CC/WD 36010:2019

Lightweight document — Document metamodel

THE CALENDARING AND SCHEDULING CONSORTIUM

TC VCARD

Ronald Tse **AUTHOR**

Jeffrey Lau **AUTHOR**

Nick Nicholas **AUTHOR**

CALCONNECT STANDARD

WORKING DRAFT

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.

The Calendaring and Scheduling Consortium, Inc.
4390 Chaffin Lane
McKinleyville
California 95519
United States of America

copyright@calconnect.org
www.calconnect.org

CONTENTS

Foreword

Introduction

1. Scope

2. Normative references

3. Terms and definitions

4. Modelling

    4.1. Intent

    4.2. Specialization

    4.3. Structure

5. BasicDocument model

6. Metadata and bibliographic information models

7. Section models

8. Block models

    8.1. General

    8.2. Paragraph

    8.3. Multi-paragraph blocks

    8.4. Table

    8.5. List

    8.6. Ancillary blocks

9. Inline element models

    9.1. General

    9.2. Text Elements

    9.3. Empty Elements

    9.4. ID Elements

    9.5. Reference Elements

    9.6. Footnote

10. Data type models

    10.1. Basic Data Types

    10.2. Contribution Element Metadata

11. Change models

11.1. General

11.2. Change

11.3. Change set

11.4. Unique identifier

11.5. Content change

11.6. Attribute change

11.7. Node change

Bibliography

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*.

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 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.

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

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 modelling of bibliographies and citations is part of the reference model, but is the subject of a separate document, ISO 6900:--, 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.

— ISO 639 (all parts), *Codes for the representation of names of languages*

— ISO 8601-1, *Date and time — Representations for information interchange — Part 1: Basic rules*

— ISO 8601-2, *Date and time — Representations for information interchange — Part 2: Extensions*

— ISO/IEC 10118 (all parts), *Information technology — Security techniques*

— ISO/IEC 14888 (all parts), *Information technology — Security techniques*

— ISO 15924, *Information and documentation — Codes for the representation of names of scripts*

— ISO 6900:—, *Information and documentation — Bibliographic reference model and serialization*

— IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

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

3.8. 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. MODELLING

4.1. 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.

4.2. 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.

4.3. Structure

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

— Sections (Clause 7)

— Blocks (paragraph-level groupings of text) (Clause 8)

— Inline elements (groupings of text smaller than a paragraph, including plain strings) (Clause 9)

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 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 ISO 6900:--.

5. BASICDOCUMENT MODEL

The *BasicDocument* model treats all documents as collections of *sections* (Clause 7). It also adds the following metadata as part of the document model:

`identifier` — a 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 10.2).

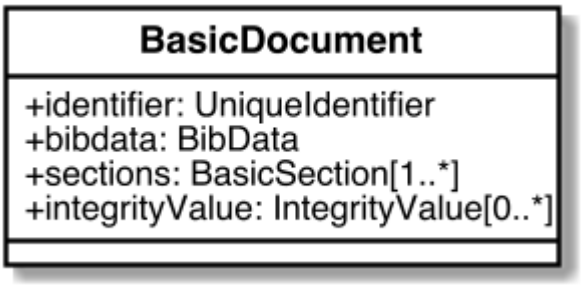


Figure 1 – Basic Document model: Document

6. METADATA AND BIBLIOGRAPHIC INFORMATION MODELS

The modelling of `bibdata` follows the *BibliographicItem* class in ISO 6900:--, 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, 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 ISO 6900:--.

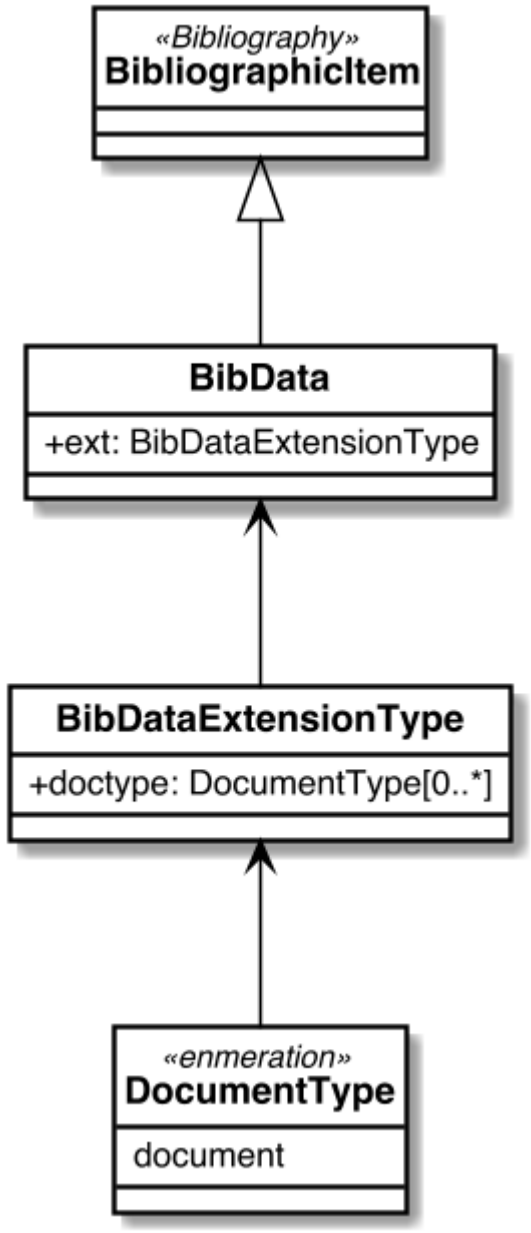


Figure 2 — Basic Document model: Bibdata

7. 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 models such as Metanorma, which differentiates 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 ISO 6900:--), 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.)

`blocks` zero or more text blocks, containing the textual content of the section (but excluding subsections, which are only present in Hierarchical Sections).

`notes` any notes whose scope is the entire section, rather than a specific block. Notes are modelled as a sequence of zero or more paragraphs.

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.

