PreTeXt RELAX-NG Schema

Robert A. Beezer
Department of Mathematics and Computer Science
University of Puget Sound
Tacoma, Washington, USA
beezer@pugetsound.edu

April 25, 2024

Contents

This is a literate programming version of the RELAX-NG schema for PreTeXt. As such, it is used to generate the RELAX-NG compact syntax version (`pretext.rnc`) and other versions are derived from the compact version with standard tools.

We intend this to be helpful for both authors and implementers. The schema is the contract between authors and implementers. If an author's source validates against the schema, then an implementer's conversion should render the content accurately, or warn about why it cannot. That said, it is still a work in progress:

- New features are not added until they are reasonably stable. Validating the sample article can be a good way to see what these are.
- Even for stable features, the schema will sometimes lag behind the code.
- There will be other inaccuracies here, so reports or pull requests are welcome.

The RELAX-NG syntax is built on **patterns**, which describe how XML elements and attributes may be combined. It begins with a `start` pattern. Patterns separated by commas must appear in that order. Elements separated by a vertical bar represent a choice. Parentheses are used for grouping. Braces are basic syntax, reminiscent of the syntax for Java. An equals sign is assignment and `|=` is a continuation of an assignment. Finally, optional and/or multiple occurrences can be specified with modifiers:

`?`	Zero or one. Optional, at most one.
`*`	Zero or more. Optional, with no limit.
`+`	One or more. Required, with no limit.

Appendix A contains a list of all the fragments described here, in order of appearance, and may be useful if you are looking for some particular topic, element, or attribute.

1 Start Elements

To support modular source files, we specify which elements can naturally be the root of a *fragment* file in a PreTeXt document. These include the `pretext` element itself, as well as most divisions. All of these are defined as elements later in the schema.

⟨1 Start elements⟩ ≡

```
start = Pretext | DocInfo | Part | Chapter | Section | Subsection | Subsubsection | Paragraphs |
```

2 Gross Structure

A PreTeXt document is always a single `pretext` element below the root. There are two divisions, a `docinfo`, which is a database of sorts about the document, along with a sibling element that indicates the type of the document and contains all the content.

⟨2 Gross structure⟩ ≡

```
Pretext =
    element pretext {
        XMLLang?,
        DocInfo?,
        (Book | Article | Letter | Memorandum)
    }
```

3 Document Types

`letter` and `memo` elements are not documented.

⟨3 Document types⟩ ≡

```
Article =
    element article {
        MetaDataLinedSubtitle,
        ArticleFrontMatter?,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Section,
                (Section | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?),
                ArticleBackMatter?
            )
        )
    }
Book =
    ## Here is what a book looks like.
    element book {
        MetaDataLinedSubtitle,
        BookFrontMatter?,
        (Part+ | Chapter+ ),
        BookBackMatter?
    }
Letter =
    element letter {empty}
Memorandum =
    element memo {empty}
```

4 Document Structure

A document is typically divided into sections. But we reserve the word `section` for one very specific type of division. To avoid confusion, we speak generically

of **divisions**. So, for example, a `section` is a division of a `chapter`. Here we list all of the possible divisions, even if they are not available in each document type.

An `appendix` looks like a chapter of a `book`, with the option to have a `notation-list` as its entire contents. It is possible this is not the best structure for an `article`, which might best be divided by `subsection`.

There are several things to note (expand this): always a title, dead-end with blocks, or subdivide with optional intro and conclusion.

⟨4 Divisions⟩ ≡

```
Part =
    element part {
        MetaDataLinedTitle, Chapter+
    }
Chapter =
    element chapter {
        MetaDataLinedTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Section,
                (Section | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Section =
    element section {
        MetaDataLinedTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Subsection,
                (Subsection | ReadingQuestions | Exercises |
```

```
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Subsection =
    element subsection {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Subsubsection,
                (Subsubsection | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Subsubsection =
    element subsubsection {
        MetaDataAltTitle,
        AuthorByline*,
        Objectives?,
        (BlockDivision | Paragraphs | Commentary)+,
        (ReadingQuestions? & Exercises? &
         Solutions? & References? & Glossary?),
        Outcomes?
    }
ArticleAppendix =
    element appendix {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary |
                 NotationList)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Subsection,
```

```
                (Subsection | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
BookAppendix =
    element appendix {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary |
                 NotationList)+,
                (ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                Section,
                (Section | ReadingQuestions | Exercises | Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
IndexDivision =
    element index {
        MetaDataAltTitleOptional,
        IndexList
    }
```

5 Lightweight Divisions

The `paragraphs` element, which is not to be confused with a *real* paragraph as implemented by the `p` element, is an exceptional type of division (both in design and utility). It must have a `title`, can appear anywhere within any of the divisions, cannot be further subdivided, and is not ever numbered. Its contents are conceptually a run of paragraphs, but as described here allow much more than that.

It is especially useful in a short document (like a class handout, letter, memorandum, or short proposal) where numbered divisions might feel like overkill.

The `NoNumber` variant allows for light-weight sectioning of un-numbered divisions, such as a Preface.

`<commentary>` is elective and is only functional with version support, hence a `@component` attribute. (2024-02-16: `<commentary>` could be eliminated in favor of a full-on use of version support, so try that first.

⟨5 Paragraphs division⟩ ≡

```
Paragraphs =
    element paragraphs {
        MetaDataTitle,
        Index*,
        BlockDivision+
    }
ParagraphsNoNumber =
    element paragraphs {
        MetaDataTitle,
        Index*,
        BlockStatementNoCaption+
    }
Commentary =
    element commentary {
        attribute component { text }
    }
```

6 Specialized Divisions

We add specialized divisions, which may appear within any of the above divisions. Titles will be provided as defaults.

⟨6 Specialized divisions⟩ ≡

```
ReadingQuestions =
    element reading-questions {
        MetaDataAltTitleOptional,
        IntroductionDivision?,
        Exercise+,
        ConclusionDivision?
    }
Exercises =
    element exercises {
        MetaDataAltTitleOptional,
        IntroductionDivision?,
        (
            (Exercise | ExerciseGroup)+ |
            Subexercises+
        ),
        ConclusionDivision?
    }
Subexercises =
    element subexercises {
        MetaDataAltTitleOptional,
        IntroductionDivision?,
        (Exercise | ExerciseGroup)+,
        ConclusionDivision?
    }
Solutions =
    element solutions {
        MetaDataAltTitleOptional,
        attribute inline {text}?,
        attribute divisional {text}?,
```

```
        attribute project {text}?,
        attribute admit {"all"|"odd"|"even"}?,
        IntroductionDivision?,
        ConclusionDivision?
    }
References =
    element references {
        MetaDataAltTitleOptional,
        IntroductionDivision?,
        BibliographyItem+,
        ConclusionDivision?
    }
Glossary =
    element glossary {
        MetaDataAltTitleOptional,
        HeadNote?,
        GlossaryItem+
    }
```

7 Solutions (experimental)

The solutions division can now have additional attributes: `@scope`, `@reading`, and `@worksheet`. We collect these three here.

⟨7 Solutions (experimental)⟩ ≡

```
Solutions |=
    element solutions {
        MetaDataAltTitleOptional,
        attribute inline {text}?,
        attribute divisional {text}?,
        attribute project {text}?,
        attribute worksheet {text}?,
        attribute reading {text}?,
        attribute scope {text}?,
        attribute admit {"all"|"odd"|"even"}?,
        IntroductionDivision?,
        ConclusionDivision?
    }
```

8 Worksheets (experimental)

A worksheet is a specialized division, allowing for some additional control of spacing, to allow for workspace.

