chicago, IL 60606</text>
       <text>Attention: General Counsel</text></block>
</party-signature>
```

48. person-record

48.1. Synopsis

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

48.1.1. Content Model

`person-record` has a mixed content model.

```
person-record = element person-record {

    (text | name | address | field | term)*,

    person-record.attlist
}


person-record.attlist =
    common.attributes,
    person-record.class.attribute,
    person-record.party-id.attribute,
    person-record.attlist.extensions


person-record.class.attribute =     standard.class
person-record.party-id.attribute =  attribute party-id { xsd:string }?
person-record.attlist.extensions =  empty
```

person-record ::=

- Zero or more
  - `address`
  - `field`
  - `name`
  - `term`
  - `text`

48.1.2. Attributes

Common Attributes Class Attributes

Additional attributes:

- `party-id`

48.2. 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.

48.2.1. 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.

48.2.2. Parents

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

48.2.3. Children

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

48.2.4. See Also

`party`

48.2.5. Examples

See `parties`

49. phrase

49.1. Synopsis

49.1.1. 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`

49.1.2. Attributes

Common Attributes and Class Attributes

49.2. 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.

49.2.1. Processing Expectations

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

49.2.2. Attributes

Common Attributes and Class Attributes

49.2.3. 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`

49.2.4. Children

The following elements occur inside `phrase`:

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

49.2.5. See Also

`note` and `note-in-line`

50. reference

50.1. Synopsis

50.1.1. 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

50.1.2. Attributes

Common Attributes and Class Attributes

Additional attributes:

- destination-lang
- destination-type
- href
- print-url

50.2. 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.

50.2.1. Processing Expectations

The content of this element is rendered inline.

50.2.2. 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 lagnuage 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/committes/documents.php?wg_abbrev=legalxml-econtracts)
```

50.2.3. 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

50.2.4. Children

The following elements occur inside `reference`:

`citation` `em` `field` `reference` `statutory-em` `strike` `sub` `sup`

50.2.5. See Also

`term`

50.2.6. Examples

Example B.27. 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
          <reference href="http://www.elkera.com" print-url="true">
can be read on the web easily!</reference>.</text></block>
    </attachment>
 </attachments>

</contract>
```

51. row

51.1. Synopsis

This is a table row.

51.1.1. Content Model

```
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`

51.1.2. Attributes

Common Attributes

Additional Attributes:

- rowsep
- valign

51.2. 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].

51.2.1. Attributes

This section only provides a brief description of each attribute's purpose. Please refer to the OASIS Exchagne 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 B.7. (enumeration)

`top`	Align the content at the top of the row.
`middle`	Align the content within the row.
`bottom`	Align the content at the bottom of the row.

51.2.2. Parents

These elements contain `row`: `thead` and `tbody`.

51.2.3. Children

The following element occur inside `row`: `entry`

51.2.3.1. Example:

Please see `table`.

52. signatory

52.1. Synopsis

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

52.1.1. Content Model

```
signatory = element signatory {

    (signature-line | block)*,

    signatory.attlist
}
```

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


signatory.person-record-id.attribute = attribute person-record-id { xsd:string }?
signatory.attlist.extensions = empty
```

signatory ::=

- Zero or more of
  - `signature-line`
  - `block`

52.1.2. Attributes

Common Attributes

Additional Attributes

- person-record-id

52.2. Description

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

52.2.1. 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`.

52.2.2. Parents

The elements contain `signatory`: `signatory-record`

52.2.3. Children

The following elements occur inside: `signatory`

`signature-line` and `block`

52.2.4. See Also

`witness`

52.2.5. Examples

Please see party-signature for extensive examples.

Example B.28. Shows use of person-record-id

```
<?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>Showinging person-record-id</text></title>
<contract-front>
```

```
<parties>
<party><person-record id="AA2"><name>John Smith</name>
</person-record>
 and <person-record id="AA3"><name>Jaime Velazquez</name></person-record></party>
</parties></contract-front>
<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 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>
```

53. signatory-group

53.1. Synopsis

53.1.1. Content Model

```
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`

53.1.2. Attributes

Common Attributes

Additional attributes:

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

53.2. 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.

