© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

CalConnect TC VCARD

Lightweight document — Document metamodel

Published Standard

Warning for drafts

This document is not a CalConnect Standard. It is distributed for review and comment, and is subject to change without notice and may not be referred to as a Standard. Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

The Calendaring and Scheduling Consortium, Inc. 2026

© 2026 The Calendaring and Scheduling Consortium, Inc.

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from the address below.

The Calendaring and Scheduling Consortium, Inc.

4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

Contents

Abstract v
Foreword vi
Introduction vii
1. Scope 1
2. Normative references 1
3. Terms and definitions 2
4. Conformance 3
5. Modelling 4
5.1. Intent 4
5.2. Specialization 4
5.3. Constructs, types, and specialization 4
5.4. Tailoring and extension 4
5.5. Attribute register 5
5.6. Composition 5
5.7. Variables and raw content 5
5.8. Structure 5
6. BasicDocument model 6
7. Metadata and bibliographic information models 6
8. Section models 8
9. Block models 9
9.1. General 9
9.2. Paragraph 10
9.3. Multi-paragraph blocks 10
9.4. Table 12
9.5. List 13
9.6. Ancillary blocks 14
9.7. Amend 17
10. Inline element models 18
10.1. General 18
10.2. Text Elements 19
10.3. Empty Elements 20
10.4. ID Elements 20
10.5. Reference Elements 22
10.6. Footnote 24
11. Data type models 24
11.1. Basic Data Types 24
11.2. Contribution Element Metadata 24
12. Change models 25
12.1. General 25
12.2. Addressing model 26
12.3. Change 26
12.4. Change set 26
12.5. Unique identifier 26
12.6. Content change 27
12.7. Attribute change 27
12.8. Node change 28
12.9. Table changes 29
12.10. Computed content 32
12.11. Ordering and concurrent changes 32
Appendix A Mapping of lightweight text markup constructs 33
A.1. General 33
A.2. AsciiDoc 33
A.3. Markdown (CommonMark, GFM, Pandoc) 33

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

A.4. reStructuredText **34**

Appendix B Accommodating specializations **35**
B.1. Type specialization **35**
B.2. Attribute specialization **35**
B.3. Class specialization **35**
B.4. Relationship to publishing document models **35**

Appendix C Accommodating extensions **36**
C.1. The attribute register **36**
C.2. Opaque and raw content **36**
C.3. Variables and composition **36**
C.4. When a markup dialect gains a construct **36**

Appendix D Abstract test suite **37**
D.1. General **37**
D.2. Conformance classes **37**
D.3. Test material **37**
D.4. Verdicts **37**

Bibliography **39**

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Abstract

This document provides a reference lightweight document model called BasicDocument for generic documents, intended as an interoperable basis that allows systems to interchange generic document content while preserving semantics.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Foreword

The Calendaring and Scheduling Consortium (“CalConnect”) is a global non-profit organization with the aim to facilitate interoperability of collaborative technologies and tools through open standards.

CalConnect works closely with international and regional partners, of which the full list is available on our website (https://www.calconnect.org/about/liaisons-and-relationships).

The procedures used to develop this document and those intended for its further maintenance are described in the CalConnect Directives.

In particular the different approval criteria needed for the different types of CalConnect documents should be noted. This document was drafted in accordance with the editorial rules of the CalConnect Directives.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CalConnect shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be provided in the Introduction.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

This document was prepared by Technical Committee *VCARD*.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Introduction

There is a general need for interchange and interoperability of structured text amongst information systems.

While there is a plethora of richly-formatted text documents, ranging from complex document formats like OOXML and ODF, hypertext formats like HTML, to older formats such as RTF and presentation-focused formats like PDF, there is no standardized method of transmitting an interoperable text structure to another system.

In addition, the popularity of lightweight text markup syntaxes such as Markdown and AsciiDoc has been hampered by practical concerns that documents generated by such syntaxes have inherited identical issues as have the document formats described above.

Such issues have limited the exchange of structured text down to the lowest common denominator in parsing support, which is plain text, with no defined human- or machine-understandable structure.

The *BasicDocument* model (also known as *BasicDoc* or *SecureDoc*) is created to express the structure of generic documents in a lightweight form.

BasicDocument achieves a number of goals:

— Serve as an interoperable basis that allows systems to interchange general documents while preserving semantics;
— Allows document formats to be mapped to it and out of it, effectively enabling document formats to be converted to presentation formats without loss of meaning;
— Establishes a central data model for supporting multiple input formats (and syntaxes) with multiple output formats; and
— Enables changes to be applied incrementally in a defined manner.

Business documents exchanged between organizations — orders, invoices, contracts, tenders, specifications — combine such structured data with rich text. Standards exist for the data elements of commerce, for time (ISO 8601) and for organization identification (ISO 6523); the document layer that carries the remaining rich text — terms, notes, line-item tables, and the revisions exchanged between the parties — has no common model, and degrades to unstructured text on every hop between systems. *BasicDocument* is intended to serve as that carrier layer as well; Annex E illustrates its application to business document interchange without adding any construct to the model.

BasicDocument is deliberately not a detailed document model, unlike DocBook and TEI: it is intended to be a base document model which other document models can map to or specialize upon.

The *BasicDocument* document model has been designed to maintain compatibility with several existing models, with a view to reflecting the expressiveness of existing document structures, but also to align with the capability of established document production tools.

The markup languages and document production tools that *BasicDocument* sought to align with include:

— hypertext formats: W3C HTML 5 and HTML 4
— document formats: OOXML, Microsoft RTF, DocBook
— text markup syntaxes: Markdown and AsciiDoc

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Lightweight document — Document metamodel

1. Scope

This document provides a reference lightweight document model called *BasicDocument* for generic documents, intended as an interoperable basis that allows systems to interchange generic document content while preserving semantics.

This document model can be directly utilized to represent interchangeable document content, or adopted, superseded or internalized by document representation models and formats in order to represent document content in other forms.

The application of the reference model to particular classes of documents, such as business documents exchanged in commerce, is illustrated in Annex E.

The modelling of bibliographies and citations is part of the reference model, but is the subject of a separate document, CalConnect CC 6900:2019, and are hence not described in detail in this document.

The following aspects are excluded from scope:

— Specialization and profiling of this model to specialized classes of documents;
— Implementation of the reference model and serialization formats;
— Mapping of the reference model to output formats; and
— Markup schemes and syntaxes that enable creation of documents that fit the reference model.

2. Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

CalConnect CC 6900:2019, CalConnect, RONALD TSE, NICK NICHOLAS and CalConnect. *Information and documentation — Bibliographic reference model and serialization*. First edition. 2019. CalConnect.

ISO/IEC 19757-2, International Organization for Standardization (committee). *Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*. Second edition. Geneva: International Organization for Standardization and International Electrotechnical Commission. https://www.iso.org/standard/52348.html.

IETF RFC 3986, BERNERS-LEE, T., R. FIELDING and L. MASINTER. *Uniform Resource Identifier (URI): Generic Syntax*. 2005. RFC Publisher. https://www.rfc-editor.org/info/rfc3986.

ISO 639 (all parts), International Organization for Standardization (committee). *Codes for the representation of names of languages*. First edition. 2002. Geneva: International Organization for Standardization. https://www.iso.org/standard/22109.html.

ISO 8601-1:2019, International Organization for Standardization (committee). *Date and time — Representations for information interchange — Part 1: Basic rules*. First edition. 2019. Geneva: International Organization for Standardization. https://www.iso.org/standard/70907.html.

ISO 8601-2:2019, International Organization for Standardization (committee). *Date and time — Representations for information interchange — Part 2: Extensions*. First edition. 2019. Geneva: International Organization for Standardization. https://www.iso.org/standard/70908.html.

ISO/IEC 10118 (all parts), International Organization for Standardization (committee). *Information technology — Security techniques*. Third edition. 2016. Geneva: International Organization for Standardization and International Electrotechnical Commission. https://www.iso.org/standard/64213.html.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

ISO/IEC 14888 (all parts), International Organization for Standardization (committee). *Information technology — Security techniques*. Second edition. 2008. Geneva: International Organization for Standardization and International Electrotechnical Commission. https://www.iso.org/standard/44226.html.

ISO 15924, International Organization for Standardization (committee). *Information and documentation — Codes for the representation of names of scripts*. Second edition. Geneva: International Organization for Standardization. https://www.iso.org/standard/81905.html.

ISO 3166 (all parts), International Organization for Standardization (committee). *Codes for the representation of names of countries and their subdivisions*. Fourth edition. 2020. Geneva: International Organization for Standardization. https://www.iso.org/standard/72482.html.

ISO 24229, International Organization for Standardization (committee). *Information and documentation — Codes for written language conversion systems*. First edition. Geneva: International Organization for Standardization. https://www.iso.org/standard/78143.html.

3. Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.
class
structure containing a description of an entity in terms of its components

3.2.
subclass
class (Clause 3.1) which inherits from another class its component descriptions, and optionally adds to them its own component descriptions

3.3.
document model
model
formal specification of the structure of a document in terms of its components and their arrangement, expressed through *classes* (Clause 3.1)

3.4.
paragraph
subdivision of running text, normally run on throughout, that is separated from text before and after by a change of line and stands below any chapters or sections (Clause 3.7)

[SOURCE: ISO 5127:2017, Clause 3.5.8.07]

3.5.
block
paragraph (Clause 3.4)-level grouping of text

3.6.
inline element
grouping of text that can be contained within a *paragraph* (Clause 3.4), including plain strings

3.7.
section
hierarchical subdivision of a document, consisting of one or more *blocks* (Clause 3.5), and/or one or more sections

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

3.8.
construct
generic model element provided by this document (a class, an enumeration, or a data type) which markup languages and document models specialize as types or subclasses

3.9.
attribute register
open set of key-value entries, optionally qualified by a scheme, attachable to any construct, into which markup languages bind their own attribute types

3.10.
spelling system
combination of language (ISO 639), script (ISO 15924), and optionally country (ISO 3166) identifying a written-language variant, as registered under ISO 24229

3.11.
composition
attachment of an entire document, as a block, as child content of any container whose content model accepts blocks

3.12.
identifier
character, or group of characters, used to identify or name an item of data and possibly to indicate certain properties of that item

[SOURCE: 714-21-07]

4. Conformance

The definition modules of the model repository are the definitive expression of this document model.

The model is independent of serialization. It may be implemented in one, many, or multiple serialization formats — XML, YAML, or any format capable of representing the model's constructs — and a document can be transformed from one serialization format to another with no loss. This lossless transformability is a defining property of the model-driven architecture of this document: the model, not any of its serializations, is what conforming implementations share.

Reference serializations accompany the model, each a complete and machine-checkable realization of it: an XML serialization whose schema is a RelaxNG Compact grammar conforming to ISO/IEC 19757-2, and a YAML serialization defined by the instance conventions of the model repository (see Appendix D). They do not constrain the choice of format; they demonstrate that the model is fully implementable, and serve as equivalence targets for testing other serializations. The diagrams reproduced in this document are informative.

A serialization or implementation conforms to this model if it can represent every construct of the model without loss. This model is intended as a complete superset of lightweight markup languages (AsciiDoc, Markdown and its flavours, reStructuredText) and richer document formats: a construct of such a format is covered if it maps to:

1) an existing construct of this model;
2) a type-specialization of such a construct; or
3) entries in the attribute register, together with relaxed or opaque (raw) content.

The coverage of the principal lightweight text markup languages is demonstrated construct by construct in Appendix A. Specialization of constructs is described in Appendix B; extension without model change (register entries, opaque content, variables) in Appendix C.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

The following are preprocessing or rendering concerns, and are out of scope: attribute conditionals, content inclusion, table-of-content generation, smart punctuation, and citation rendering modes.

5. Modelling

5.1. Intent

The *BasicDocument* model expresses the structure of generic documents in a lightweight form.

It is deliberately not designed to fully represent semantics of document formats like DocBook and TEI: the model is intended to be a base document model which other document models can map to or specialize upon, or for the interchange of generic document content.

5.2. Specialization

Specialization of a model consists of:

— Adding classes to a base model.
— Changing attributes of a base model class. This is not restricted to adding attributes, as is the case in typical entity subclassing; it can also include removing attributes from a class, changing their obligation and cardinality, and changing their type, including changing enumerations. Attributes can be overruled at any level; for example, standards-specific models routinely enhance the bibliographic model at the base of the hierarchy.
— For reasons of clarity, renaming classes and attributes is avoided in specialisation.

5.3. Constructs, types, and specialization

The *BasicDocument* model provides generic *constructs*; markup languages and document models *specialize them as types*. The model does not enumerate a class per format feature: admonition kinds such as Note, Warning, Caution are values of the *AdmonitionType* of a single *Admonition* construct, and markup-specific kinds arrive through type values or subclassing. A foreign construct is covered by this model if it maps to an existing construct, a type-specialization of one, or register entries with relaxed or opaque content (see Clause 4).

5.4. Tailoring and extension

Users adapt this model through three layers, none of which modifies the basis (open-closed):

— **Extension of instances** — the attribute register, format-qualified raw content, and variable references carry dialect-specific information losslessly, with no tooling beyond the base model.
— **Extension of the model** — a specialization module defines new classes under the construct roots, extends enumerations, and narrows inherited attributes, as described above; it is a layer above this model, never a change to it.
— **Tailoring** — a *profile* declares a subset of this model: the constructs it excludes, the cardinalities it narrows, the enumeration values it admits, and the closed vocabulary of register keys it permits.