The attributes on a worksheet include margin information to control layout. Inside a worksheet we can have either a number of `<page>` elements that holding the content or just the content itself.

The contents of a worksheet can include the same blocks as a division, namely BlockDivision.

⟨8 Worksheets (experimental)⟩ ≡

```
WorksheetAttributes =
    attribute margin { text }?,
    attribute top { text }?,
    attribute bottom { text }?,
    attribute right { text }?,
    attribute left { text }?

WorksheetBlock =
    BlockStatement | Remark | Computation | Theorem | Proof | Definition |
    Axiom | Example | WorksheetExercise | Project |
    Poem | Assemblage | ListGenerator | Fragment |
    WorksheetSideBySide
# Allow exercise in sidebyside
WorksheetSideBySide =
    element sidebyside {
        SidebySideAttributes,
        (
            Figure |
            Poem |
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List |
            Stack |
            WorksheetExercise |
            WorksheetTask
        )+
    }
# Exercises and tasks can have workspace if they don't contain additional tasks:
WorksheetExercise =
    element exercise {
        MetaDataTitleOptional,
        attribute number {text}?,
        attribute workspace {text}?,
        (
        ExerciseBody |
        (StatementExercise, Hint*, Answer*, Solution*) |
        (IntroductionText?, WebWork, ConclusionText?)
        )
    }
WorksheetExercise |=
    element exercise {
        MetaDataTitleOptional,
        attribute number {text}?,
        attribute workspace {text}?,
        (IntroductionStatement?, WorksheetTask+, ConclusionStatement?)
    }
WorksheetTask =
    element task {
        MetaDataTitleOptional,
```

```
        attribute workspace {text}?,
        (
            BlockStatement+ |
            (Statement, Hint*, Answer*, Solution*)
        )
    }
WorksheetTask |=
    element task {
        MetaDataTitleOptional,
        attribute workspace {text}?,
        (IntroductionStatement?, WorksheetTask+, ConclusionStatement?)
    }
# Main worksheet definition
Worksheet =
    element worksheet {
        WorksheetAttributes,
        MetaDataAltTitleOptional,
        (Objectives? & IntroductionDivision?),
        (
        element page {WorksheetBlock+|empty}+ | WorksheetBlock+),
        (Outcomes? & ConclusionDivision?)
    }

# Insert worksheets into divisions (merge with division when adopted)
Chapter |=
    element chapter {
        MetaDataLinedTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (Worksheet? & ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                (Section | Worksheet),
                (Section | Worksheet | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Section |=
    element section {
        MetaDataLinedTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
```

```
                (Worksheet? & ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                (Subsection | Worksheet),
                (Subsection | Worksheet | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Subsection |=
    element subsection {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary)+,
                (Worksheet? & ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                (Subsubsection | Worksheet),
                (Subsubsection | Worksheet | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
Subsubsection |=
    element subsubsection {
        MetaDataAltTitle,
        AuthorByline*,
        Objectives?,
        (BlockDivision | Paragraphs | Commentary)+,
        (Worksheet? & ReadingQuestions? & Exercises? &
         Solutions? & References? & Glossary?),
        Outcomes?
    }
ArticleAppendix |=
    element appendix {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary |
```

```
                 NotationList)+,
                (Worksheet? & ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                (Subsection | Worksheet),
                (Subsection | Worksheet | ReadingQuestions | Exercises |
                 Solutions | References | Glossary)*,
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
BookAppendix |=
    element appendix {
        MetaDataAltTitle,
        AuthorByline*,
        (
            (
                Objectives?,
                (BlockDivision | Paragraphs | Commentary |
                 NotationList)+,
                (Worksheet? & ReadingQuestions? & Exercises? &
                 Solutions? & References? & Glossary?),
                Outcomes?
            )
        |
            (
                (Objectives? & IntroductionDivision?),
                (Section | Worksheet),
                (Section | Worksheet | ReadingQuestions | Exercises | Solutions | References | G
                (Outcomes? & ConclusionDivision?)
            )
        )
    }
```

9 Paragraphs

Most PreTeXt elements are about delineating structure. What you actually write happens in very few places. Principally paragraphs, but also titles, captions, index headings, and other short bursts. The shorter the burst, the more likely the text will be recycled in other places (Table of Contents, List of Figures, or Index perhaps). And the more text gets re-purposed, the more care we need to take with its contents.

Simple text is simply runs of characters, some of which is accomplished with empty elements. This is used for names of people, etc. It should not be confused with the RELAX-NG keyword `text` which matches runs of (Unicode) characters, with no intervening markup. So the latter is used for things like URLs, internal identifiers, configuration parameters, and so on.

Short text is used for titles, subtitles, names, index headings, and so on. It allows a variety of characters, font styling, groupings, and convenience con-

structions. It does not allow for references, nor anything that typographically requires more than the linearity of a sentence. In other words, no lists, no images, no tables, no displayed equations. Because of the potential for movement, we also do not include footnotes within short text.

Long text is everything that is short text, but also allows for references, both external (internet URLs) and internal (cross-references). It is used for the content of footnotes and captions. The WeBWorK variant allows for variables in inline mathematics.

⟨9 Running text⟩ ≡

```
TextSimple = mixed {
    Character* }
TextShort = mixed { (
    Character |
    Generator |
    Verbatim |
    Group |
    MathInline |
    Music)* }
TextLong =  mixed { (
    Character |
    Generator |
    Verbatim |
    Group |
    MathInline |
    Music |
    Reference |
    WWVariable)* }
```

A paragraph is a key bottleneck between structure and prose. You can use a variety of constructs in a paragraph, and you may use a paragraph in many places. So the name of the element is very simple, just a `p`. Now you can include footnotes, display mathematics, display verbatim text, and lists. Note that a list can *only* occur in a paragraph, so to make nested lists you must structure a list item of the exterior list with a paragraph to contain the interior list. A paragraph can contain some metadata, like index entries and mathematical notation. It does not have a title, nor is it ever numbered. It can be the target of a cross-reference, but only with some care.

A **lined paragraph** is a variant, for use when the line-by-line structure is necessary. The WeBWorK variant of a `p` element allows for using the `var` element as an answer blank or generated content, possibly inside mathematics, and possibly inside lists.

Note: A paragraph effectively could have the `MetaDataTarget` pattern, except that we allow index elements (`<idx>`) to go anywhere within the paragraph.

⟨10 Paragraphs⟩ ≡

```
TextParagraph = mixed { (
    Character |
    Generator |
    Verbatim |
    Group |
```

```
    WWVariable |
    MathInline |
    Music |
    Reference |
    CodeDisplay |
    MathDisplay |
    List |
    Footnote |
    Notation |
    Index)* }
Paragraph =
    element p {
        UniqueID?,
        LabelID?,
        PermanentID?,
        Component?,
        TextParagraph
    }
ParagraphLined =
    element p {
        UniqueID?,
        LabelID?,
        PermanentID?,
        Component?,
        element line {TextShort}+
    }
```

Fundamentally PreTeXt allows for conversion to other markup languages, such as LaTeX or HTML, and of course XML is a syntax for designing a markup vocabulary. As such, certain characters traditionally found on keyboards have been co-opted for special purposes. And once you actually want one of those special characters, you need an escape character to indicate a "normal" use. For these reasons, certain characters have empty elements to represent them.