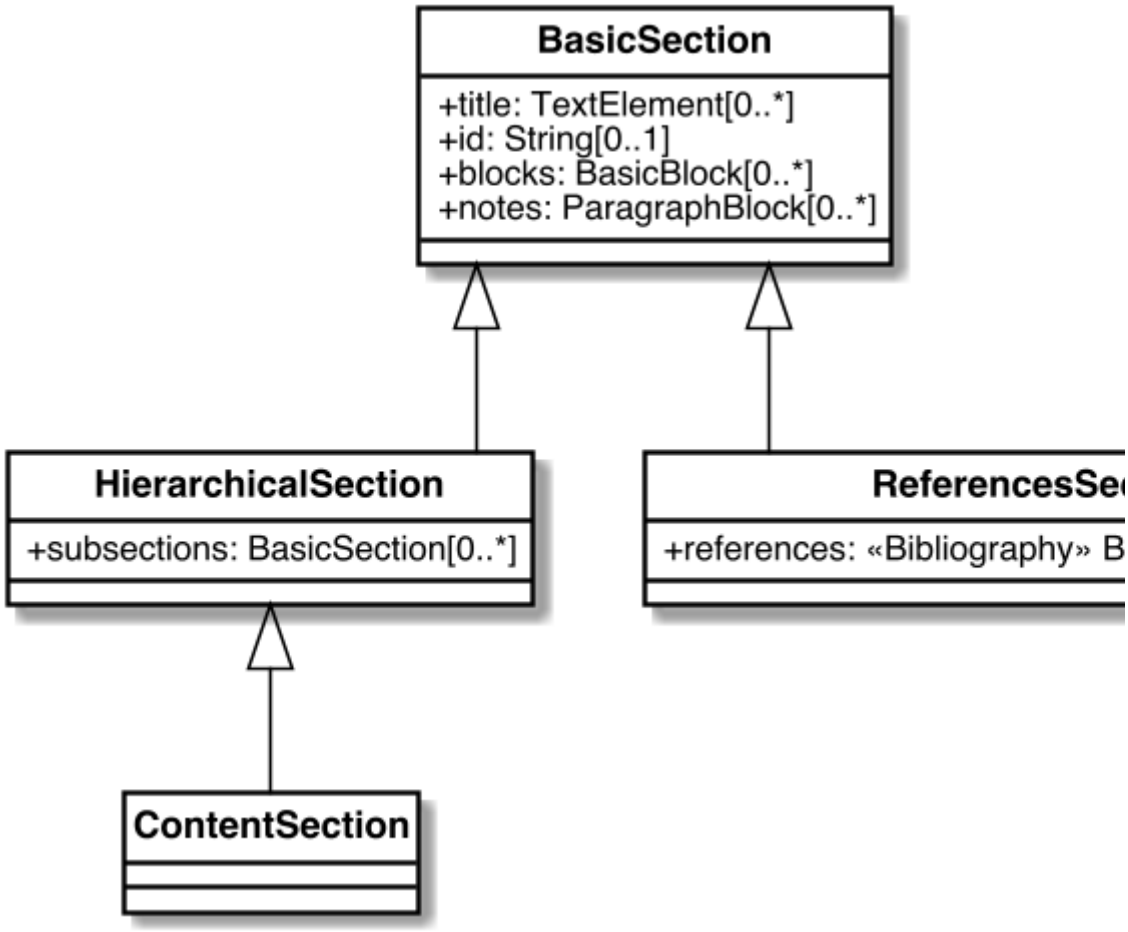


Figure 3 – Basic Document model: Section

8. BLOCK MODELS

8.1. 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.
`contributionelementmetadata`	attribution of the block to a specific contributor (Clause 10.2), to be used in change management of documents.

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

— Paragraphs (Clause 8.2)

— Multi-paragraph blocks: Blockquotes, Reviewer comments, Admonitions (Clause 8.3)

— Tables (Clause 8.4)

— Lists (Clause 8.5): Unordered lists, Ordered lists, Definition lists

— Ancillary blocks: Figures (Clause 8.6.2), Sourcecode (Clause 8.6.3), Formulas (Clause 8.6.4), Examples (Clause 8.6.6), Pre-formatted blocks (Clause 8.6.5)

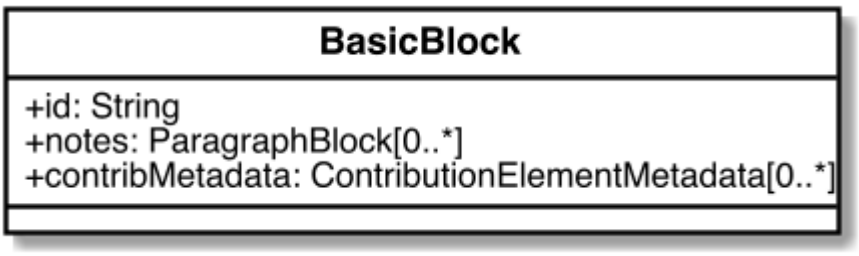


Figure 4 — Basic Document model: Block

8.2. Paragraph

Paragraphs can contain any sequence of inline elements (Clause 9), 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 9.6). 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 8.6.3).

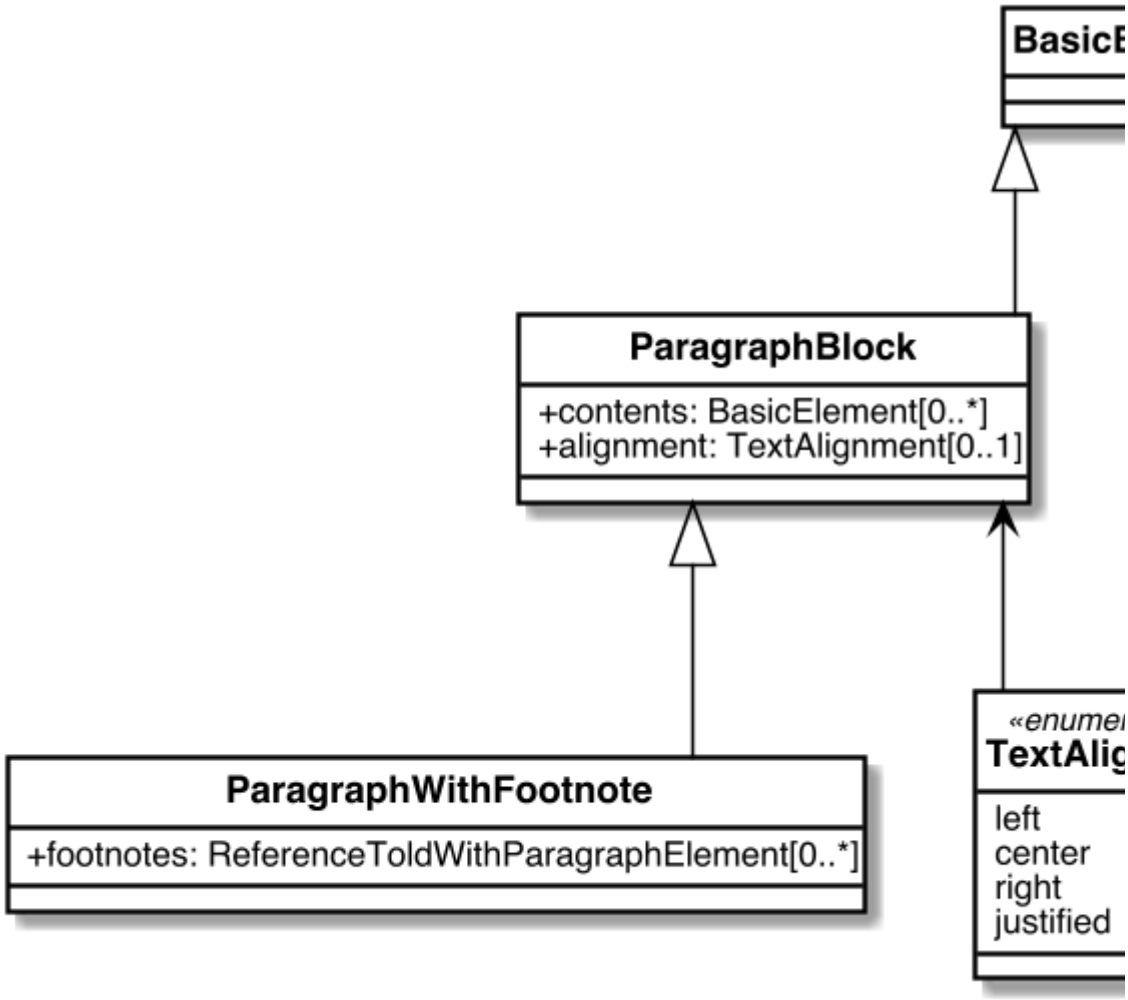


Figure 5 — Basic Document model: Paragraph

8.3. Multi-paragraph blocks

The following classes of block are modelled as containing zero or more paragraphs:

— *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.