A profile shall only narrow the base: it shall not widen any cardinality, admit enumeration values outside the base, add constructs, or rename them. Extensions live above a profile, as specialization modules. Profiles are expressed as declarative artifacts alongside the model, and are validated mechanically with the model repository's test suite (see Appendix D); an instance that conforms to a profile is always a conforming instance of the base model.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

5.5. Attribute register

Attributes form an open *register*: any construct (document, section, block, inline element) carries a set of key-value *Attribute* entries, optionally qualified by a scheme. Markup languages bind their own attribute types as register entries; this model deliberately does not enumerate them. Keys with established cross-format meaning include `id`, `class`, `lang`, `script`, `unnumbered`, `subsequence`, and `format`. The set of well-known keys is maintained as a public registry by the registration authority; additional keys are registered according to its registration procedure. A profile may close the permitted vocabulary of register keys (see *Tailoring and extension*).

5.6. Composition

A document is itself a block: entire documents may be attached as child content of any container whose content model accepts blocks. Content models of containers (list items, multi-paragraph blocks, definition-list definitions, table cells, examples) are accordingly *relaxed*: they contain zero or more blocks, not only paragraphs.

5.7. Variables and raw content

References to variables (attribute references in AsciiDoc, substitutions in RST, template variables) are modelled as a *ReferenceToVariable* inline element resolving, at processing time, against the attribute register. Raw or passthrough content is non-markup content the model does not interpret: inline raw content is a *FormattedString* (whose `format` attribute exists precisely to admit markup into strings), and block-level raw content is a string-carrying construct qualified by a `format` register entry.

5.8. Structure

The classes involved in the document model are of three classes:

— Sections (Clause 8)
— Blocks (paragraph-level groupings of text) (Clause 9)
— Inline elements (groupings of text smaller than a paragraph, including plain strings) (Clause 10)

In the *BasicDocument* model, the classes are in a strict hierarchical relation:

— Documents consist of sections, which consist of blocks, which consist of inline elements.
— Sections can be nested within sections (e.g. clauses and subclauses); blocks can be nested within blocks (e.g. nested lists);
— Inline elements can also be embedded within other inline elements (e.g. bold + italics).

However, sections should not be siblings of blocks, nor blocks of inline elements. For this reason, paragraphs cannot contain other block elements, such as lists or tables.

a list (block) is not expected to occur next to inline text within a paragraph.
NOTE This constraint is imposed in order to maintain structural simplicity of this model. While it sacrifices some expressive potential, the difference is minor, particularly with regards to the rendering of paragraphs. This is a major difference between this model and the more flexible document models in XML-based schemas, such as HTML, TEI-C, and DocBook, which do not have this limitation.

The *BasicDocument* model is most fleshed out at the level of blocks and inline elements. Specialization of the model is expected to take place mostly at the level of prescribing particular arrangements of sections.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

The Basic Document model depends on the ISO 690 bibliography model for its expression of bibliographic references. The specific bibliography model instantiation and serialization it uses is described in CalConnect CC 6900:2019.

6. BasicDocument model

The *BasicDocument* model treats all documents as collections of *sections* (Clause 8). It also adds the following metadata as part of the document model:

`identifier` an optional globally unique identifier for the document in an agreed identifier schema. The identifier is to be used for tracking interactions with the document without depending on formal document registries; it would be exemplified by a GUID, rather than a document registry identifier such as “ISO 639”, which belongs to `bibdata`.

`bibdata` a bibliographic description, capturing bibliographic metadata about the document itself, including authors, title, and date of production.

`integrityValue` an optional digital signature of the document (Clause 11.2).

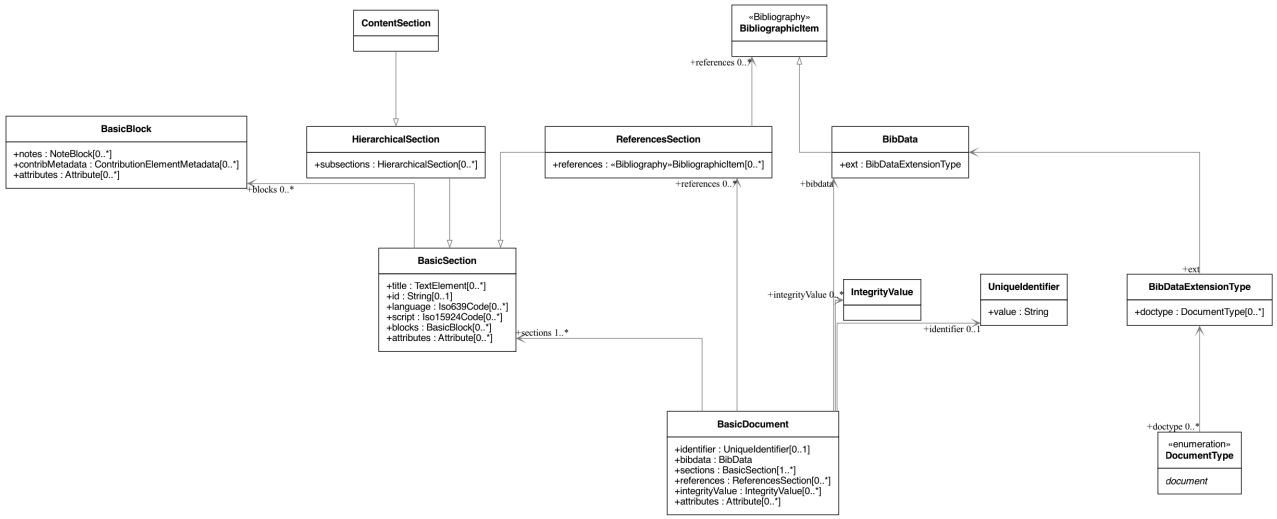

Figure 1 — Basic Document model: Document

7. Metadata and bibliographic information models

The modelling of `bibdata` follows the *BibliographicItem* class in CalConnect CC 6900:2019, and readers are referred to that specification. The *BibliographicItem* class is intended to capture document citations, and to be applicable to any document type, without any further specialization; that is because a document can cite documents of any type.

The `bibdata` class allows the *BibliographicItem* class to be extended with metadata specific to a document class, which appears as document metadata rather than as citation data; this information is modelled in `BibDataExtensionType`. The only extension point modelled as generically applicable in the Basic Document model is the document type, `doctype`, which is populated in the generic instance with the single value “document”. It is assumed that particular specialisations of the document model will substitute their own enumerations of particular subclasses of document, which will be more granular than the intentionally generic classes of document modelled in CalConnect CC 6900:2019.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

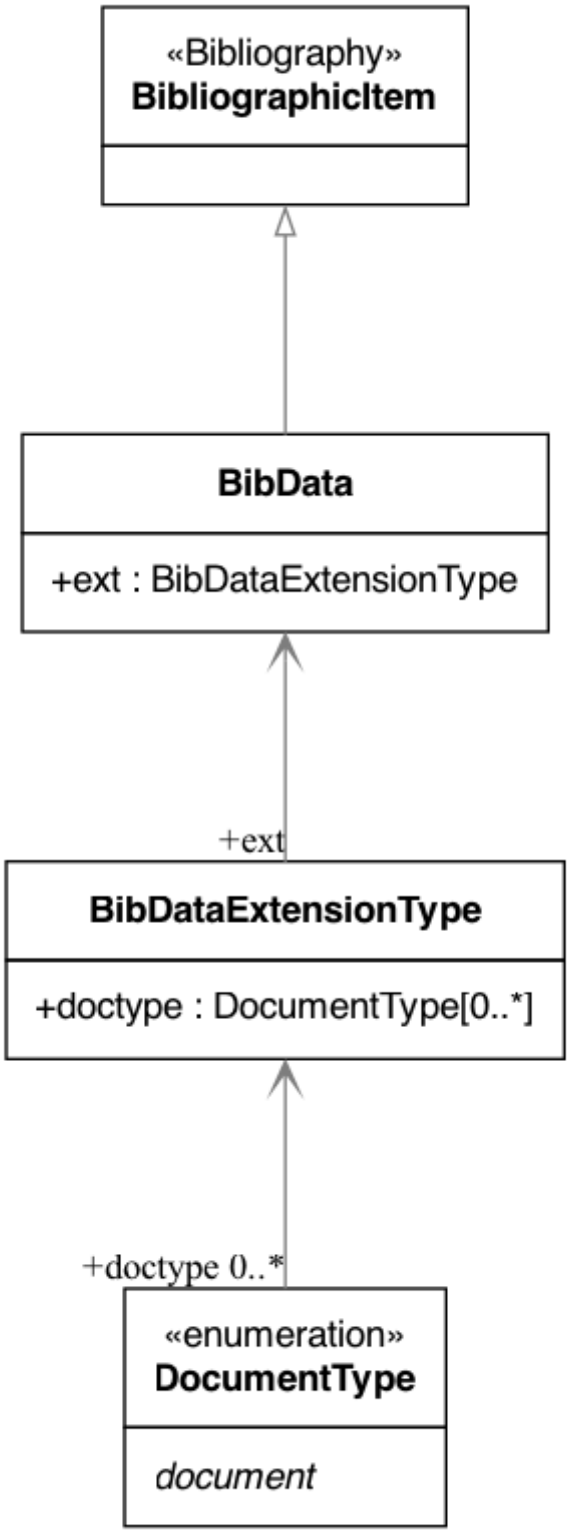

Figure 2 — Basic Document model: Bibdata

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

8. Section models

The *BasicDocument* model is generic in its application, and accordingly does not do detailed modelling of the differences between sections; that is deferred for specialisations (such as the *StandardsDocument* model). The *BasicDocument* model only recognizes the following classes of section:

— *Basic Sections*, which are leaf nodes (do not contain any sections).
— *Hierarchical Sections*, which can contain other sections. Hierarchical Sections are modelled as a subclass of Basic Sections, so the hierarchical arrangement of sections can be arbitrarily deep.
— *Content Sections*, which in the *BasicDocument* model are treated as equivalent to Hierarchical Sections. (The distinction is currently reserved for downstream document models, which may differentiate structurally between prefatory sections and sections in the main body of the text; differences between the two may be introduced in the Basic Document model at a later date.)
— *References Sections*, which are leaf nodes, and contain zero or more bibliographical items (as described in CalConnect CC 6900:2019), along with any prefatory text.

All sections are modelled as having the following attributes:

`title` an optional title of the section.

`id` an optional identifier for the section, to be used for cross-references within the document. (Citations of references are modelled as cross-references to the corresponding bibliographical item in the References section.)

`language` zero or more language tags for the section, coded as ISO 639 codes.

`script` zero or more script tags for the section, coded as ISO 15924 codes. The combination of language and script identifies a spelling system as registered under ISO 24229.

`blocks` zero or more text blocks, containing the textual content of the section (but excluding subsections, which are only present in Hierarchical Sections). Notes are not given a separate attribute: a note is itself a block, modelled as an admonition of type *note*.

NOTE In the *BasicDocument* model, a section can contain both blocks of text and subsections; in rendering, the blocks of text are presumed to come before the subsections. In some classes of document the co-occurrence of text and subsections in a clause (or in a section) is proscribed as “hanging paragraphs”: in order for text to be clearly identifiable by section number, those models prevent text and subsections from being siblings. That proscription is modelled as an override of this model specific to Standards documents.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

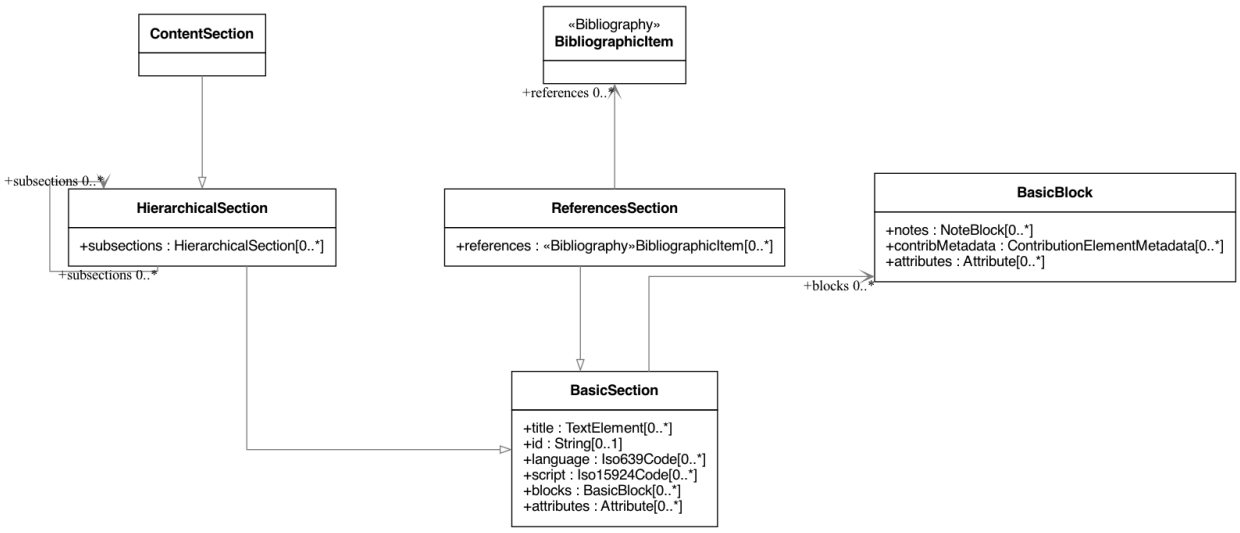

Figure 3 — Basic Document model: Section

9. Block models

9.1. General

Blocks of text are all modelled as subclasses of the *BasicBlock* class, which has the following attributes common to all blocks:

`id` an optional identifier for the block, to be used for cross-references.