Special characters for XML are the ampersand, less than, greater than, single quote and double quote: `&`, `<`, `>`, `'`, `"`. The ampersand is the escape character for XML. In practice, the first two characters are the most important, since processing of your XML will be confused by any attempt to use them directly. So in regular text (not mathematics, not verbatim), always use the the escaped versions: `&`, `<`, and perhaps `>`.

See below for elements that can be used to form *groupings* with left and right delimiters. For example, a simple quotation should use a left double quote and a right double quote, and these characters should look different (so-called **smart** quotes). Notice that a keyboard only has a single **dumb** quote. If you need these characters *in isolation* (i.e., not in pairs), these elements are the best way to ensure you get what you want in all possible conversions. Note that left and right braces , `{`, `}` ("curly brackets"); brackets, `[`, `]`; may be used directly. To create individual, left or right, create angle brackets us the elements here, not the keyboard characters (which are different).

⟨11 Delimiter characters⟩ ≡

```
Character =
    element lsq {empty} |
    element rsq {empty} |
    element rq {empty} |
```

```
    element lq {empty} |
    element langle {empty}|
    element rangle {empty}
```

A space is a space. But sometimes you want a space between two associated items which will not get split across two lines (e.g., Chapter 23). An element will create a **non-breaking space** using the right technique for the conversion at hand.

There is a variety of dashes of various lengths. Use the keyboard character for a **hyphen**, use an **ndash** to separate a range of numbers or dates, and use an **mdash** as punctuation within a sentence to isolate a clause. These are implemented differently for different conversions, so their use is strongly encouraged.

⟨12 Dash characters⟩ ≡

```
Character |=
    element nbsp {empty} |
    element ndash {empty} |
    element mdash {empty}
```

We define a few characters to help with simple arithmetic expressions authored within regular text. (Perhaps you are writing a novel with PreTeXt.) These are for simple uses in regular text, not for actual mathematics, which is described later. The **solidus** is slightly different from the **slash** found on a keyboard and is used for fractions and ratios. The `<minus/>` is for subtraction and negation, and is not a hyphen or dash. An **obelus** is better known as a division sign. `<degree/>`, `<prime/>`, and `<dblprime/>` are designed for specifying coordinates in degrees, minutes, and seconds. Use the unambiguous `+` keyboard character for addition.

⟨13 Arithmetic characters⟩ ≡

```
Character |=
    element minus {empty} |
    element times {empty} |
    element solidus {empty} |
    element obelus {empty} |
    element plusminus {empty} |
    element degree {empty} |
    element prime {empty} |
    element dblprime {empty}
```

The following are largely conveniences. They are typically not available on keyboards, and their implementations for various conversions can involve some subtleties. Again, their use is encouraged for the best quality output.

⟨14 Exotic characters⟩ ≡

```
Character |=
    element ellipsis {empty} |
    element midpoint {empty} |
    element swungdash {empty} |
    element permille {empty} |
    element pilcrow {empty} |
```

```
    element section-mark {empty} |
    element copyleft {empty} |
    element copyright {empty} |
    element registered {empty} |
    element trademark {empty} |
    element phonomark {empty} |
    element servicemark {empty}
```

Icons are available through a `@name` attribute, which is meant to usually be more semantic than just a description of the picture, though that may sometimes be the case. These are intended for use when describing elements of computer interfaces. Icons which are decorative should be supplied as part of styling, not as part of the source language.

⟨15 Icon characters⟩ ≡

```
Character |=
    element icon {
        attribute name {text}
    }
```

The `<kbd>` element will produce something akin to a calculator key or a keyboard key. It may have (simple) content, which will be reproduced as the label of the key, or it may have a `@name` attribute which describes a key that looks more like a graphic, such as an arrow key.

⟨16 Keyboard characters⟩ ≡

```
Character |=
    element kbd {
        (text | attribute name {text})
    }
```

We support musical notation as if they were characters: accidentals, scale degrees, notes, and chords. Implementation of these is about as complicated as inline mathematical notation, hence they have identical rules about placement.

⟨17 Music characters⟩ ≡

```
Music =
    element doublesharp {empty} |
    element sharp {empty} |
    element natural {empty} |
    element flat {empty} |
    element doubleflat {empty} |
    element scaledeg {"0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"10"} |
    element timesignature {
        attribute top {text},
        attribute bottom {text}
    } |
    element n {
        attribute pc {
            "A"|"B"|"C"|"D"|"E"|"F"|"G"|"a"|"b"|"c"|"d"|"e"|"f"|"g"|
            "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"10"
        },
```

```
        attribute acc {"doublesharp"|"sharp"|"flat"|"doubleflat"}?,
        attribute octave {"1"|"2"|"3"|"4"|"5"}?
    } |
    element chord {
        attribute root {text}?,
        attribute mode {text}?,
        attribute bps {text}?,
        attribute bass {text}?,
        attribute suspended {"yes"|"no"}?,
        attribute parentheses {"yes"|"no"}?,
        element alteration {
            (TextSimple |
            element sharp {empty} |
            element flat {empty})*
        }*
    }
```

⟨18 Characters⟩ ≡
⟨Dash characters 12 [15]⟩
⟨Delimiter characters 11 [14]⟩
⟨Arithmetic characters 13 [15]⟩
⟨Exotic characters 14 [15]⟩
⟨Icon characters 15 [16]⟩
⟨Keyboard characters 16 [16]⟩
⟨Music characters 17 [16]⟩

There are empty elements to generate certain items, like the date, or names of commonly referenced tools, such as PreTeXt itself. These include some common **Latin abbreviations**, for the purpose of handling the periods properly in conversions to LaTeX.

⟨19 Text generators⟩ ≡

```
Generator =
    element today {empty} |
    element timeofday {empty} |
    element tex {empty} |
    element latex {empty} |
    element xetex {empty} |
    element xelatex {empty} |
    element pretext {empty} |
    element webwork {empty} |
    element ad {empty} |
    element am {empty} |
    element bc {empty} |
    element ca {empty} |
    element eg {empty} |
    element etal {empty} |
    element etc {empty} |
    element ie {empty} |
    element nb {empty} |
    element pm {empty} |
    element ps {empty} |
    element vs {empty} |
    element viz {empty}
```

A `fillin` blank is not really a character, but maybe a really long, low dash? The `characters` attribute controls the length. It is atomic, indivisible, and content-less, like all the other characters. `fillin` is also unusual due to its allowed use within mathematics.

⟨20 Fill-in blank character⟩ ≡

```
FillInText =
    element fillin {
        attribute characters {xsd:integer}?,
        attribute rows {xsd:integer}?,
        attribute cols {xsd:integer}?,
        empty
    }
Generator |=
    FillInText
```

A large class of similarly indivisible items are units on physical quantities. The `<quantity>` element is allowed to be empty, and the code should silently produce no output. Expressing non-emptiness here might get a bit messy, so a Schematron warning could be a good alternative.

⟨21 SI units⟩ ≡

```
UnitSpecification =
        attribute prefix {text}?,
        attribute base {text},
        attribute exp {xsd:integer}?
Generator |=
    element quantity {
        element mag {text}?,
        element unit {UnitSpecification}*,
        element per {UnitSpecification}*
    }
```

Some markup is for just ASCII characters, in other words, unadorned verbatim text.