— *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, 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 autonumbered differently, even if they are of the same type. The defined types for the *BasicDocument* model are *Warning, Note, Tip, Important, Caution, and Statement*. (In rendering, distinct admonition types are often associated with distinct icons or rendering.)

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.

— *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; a mandatory 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`).

NOTE 2 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`.

NOTE 3 Admonition notes are modelled to be distinct from notes under sections or blocks.

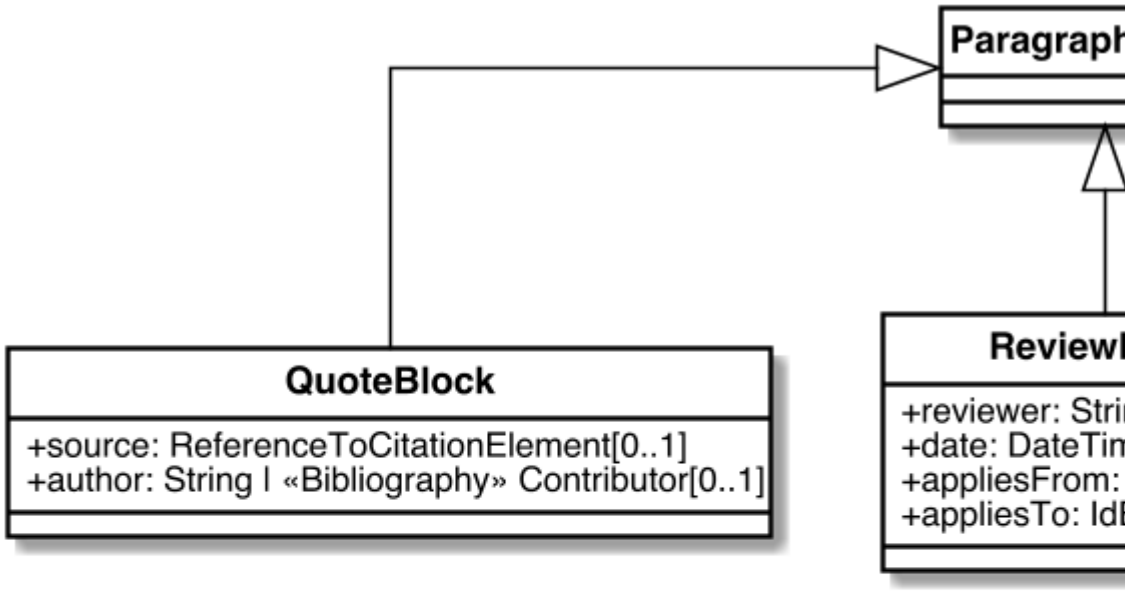


Figure 6 — Basic Document model: Multi-paragraph Block

8.4. Table

Tables are modelled following the same principles as HTML tables. 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 8.5) 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.

`uri` a URI (in case the admonition 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.

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.

Table cells contain either zero or more paragraphs with footnotes (Clause 8.2), or zero or more text elements (Clause 9.2). 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.