53.2.1. 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.

53.2.2. Attributes

Common Attributes Additional Attributes:

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

  Table B.8. (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 B.9. (enumeration)

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

53.2.3. Parents

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

53.2.4. Children

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

53.2.5. See Also

signatory-record

53.2.6. Examples

See party-signature

54. signatory-record

54.1. Synopsis

54.1.1. 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`

54.1.2. Attributes

Common Attributes

Additional attributes:

- `align-signatory-witness` (enumeration)
  - "horizontal"
  - "vertical"
- `brace`(enumeration)
  - "block-only"

54.1.3. 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.

54.2. 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.

54.3. 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 colun separator to the right of the left column's content. See the second to last example under party-signature.

54.4. Attributes

Common Attributes

Additional Attributes:

align-signatory-witness

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

Table B.10. (enumeration)

horizontal	The signature information is laid out across the page.
vertical	The signature inforamtion is laid out vertically, with one item underneath another

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 B.11. (enumeration)

"horizontal"	The signature information is laid out across the page.
"vertical"	The signature information is laid out with one item underneath another.

54.5. Parents

These elements contain signatory-record: party-signature, signatory-group, and signatory-record

54.6. Children

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

54.7. See Also

signatory

54.8. Examples

See party-signature

55. signature-line

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

55.1. Synopsis

55.1.1. Content Model

```
signature-line = element signature-line {

   (\text* & field )*,

    signature-line.attlist
}

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

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

signature-line ::

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

55.1.2. Attributes

Common Attributes

55.2. Description

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

55.2.1. Attributes

Common Attributes

55.2.2. Parents

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

55.2.3. Children

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

55.2.4. Examples

Please see party-signature for extensive examples.

56. statutory-em

56.1. Synopsis

56.1.1. Content Model

`statuory-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
  - field
  - reference
  - statutory-em
  - strike
  - sub
  - sup
  - text

56.1.2. Attributes

Common Attributes and Class Attributes

56.2. 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].

56.2.1. Attributes

Common Attributes and Class Attributes

56.2.2. 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

56.2.3. Children

The following elements occur inside: `statutory-em`:

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

56.2.4. See Also

em strike

56.2.5. Examples

Example B.29. 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>
      <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>
```

57. strike

57.1. Synopsis

57.1.1. 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`

57.1.2. Attributes

Common Attributes and Class Attributes

57.2. Description

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

57.2.1. Attributes

Common Attributes and Class Attributes

57.2.2. Parents

These elements contain `strike`: `text`

57.2.3. Children

The following elements occur inside `strike`:

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

57.2.4. See Also

`sub` `sup`

57.2.5. Examples

Example B.30. Example of the strike element

```
<?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>
```

58. sub

58.1. Synopsis

58.1.1. 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

58.1.2. Attributes

Common Attributes and Class Attributes

58.2. Description

This is used to mark up content to be subscripted.

58.2.1. Attributes

Common Attributes and Class Attributes

58.2.2. Parents

These elements cotnain 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

58.2.3. Children

The following elements occur inside sub:

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

58.2.4. See Also

sup strike

58.2.5. Examples

Example B.31. Example of sub and sup 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 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>
```

59. subtitle

59.1. Synopsis

59.1.1. 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`

59.1.2. Attributes

Common Attributes

59.1.3. Description

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

59.1.3.1. Attributes

Common Attributes

59.1.4. Parents

These elements contain `subtitle`: `attachment` and `contract`

59.1.5. Children

The following elements occur inside `subtitle`: `text`

59.1.6. See Also:

`title`

59.1.7. Examples

Example B.32. Example of a subtitle element

```
<?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 specifcation draft
  </subtitle>




  <body></body>

  <back></back>

  <attachments>

    <attachment><title><text>Attachment Title</text></title><subtitle>Attachment
Subtitle</subtitle><subtitle>More than One Subtitle
Allowed</subtitle></attachment>
  </attachments>

</contract>
```

60. sup

60.1. Synopsis

60.1.1. Content Model

`sup` has a mixed content model.