⟨22 Verbatim text⟩ ≡

```
Verbatim =
    element c {text} |
    element email {text}
```

Simple markup is groupings of text that gets a different typographic appearance, either through font changes or through delimiters. Examples are emphasis or paired quotations, non-examples are cross-references or footnotes.

Abbreviations are sequences of characters that shorten some longer word or words (e.g. VS. for the Latin *versus*), initialisms are formed from the first letters of a sequence of words (e.g. HTML), acronyms are pronounceable as words (e.g. SCUBA).

⟨23 Abbreviations⟩ ≡

```
Group |=
    element abbr {TextSimple} |
    element acro {TextSimple} |
    element init {TextSimple}
```

Notice that long text can be part of a grouping construction, and that long text can can contain a group construction. The effect is that these groupings can be nested arbitrarily deep.

⟨24 Delimited groups⟩ ≡

```
Group |=
    element q {TextLong} |
    element sq {TextLong} |
    element angles {TextLong} |
    element dblbrackets {TextLong}
```

⟨25 Highlighted groups⟩ ≡

```
Group |=
    element em {TextLong} |
    element term {TextLong} |
    element alert {TextLong} |
    element pubtitle {TextLong} |
    element articletitle {TextLong} |
    element foreign {
      XMLLang?,
      TextLong
    }
```

⟨26 Editing groups⟩ ≡

```
Group |=
    element delete {TextLong} |
    element insert {TextLong} |
    element stale {TextLong}
```

We use elements to get consistent typography when discussing PreTeXt itself. We could probably limit the content of these elements to lowercase letters and a hyphen. The definitions here will preclude any contained markup.

⟨27 XML syntax groups⟩ ≡

```
Group |=
    element tag {text} |
    element tage {text} |
    element attr {text}
```

An empty `taxon` will match either version.

⟨28 Taxonomic groups⟩ ≡

```
Group |=
    element taxon {
        attribute ncbi {xsd:integer}?,
        (
            text |
            (
                element genus {text}?,
                element species {text}?
            )
        )
    }
```

⟨29 Text groups⟩ ≡
⟨Abbreviations 23 [18]⟩
⟨Delimited groups 24 [19]⟩
⟨Highlighted groups 25 [19]⟩
⟨Editing groups 26 [19]⟩
⟨XML syntax groups 27 [19]⟩
⟨Taxonomic groups 28 [19]⟩

10 Mathematics

All mathematics appears inside paragraphs, and the syntax is that of LaTeX, as supported by MathJax, whose supported commands and macros are meant to be very similar to those of the AMSMath package. Note that the content is typically unstructured, excepting "fill-in-the-blank", WeBWorK variables (see variants), and internal cross-references in multi-row display mathematics. Also, `md` and `mdn` are not targets of cross-references, though their rows can be. Fill-in blanks have a variant attribute `@fill` more suited for mathematics.

⟨30 Mathematics⟩ ≡

```
FillInMath = element fillin {
                (attribute fill{text}?|attribute characters {xsd:integer}?),
                empty
             }
MathInline =
    element m {
        mixed {(FillInMath | WWVariable)*}
    }
MathRow =
    element mrow {
        MetaDataTarget,
        (
            attribute number {"yes" | "no"} |
            attribute tag {"star" | "dstar" | "tstar" |
                           "dagger" | "ddagger" | "tdagger" |
                           "daggerdbl" | "ddaggerdbl" | "tdaggerdbl" |
                           "hash" | "dhash" | "thash" |
                           "maltese" | "dmaltese" | "tmaltese" }
        )?,
        attribute break {"yes" | "no"}?,
```

```
        mixed {(Xref | FillInMath | WWVariable)*}
    }
MathIntertext = element intertext {TextLong}
MathDisplay =
    element me {
        mixed {(FillInMath | WWVariable)*}
    } |
    element men {
        MetaDataTarget,
        mixed {(FillInMath | WWVariable)*}
    } |
    element md {
        attribute number {"yes" | "no"}?,
        attribute break {"yes" | "no"}?,
        attribute alignment {text}?,
        attribute alignat-columns {text}?,
        MathRow,
        (MathRow | MathIntertext)*
    } |
    element mdn {
        attribute number {"yes" | "no"}?,
        attribute break {"yes" | "no"}?,
        attribute alignment {text}?,
        attribute alignat-columns {text}?,
        MathRow,
        (MathRow | MathIntertext)*
    }
```

11 Mathematics (experimental)

We include some additions to math elements, including allowing permid on displayed math and allowing xrefs inside displayed math.

⟨31 Mathematics (experimental)⟩ ≡

```
MathDisplay |=
    element me {
        MetaDataTarget?,
        mixed {(Xref | FillInMath | WWVariable)*}
    } |
    element men {
        MetaDataTarget?,
        mixed {(Xref | FillInMath | WWVariable)*}
    } |
    element md {
        attribute number {"yes" | "no"}?,
        attribute break {"yes" | "no"}?,
        attribute alignment {text}?,
        attribute alignat-columns {text}?,
        MathRow,
        (MathRow | MathIntertext)*
    } |
    element mdn {
```

```
        attribute number {"yes" | "no"}?,
        attribute break {"yes" | "no"}?,
        attribute alignment {text}?,
        attribute alignat-columns {text}?,
        MathRow,
        (MathRow | MathIntertext)*
    }
```

12 Blocks

A **text block** is very similar to a paragraph. It can be an actual paragraph, a sequence of paragraphs enclosed as a block quote (with attribution, perhaps), or a large chunk of unformatted text presented typically in a monospace font. Certain "atomic" objects, such as an `<image>` may be placed as peers of paragraph-like objects.

A **statement block** is used in statements. What are those? Theorems have statements, exercises have statements, questions have statements. Some of these blocks with statements also have peers of statements that are proofs, hints, answers, and solutions. In statements, and their peers, we include text blocks, captioned items, asides, side-by-side layouts, and Sage computations, but exclude many of the numbered and titled division blocks. A slight extension is a **solution block**, which is everything that can go in a `<statement>`, plus one or more `<proof>`, only as part of a `<hint>`, `<answer>`, or `<solution>`.

A **division block** includes text blocks, statement blocks, plus topical chunks of text that can have numbered headings or numbered captions, with optional titles, and are set apart slightly from the surrounding narrative. These are placed mostly as children of divisions, and so one cannot contain another. They certainly contain paragraphs, and all that goes into them, such as mathematics (inline and display) and figures (and other captioned items). The `sidebyside` element can be used to illustrate a division block with a variety of images and displayed text in flexible layouts.

A `<fragment>` is used for literate programming, and is numbered, so it is allowed places where other numbered items go.

Other division blocks include `poem`, `aside`, and `assemblage`. These are never numbered, but can have titles. The `list-of` mechanism is a convenience device to automatically create lists of contents, and so we leave surrounding divisional structure to the author. A `sidebyside`, and its cousin, `sbsgroup`, are strictly layout devices. The `sage` element is unique for its possibilities in certain electronic formats.