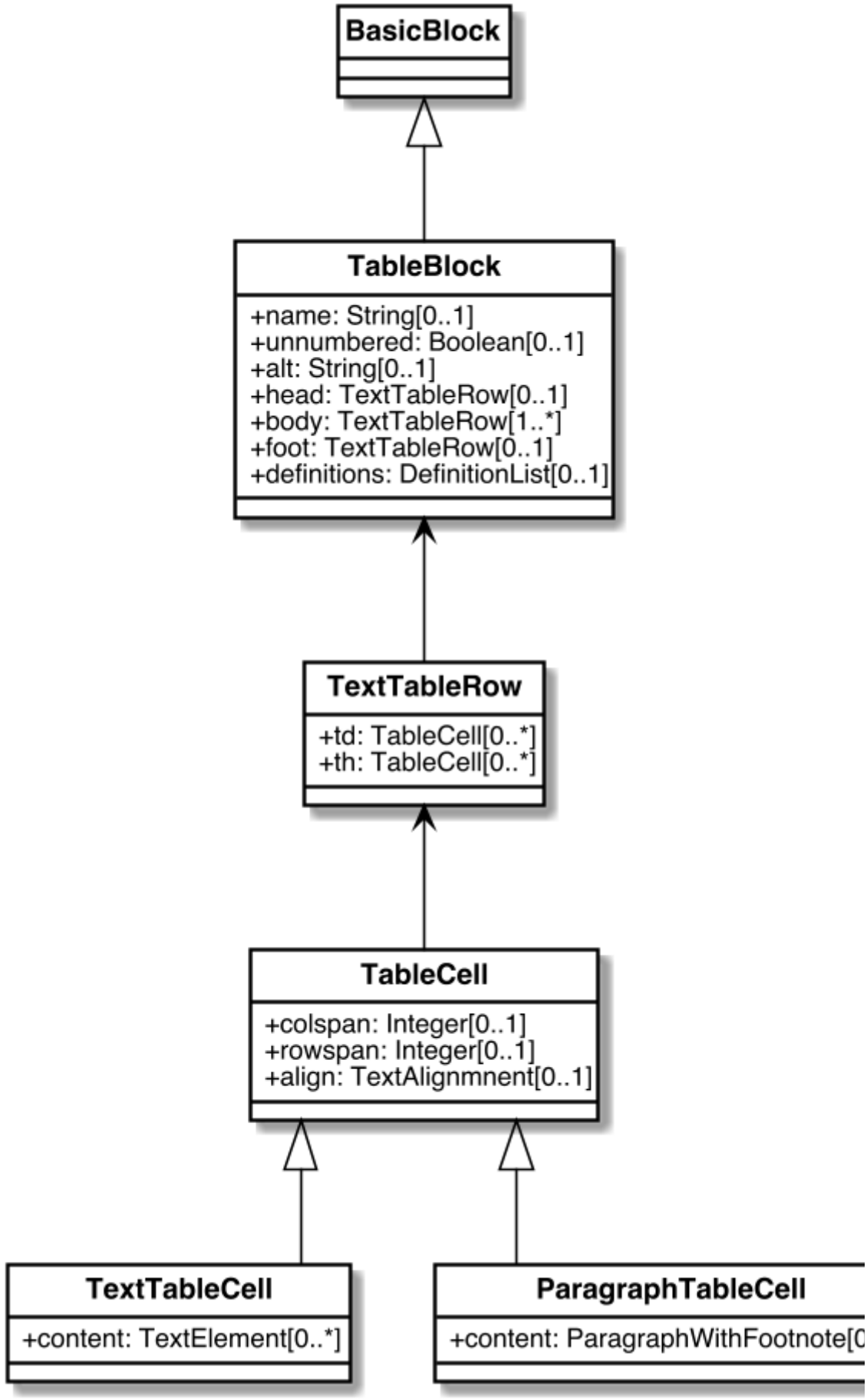

BasicBlock
TableBlock
+name: String[0..1]
+unnumbered: Boolean[0..1]
+alt: String[0..1]
+head: TextTableRow[0..1]
+body: TextTableRow[1..*]
+foot: TextTableRow[0..1]
+definitions: DefinitionList[0..1]
TextTableRow
+td: TableCell[0..*]
+th: TableCell[0..*]
TableCell
+colspan: Integer[0..1]
+rowspan: Integer[0..1]
+align: TextAlignmnent[0..1]
TextTableCell
+content: TextElement[0..*]
ParagraphTableCell
+content: ParagraphWithFootnote[0

Figure 7 — Basic Document model: Table

8.5. List

Lists are modelled following the same principles as HTML lists. All lists contain zero or more *list items*, which by default consist of an identifier (`id`), and one or more paragraphs with footnotes (Clause 8.2). This allows individual list items in a list to be cross-referenced within the document.

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*, *roman_upper*, *alphabet_upper*, 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 9.2) and `definition` (zero or more paragraphs with footnotes: Clause 8.2).

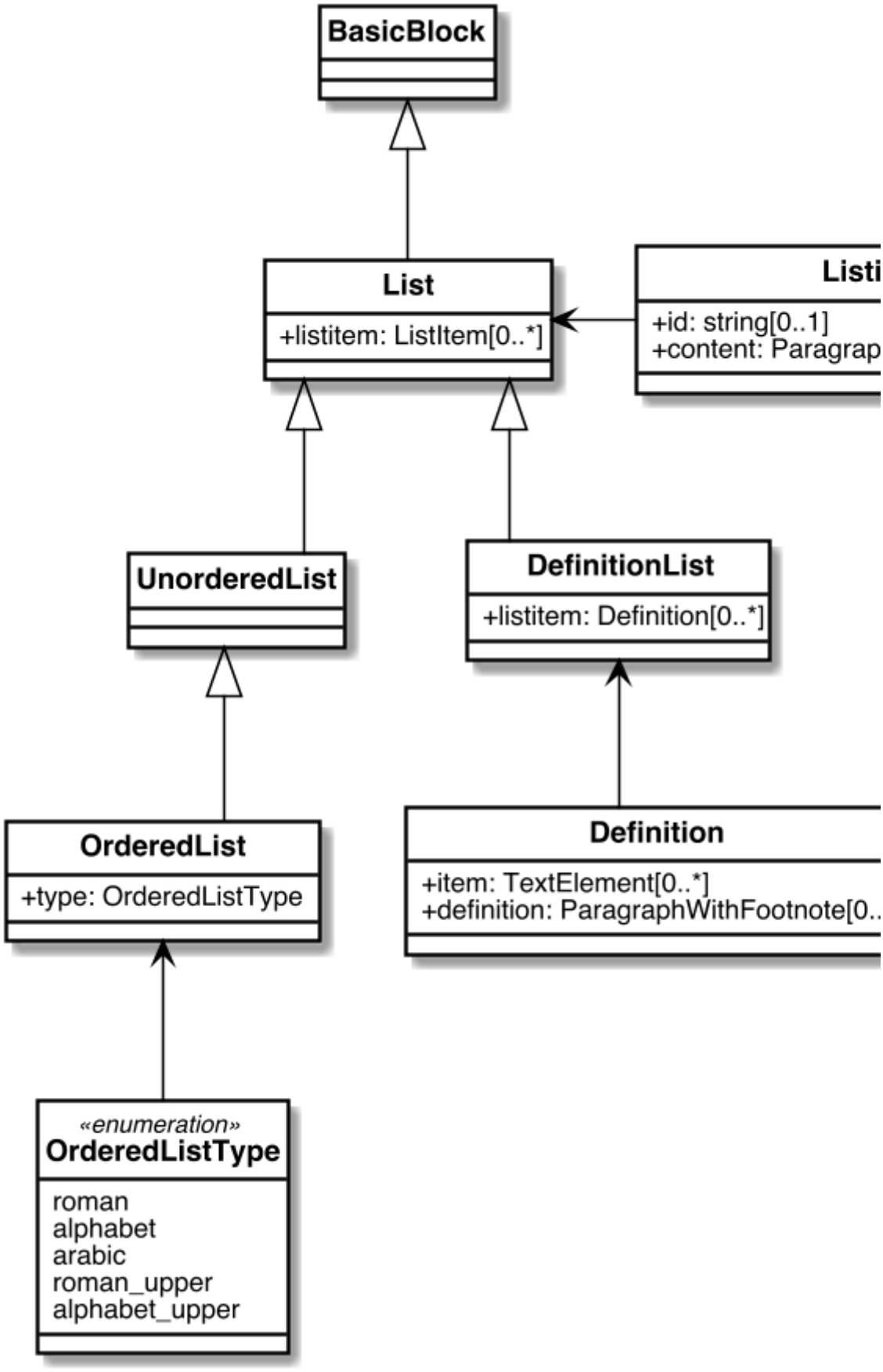


Figure 8 – Basic Document model: List

8.6. Ancillary blocks

8.6.1. 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.