```
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

60.1.2. Attributes

Common Attributes and Class Attributes

60.2. Description

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

60.2.1. Attributes

Common Attributes and Class Attributes

60.2.2. 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`

60.2.3. Children

The following elements occur inside `sup`:

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

60.2.4. See Also

`sub` `strike`

60.2.5. Examples

Example B.33. Example showing the sub and sup 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 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>
```

61. table

61.1. 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]

61.1.1. Content Model

```
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`

61.1.2. Attributes

Common Attributes and orient.attribute

Additional Attributes

- frame
  - "top"
  - "bottom"
  - "topbot"
  - "all"
  - "sides"
  - "none"
- colsep
- rowsep
- pgwide
- summary
- title

61.2. 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].

61.2.1. 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 B.12. (enumeration)

`top`	below title
`sides`	(left and right sides)
`bottom`	after information for last `row`
`topbot`	both `top` and `bottom`
`none`	none of the above

`all`	all of the above

colsep

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

rowsep

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.

61.2.2. Parents

These elements contain `table`: `block`

61.2.3. Children

The following elements occur inside `table`: `tgroup` and `title`

61.2.4. 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`.

61.2.5. Example

Example B.34. Example showing a table

```
<?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>
        <tgroup cols="1">
          <colspec colnum="1" colname="C1" colwidth="10" colsep="true"
                rowsep="false" align="justify"/>
          <thead>
            <row><entry><block></block></entry></row>
          </thead>
          <tbody>
            <row><entry><block></block></entry></row>
          </tbody>
        </tgroup>
      </table>
    </block>
  </body>

</contract>
```

62. tbody

62.1. Synopsis

62.1.1. 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

62.1.2. Attributes

Common Attributes

Additional Attributes

- valign

62.2. 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].

62.2.1. 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 B.13. (enumeration)

`top`	Align content at the top of the row.
`middle`	Center content within the row (default)
`bottom`	Align content at the bottom of the row.

62.2.2. Parents

These elements contain `tbody`: `tgroup`

62.2.3. Children

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

62.2.4. Example:

Please see table.

63. term

63.1. Synopsis

63.1.1. 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

63.1.2. Attributes

Common Attributes and Class Attributes

Additional Attributes

- "abbreviation"

63.2. Description

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

63.2.1. 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.

63.2.2. Parents

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

63.2.3. Children

The following elements occur inside `date`:

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

63.2.4. See Also

`phrase` and `field`

63.2.5. Examples

Please see `terms`

64. terms

64.1. Synopsis

64.1.1. 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`

64.1.2. Attributes

Common Attributes

64.2. 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.

64.2.1. Attributes

Common Attributes

64.2.2. Parents

These element contain `terms`: `definition`

64.2.3. Children

The following elements occur inside `terms`:

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

64.2.4. Examples

Example B.35. Examples of term and terms 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
               - 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>
```

65. text

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

65.1. Synopsis

65.1.1. 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

65.1.2. Attributes

Common Attributes and Class Attributes

Additional attributes:

- `textflow` (enumeration)
  - "runon"
- `xml:space` (enumeration)
  - "default"
  - "preserve"

65.2. 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.

65.2.1. 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 B.14. (enumeration)

"runon"	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`.

- `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 B.15. (enumeration)

"default"	Use application default
"preserve"	Preserve white space

65.2.2. 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`, `subtitle`, `strike`, `term` `terms`, and `title`

65.2.3. 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`

65.2.4. See Also

`term`, `note-in-line`

65.2.5. Examples

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

Example B.36. An example showing `text` element

```
<?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>
```

Example B.37. Another example showing `text` element including inline elements