⟨32 Blocks⟩ ≡

```
BlockText =
    Paragraph | BlockQuote | Preformatted |
    Image | Video | Program | Console | Tabular
BlockStatementNoCaption =
    BlockText | Aside |
    SideBySideNoCaption | SideBySideGroupNoCaption
BlockStatement =
    BlockText |
    Figure | Aside |
    SideBySide | SideBySideGroup | Sage
BlockSolution =
```

```
    BlockStatement | Proof
BlockDivision =
    BlockStatement |
    Remark | Computation | Theorem | Proof | Definition |
    Axiom | Example | Exercise | Project |
    Poem | Assemblage | ListGenerator | Fragment
```

Blocks are often structured, in a light way. Hints, answers, and solutions adorn exercises, examples, and projects. A simple introduction or conclusion is sometimes useful. A `prelude` or `postlude` are authored inside a block and so are associated with it. But they are presented before and after the block visually. An `interlude` will be used between the statement of a theorem and its proof.

When a block is structured to allow some of the ancillary parts, a `statement` element is used to structure the main part. Hints, answers, and solutions can be the target of cross-references, but do not get author-supplied titles.

⟨33 Common components of blocks⟩ ≡

```
Prelude =
    element prelude {BlockText+}
Interlude =
    element interlude {BlockText+}
Postlude =
    element postlude {BlockText+}
Statement =
    element statement {
        BlockStatement+
    }
Hint =
    element hint {
        MetaDataTitleOptional,
        BlockSolution+
    }
Answer =
    element answer {
        MetaDataTitleOptional,
        BlockSolution+
    }
Solution =
    element solution {
        MetaDataTitleOptional,
        BlockSolution+
    }
```

13 Introductions, Conclusions, and Headnotes

The `introduction` and `conclusion` containers can be used in a variety of other structured elements. They come in three levels, according to what they can contain, and are meant to be consonant with their surroundings. As children of a division, they may carry a `title`, which in turn allows them to be cross-referenced by that text.

A `<headnote>` is like an `<introduction>`, but does not have a symmetric concluding element, and is typically meant for specialized divisions, such as a

<glossary>.

⟨34 Introductions, conclusions, headnotes⟩ ≡

```
IntroductionText =
    element introduction {BlockText+}
ConclusionText =
    element conclusion {BlockText+}
IntroductionStatementNoCaption =
    element introduction {BlockStatementNoCaption+}
ConclusionStatementNoCaption =
    element conclusion {BlockStatementNoCaption+}
IntroductionStatement =
    element introduction {BlockStatement+}
ConclusionStatement =
    element conclusion {BlockStatement+}
IntroductionDivision =
    element introduction {
        MetaDataTitleOptional,
        BlockDivision+
    }
ConclusionDivision =
    element conclusion {
        MetaDataTitleOptional?,
        BlockDivision+
    }
HeadNote =
    element headnote {BlockStatementNoCaption+}
```

14 References

There are a variety of referencing mechanisms, external references, internal cross-references, index entries, and specialized support for a table of mathematical notation.

⟨35 Cross-references⟩ ≡

```
XrefTextStyle =
    "local" | "global" | "hybrid" | "type-local" | "type-global" |
    "type-hybrid" | "phrase-global" | "phrase-hybrid" |
    "title" | "custom"
Reference = Url | Xref
Url =
    element url {
        attribute href {text},
        (
            (attribute visual {text},
            TextShort
            )|(
            attribute visual {text}?
            )
        )
    }
```

```
Xref =
    element xref {
            (
                attribute ref {text} |
                (attribute first {text}, attribute last {text}) |
                attribute provisional {text}
            ),
            attribute text { XrefTextStyle }?,
            attribute detail {text}?,
            TextShort
    }
Notation =
    element notation {
        element usage {MathInline},
        element description {
            TextShort
        }
    }
```

Footnotes are especially dangerous. They should contain quite a bit of content, and should be targets of cross-references. So the content is not as expansive as a regular paragraph, which is possibly too restrictive.

⟨36 Footnotes⟩ ≡

```
Footnote =
    element fn {
        MetaDataTarget,
        TextLong
    }
```

Index entries have two forms, simple and structured. The `start` and `finish` attributes are meant to use `xml:id` to create an index range that crosses XML boundaries. (Replace principal tags with `idx/h/h`.)

The actual index is generated within the `index-part` via the `index-list` element.

Note that we might point to another index entry as part of a "see also" mechanism.

⟨37 Index entries⟩ ≡

```
IdxHeading =
    element h {
        attribute sortby {text}?,
        TextShort
    }
Index =
    element idx {
        MetaDataTarget,
        attribute sortby {text}?,
        attribute start {text}?,
        attribute finish {text}?,
        (
            TextShort
        |
```

```
            (
            IdxHeading,
            IdxHeading?,
            IdxHeading?,
            (element see {TextShort} | element seealso {TextShort})?
            )
        )
    }
IndexList = element index-list {empty}
```

15 References (experimental)

Here we collect experimental versions of reference-like elements. Specifically, changes to Url, that only recommend a visual element when the URL element contains content.

⟨38 References (experimental)⟩ ≡

```
Url |=
    element url {
        attribute href {text},
        (
            (attribute visual {text},
            TextShort
            )| TextShort |
            (
            attribute visual {text}?
            )
        )
    }
```

16 Objectives

A division may lead (first) with an optional list of objectives for the division and may be followed by a (final) optional list of outcomes. The element names are only chosen to reflect a pre- and post- behavior and so could be used for objectives, outcomes, and standards in a variety of ways.

⟨39 Objectives and outcomes⟩ ≡

```
Objectives =
    element objectives {
        MetaDataTitleOptional,
        IntroductionText?,
        List,
        ConclusionText?
    }
Outcomes =
    element outcomes {
        MetaDataTitleOptional,
        IntroductionText?,
        List,
```

```
        ConclusionText?
    }
```

17 Block Quotes

These are a run of paragraphs, but may optionally have an `attribution`.

⟨40 Block quotes⟩ ≡

```
BlockQuote =
    element blockquote {
        MetaDataTitleOptional,
        Paragraph+,
        Attribution?
    }
SimpleLine =
    element line {TextSimple}
ShortLine =
    element line {TextShort}
LongLine =
    element line {TextLong}
```

18 Verbatim Text

Large blocks of verbatim material, rather than just little bits in a sentence. A code display, `cd`, is an analog of a math display, and meant to be used *within* a paragraph, either as a single line of text, or optionally structured as several lines by using code lines, `cline`. `pre` is a block, which preserves line breaks and sanitizes whitespace to the left. It can be optionally structured as code lines. It should be thought of as a monospace analogue of a "regular" paragraph, minus indentation and automatic line-breaking.

⟨41 Verbatim displays⟩ ≡

```
CodeLine =
    element cline {text}
CodeDisplay =
    element cd {
        attribute latexsep {text}?,
        (text | CodeLine+)
    }
Preformatted =
    element pre {
        text | CodeLine+
    }
Console =
    element console {
        PermanentID?,
        Component?,
        attribute prompt {text}?,
        attribute width {text}?,
        attribute margins {text}?,
```

```
        (
            element input {
                attribute prompt {text}?,
                text
            },
            element output {text}?
        )+
    }
Program =
    element program {
        PermanentID?,
        Component?,
        attribute width {text}?,
        attribute margins {text}?,
        attribute language {text}?,
        attribute line-numbers {"yes"|"no"}?,
        attribute highlight-lines {text}?,
        attribute interactive {"codelens"}?,
        element input {text}
    }
```

19 Lists

Are complicated. Maybe we need a special type of paragraph which does not allow nesting a description list down into some other list?