`notes` any notes whose scope is the block. Notes are modelled as a sequence of zero or more paragraphs.

`contribMetadata` attribution of the block to a specific contributor (Clause 11.2), to be used in change management of documents.

The *BasicDocument* model recognizes the following classes of block:

— Paragraphs (Clause 9.2)
— Multi-paragraph blocks: Blockquotes, Reviewer comments, Admonitions (Clause 9.3)
— Tables (Clause 9.4)
— Lists (Clause 9.5): Unordered lists, Ordered lists, Definition lists
— Ancillary blocks: Figures (Clause 9.6.2), Sourcecode (Clause 9.6.3), Formulas (Clause 9.6.4), Pre-formatted blocks (Clause 9.6.5), Examples (Clause 9.6.6)

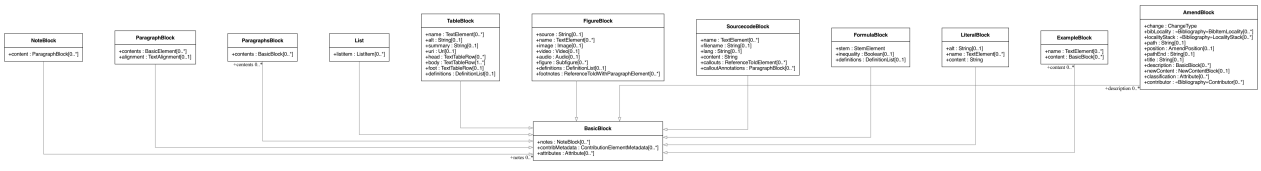

Figure 4 — Basic Document model: Block

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

9.2. Paragraph

Paragraphs can contain any sequence of inline elements (Clause 10), and optionally a text alignment (`alignment`). Unlike the case for other document models, paragraphs *cannot* contain other blocks, such as lists, tables, or figures: they are modelled as a basic building block of text.
NOTE Text alignment is the only concession the modelling of paragraphs makes to rendering, and is there because the application of alignment to paragraphs, while rare, can be unpredictable from paragraph semantics. Other rendering attributes of paragraphs, such as spacing before and after, are considered to be semantically predictable and are relegated to document stylesheets.

Paragraphs have the following subclass:

— *Paragraph With Footnote*, which can also contain footnotes (Clause 10.6). While most paragraphs in a document can contain footnotes, the distinction is necessary, as footnotes are not appropriate for all instances of paragraph content in a document (e.g. sourcecode annotations) (Clause 9.6.3).

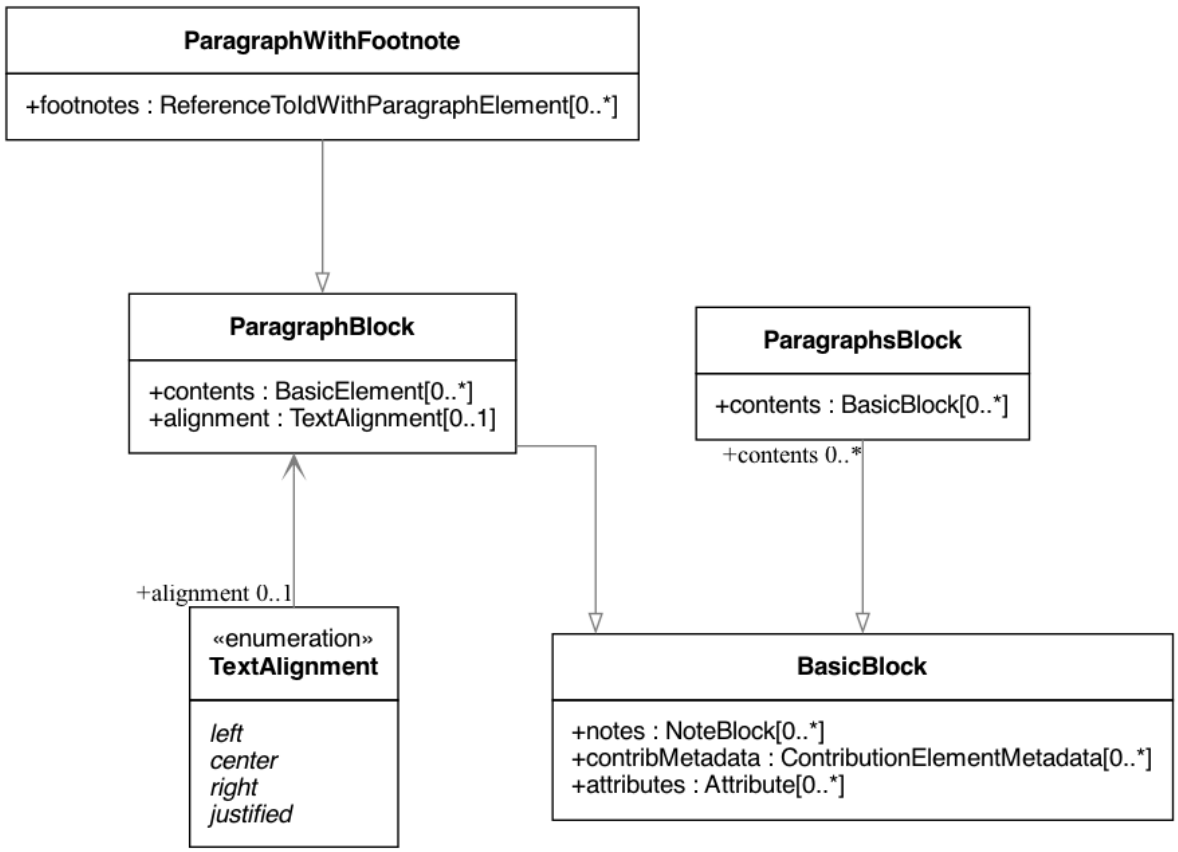

Figure 5 — Basic Document model: Paragraph

9.3. Multi-paragraph blocks

9.3.1. General

The following classes of block are modelled as containers of zero or more blocks; the paragraphs of their name are the typical case.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

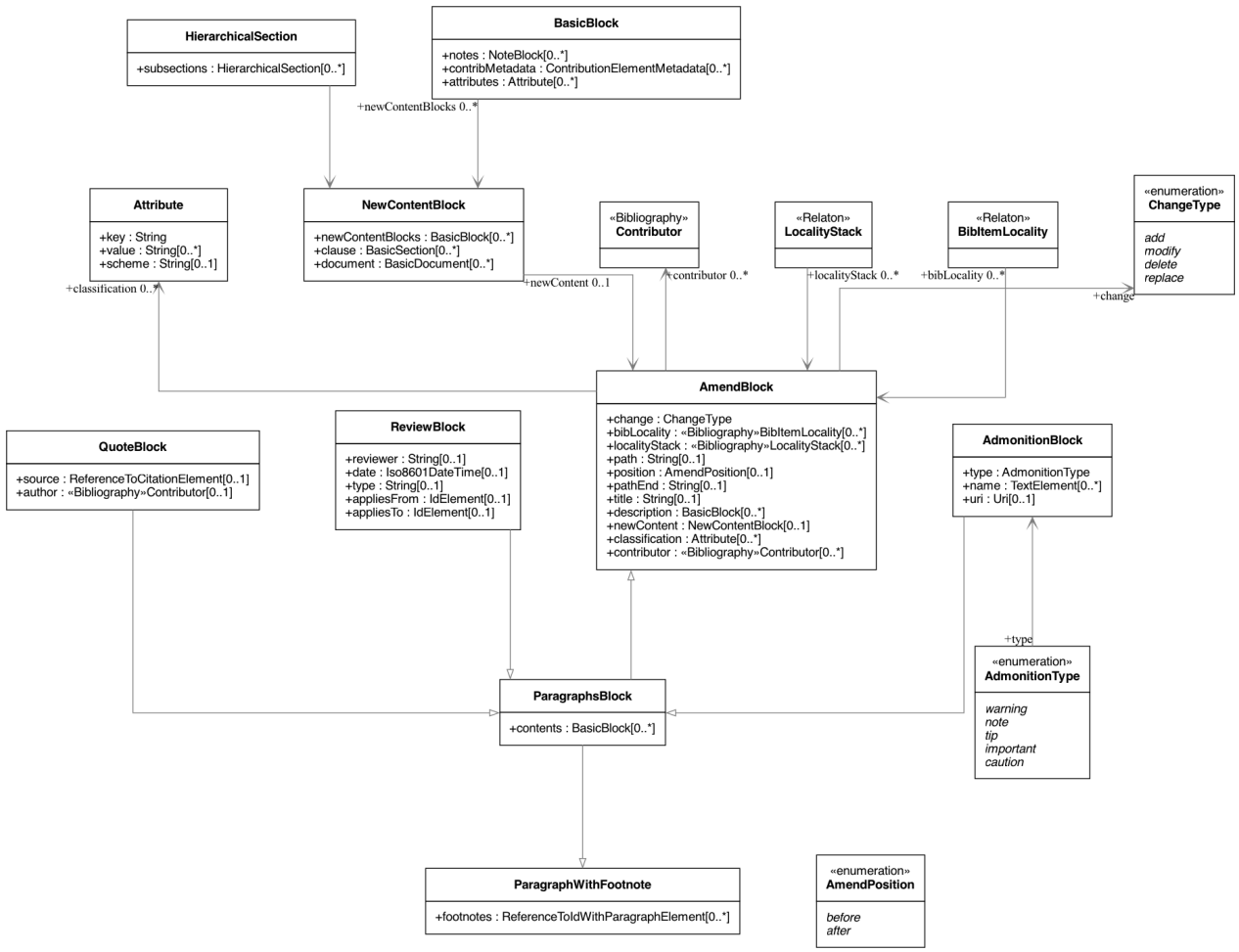

Figure 6 — Basic Document model: Multi-paragraph Block

9.3.2. Blockquote

Blockquote, which also contains an optional bibliographic citation for the quotation (`source`), and an optional `author` of the quotation. The `author` attribute of the blockquote is redundant with the citation, since it restates information about the author that should be recoverable from the citation itself. It is included for convenience, in case processing the citation to extract the author is prohibitive for rendering tools.

9.3.3. Admonition

Admonition, which captures sidebars to the main text conveying particular warnings or supplementary text to the reader. The Admonition block includes a `name` (title), a `class` (subcategory), a `uri` (in case the admonition is available as a separate external document), and a `type` of admonition.

The `type` of admonition determines the rendering of the admonition; the `class` allows different runs of admonitions to be labelled and auto-numbered differently, even if they are of the same type. The types defined for the *BasicDocument* model are *warning, note, tip, important,* and *caution* (in rendering, distinct admonition types are often associated with distinct icons or rendering). Further kinds, such as *statement*, *danger*, *error,* or *hint,* are provided by specializations of this model, not by this model itself.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

NOTE 1 Statement is intended for typographically separate statements in mathematics, such as propositions, proofs, or theorems. Statement conflates all of these for rendering, while Proposition, Proof, Theorem etc. can be treated as distinct classes.

NOTE 2 Admonition notes are modelled to be distinct from notes under sections or blocks.

9.3.4. Review

Review, which is intended to capture reviewer comments about some text in the document. The Review block includes the following attributes: an optional string identifying the `reviewer` who offered the comment; an optional `date` when the comment was made; an optional `type` of reviewer comment; an optional identifier for the start of the text to which the comment applies (`appliesFrom`), and an optional identifier for the end of the text to which the comment applies (`appliesTo`). If `appliesFrom` is absent, the comment applies in the vicinity of the place it has been inserted into the text.

NOTE If the `appliesTo` identifier of a Review block is absent, the comment applies only to the span of text identified by the `appliesFrom` identifier; if it is present, the comments applies to the span of text between the start of the span of text identified by `appliesFrom`, and the end of the span of text identified by `appliesTo`.

9.4. Table

9.4.1. General

Tables are modelled following the same principles as HTML tables.