```
<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>
```

66. tgroup

66.1. Synopsis

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

66.1.1. 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`

66.1.2. Attributes

Common Attributes

Additional Attributes:

- cols
- colsep
- rowsep
- align

66.2. 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].

66.2.1. 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 B.16. (enumeration)

`left`	align to the left (default)
`right`	align to the right (default)
`center`	center the text
`justify`	justify the text

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`.

66.2.2. Parents

These elements occur inside `tgroup`: `table`

66.2.3. Children

The following elements occur inside `tgroup`: `colspec`, `thead` and `tbody`

66.2.4. Example:

Please see `table`.

67. thead

67.1. Synopsis

This is the table header.

67.1.1. 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

67.1.2. Attributes

Common Attributes

Additional Attributes

- valign

67.2. 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].

67.2.1. 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 B.17. (enumeration)

`top`	Align content at the top of cell.
`middle`	Centered content within the cell.
`bottom`	Align content at the bottom of the cell.

67.2.2. Parents

These elements contain `thead`: `tgroup`

67.2.3. Children

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

67.2.4. Example:

Please see `table`.

68. title

68.1. Synopsis

68.1.1. 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
  - text

68.1.2. Attributes

Common Attributes

68.2. Description

This is used to indicate the title for the contract itself, attachments, elements in lists, and many other elements, as listed below.

68.2.1. Parents

These elements contain `title`: attachment background back body contract inclusion item block. item parties table

68.2.2. Children

The following elements occur inside `title`: text

68.2.3. See Also:

subtitle

68.2.4. Examples

Example B.38. To illustrate a title in an otherwise minimal contract.

```
<?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>
```

69. witness

69.1. Synopsis

witness (a person who witnesses the signing of a contract or document)

69.1.1. Content Model

```
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`

69.1.2. Attributes

Common Attributes

69.2. Description

This is the person who witnesses the signing of the document.

69.2.1. Attributes

Common Attributes

69.2.2. Parents

These elements contain `witness`: `signatory-record`

69.2.3. Children

The following elements appear inside `witness`: `signature-line` and `block`.

69.2.4. See Also

`signatory`

69.2.5. Examples

Please see party-signature

70. xi:fallback

70.1. Synopsis

70.1.1. Content Model

`xi:fallback` has a mixed content model.

```
    xiInclude = element xi:include {

        element xi:fallback {

            (xiInclude.fallback.model)*

        }?,
        attribute href { xsd:anyURI }?,
        xiInclude.extensions

    }

    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

70.1.2. Attributes

None.

70.2. Description

This is xi:include fallback text. This element and all other elements contained by xi:include are taken from the World Wide ConsoritumXML Inclusions recommendation. Please refer to that specification for full details of this element.

70.2.1. Parents

These elements contain xi:fallback: xi:include

70.2.2. Children

The following elements occur inside xi:fallback:

conditional, em, field, reference, statutory-em, strike, sub, sup xi:include

70.2.3. See Also:

Please do not confuse this with fallback

xi:fallback can only be used inside xi:include

70.2.4. Example

Please see the example for xi:include

71. xi:include

71.1. Synopsis

This allow including or bringing in content from other files. It uses the World Wide Web Consortium XML Inclusions recommendation:

http://www.w3.org/TR/2004/PR-xinclude-20040930

71.1.1. Content Model

```
   xiInclude = element xi:include {

       element xi:fallback {

           (xiInclude.fallback.model)*

       }?,
       attribute href { xsd:anyURI }?,
       xiInclude.extensions
```

```
    }


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

    inline.content.inner |= conditional
    xiInclude.fallback.model = xiInclude | inline.content.inner
    xiInclude.extensions = empty

}
```

xi:include::=

- Zero or one `xi:fallback`

71.1.2. Attributes

href

71.2. 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:

```
<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 ifnormation to be included when the indicated file is not located.

71.2.1. Attributes

href

This specifies the URI (possibly after escaping characters) from which to load the content.