As a container, the lists themselves get no metadata. But the numbered or titled list items do get metadata. To point to an entire list, make it a **named list** and point to that.

⟨42 Lists⟩ ≡

```
List =
    element ol {
        PermanentID?,
        Component?,
        attribute cols {"2"|"3"|"4"|"5"|"6"}?,
        attribute marker {text}?,
        element li {
            (
                (MetaDataTarget, TextParagraph)
            |
                (MetaDataTitleOptional, BlockStatement+)
            )
        }+
    } |
    element ul {
        PermanentID?,
        Component?,
        attribute cols {"2"|"3"|"4"|"5"|"6"}?,
        attribute marker {"disc" | "circle" | "square" | ""}?,
        element li {
            (
                (MetaDataTarget, TextParagraph)
```

```
            |
                (MetaDataTitleOptional, BlockStatement+)
            )
        }+
    } |
    element dl {
        PermanentID?,
        Component?,
        attribute width {"narrow" | "medium" | "wide"}?,
        element li {
            MetaDataTitle,
            BlockStatement+
        }+
    }
```

20 Definitions

Definitions are special, there is nothing else quite like them. A statement, no proof, and also a natural place for notation entries.

⟨43 Definitions⟩ ≡

```
DefinitionLike =
    MetaDataTitleOptional,
    Notation*,
    Statement
Definition =
    element definition {DefinitionLike}
```

21 Theorems, And Other Results

Theorems, corollaries, lemmas — they all have statements, and should have proof(s). Otherwise they are all the same. A proof may be divided with cases, in no particular rigid way, just as a marker of any number of different, non-overlapping portions of a proof. Titles can be used to describe each case, or implication arrows may be used (typically with a proof of an equivalence). A `proof` is also allowed to stand on its own as a block, independent of a structure like a `theorem` or `algorithm`.

⟨44 Theorems, and similar⟩ ≡

```
Case =
    element case {
       MetaDataTitleOptional,
       attribute direction {text}?,
       BlockStatement+
       }
Proof =
    element proof {
        MetaDataTitleOptional,
        (BlockStatement | Case)+
    }
```

```
TheoremLike =
    MetaDataTitleCreatorOptional,
    (BlockStatement+ | (Statement, Proof*))
Theorem =
    element theorem {TheoremLike} |
    element lemma {TheoremLike} |
    element corollary {TheoremLike} |
    element claim {TheoremLike} |
    element proposition {TheoremLike} |
    element algorithm {TheoremLike} |
    element fact {TheoremLike} |
    element identity {TheoremLike}
```

22 Proof-like (experimental)

We extend the types of elements that are types of proofs, as well as create a `ProofLike` named pattern for what can go in them.

⟨45 Proofs, and similar⟩ ≡

```
ProofLike =
    MetaDataTitleOptional,
    (BlockStatement | Case)+
Proof |=
    element proof {ProofLike} |
    element argument {ProofLike} |
    element justification {ProofLike} |
    element reasoning {ProofLike} |
    element explanation {ProofLike}
```

23 Axioms and Other Mathematical Statements

Mathematical statements that do not have proofs (in other words, no proof is known, or a proof is not appropriate).

⟨46 Axioms, and similar⟩ ≡

```
AxiomLike =
    MetaDataTitleCreatorOptional,
    Statement
Axiom =
    element axiom {AxiomLike} |
    element principle {AxiomLike} |
    element conjecture {AxiomLike} |
    element heuristic {AxiomLike} |
    element hypothesis {AxiomLike} |
    element assumption {AxiomLike}
```

24 Projects and Activities

A favorite of Inquiry-Based Learning textbooks. Numbered independently. Possibly structured with `task`. Three different ways to structure this, we

combine the second two so that the derived XML Schema (XSD) version is less-confusing to certain tools (e.g. the Red Hat XML schema validator used within VS Code).

⟨47 Projects, and similar⟩ ≡

```
ProjectLike =
    MetaDataTitleOptional,
    (
        (BlockStatement+) |
        (
           Prelude?,
           (
              (Statement, Hint*, Answer*, Solution*) |
              (IntroductionStatement?, Task+, ConclusionStatement?) |
              (IntroductionText?, WebWork, ConclusionText?)
           ),
           Postlude?
        )
    )
Project =
    element activity {ProjectLike} |
    element investigation {ProjectLike} |
    element exploration {ProjectLike} |
    element project {ProjectLike}
Task =
    element task {
        MetaDataTitleOptional,
        (
            BlockStatement+ |
            (Statement, Hint*, Answer*, Solution*) |
            (IntroductionStatement?, Task+, ConclusionStatement?)
        )
    }
```

25 Remarks and Other Comments

Really simple blocks, they do not have much structure, and so are just runs of paragraphs, though `<figure>`, `<table>`, `<listing>`, and `<list>` may be included.

⟨48 Remarks, and similar⟩ ≡

```
RemarkLike =
    MetaDataTitleOptional,
    BlockStatement+
Remark =
    element remark {RemarkLike} |
    element convention {RemarkLike} |
    element note {RemarkLike} |
    element observation {RemarkLike} |
    element warning {RemarkLike} |
    element insight {RemarkLike}
```

26 Computations and Technology

Somewhat simple blocks, they do not have much structure, but can hold more than a Remark.

⟨49 Computation, and similar⟩ ≡

```
ComputationLike =
    MetaDataTitleOptional,
    BlockStatement+
Computation =
    element computation {ComputationLike} |
    element technology {ComputationLike} |
    element data {ComputationLike}
```

27 Asides

An aside is a deviation from the narrative, and might physically move in the presentation (say, to a margin, or to a knowl). `biographical` and `historical` may be further developed.

⟨50 Asides, and similar⟩ ≡

```
AsideLike =
    MetaDataTitleOptional,
    BlockText+
Aside =
    element aside {AsideLike} |
    element biographical {AsideLike} |
    element historical {AsideLike}
```

28 Assemblages

Since an `assemblage` is meant to accumulate significant content (as a review or summary, or for initial presentation) lists are allowed here, an exception to their restriction to paragraphs. We are also mildly restrictive about what can be content here—in particular blocks are excluded, despite not strictly being blocks themselves.

⟨51 Assemblages⟩ ≡

```
Assemblage =
    element assemblage {
        MetaDataTitleOptional,
        (BlockText | SideBySideNoCaption | SideBySideGroupNoCaption)+
    }
```

29 Figures, Tables, Listings and Named Lists

These are containers that all carry titles (mandatory and optional), captions for two, and numbers. They need to be filled with other (atomic) items, which

we generally call **planar** due to their two-dimensional and rigid characteristics. These have also called **captioned items** in the code, even if not all allow a caption. The option for a lanscape orientation is only relevant for print, and not within a `sidebyside`.

⟨52 Captioned and titled displays⟩ ≡

```
Caption =
    element caption {TextLong}
Landscape =
    attribute landscape {"yes" | "no"}
Figure =
    element figure {
        MetaDataCaption,
        Landscape?,
        (
            Image |
            Video |
            SideBySide |
            SideBySideGroup |
            MuseScore
        )
    } |
    element table {
        MetaDataAltTitle,
        Landscape?,
        Tabular
    } |
    element listing {
        MetaDataCaption,
        Landscape?,
        (
            Program |
            Console
        )
    } |
    element list {
        MetaDataAltTitle,
        Landscape?,
        IntroductionText?,
        List,
        ConclusionText?
    }
```

The guts of a table go in a `tabular` element.