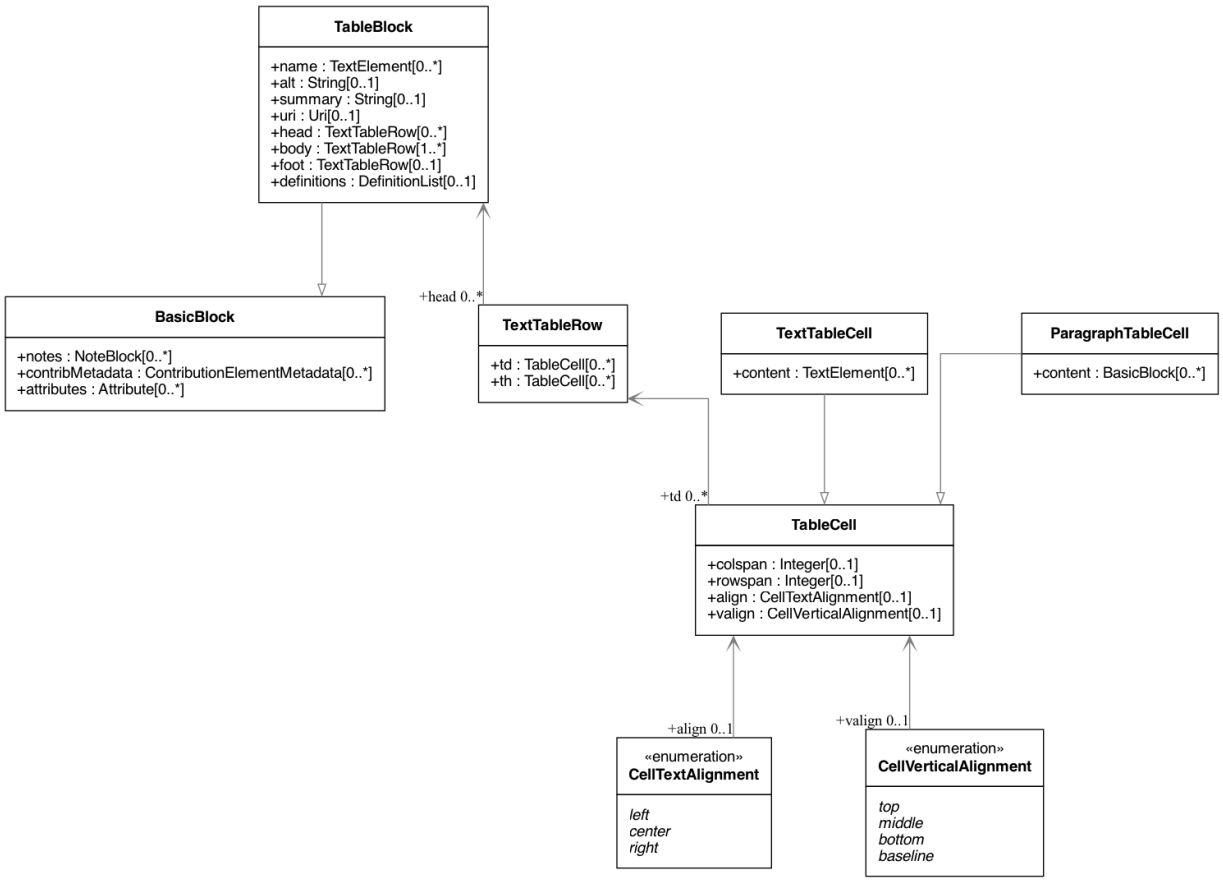

Figure 7 — Basic Document model: Table

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

They contain the following elements:

`name` An optional label for the table.

`head` Zero or more table rows constituting the table header.

`body` One or more table rows constituting the table body.

`foot` Zero or more table rows constituting the table footer.

`definitions` An optional definitions list (Clause 9.5) defining any symbols used in the table. (This reflects practice in standards documents such as ISO/IEC DIR 2:2018.)

`alt` Alternate text to be provided for accessibility purposes, in case the table cannot be rendered accessibly.

`summary` Alternative more extensive summary of table to be provided for accessibility purposes, in case the table cannot be rendered accessibly.

`uri` a URI (in case the table is available as a separate external document),

`unnumbered` An optional boolean attribute indicating that the table should be excluded from any automatic numbering of tables in the document.

`subsequence` A token indicating that all assets with the same subsequence token are to be autonumbered in the same subsequence (e.g. as 2a, 2b, 2c... rather than as 2, 3, 4...)

9.4.2. Table rows

Table rows are defined as a sequence of zero or more header cells and data cells (corresponding to HTML `th` and `td`), both classes being instances of table cells.

9.4.3. Table cells

Table cells contain either zero or more text elements (Clause 10.2), footnotes included, or zero or more blocks (Clause 9) — the cell content model is relaxed, matching the container rule of Clause 4; markup that admits block content in cells (e.g. AsciiDoc `a|` cells) maps directly. In addition, they have the following optional rendering attributes, which are aligned with HTML:

`colspan` Number of columns in the underlying table grid which the cell spans.

`rowspan` Number of rows in the underlying table grid which the cell spans.

`align` Textual alignment of the cell.

`valign` Vertical alignment of the cell.

9.5. List

Lists are modelled following the same principles as HTML lists. All lists contain zero or more *list items*, which consist of an optional identifier (`id`), and zero or more blocks (Clause 9) — list item content is relaxed, so markup whose items contain arbitrary blocks (loose Markdown lists, AsciiDoc continuations, RST items) maps directly. The identifier allows individual list items to be cross-referenced within the document.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Three subclasses of List are modelled.

— *Unordered lists* are equivalent to the List base class.
— *Ordered lists* are Lists with a `type` attribute, describing the kind of numeration applied to the List; the values allowed under the *BasicDocument* model are *roman, alphabet, arabic, romanUpper, alphabetUpper*, corresponding to lowercase Roman numerals, lowercase alphabetic letters, Arabic numerals, uppercase Roman numerals, and uppercase alphabetic letters.
— *Definition lists* override the definition of the List Item to be a pair of `item` (zero or more text elements: Clause 10.2) and `definition` (zero or more paragraphs with footnotes: Clause 9.2).

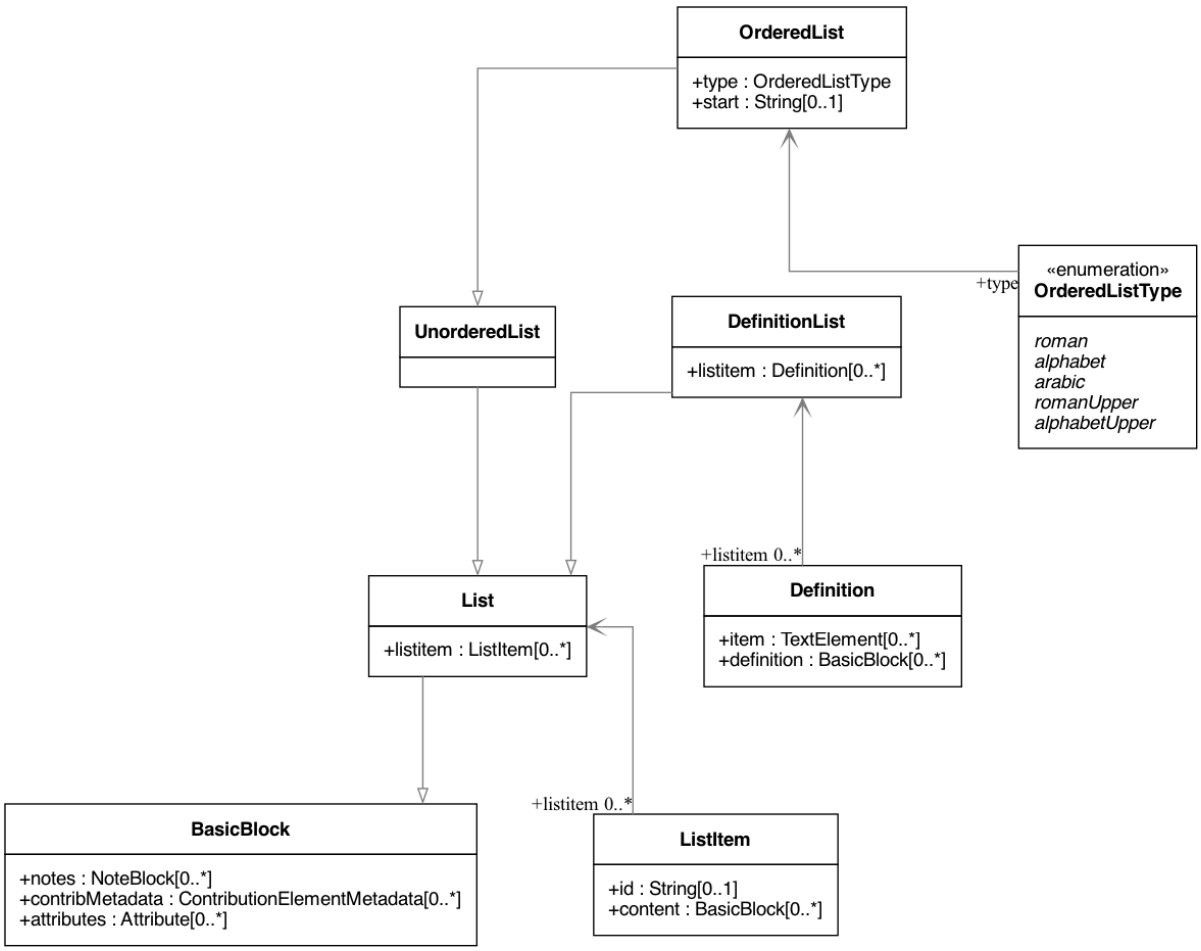

Figure 8 — Basic Document model: List

9.6. Ancillary blocks

9.6.1. General

Functionally, figures, sourcecode, formulas, pre-formatted blocks and examples all play a similar role, as providing illustrative content that is ancillary to the main content. However each class has its own particular structure.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

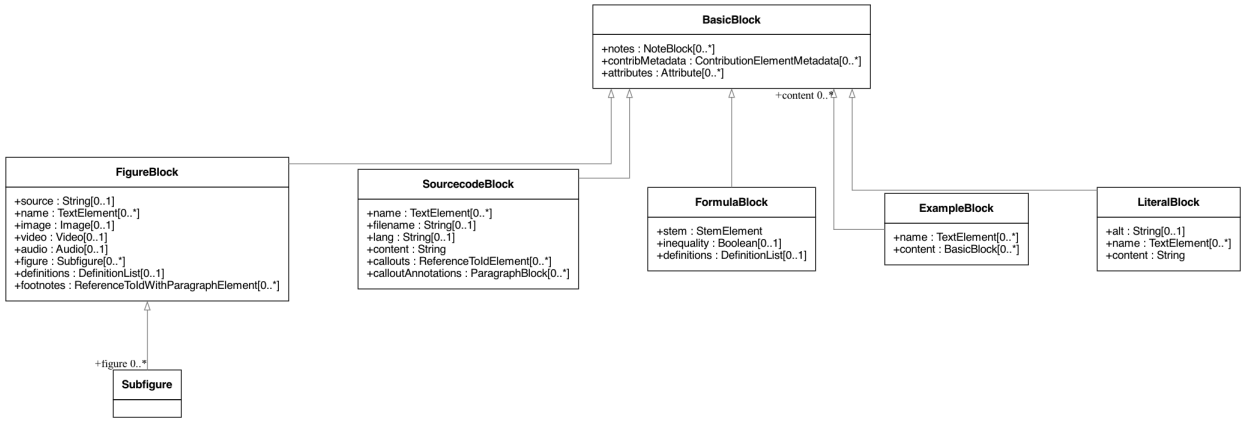

Figure 9 — Basic Document model: Figure, Sourcecode, Formula, Example

9.6.2. Figure

Figures are wrappers for images, and may themselves contain figures (*Subfigure* class). They contain the following elements, all of which are optional:

`name` A label for the figure.

`class` A class for the figure; this is to allow different classes of figure (e.g. *Plate*, *Chart*, *Diagram*) to be auto-numbered and captioned differently.

`image` An image file (Clause 10.4).

`video` A video file (Clause 10.4).

`audio` An audio file (Clause 10.4).

`source` A URI or other reference intended to link to an externally hosted image (or equivalent).

`definitions` An optional definitions list (Clause 9.5) defining any symbols used in the figure.

NOTE 1 This reflects practice in ISO/IEC DIR 2:2018.

`footnotes` Optional footnotes specific to the figure. (This reflects practice in ISO/IEC DIR 2:2018.)

`figure` Zero or more embedded figures.

NOTE 2 This reflects practice in e.g. ISO/IEC DIR 2:2018, and subfigures are intended to be mutually exclusive with `image`, `source`: the latter are intended for leaf node figures.

`unnumbered` An optional boolean attribute indicating that the figure should be excluded from any automatic numbering of figures in the document.

`subsequence` A token indicating that all assets with the same subsequence token are to be autonumbered in the same subsequence (e.g. as 2a, 2b, 2c... rather than as 2, 3, 4...)

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

9.6.3. Sourcecode

Sourcecode blocks are wrappers for computer code or comparable text. They contain the following elements:

`name`	A label for the source code.
`filename`	A file name associated with the source code (and which could be used to extract the source code fragment to from the document, or to populate the source code fragment with from the external file, in automated processing of the document).
`lang`	The computer language or other notational convention that the source code is expressed in.
`content`	The computer code or other such text presented in the block, as a single unformatted string. (The string should be treated as pre-formatted text, with whitespace treated as significant.)
`callouts`	Zero or more cross-references (Clause 10.5); these are intended to be embedded within the `content` string, and link to annotations.
`calloutAnnotations`	These are annotations to the source code; each annotation consists of zero or more paragraphs, and is intended to be referenced by a callout within the source code.
`unnumbered`	An optional boolean attribute indicating that the sourcecode block should be excluded from any automatic numbering of sourcecode blocks in the document.
`subsequence`	A token indicating that all assets with the same subsequence token are to be autonumbered in the same subsequence (e.g. as 2a, 2b, 2c... rather than as 2, 3, 4...)

9.6.4. Formula

Formula blocks are wrappers for mathematical or other formulas. They contain the following elements:

`stem`	A STEM element (Clause 10.2), constituting the content of the formula
`definitions`	An optional definitions list (Clause 9.5) defining any symbols used in the formula.

NOTE This reflects practice in ISO/IEC DIR 2:2018.

`unnumbered`	An optional boolean attribute indicating that the formula should be excluded from any automatic numbering of formulas in the document.
`subsequence`	A token indicating that all assets with the same subsequence token are to be autonumbered in the same subsequence (e.g. as 2a, 2b, 2c... rather than as 2, 3, 4...)
`inequality`	An optional boolean attribute indicating that the formula is to be labelled as an Inequality, if inequalities are differentiated from equations.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

9.6.5. Pre-formatted Blocks

Pre-formatted blocks are wrappers for text to be rendered with fixed-width typeface, and preserving spaces including line breaks. They are intended for a restricted number of functions, most typically ASCII Art (which is still in prominent use in some standards documents), and computer output. In most cases, Sourcecode blocks (Clause 9.6.3) is more appropriate in markup, as it is more clearly motivated semantically.