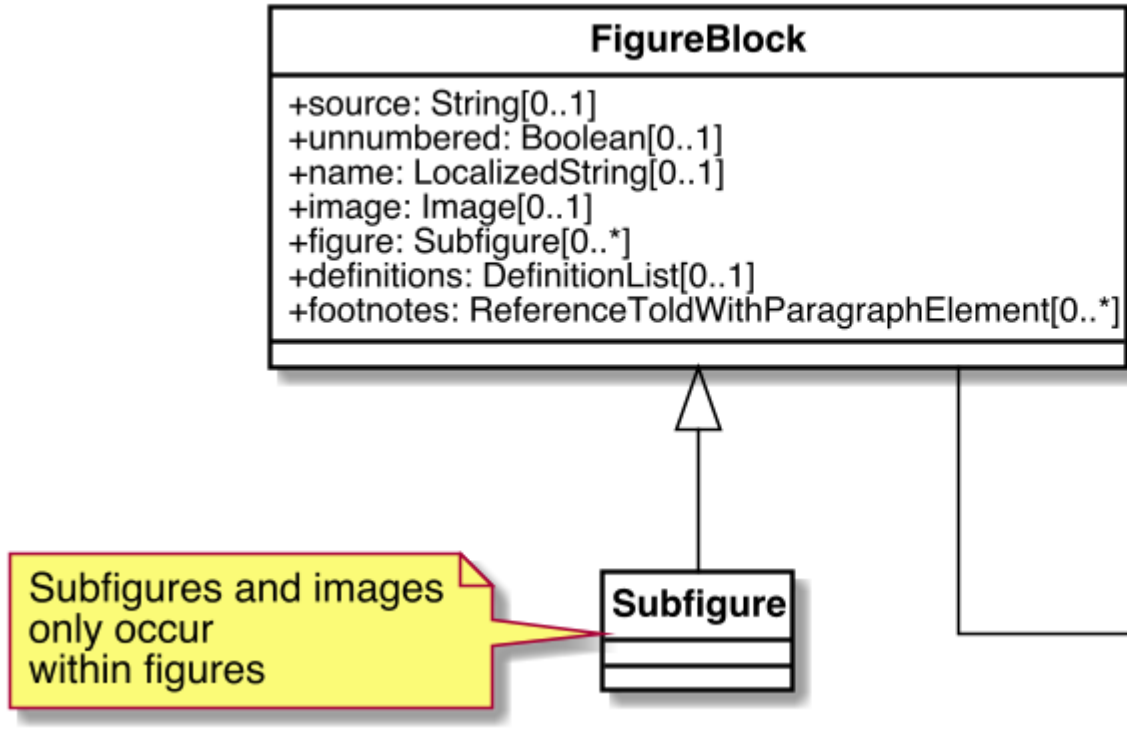


Figure 9 — Basic Document model: Figure, Sourcecode, Formula, Example

8.6.2. 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 diffent classes of figure (e.g. *Plate*, *Chart*, *Diagram*) to be autonumbered and captioned differently.
`image`	An image (Clause 9.4).
`source`	A URI or other reference intended to link to an externally hosted image (or equivalent).

`definitions`	An optional definitions list (Clause 8.5) defining any symbols used in the figure. (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. (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.

8.6.3. 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 9.5); 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.

8.6.4. Formula

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

`stem` — A STEM element (Clause 9.2), constituting the content of the formula

`definitions` — An optional definitions list (Clause 8.5) defining any symbols used in the formula. (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.

8.6.5. 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 8.6.3) 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.)

8.6.6. Example

Example blocks are wrappers for open-ended example text. They consist of a combination of any of the following blocks:

- Formula
- List
- Blockquote (which is how generic text is included in an example)
- Sourcecode
- Paragraph

It also contains the following elements:

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

9. INLINE ELEMENT MODELS

9.1. 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:

`contributionelementmetadata`	attribution of the element to a specific contributor (Clause 10.2), to be used in change management of documents.

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 9.2)

— ID elements, which contain identifiers. (Clause 9.4)

— Reference elements, which contain references to identifiers. (Clause 9.5)

Footnotes (Clause 9.6) 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 8.2).

9.2. 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 10).

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 their HTML 5 equivalents;

- ruby
- rt
- rp

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

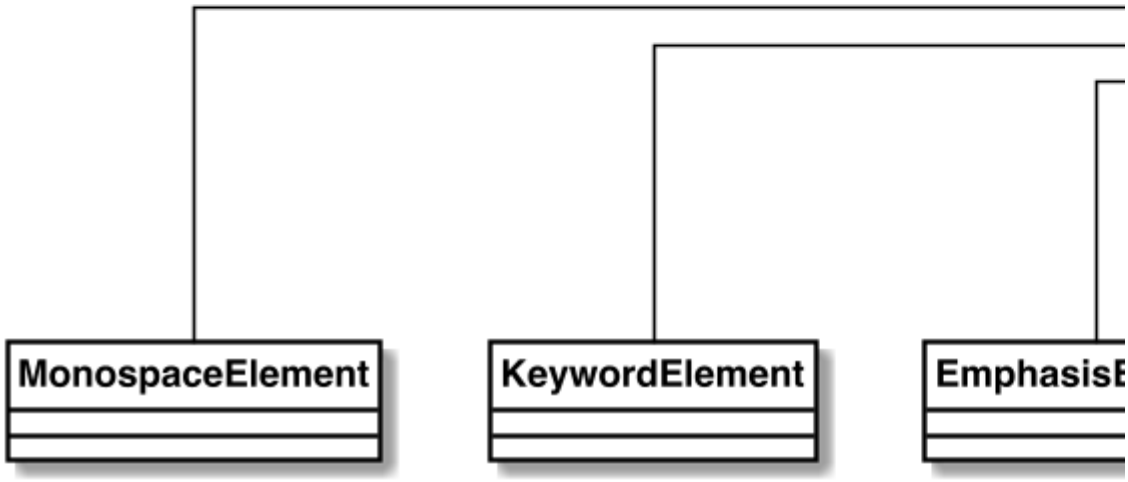


Figure 10 — Basic Document model: Text Elements

9.3. 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`)

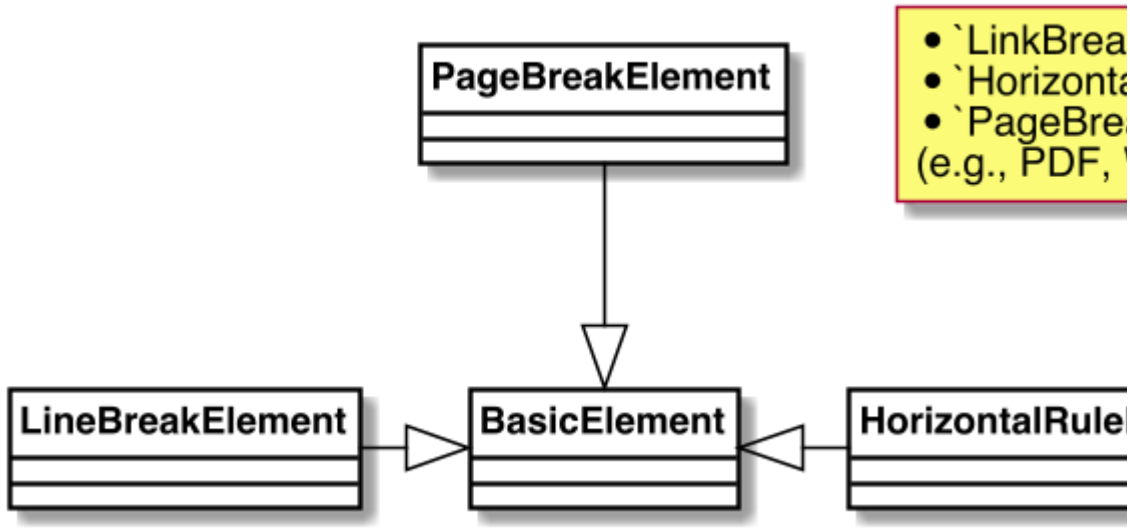


Figure 11 — Basic Document model: Empty Elements

9.4. ID Elements

ID Elements are inline elements that have an identifier (`id`), which permits them to be cross-referenced by Reference Elements (Clause 9.5). 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 Review block (review) 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 (RFC 2045)