⟨53 Tabular display⟩ ≡

```
BorderThickness = "none" | "minor" | "medium" | "major"
BorderTop =
    attribute top {BorderThickness}
BorderBottom =
    attribute bottom {BorderThickness}
BorderLeft =
```

```
    attribute left {BorderThickness}
BorderRight =
    attribute right {BorderThickness}
AlignmentHorizontal =
    attribute halign {"left" | "center" | "right" | "justify"}
AlignmentVertical =
    attribute valign {"top" | "middle" | "bottom"}

TableCell =
    element cell {
        AlignmentHorizontal?,
        BorderBottom?,
        BorderRight?,
        attribute colspan {text}?,
        (
            TextLong |
            LongLine+ |
            Paragraph+
        )
    }
TableRow =
    element row {
        attribute header {"yes" | "no" | "vertical"}?,
        AlignmentHorizontal?,
        AlignmentVertical?,
        BorderBottom?,
        BorderLeft?,
        TableCell+
    }
TableColumn =
    element col {
        AlignmentHorizontal?,
        BorderTop?,
        BorderRight?,
        attribute width {text}?
    }
Tabular =
    element tabular {
        PermanentID?,
        Component?,
        attribute width {text}?,
        attribute margins {text}?,
        attribute row-headers {"yes" | "no"}?,
        AlignmentHorizontal?,
        AlignmentVertical?,
        BorderTop?,
        BorderBottom?,
        BorderLeft?,
        BorderRight?,
        TableColumn*,
        TableRow+
    }
```

30 Figure (experimental)

We add tabular as a valid child of a figure.

⟨54 Figure (experimental)⟩ ≡

```
Figure |=
    element figure {
        MetaDataCaption,
        Tabular
    }
```

31 Side-By-Side Layout

Page width or screen width, both are at a premium. Height goes on forever (barring physical page breaks) and we have many devices for demarcating that flow. But sometimes you need to organize items horizontally, i.e. side-by-side. We place the components of a `sidebyside` into generic regions of specified width called **panels**.

This is a pure layout device. So you cannot title it, nor caption it. It does not admit a `xml:id` attribute, since you cannot make it the target of a cross-reference. Nor can you reference it from the index (but you can point to its surroundings from the index).

Because of its utility, it can go anywhere a block can go (i.e., as a child of a division) and it can go many other places as a sibling of a paragraph (such as to illustrate an `example`).

Note that widths give on a `sidebyside` override any width given to the components of the panels.

A `<stack>` allows non-captioned, non-titled elements to accumulate vertically in a single panel. It is a basic container.

A group of side-by-sides is designed to stack vertically with common controls on widths, etc. Its implementation is entirely experimental right now, even if we are relatively confident of the markup.

⟨55 Side-by-side layouts⟩ ≡

```
Stack =
    element stack {
        (
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List
        )+
    }
SidebySideAttributes =
    PermanentID?,
    Component?,
    attribute margins {text}?,
```

```
    (attribute width {text} | attribute widths {text})?,
    (AlignmentVertical | attribute valigns {text})?
SideBySide =
    element sidebyside {
        SidebySideAttributes,
        (
            Figure |
            Poem |
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List |
            Stack
        )+
    }
SideBySideNoCaption =
    element sidebyside {
        SidebySideAttributes,
        (
            Poem |
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List |
            Stack
        )+
    }
SideBySideGroup =
    element sbsgroup {
        SidebySideAttributes,
        SideBySide+
    }
SideBySideGroupNoCaption =
    element sbsgroup {
        SidebySideAttributes,
        SideBySideNoCaption+
    }
```

32 Images and Graphics

Raster, and described by languages, plus 100% duplicates. The WeBWorK variant is quite different.

Note: the `ImageCode` pattern allows an `@xml:id` attribute since it is used to construct a filename.

⟨56 Images⟩ ≡

```
Image = ImageRaster | ImageCode
ImageRaster =
    element image {
        UniqueID?,
        PermanentID?,
        Component?,
        attribute width {text}?,
        attribute margins {text}?,
        attribute rotate {text}?,
        attribute archive {text}?,
        attribute source {text},
        (
          attribute decorative {"yes"} |
          (
            attribute decorative {"no"}?,
            (
              element shortdescription {text}? &
              element description {(Paragraph | Tabular)+}?
            )
          )
        )
    }
ImageCode =
    element image {
        UniqueID?,
        PermanentID?,
        Component?,
        attribute width {text}?,
        attribute margins {text}?,
        attribute archive {text}?,
        (
          attribute decorative {"yes"} |
          (
            attribute decorative {"no"}?,
            (
              element shortdescription {(text | WWVariable)+}? &
              element description {(Paragraph | Tabular)+}? &
              (
                element latex-image {
                  LabelID?,
                  Component?,
                  text
                } |
                element asymptote {
                  LabelID?,
                  Component?,
                  text
                } |
                element sageplot {
                    LabelID?,
                    Component?,
                    attribute variant {'2d'|'3d'}?,
                    attribute aspect {text}?,
```

```
                    text
                }
              )
            )
          )
        )
    }
ImageWW =
    element image {
        attribute pg-name {text}?,
        attribute width {text}?,
        (
          attribute decorative {"yes"} |
          (
            attribute decorative {"no"}?,
            (
              element shortdescription {(text | WWVariable)+}? &
              element description {(Paragraph | Tabular)+}? &
              element latex-image {
                text
              }?
            )
          )
        )
    }
```

33 Sage Code

Sage is integral.

⟨57 Sage code⟩ ≡

```
Sage = element sage {
    PermanentID?,
    Component?,
    attribute doctest {text}?,
    attribute tolerance {text}?,
    attribute auto-evaluate {'no'|'yes'}?,
    attribute language {text}?,
    attribute type {text}?,
    (element input {text}, element output {text}?)?
}
```

34 Legacy Interactive Elements

Some specific interactive goodies. These are being phased-out in favor of a more general <interactive> element.

⟨58 Interactives⟩ ≡

```
MuseScore =
    element score {
```

```
        attribute musescoreuser {text},
        attribute musescore {text}
    }
```

35 Interactive Elements (experimental)

A general <interactive> element.