It contains the following elements (which are a subset of the elements of Sourcecode blocks):

`name` A label for the pre-formatted text.

`content` The pre-formatted text presented in the block, as a single unformatted string. (Whitespace is treated as significant.)

9.6.6. Example

Example blocks are wrappers for open-ended example content: they contain zero or more blocks (Clause 9), an extension point that higher layers may extend additively. They also contain:

`unnumbered` An optional boolean attribute indicating that the example should be excluded from any automatic numbering of examples in the document.

`subsequence` A token indicating that all assets with the same subsequence token are to be autonumbered in the same subsequence (e.g. as 2a, 2b, 2c... rather than as 2, 3, 4...)

`name` A label for the example.

9.7. Amend

The *AmendBlock* describes a change in a document for human readers, as opposed to the machine-readable *Change* models (Clause 12).

`change` the type of change described: one of *add, modify, delete, replace* (*ChangeType*).

`bibLocality`, `localityStack` the location in the original document which has undergone the change, as bibliographic localities (CalConnect CC 6900:2019).

`path`, `pathEnd` the span within the location where the change applies, if the location defines a container larger than the scope of the change.

`title` an optional caption of this block.

`description` zero or more blocks describing the change.

`newContent` a *NewContentBlock* (see below).

`classification` zero or more key-value classifications of the change, as attribute register entries.

`contributor` zero or more contributors responsible for the change (CalConnect CC 6900:2019).

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

`position` for an *add* change, whether the change is added *before* or *after* the location.

9.7.1. New content

The *NewContentBlock* is the composition container for amend content: it carries zero or more blocks (`newContentBlocks`), zero or more sections (`clause`), and zero or more entire documents (`document`) — a document is a block (see Clause 4), so whole documents compose as child content.

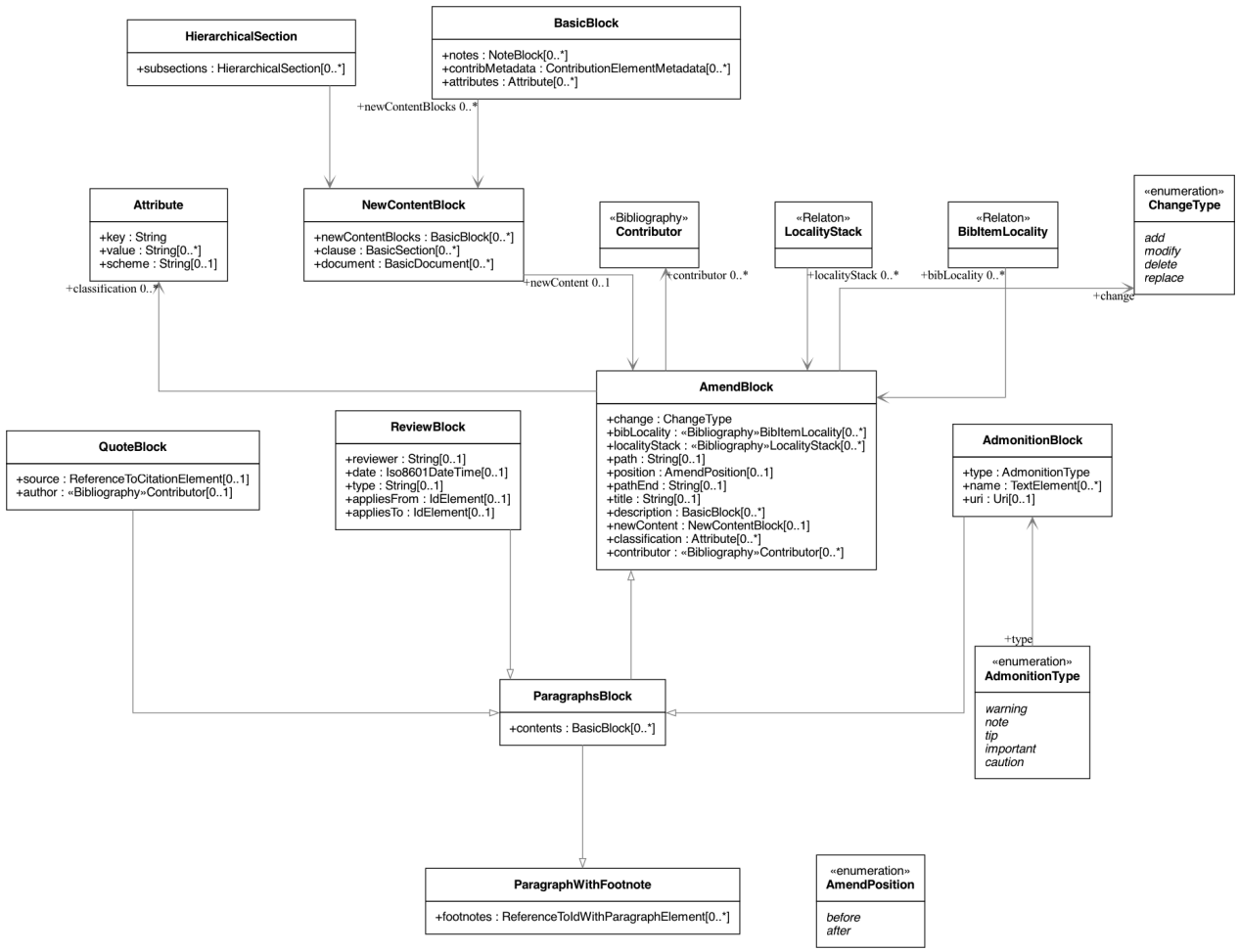

Figure 10 — Basic Document model: Multi-paragraph and Amend Blocks

10. Inline element models

10.1. General

Inline elements represent the components of text blocks. They are modelled in the *BasicDocument* model as *Basic Elements*.

All Basic Elements have the following attribute:

`contribMetadata` attribution of the element to a specific contributor (Clause 11.2), to be used in change management of documents.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Three subclasses of Basic Elements are modelled:

— Text elements, which contain text and associated formatting information, but which do not contain any associated identifiers. (Clause 10.2)
— ID elements, which contain identifiers. (Clause 10.4)
— Reference elements, which contain references to identifiers. (Clause 10.5)

Footnotes (Clause 10.6) are a special case of Reference element, which are not included under Basic Elements because of the need to exclude them from certain classes of paragraph (Clause 9.2).

10.2. Text Elements

The modelling of text elements is substantially derived from HTML, and encompasses semantically significant formatting of text. For example, it encompasses italics and boldface (under their semantic guise of `emphasis` and `strong`); but it omits different sizes of font, as information that is typically semantically predictable, and relegated to stylesheets.

All text elements contain a localized string (Clause 11).

The following elements indicate formatting, and have no further attributes:

— Monospace (corresponding to HTML `tt`, `code`)
— Keyword
— Emphasis (corresponding to HTML `em`, `i`)
— Strong (corresponding to HTML `strong`, `b`)
— Superscript (corresponding to HTML `sup`)
— Subscript (corresponding to HTML `sub`)
— Strike (corresponding to HTML 4 `s`)
— Underline (corresponding to HTML 4 `u`)
— Small Caps

The following elements are used for Ruby annotations in East Asian languages, and correspond to the HTML 5 `ruby`, `rt`, `rp` elements:

— *RubyElement* (the annotation base)
— *RubyPronunciation* (`rt`)
— *RubyAnnotation* (`rp`)

All strings are Unicode text. Directionality and line-breaking rules are rendering concerns, relegated to stylesheets and rendering processors; where an author must assert text direction explicitly, the `dir` attribute register key carries the assertion. Unicode normalization is a serialization concern and is not constrained by this model.

Text elements are distinguished from *Pure Text Elements*, which are restricted to contain only other pure text elements — no identifiers and no references — so that consumers that process plain runs of text can rely on the distinction.

Text Elements also include the STEM element, representing mathematical and other formulas. This consists of a `stemType`, indicating which language is used to express the formula, and the content (`stemValue`) of the formula itself. The *BasicDocument* model allows AsciiMath, MathML, and LaTeX in STEM elements.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

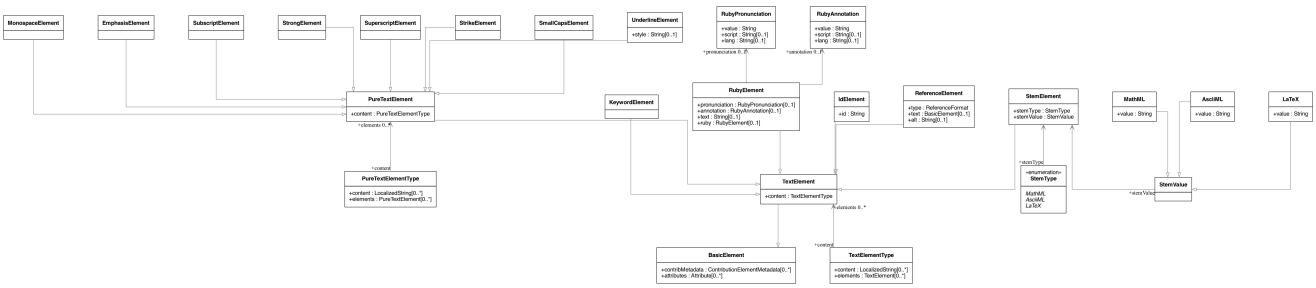

Figure 11 — Basic Document model: Text Elements

10.3. Empty Elements

The following elements are subclasses of Basic Element that contain no text, and are intended to represent semantically significant formatting elements:

— Line Break (corresponding to HTML `br`)
— Page Break
— Horizontal Rule (corresponding to HTML `hr`)
— Index Term
  — Contains Primary, and optionally Secondary and Tertiary index terms to be associated with that location in the next, but no textual content.
  — Contains optional attribute `to`, a reference to an ID element (Clause 10.5), to indicate that the index range covers a range of locations; the attribute should reference a bookmark.

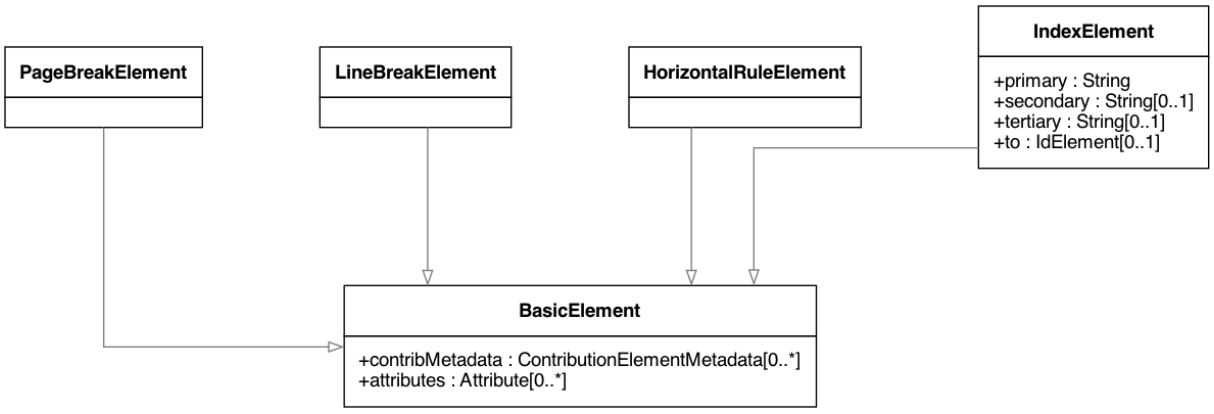

Figure 12 — Basic Document model: Empty Elements

10.4. ID Elements

ID Elements are inline elements that have an identifier (`id`), which permits them to be cross-referenced by Reference Elements (Clause 10.5). ID Elements in the *BasicDocument* model do not contain text.

There are two subclasses of ID Elements in the *BasicDocument* model:

— *Bookmarks* are intended as anchors for cross-references which do not have scope over blocks or sections. Anchors within a block under the *BasicDocument* model cannot span across a number of inline elements; bookmarks are intended as point anchors. For that reason, the

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Review block (Clause 9.3.4) has a starting reference and an optional ending reference, which can be bookmarks as well as block or section references.