71.2.2. Parents

`xi:include` appears inside: `xi:fallback` `attachment` `back` `background` `body` `entry` `inclusion` `itemregular`

71.2.3. Children

The following elements occur inside `xi:include`: `xi:fallback`

71.2.4. Example:

This will bring in **Inventry01.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 B.39. Example of xInclude element as well as fallback element

```
<?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 xi:include.</text></title>

  <body>
    <xi:include href="Inventry01.xml">
      <xi:fallback>
        <xi:include href="Acct.xml">
          <xi:fallback>This is the fallback of Acct.xml.</xi:fallback>
        </xi:include>
      </xi:fallback>
    </xi:include>
    <xi:include href="Terminal.xml"/>
  </body>

</contract>
```

C. Acknowledgments (Non-Normative)

The following members of the committee have participated in the creation of this specification:

- Rolly Chambers, American Bar Association
- Daniel Greenwood, Electronic Commerce Architecture Project, Massachussets Institute of Technology
- Dr. Laurence Leff, Individual Member
- Mr. Dave Marvit, Fujitsu Corporation
- Mr. Peter Meyer, Associate Member
- Dr. Zoran Milosevic, Individual Member

The following persons assisted in the writing and review of this specification. There contributions are gratefully acknowledged:

- Mr. Nobuhiro Sasagawa
- Mr. Andrew Squire
- Dr. Hoylen Sue

Contributions by the following persons who were formerly members of the Technical Committee are gratefully acknowledged:

- Mr. Jason Harrop, Individual Member
- Mr. John McClure, Individual Member

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

[dc] *Dublin Core Metadata Element Set, Version 1.1: Reference Description* 2004-12-20, http://dublincore.org/documents/dces

[XInclude] *XML Inclusions (XInclude) Version 1.0* W3C Proposed Recommendation 30 September 2004

[RFC 2119] S. Bradner. *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. IETF (Internet Engineering Task Force). 1997.

Non-normative

[CM] eContracts Structure sub-committee, Harrop, Jason, editor, eContracts Structure Markup - Preliminary, July 19, 2004. http://www.oasis-open.org/committees/documents.php?wg_abbrev=legalxml-econtracts

[crc] Australian Government, Department of Education, Science and Training, 2006 CRC Commonwealth Agreement, https://www.crc.gov.au/HTMLDocuments/Documents/PDF/Comm_Agreement.pdf

[grand93] Grand, Mark, "Mime Overview" Revised October 26, 1993 http://mgrand.home.mindspring.com/mime.html

[meyer03] Meyer, Peter and Harrop, Jason: OASIS Legal XML eContracts TC Requirements Process: Requirements for Clause Model, Draft Number, May 27, 2003. http://www.oasis-open.org/committees/documents.php?wg_abbrev=legalxml-econtracts http://lists.oasis-open.org/archives/legalxml-econtracts

[meyer05] Meyer, Peter, "A Proposed XML Standard for Contract Documents" Presented at IDEAlliance XML2005 in Atlanta Georgia (www.elkera.com/cms/articles/seminars_and_presentations/xml_2005_atlanta_ga)

[meyer06] Meyer, Peter, "Extracting More Value From and Reducing the Cost of Precedents" Presentation at the Sinch Precedents Automation Conference, Sydney, 26 October 2006, www.ccthefile.com/sinchseminars/spac06

[oxford] Agreement for the Supply of Services by External Consultancy, http://www.admin.ox.ac.uk/rso/contracts/CA13.doc

[RD] OASIS Legal XML Contracts TC *Requirements for a Technical Specification* On OASIS LegalXML eContracts TC Public Documents. Version Number 1.0, May 20, 2005

[Relax] Clark, James, Cown, John and Murata, Makoto, "Relax NG Compact Syntax Tutorial" Working Draft 26 March 2003, http://relaxng.org/compact-tutorial-20030326.html

[sperberg] www.w3.org/2000/04/26-csrules.html C. M. Sperberg-McQueen, *Context-sensitive Rules in XML Schema* rev. 25 April 2000

[table1] CALS "XML Exchange Table Model" www.oasis-open.org/specs/soextblx.dtd

[table2] Norman Walsh OASIS CALS Table Model Technical memorandum TR 9901:1999, September 29th 1999. http://www.oasis-open.org/specs/tm9901.htm

[table3] Harvey Bingham, *CALS Table Model Document Type Definition* OASIS Technical Memorandum TM 9502:1995 www.oasis-open.org/specs/a502.htm October 19, 1995.

[XSLT] World Wide Web Consortium, *XSLT Transformations (XSLT)* Version 1.0, W3C Recommendation 16 November 1999.