`alt` alternate text, 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, which can be an integer or “auto”

`width` optional attribute, which can be an integer 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 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

`height` optional attribute, which can be an integer or “auto”

`width` optional attribute, which can be an integer or “auto”

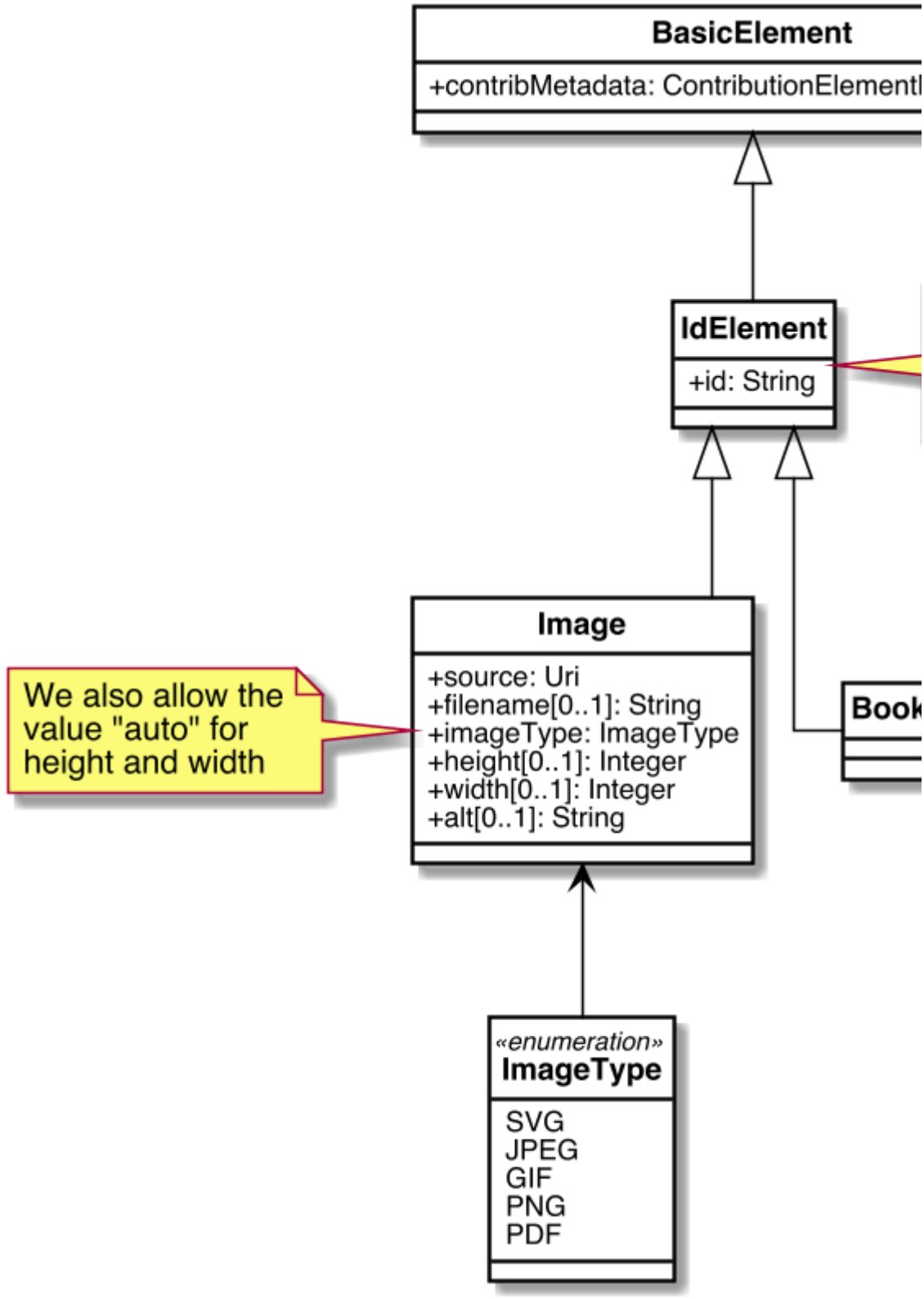


Figure 12 – Basic Document model: ID Elements

9.5. 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 8.6.3).

`alt` Alternate text, used for accessibility.

The following subclasses of Reference Elements are modelled.

- 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 ISO 6900:-- 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.

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 8.2).

9.6. 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.

10. DATA TYPE MODELS

10.1. Basic Data Types

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

- String
- URI, as defined in RFC 3986.
- Language names, as defined in ISO 639 (all parts).
- Dates and times, as defined in ISO 8601-1.
- Hash algorithms, as defined in ISO/IEC 10118 (all parts).
- Digital signature algorithms, as defined in ISO/IEC 14888 (all parts).
- Script names, as defined in ISO 15924.
- Localized Strings, specifying a String with optional Language and Script attributes.
- Formatted Strings, specifying a Localized String with a `format` attribute, in order to admit markup into Strings (whether XML-based or not).