— *Media* are containers for media content. They have the following attribues:
  - `source` indicating the URI of the media file
  - `filename` indicating a file name corresponding to the media, to which the media can be extracted if it is represented inline (e.g. in Base64 encoding in the URI)
  - `type` indicating the type of the image file; the Basic Document model leaves the text to be used here open, but recommends the use of MIME types (IETF RFC 2045)
  - `alt` alternate text, supplied for accessibility
  - `title` title, supplied for accessibility
  - `longdesc` URI pointing to more extensive alternate text description, supplied for accessibility

Media files are of three subtypes, each of which has its own element name:

— *Image* is for image files, and has the following additional attributes
  - `height` optional attribute (Clause 11): a real number with an optional percent sign, or “auto”
  - `width` optional attribute (Clause 11): a real number with an optional percent sign, or “auto”

— *Audio* is for audio files, and has the following additional attributes
  - `altsource` zero or more specifications of alternative files to use as media. These specifications in turn consist of an optional `filename`, a `source`, and a `type`, as with the parent *Media* class

— *Video* is for video files, and has the following additional attributes
  - `altsource` zero or more specifications of alternative files to use as media. These specifications in turn consist of an optional `filename`, a `source`, and a `type`, as with the parent *Media* class
  - `height` optional attribute, which can be an integer or “auto”
  - `width` optional attribute, which can be an integer or “auto”

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

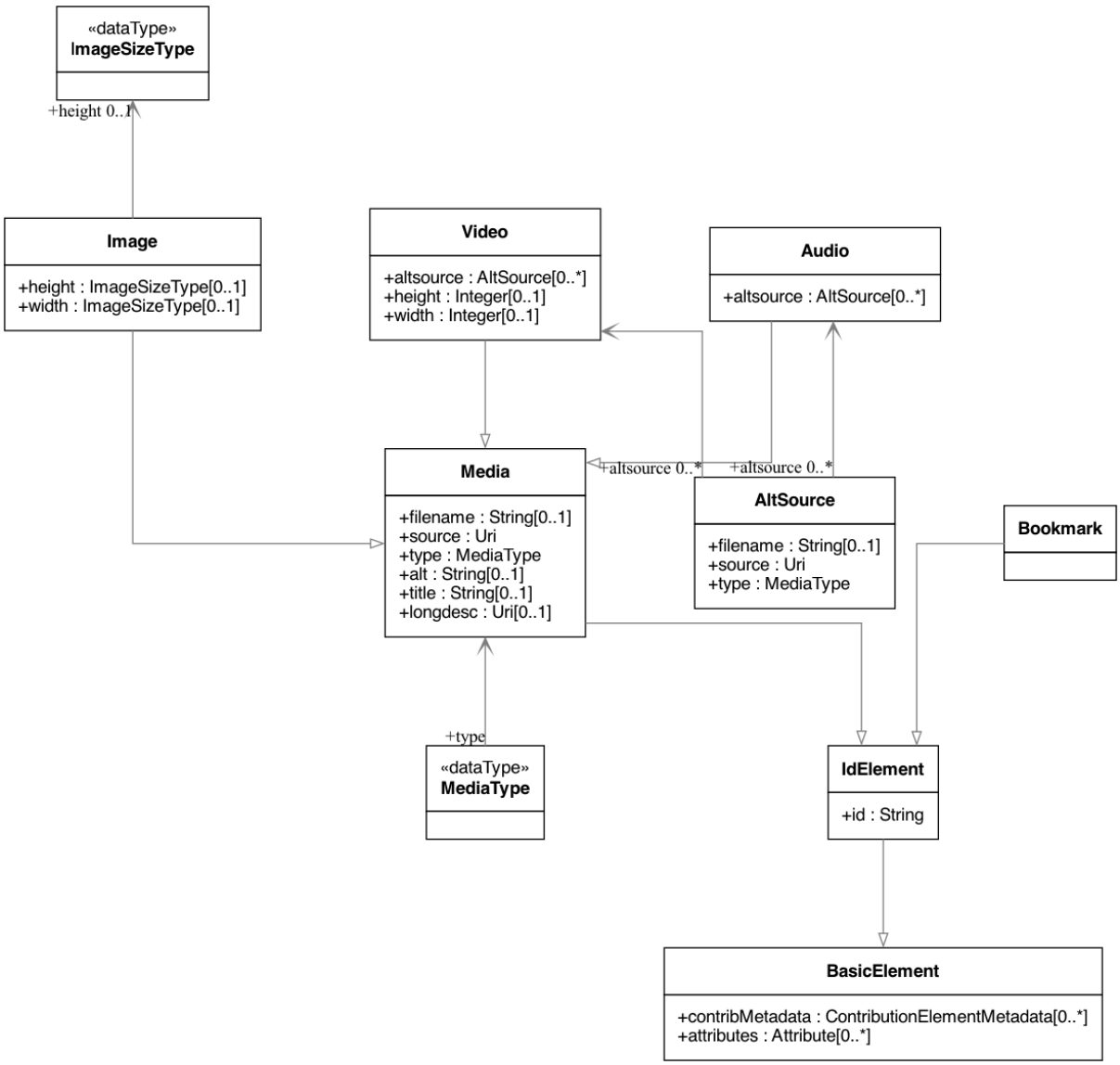

Figure 13 — Basic Document model: ID Elements

10.5. Reference Elements

Reference Elements are inline elements which reference other elements in the document, or other documents. All Reference Elements are modelled as containing the following attributes:

`text` The optional, unformatted textual content of the reference element.

`type` The type of Reference Element, prescribing how it is to be rendered. The *BasicDocument* model recognises four types: *inline* (referencing another element in the same document), *external* (referencing an external document), *footnote* (an inline reference to be rendered as a footnote), and *callout* (an inline reference to be rendered as a callout: Clause 9.6.3).

`alt` Alternate text, used for accessibility.

The following subclasses of Reference Elements are modelled.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

— Reference to Link Element: An external reference, whose `target` is defined as a URI. An optional `alt` attribute is also permitted, summarising the link content for accessibility.
— Reference to Citation Element: An external reference to a bibliographic entity, as modelled in CalConnect CC 6900:2019 as a *citation*. In addition to the attributes of *citation*, the reference has an optional `normative` attribute (which may be used by those standards which differentiate normative and informative references), and optional `citeAs` attributes prescribing how the bibliographic citation should be rendered in the text.
— Reference to ID Element: An internal reference, whose `target` corresponds to the identifier of a section, block or ID Element within the current document.
— Reference to Index Element: Contains Primary, and optionally Secondary and Tertiary index terms, to be cross-referenced to an alternate index entry, either as “see” or “see also”, depending on an `also` attribute

The Reference to ID Element class in turn has the following subclasses modelled:

— Callout, for which the `type` is set to *callout*, and the `text` is constrained to be a single mandatory string. The target of the callout is understood to be the location of the callout within the source code; the extent of the target is not expressed overtly.
— Reference To ID With Paragraph Element, which associates both `text` and `content` to the cross-reference; the `content` is a sequence of one or more paragraphs (Clause 9.2).

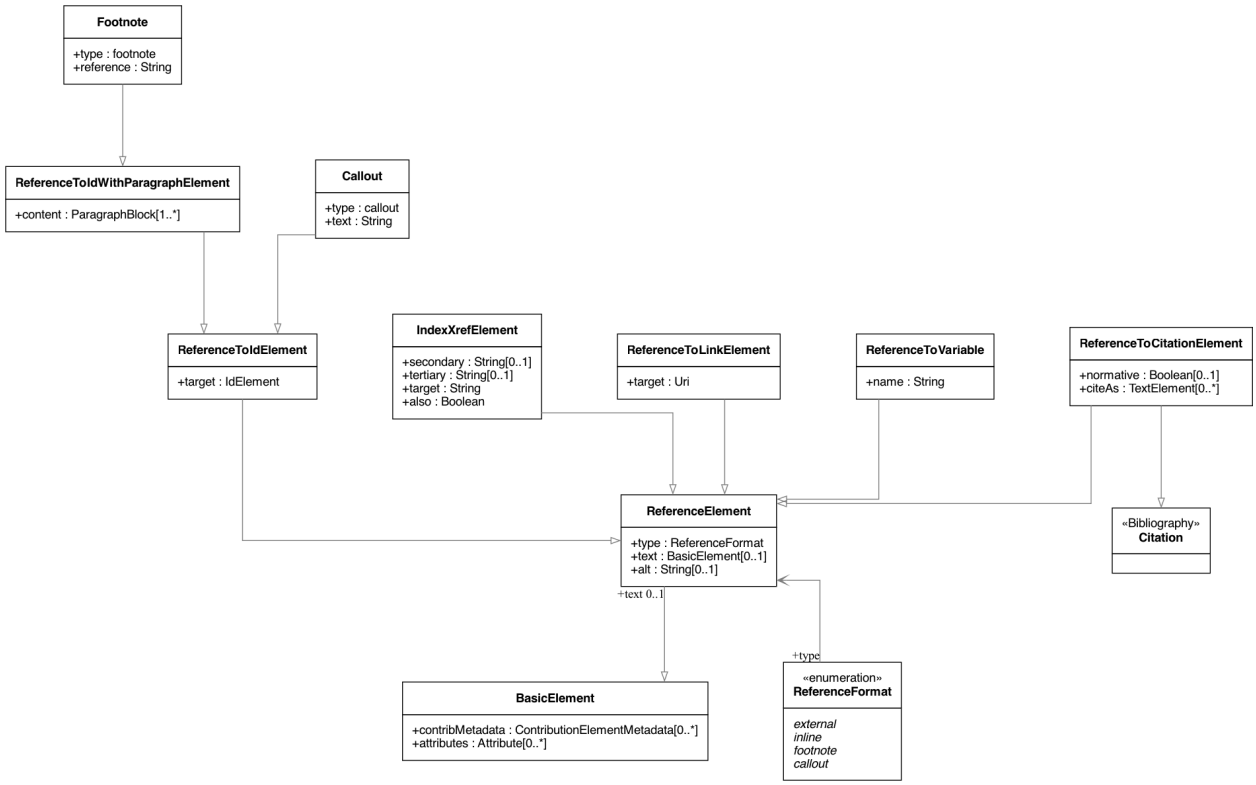

Figure 14 — Basic Document model: Reference Elements

10.5.1. Reference to Variable

The *ReferenceToVariable* element references a variable: it resolves, at processing time, to the value registered under its `name` in the attribute register (Clause 4). It is the model home for attribute references (AsciiDoc `{attr}`), substitutions (RST `|substitution|`), and template variables.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

10.6. Footnote

Footnotes are modelled as a subclass of Reference To ID With Paragraph Element, which constrain their `type` to be *footnote*. The `text` attribute is the footnote reference, and the `content` attribute is the footnote contents. The target of the footnote is understood to be the location of the footnote within the text; the extent of the target is not expressed overtly.

NOTE Endnotes are not modelled separately from footnotes in the *BasicDocument* model, and the use of footnotes and endnotes as realisations of annotations are normally stylistic alternatives, which would be relegated to a stylesheet.

11. Data type models

11.1. Basic Data Types

The following basic types are used in the definition of the *BasicDocument* model.

— String
— Attribute register entries: key-value pairs with an optional qualifying scheme (see Clause 4)
— Image sizes: a real number with an optional percent sign, or “auto”
— URI, as defined in IETF RFC 3986.
— Country codes, as defined in ISO 3166.
— Language names, as defined in ISO 639.
— Dates and times, as defined in ISO 8601-1:2019.
— Hash algorithms, as defined in ISO/IEC 10118.
— Digital signature algorithms, as defined in ISO/IEC 14888.
— Script names, as defined in ISO 15924.
— Localized Strings, specifying a String with optional language (ISO 639), script (ISO 15924), and country (ISO 3166) attributes. A language-script(-country) combination identifies a *spelling system* as registered under ISO 24229, which also registers *system codes* for written-language conversion systems (e.g. romanization and transliteration schemes); a conversion system applied to content may be identified through such a system code.
— Formatted Strings, specifying a Localized String with a `format` attribute, in order to admit markup into Strings (whether XML-based or not).

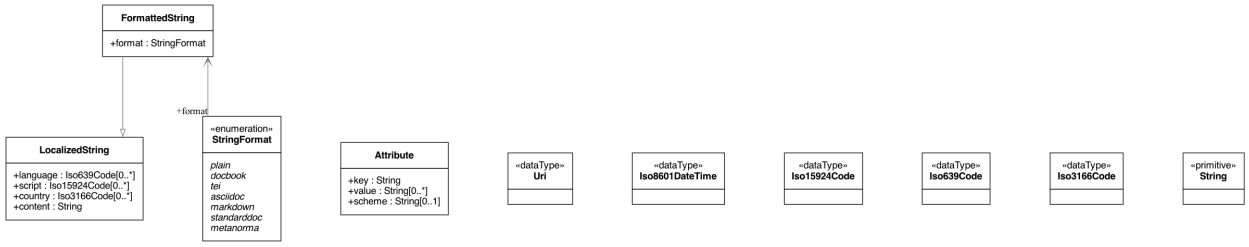

Figure 15 — Basic Document model: Data Types

11.2. Contribution Element Metadata

In addition to the Basic Data Values, the *BasicDocument* model defines a container encoding the contribution made by a party towards a particular element in the document, with the following attributes:

`dateTime` The date and time when the contribution was made.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

`contributor` The party who made the contribution, as described through the `contributor` element in CalConnect CC 6900:2019.

`integrityValue` zero or more *IntegrityValue* entries, digital signatures of the contribution.

The *IntegrityValue* class consists of a `hash` value and a `signature`, with an associated `publicKey`; hash algorithm identifiers follow ISO/IEC 10118 and digital signature algorithm identifiers follow ISO/IEC 14888. This document does not specify security mechanisms, protocols, or verification procedures: integrity values carry evidence whose semantics are defined by the referenced standards.

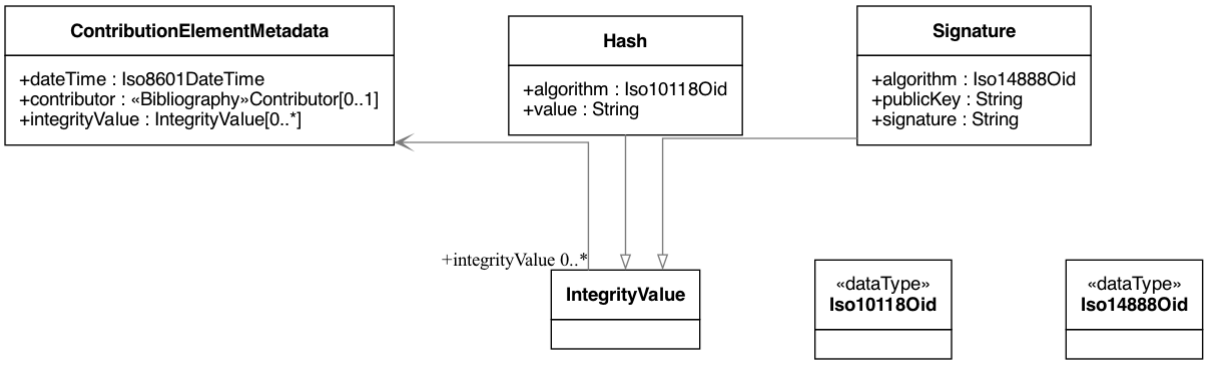

Figure 16 — Basic Document model: Contribution Element Metadata

12. Change models

12.1. General

The change models are provided in *BasicDocument* to enable changes to be applied incrementally towards a *BasicDocument* in a defined manner.

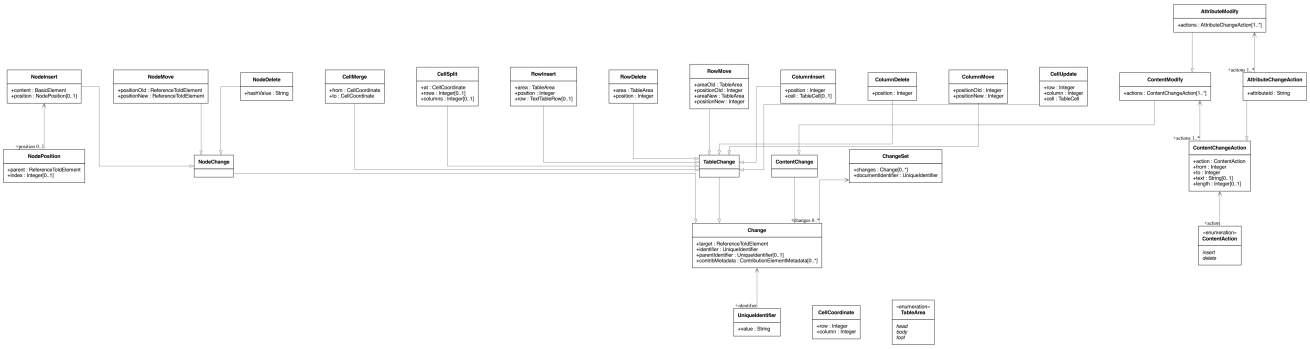

Figure 17 — Basic Document model: Changes

The change models are distinct from in-document change markup (Clause 9.7): a change set carries operational changes to be applied to a document, while amend markup records, inside a document, changes that have been applied to an earlier version of it. An applied change may additionally be recorded as amend markup.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.2. Addressing model

Changes locate their target and express their effect using three addressing modes:

— **identifier reference**: the `target` attribute of the *Change* model identifies an element of the document by the well-known `id` register key (see Clause 4). Elements that are the target of changes SHOULD carry an `id` register entry, so that they remain addressable across revisions.
— **ordinal position**: a *NodePosition* locates a point in a child sequence as a container node and a zero-based ordinal; table changes locate rows, columns and cells in the grid of the table (see Clause 12.9).
— **text cursor range**: a *ContentChangeAction* locates a textual edit by the zero-based character positions `from` and `to` within the textual content of an element or of a register entry.

Repeated values — the `value` set of a register entry, or repeated attributes such as language tags — are addressed as an ordered sequence: values are inserted or removed by ordinal or by cursor range, in the same way as ordered children.

12.3. Change

The *Change* model defines an action to be performed on an element within *BasicDocument*.

It contains the following attributes:

`target`	The element that this action should be applied to.
`identifier`	A unique identifier of this change.
`parentIdentifier`	zero or more unique identifiers of *Change* objects, that this change is supposed to follow after; a change with no parent is a root of a change sequence.
`contribMetadata`	Metadata of the contributor, see Clause 11.2.

12.4. Change set

The *ChangeSet* model defines a collection of *Change* data, and specifies a unique identifier that identifies the *BasicDocument* where this *ChangeSet* should be applied to.

`changes`	The set of *Change* data.
`documentIdentifier`	The unique identifier that identifies the *BasicDocument* where this *ChangeSet* should be applied to.

12.5. Unique identifier

The unique identifier is used to uniquely identify a *BasicDocument*.

It contains the attribute:

`value`	A string that uniquely identifies a *BasicDocument*.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.6. Content change

12.6.1. General

The *ContentChange* model defines possible actions that involve modification of content within a *BasicDocument* data element.

12.6.2. Content change action

The *ContentChangeAction* model is used to indicate the actual content changes that applies to the specified portion of textual content. This is used both by the *ContentModify* and *AttributeModify* models as their content are treated as pure text.

It provides the following attributes:

`action` A *ContentAction* value, where it could be either `insert` or `delete`, indicating text to be inserted or deleted from the content.

`from` An `Integer` that specifies the beginning cursor position of a textual change.

`to` An `Integer` that specifies the ending cursor position of a textual change.

`text` In the case of an `insert`, a `String` to be inserted or replace the substring referred to by `from` to `to`.

`length` In the case of a `delete`, an `Integer` to indicate how many characters to be removed from the `from` position. In the case of an `insert`, an `Integer` to indicate the length of the `text` attribute.

12.6.3. Content modify

The *ContentModify* class provides a container for a multiple *ContentChangeAction* data.

It has the following attribute:

`actions` One or more `ContentChangeAction` data

12.7. Attribute change

The *AttributeChange* model defines possible actions that involve modification of an attribute within a *BasicDocument* data element.

12.7.1. Attribute change action

Similar to *ContentChangeAction* which it inherits from, the *AttributeChangeAction* class is used to specify an actual change to an attribute.

It has the following attribute:

`attributeId` A `String` that identifies the attribute where the attribute change should apply to.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.7.2. Attribute modify

The *AttributeModify* class provides a container for a multiple *AttributeChangeAction* data.

It has the following attribute:

`actions` One or more `AttributeChangeAction` data

12.8. Node change

12.8.1. General

The *NodeChange* model defines possible actions that involve modification of data elements at the node level within a *BasicDocument*.

The `target` attribute inherited from the *Change* model indicates the node this *NodeChange* action applies to.

12.8.2. Node position

The *NodePosition* model locates an insertion point in the document tree: a container node and a zero-based ordinal among its children.

It has the following attributes:

`parent` A *ReferenceToIdElement* that indicates the node whose child sequence the position refers to.

`index` A zero-based ordinal among the children of the parent node. When omitted, the position refers to the end of the child sequence.

12.8.3. Node insert

The *NodeInsert* class specifies the insertion of a data node in a *BasicDocument*.

It has the following attributes:

`content` A data element conforming to *BasicElement* to be inserted into the specified *BasicDocument*.

`position` A *NodePosition* that indicates where the content is to be inserted. When omitted, the content is appended to the end of the child sequence of the target node.

12.8.4. Node delete

The *NodeDelete* class specifies the deletion of a data node in a *BasicDocument*.

It has the following attributes:

`hashValue` An optional string that contains the hash value of the node to be deleted for verification purposes.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.8.5. Node move

The *NodeMove* class specifies moving of a particular node in a *BasicDocument* to another location within the same *BasicDocument*.

It has the following attributes:

`positionOld` A `ReferenceToIdElement` that indicates the position of the node's parent. While this seems redundant to the `target` attribute inherited from the *Change* model, it is useful for verifying that the location has not changed.

`positionNew` A `ReferenceToIdElement` that indicates the new parent or sibling of the node.

12.9. Table changes

12.9.1. General

The *TableChange* model defines possible actions that modify the structure or the content of a table: insertion, deletion and reordering of rows and columns, and changes to individual cells, including merging and splitting. The `target` attribute inherited from the *Change* model indicates the *TableBlock* that the action applies to.

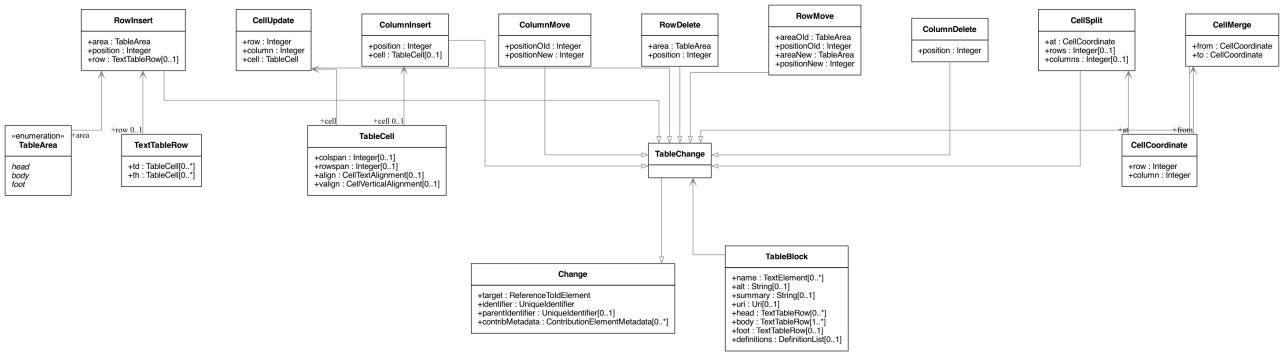

Figure 18 — Basic Document model: Table changes

Row and column ordinals index the **grid** of the table, not the sequence of cells in a row: a cell that spans multiple slots through `colspan` or `rowspan` occupies all of the covered slots, and grid coordinates refer to the origin slot of a cell. Row operations address one of the row containers (head, body or `foot`) and the ordinal within that container; cell operations address grid coordinates across the whole table.

12.9.2. Table area

The *TableArea* enumeration identifies the row container of a table that a row-level change applies to.

head	The table header rows.
body	The table body rows.
`foot`	The table footer rows.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.9.3. Cell coordinate

The *CellCoordinate* model locates a cell in the grid of a table by the grid row and grid column of its origin slot. Indexes are zero-based.

It has the following attributes:

`row` Zero-based index of the grid row of the cell's origin slot.

`column` Zero-based index of the grid column of the cell's origin slot.

12.9.4. Row insert

The *RowInsert* model specifies the insertion of a row into a table. Rows at or after the insertion position shift down by one position within the same row container.

It has the following attributes:

`area` A *TableArea* value indicating the row container that the row is inserted into.

`position` Zero-based ordinal within the row container. Rows at or after this position shift down by one.

`row` A *TextTableRow* to be inserted. When omitted, an empty row sized to the grid of the table is inserted.

12.9.5. Row delete

The *RowDelete* model specifies the deletion of a row from a table. Rows after the deleted position shift up by one within the same row container.

It has the following attributes:

`area` A *TableArea* value indicating the row container that the row is deleted from.

`position` Zero-based ordinal of the row within the row container.

12.9.6. Row move

The *RowMove* model specifies moving a row of a table to another position, possibly into another row container. Other rows shift accordingly in both the old and the new container.

It has the following attributes:

`areaOld` The row container that the row is moved from.

`positionOld` Zero-based ordinal of the row within its old container. Useful for verifying that the location has not changed.

`areaNew` The row container that the row is moved to.

`positionNew` Zero-based ordinal of the row within its new container.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

12.9.7. Column insert

The *ColumnInsert* model specifies the insertion of a column into a table. Occupied grid slots at or after the insertion column shift right by one in every row; each row that has a slot at the insertion column gains a cell there.

It has the following attributes:

`position` Zero-based index of the grid column to insert the column at.

`cell` A *TableCell* template used as the initial content of the cells created in the new column. When omitted, empty text cells are created.

12.9.8. Column delete

The *ColumnDelete* model specifies the deletion of a column from a table. Cells that originate in the deleted grid column are removed; cells that span it have their column span decremented by one. Occupied slots after the deleted column shift left by one in every row.

It has the following attributes:

`position` Zero-based index of the grid column to be deleted.

12.9.9. Column move

The *ColumnMove* model specifies moving a column of a table to another grid position. Cells move with their column; the affected cells of other columns shift accordingly.

It has the following attributes:

`positionOld` Zero-based index of the grid column to be moved. Useful for verifying that the location has not changed.

`positionNew` Zero-based grid column index the column is moved to.

12.9.10. Cell update

The *CellUpdate* model specifies the replacement of a single cell of a table, identified by the grid coordinates of its origin slot. The replacement cell replaces the existing cell wholly: its specialization, content and properties.

It has the following attributes:

`row` Zero-based index of the grid row of the cell's origin slot.

`column` Zero-based index of the grid column of the cell's origin slot.

`cell` A *TableCell* that replaces the existing cell.

12.9.11. Cell merge

The *CellMerge* model specifies merging two adjacent cells of a table: the cell at `from` absorbs the grid slots of the cell at `to`. The cells shall be adjacent on the same grid row (a horizontal merge,

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

increasing the column span) or in the same grid column (a vertical merge, increasing the row span). The content of both cells is concatenated, in document order, into the merged cell.

It has the following attributes:

`from` A *CellCoordinate* for the cell that absorbs the slots of the other cell.

`to` A *CellCoordinate* for the cell whose slots are absorbed; it ceases to exist.

12.9.12. Cell split

The *CellSplit* model specifies splitting a cell of a table that spans multiple grid slots. The cell's span is reduced to the given extents; the freed slots are filled with empty text cells, and the original cell keeps its content in its origin slot.

It has the following attributes:

`at` A *CellCoordinate* for the cell to be split, by the grid coordinates of its origin slot.