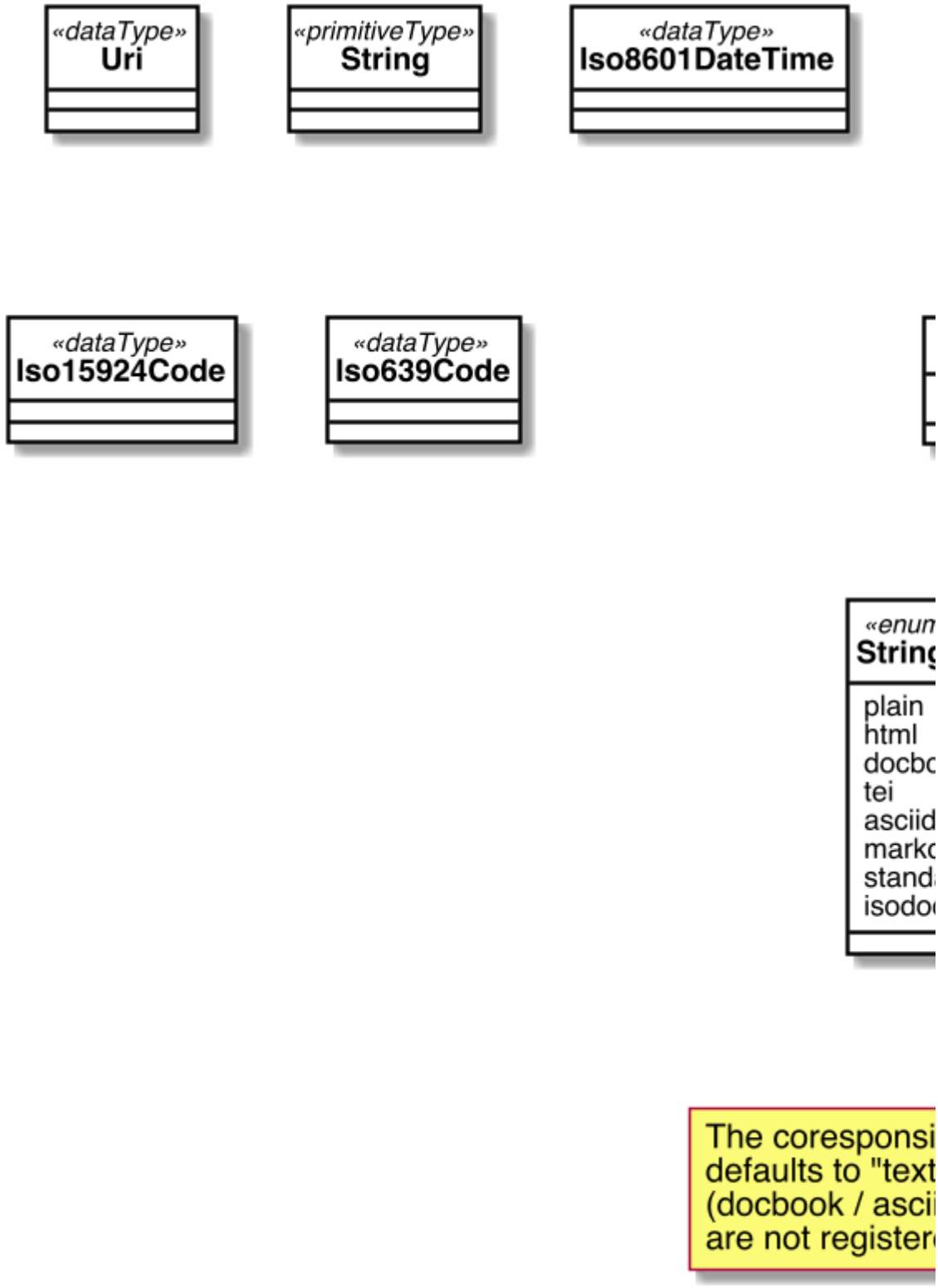


Figure 13 – Basic Document model: Data Types

10.2. 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.
`contributor`	The party who made the contribution, as described through the `contributor` element in ISO 6900:--.
`inegrityValue`	A digital signature of the contribution, consisting of a `hash` value and a `signature`, with an associated `publicKey`.

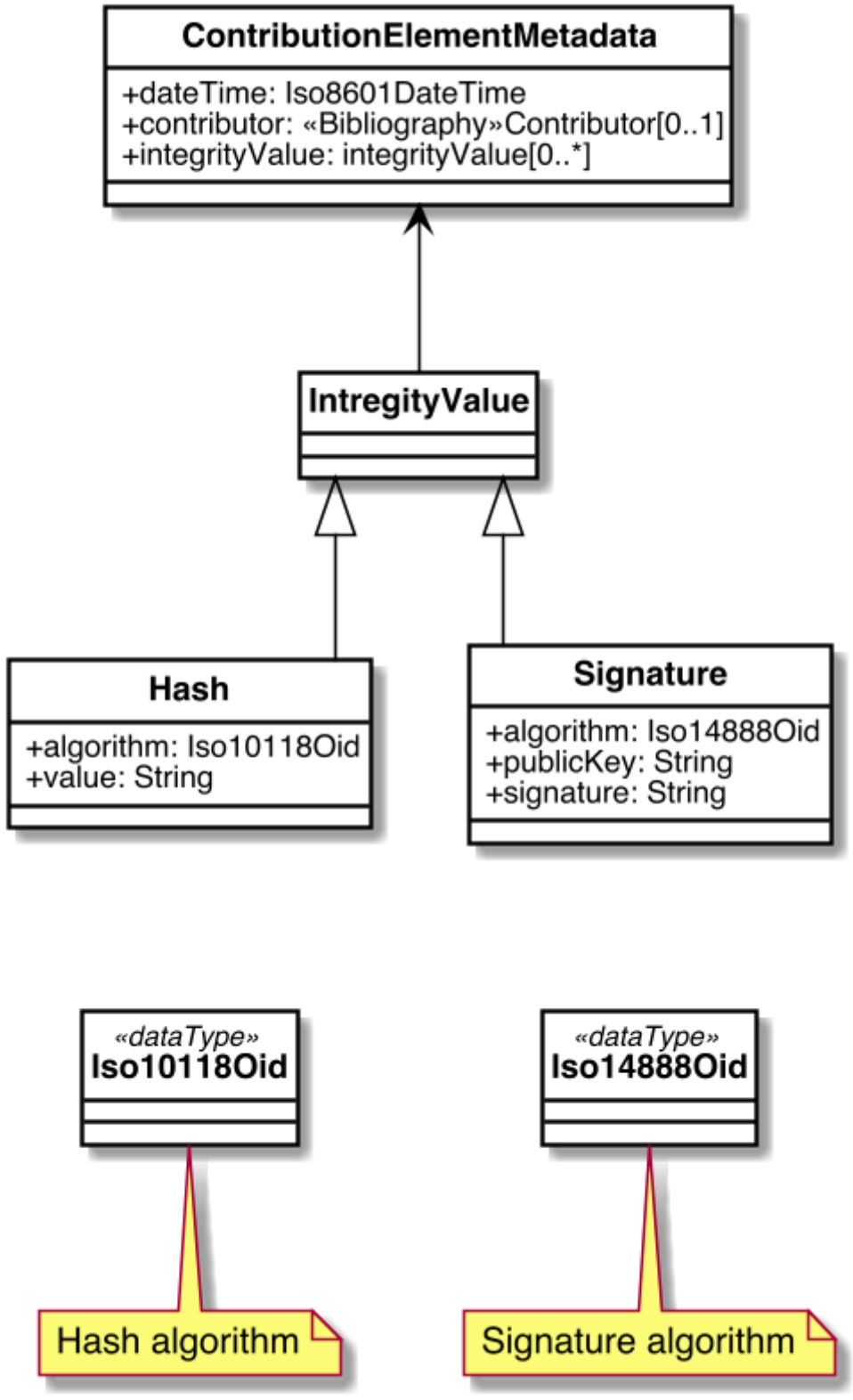
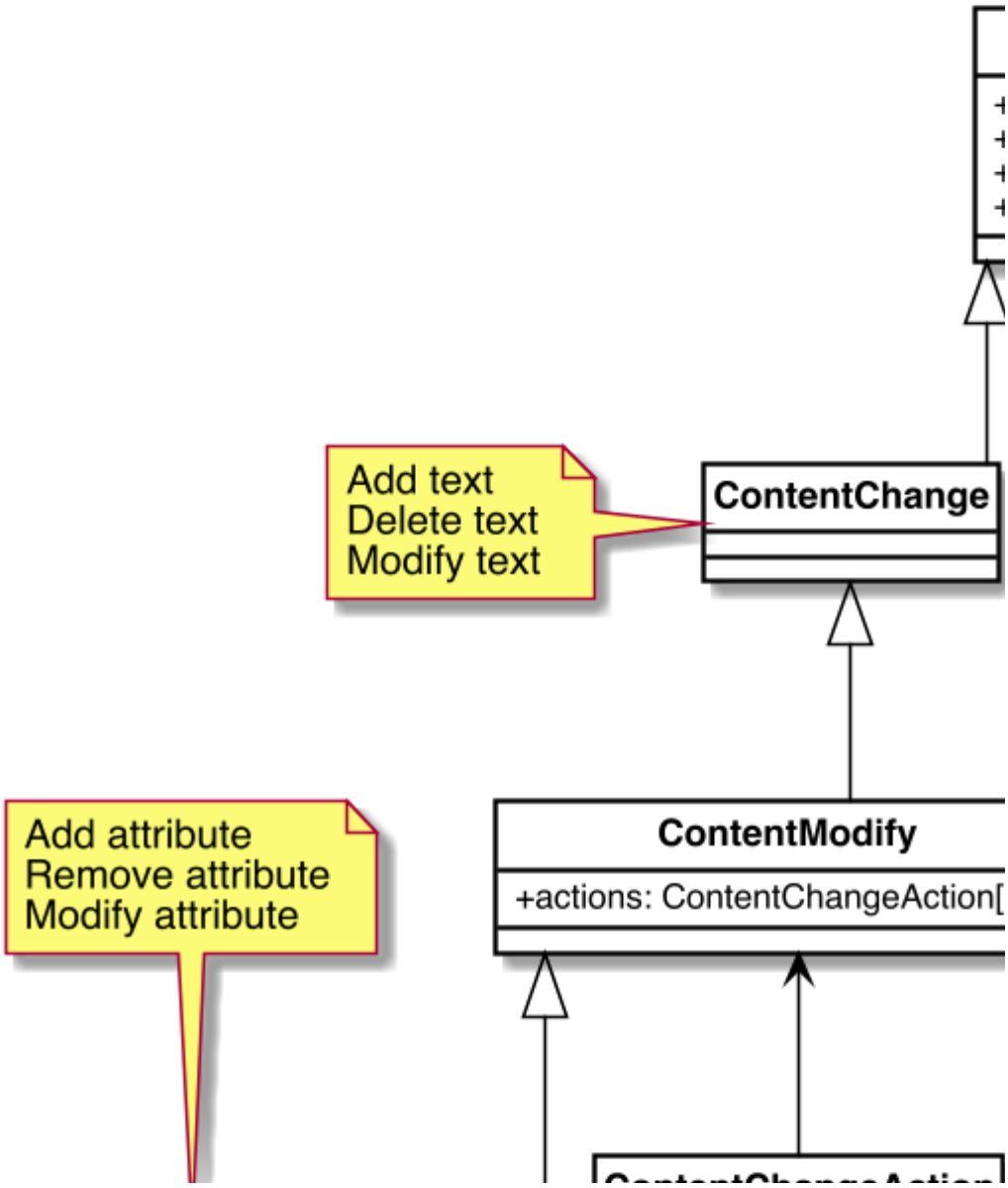