`rows` The resulting row span of the cell. When omitted, the span becomes one.

`columns` The resulting column span of the cell. When omitted, the span becomes one.

12.10. Computed content

Because the content model of a table cell is relaxed (see Clause 9.4), a cell may hold any block, including a *FormulaBlock*: a cell whose value is computed rather than literal. The formula notation is carried by the `stemType` of the *StemElement* model; formula languages beyond the stem notations, including spreadsheet-style expressions that reference other cells, are bound as register entries on the formula block, qualified by the `scheme` key of the *Attribute* model.

The model carries the expression itself. The evaluation of expressions, the resolution of cell references, and the recomputation of computed content when referenced content changes are outside the scope of this document.

12.11. Ordering and concurrent changes

Changes are operation records: a *ChangeSet* is applied to the document identified by its `documentIdentifier`, and each *Change* records exactly one operation. Ordering between changes is carried by the `parentIdentifier` attribute: a change follows the changes it names, and a change with no parent is the root of a change sequence. The resulting directed acyclic graph is the causal order in which the operations of a change set are to be applied.

How concurrent changes transform against each other — operational transformation, conflict-free replicated data types, or any other merging strategy — is outside the scope of this document, in the same way that the choice of serialization format is (see Clause 4): this document standardizes the operations and their ordering, not the algorithms that merge them.

EXAMPLE

> Two parties may each issue a change set against the same table, one inserting a column and the other updating a cell. The operations and their parent ordering are exchanged losslessly by these models; aligning the two revisions onto a common state is the concern of the collaboration algorithm, not of the interchange format.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Appendix A
(normative)
Mapping of lightweight text markup constructs

A.1. General

This annex demonstrates that the constructs of the principal lightweight text markup languages map onto the harmonized model of this document: one model, many markups. Each row names the markup construct and its model home. A construct not listed is covered by the conformance formula of Clause 4: an existing construct, a type-specialization of one, or register entries with relaxed or opaque content.

Preprocessing and rendering concerns are out of scope: content inclusion, attribute conditionals, table-of-contents generation, smart punctuation, and citation rendering modes.

A.2. AsciiDoc

Table A.1

AsciiDoc construct	Model home	Notes
Document header (title, author, revision)	`BasicDocument` `bibdata` + register	Document attributes become register entries
section titles	`BasicSection` hierarchy	anchor identifiers and roles via register keys
Paragraph	`ParagraphBlock`	`align` option maps to `alignment`
quote blocks	`QuoteBlock`	attribution/source
admonition paragraphs and blocks	`AdmonitionBlock` + `AdmonitionType`	further kinds (e.g. sidebar labels) as type-specializations
example blocks	`ExampleBlock`	relaxed block content
bullet, ordered, and definition lists	`UnorderedList`, `OrderedList`	continuations map to relaxed `ListItem` content
definition list	`DefinitionList`	
pipe table	`TableBlock`	auto-prefixed cells map to relaxed cell content
image block and inline macros	`FigureBlock` / inline `Image`	link target and roles via register
`source` block	`SourcecodeBlock`	callouts and annotations map directly
stem macro	`StemElement` (`AsciiML` or `MathML`)	
`literal` / `[verse]`	`LiteralBlock`	verse: `format` register key over preformatted text
passthroughs (pass macro, triple-plus)	`FormattedString` / `LiteralBlock` + `format`	raw content is non-markup
attribute references	`ReferenceToVariable`	resolves against the register
cross-reference macros, anchors	`ReferenceToIdElement`, `Bookmark`	
footnote macro	`Footnote`	
bibliography section	`ReferencesSection` + `BibliographicItem`	
include, ToC, and conditional directives	—	preprocessing, out of scope

A.3. Markdown (CommonMark, GFM, Pandoc)

Table A.2

Markdown construct	Model home	Notes
ATX/setext headings	`BasicSection`	heading identifiers and classes via register

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Markdown construct	Model home	Notes
emphasis/strong/ strikethrough	`EmphasisElement`, `StrongElement`, `StrikeElement`	
code span / fenced code	`MonospaceElement` / `SourcecodeBlock`	number-lines option via register
bullet/ordered lists	`UnorderedList`, `OrderedList`	loose items map to relaxed `ListItem` content
task lists	`ListItem` + register	`checkbox`/`completed` as register keys
GFM table	`TableBlock`	column alignment denormalized to cell `align`
links / autolinks	`ReferenceToLinkElement`	
images (inline)	`Image`	
footnotes (Pandoc/GFM)	`Footnote`	
highlight marks	text element specialization	emphasis-family kind
$_{\text{sub}}$ $^{\text{sup}}$	`SubscriptElement`, `SuperscriptElement`	
inline math	`StemElement` (`LaTeX`)	
citations	`Citation` + `BibliographicItem`	
fenced divs	any relaxed container + register `class`	
raw HTML/TeX	`FormattedString` / `LiteralBlock` + `format`	
YAML front matter	document register	`BibDataExtensionType` for bibliographic keys

A.4. reStructuredText

Table A.3

RST construct	Model home	Notes
section titles / transitions	`BasicSection` / `HorizontalRuleElement`	
bullet/enumerated lists	`UnorderedList`, `OrderedList`	auto-numbering via `start`
definition list	`DefinitionList`	
field list	document/block register	bibliographic keys to `BibDataExtensionType`
option list	`DefinitionList` + register `option`	
literal block / line block	`LiteralBlock`	line block: line breaks significant
block quote	`QuoteBlock`	
doctest block	`SourcecodeBlock` (`lang` = doctest)	
admonitions (note, warning, ...)	`AdmonitionBlock` + `AdmonitionType`	`danger`, `error`, `hint` as type-specializations
code directive	`SourcecodeBlock`	number-lines option via register
math directive	`StemElement`	
image and figure directives	`Image` / `FigureBlock`	target option via register
substitution references	`ReferenceToVariable`	
interpreted text roles	text element specializations	title-reference and similar roles as kinds
directives (incl. Sphinx)	register + relaxed content	directive name and options as register entries over any container
comments	—	dropped; noted as non-content

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Appendix B
(informative)
Accommodating specializations

B.1. Type specialization

Markup-specific kinds arrive as *type values* of a single construct, never as parallel classes. `AdmonitionType` defines the five generic kinds (*warning*, *note*, *tip*, *important*, *caution*); reStructuredText's *danger*, *error*, and *hint*, and domain-specific labels, are added by specializations of this model extending the enumeration. The same pattern governs `OrderedListType`, `StemType`, and `DocumentType` (whose generic instance carries the single value *document*).

B.2. Attribute specialization

The specialization rules of Clause 4 permit subclasses to add, remove, retighten, or retype inherited attributes. A Standards-document specialization proscribes "hanging paragraphs" by constraining the co-occurrence of text blocks and subsections; a typography-conscious specialization may narrow `LocalizedString` usage to spelling systems registered under ISO 24229.

B.3. Class specialization

New classes subclass existing roots: block families under `BasicBlock`, inline elements under `BasicElement`, reference kinds under `ReferenceElement`. The distinction between `TextElement` and `PureTextElement` illustrates specialization by constraint: pure text elements recurse only to pure text elements, so consumers processing plain runs of text can rely on the distinction without inspecting the full hierarchy.

B.4. Relationship to publishing document models

Publishing document models such as DocBook and DITA solve a related problem with a different centre of gravity: they are rich, XML-based vocabularies for publishing pipelines, and DITA in particular pioneered the specialization mechanism this model adopts in lighter form. This model is not a replacement for them: it is the interchange core *below* syntax — a deliberately small set of constructs into which lightweight markups parse losslessly and from which publishing vocabularies can be derived. Content reuse mechanisms such as DITA's content references are preprocessing in those frameworks; in this model, inclusion is structural (documents compose as blocks, see Appendix C), requiring no preprocessing stage.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Appendix C
(informative)
Accommodating extensions

C.1. The attribute register

Extensions that would otherwise require model changes are carried as entries in the open attribute register (see Clause 4): any construct accepts `Attribute` entries (key, value(s), optional scheme). Well-known keys used across the ecosystem are `id`, `class`, `lang`, `script`, `unnumbered`, `subsequence`, and `format`; a markup dialect binds its own keys (for example, `checkbox` for task-list items, `option` for option-list entries) without any change to this model. A scheme qualifies interpretation — including ISO 24229 system codes identifying the romanization or transliteration scheme applied to content.

C.2. Opaque and raw content

Raw or passthrough content is non-markup: inline raw content is a `FormattedString` whose `format` attribute exists precisely to admit markup into strings; block-level raw content is a string-carrying construct (`LiteralBlock`, `SourcecodeBlock`) qualified by a `format` register entry. Unknown directives — including the entire Sphinx directive ecosystem — decompose into register entries (the directive name and its options) over any relaxed container, preserving both the content and the extension identity losslessly.

C.3. Variables and composition

Variable references (`ReferenceToVariable`) resolve against the register at processing time, covering AsciiDoc attribute references, reStructuredText substitutions, and template variables. Because a document is itself a block (see Clause 4), entire documents compose as child content of any relaxed container, supporting document-level inclusion as a structural relation rather than a preprocessing step.

C.4. When a markup dialect gains a construct

A flavour of Markdown introduces a new inline construct — say, a mention syntax. No change to this model is required, and existing conforming processors remain conforming:

— the construct's information rides as register entries (for example, keys `mention` and `mention-id`) over the nearest existing construct, so documents remain losslessly interchangeable the day the flavour ships;
— a specialization module may later give the construct type semantics (a subclass under `BasicElement`), for processors that want them;
— neither path touches the base model, and both are visible to profiles through their register-key vocabulary.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Appendix D
(normative)
Abstract test suite

D.1. General

This annex defines the abstract test suite for the conformance classes of this document. The test suite is executable: it runs as part of the continuous integration of the model repository, and the reference instance documents it consumes are maintained alongside the model.

D.2. Conformance classes

Table D.1

Class	Requirement	Test
Model integrity	The LML definition modules parse, names resolve, every attribute carries an explicit visibility marker and a definition	Model lint gate
Diagram parity	Every diagram view includes its models and declares associations within its include closure; no class bodies under views	Parity gate
XML reference serialization	Every construct has a serializable XML form accepted by the accompanying grammar	XML instance suite against the compiled grammar
YAML reference serialization	Every construct has a serializable YAML form conforming to the model, inheritance included	YAML instance suite against the parsed model
Any serialization	A serialization format represents every construct without loss and round-trips to a reference serialization	Instance-equivalence check against a reference serialization
Profile conformance	A profile narrows only: names exist, cardinalities never widen, enumerations are subsets; profile instances contain no excluded construct and use permitted register keys only	Profile suite against the model and worked instances
Change operations	Every change operation has a serializable form, and the operation set addresses every construct shape (string content, key-value slot, ordered sequence, table grid)	Change twin instance suite against the compiled grammar and the parsed model

D.3. Test material

The reference instances are twin documents — one XML, one YAML — exercising the attribute register (document, section, block, and inline levels; single and repeated values), relaxed content models (blocks within list items, admonition, quotation, definition-list definitions, table cells, examples), multi-row table headers, variable references, citations with a references section and bibliographic item, ruby annotations, source code callouts and annotations, reviewer comments, anchors and cross-references, and amend composition of blocks, sections, and entire documents, and a business document instance (bibliographic data, trade register entries with scheme keys, line-item tables, computed cells). A twin change set exercises every change operation — node insert with explicit position, node delete and move, content and attribute modification, and the full family of table changes, including a computed cell. A construct without a serializable instance in the suite is treated as unfinished.

D.4. Verdicts

A processor or serialization passes a test if the instance validates against its schema or parsed model without error; a serialization in any other format passes if its instances are equivalent, construct for construct, to a valid reference-serialization instance. The twin reference instances — one per reference serialization, carrying identical content — demonstrate lossless transformability

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

between serialization formats. The gates abort on first failure; the test suite is regenerated from the model sources on every change.

 © The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved

Bibliography

[1] ISO/IEC DIR 1, ISO and IEC. *Procedures for the technical work*.

[2] ISO/IEC DIR 2:2018, ISO and IEC. *Principles and rules for the structure and drafting of ISO and IEC documents*. 2018.

[3] ISO 690, International Organization for Standardization (committee). *Information and documentation — Guidelines for bibliographic references and citations to information resources*. Fourth edition. Geneva: International Organization for Standardization. https://www.iso.org/standard/72642.html.

[4] ISO 6523, International Organization for Standardization (committee). *Data interchange — Structures for the identification of organizations*. First edition. Geneva: International Organization for Standardization. https://www.iso.org/standard/12908.html.

[5] ISO 9735, International Organization for Standardization (committee). *Electronic data interchange for administration, commerce and transport (EDIFACT) — Application level syntax rules*. First edition. Geneva: International Organization for Standardization. https://www.iso.org/standard/17592.html.

[6] ISO 5127:2017, International Organization for Standardization (committee). *Information and documentation — Foundation and vocabulary*. Second edition. 2017. Geneva: International Organization for Standardization. https://www.iso.org/standard/59743.html.

[7] IEC 60050:1975, International Electrotechnical Commission (committee). *International Electrotechnical Vocabulary (IEV) — Index*. Third edition. 1975. Geneva: International Electrotechnical Commission. https://webstore.iec.ch/publication/12502.

[8] IETF RFC 2045, FREED, N., N. BORENSTEIN and Internet Engineering Task Force. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. 1996. RFC Publisher. https://www.rfc-editor.org/info/rfc2045.

© The Calendaring and Scheduling Consortium, Inc. 2026 – All rights reserved