Figure 14 — Basic Document model: Contribution Element Metadata

11. CHANGE MODELS

11.1. General

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



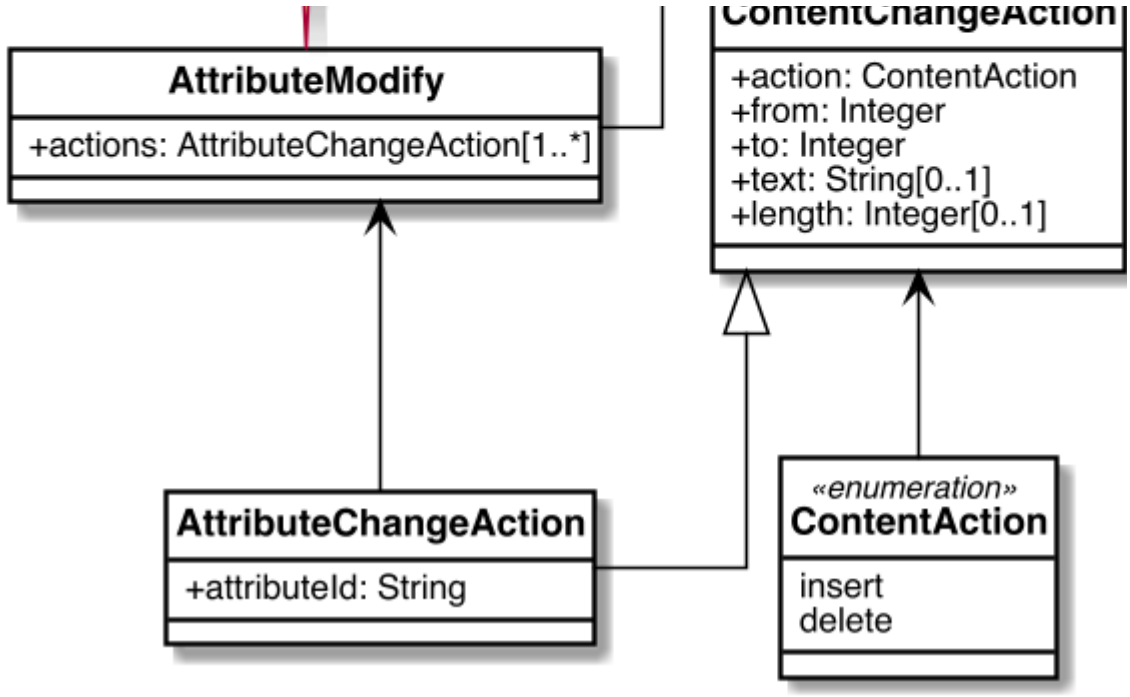


Figure 15 — Basic Document model: Changes

11.2. 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`	One or more unique identifiers of *Change* objects, that this change is supposed to follow after.
`contribMetadata`	Metadata of the contributor, see Clause 10.2.

11.3. 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.

11.4. Unique identifier

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

It contains the attribute:

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

11.5. Content change

11.5.1. General

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

11.5.2. 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.

11.5.3. Content modify

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

It has the following attribute:

`actions` One or more `ContentChangeAction` data

11.6. Attribute change

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

11.6.1. 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.

11.6.2. Attribute modify

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

It has the following attribute:

`actions` One or more `AttributeChangeAction` data

11.7. Node change

11.7.1. 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.

11.7.2. Node insert

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

It has the following attribute:

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

11.7.3. 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.

11.7.4. 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.

BIBLIOGRAPHY

— [1] ISO 690, *Information and documentation — Guidelines for bibliographic references and citations to information resources*

— [2] ISO 5127:2017, *Information and documentation — Foundation and vocabulary*

— [3] ISO/IEC DIR 1, *Procedures for the technical work*

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

— [5] IETF RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*