⟨59 Interactives⟩ ≡

```
Interactive =
    element interactive {
        UniqueID?,
        LabelID?,
        PermanentID?,
        Component?,
        attribute aspect { text }?,
        attribute width { text }?,
        attribute platform { text }?,
        attribute preview { text }?,
        attribute iframe { text }?,
        attribute source { text }?,
        attribute version { text }?,
        (
          (
            Slate |
            SideBySideNoCaption |
            SideBySideGroupNoCaption
          )* &
          element instructions { mixed { MetaDataTitleOptional, BlockText } }? &
          element static { Image }?
        )

    }

Stack |=
    element stack {
        (
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List |
            Slate
        )+
    }

Slate =
    element slate {
        UniqueID?,
```

```
        LabelID?,
        Component?,
        (
          JessieCodeAtt |
          (
            attribute surface { text },
            (
              attribute source { text } |
              attribute material { text }
            )?,
            attribute aspect { text }?,
            (
              Paragraph |
              Tabular |
              SideBySideNoCaption |
              SlateInput |
              element xhtml:button {
                attribute type { text },
                attribute id { text },
                text*
              }? |
              text*
            )*
          )
        )
    }

  JessieCodeAtt =
    attribute surface {"jessiecode"},
    attribute axis {"true" | "false"}?,
    attribute grid {"true" | "false"}?,
    (
      attribute source {text} |
      text*
    )

  SlateInput =
    element xhtml:input {
      attribute type {text}?,
      attribute value {text}?,
      attribute onkeypress {text}?,
      attribute onclick {text}?,
      attribute style {text}?
    } |
    element input {
      attribute type {text}?,
      attribute value {text}?,
      attribute onkeypress {text}?,
      attribute onclick {text}?,
      attribute style {text}?
    }

# add Interactives where used
BlockStatement |= Interactive
```

```
Figure |= element figure { MetaDataCaption, Interactive }

SideBySide |= element sidebyside {
    SidebySideAttributes,
    (Interactive | Slate)+
}

SideBySideNoCaption |= element sidebyside {
    SidebySideAttributes,
    (Interactive | Slate)+
}

Exercises |= element exercises {
    MetaDataAltTitleOptional,
    IntroductionDivision?,
    (
    (Exercise | ExerciseGroup)+ |
    Subexercises+ | Interactive
    ),
    ConclusionDivision?
}
```

36 Audio and Video

Well, just video right now. The `xml:id` is not used as a target, but rather as a name for a static preview image that is auto-generated by the `pretext` script thumbnail file, hence optional. `preview` maybe be one of two reserved switches, or the filename of a static preview image.

Note: the `Video` pattern allows an `@xml:id` attribute since it is used to construct a filename for preview images ("poster"), especially when scraped.

⟨60 Video and audio⟩ ≡

```
Video =
    element video {
        UniqueID?,
        LabelID?,
        PermanentID?,
        Component?,
        attribute width {text}?,
        attribute margins {text}?,
        attribute aspect {text}?,
        attribute start {xsd:integer}?,
        attribute end {xsd:integer}?,
        attribute play-at {"embed" | "popout" | "select"}?,
        attribute preview {"default" | "generic" | text}?,
        (AttributesSourceFile | AttributesNetwork | AttributesYouTube |
         AttributesYouTubePlaylist | AttributesVimeo)
    }
AttributesSourceFile =
    attribute source {text}
AttributesNetwork =
```

```
    attribute href {text}
AttributesYouTube =
    attribute youtube {text}
AttributesYouTubePlaylist =
    attribute youtubeplaylist {text}
AttributesVimeo =
    attribute vimeo {text}
```

37 Poetry

Poems!

⟨61 Poems⟩ ≡

```
AlignmentPoem = attribute halign {"left" | "center" | "right"}
Poem =
    element poem {
        MetaDataTitleOptional,
        AlignmentPoem?,
        element author {
            AlignmentPoem?,
            TextShort
        }?,
        (PoemLine+ | Stanza+)
    }
Stanza =
    element stanza {
        MetaDataTitleOptional,
        PoemLine+
    }
PoemLine =
    element line {
        attribute indent {xsd:integer}?,
        TextShort
    }
```

38 Exercises

Inline, divisional, and WeBWorK. Exercises use task to structure parts, where before they used ordered lists for parts of a statement (to eventually be deprecated).

⟨62 Exercises⟩ ≡

```
ExerciseBody =
    (
        BlockStatement |
        element ol {
            attribute cols {text}?,
            attribute marker {text}?,
            element li {
                MetaDataTarget,
```

```
                (TextParagraph | BlockText+)
            }+
        }
    )+
StatementExercise =
    element statement { ExerciseBody }
Exercise =
    element exercise {
        MetaDataTitleOptional,
        attribute number {text}?,
        (
        ExerciseBody |
        (StatementExercise, Hint*, Answer*, Solution*) |
        (IntroductionStatement?, Task+, ConclusionStatement?) |
        (IntroductionText?, WebWork, ConclusionText?)
        )
    }
ExerciseGroup =
    element exercisegroup {
        MetaDataTitleOptional,
        attribute cols {"2"|"3"|"4"|"5"|"6"}?,
        IntroductionStatementNoCaption,
        Exercise+,
        ConclusionStatementNoCaption?
    }
```

39 Exercises (experimental)

We can have exercises that are interactive, such as true/false, multiple choices, Parson's problems, etc.

⟨63 Exercises (experimental)⟩ ≡

```
TrueFalse =
    MetaDataTitleOptional,
    attribute number {text}?,
    element statement {
        attribute correct {"yes"|"no"},
        Paragraph
    },
    Feedback?, Hint*, Answer*, Solution*
MultipleChoice =
    MetaDataTitleOptional,
    attribute number {text}?,
    StatementExercise,
    element choices {
        attribute randomize {"yes"|"no"}?,
        Choice+
    },
    Hint*, Answer*, Solution*
Choice =
    element choice {
        attribute correct {"yes"|"no"}?,
```

```
        ((mixed {BlockText?})
        | (StatementExercise, Feedback?))
    }
Parsons =
        MetaDataTitleOptional,
        attribute number {text}?,
        attribute language {text}?,
        attribute adaptive {"yes"|"no"}?,
        attribute indentation {text}?,
        StatementExercise,
        element blocks {
            attribute layout {"horizontal"}?,
            attribute randomize {"yes"|"no"}?,
            Block+
        },
        Hint*, Answer*, Solution*
Block =
    element block {
        attribute order {xsd:integer}?,
        ((
            attribute correct {"yes"|"no"}?,
            mixed {BlockText?, CodeLine?}+
        ) |
        (
            element choice {
                attribute correct {"yes"|"no"}?,
                mixed {BlockText?, CodeLine?}+
            }+
        ))
    }
Matching =
    MetaDataTitleOptional,
    attribute number {text}?,
    StatementExercise,
    Feedback?,
    element matches {
        Match+
    },
    Hint*, Answer*, Solution*
Match =
    element match {
        attribute order {xsd:integer}?,
        element premise {
            mixed {BlockText?}
        },
        element response {
            mixed {BlockText?}
        }
    }
FreeResponse =
    MetaDataTitleOptional,
    attribute number {text}?,
    (
    (ExerciseBody, Response?) |
```

```
    (StatementExercise, Response?, Hint*, Answer*, Solution*) |
    (IntroductionStatement?, Task+, ConclusionStatement?)
    )
Response =
    element response {empty}


# Selectable areas
Area =
    element area {
        attribute correct {"yes"|"no"}?,
        TextLong
    }
TextLongAreas =  mixed { (
      Area |
      Character |
      Generator |
      Verbatim |
      GroupAreas |
      MathInline |
      Music |
      Reference |
      WWVariable)* }
GroupAreas |=
    element q {TextLongAreas} |
    element sq {TextLongAreas}
TextParagraphAreas = mixed { (
  Character |
  Generator |
  Verbatim |
  Group |
  WWVariable |
  MathInline |
  Music |
  Reference |
  CodeDisplay |
  MathDisplay |
  List |
  Footnote |
  Notation |
  Index |
  Area |
  GroupAreas)* }
ParagraphAreas =
    element p {
        UniqueID?,
        LabelID?,
        PermanentID?,
        Component?,
        TextParagraphAreas
    }
Areas =
    MetaDataTitleOptional,
    attribute number {text}?,
```

```
    StatementExercise,
    Feedback?,
    element areas {
      ParagraphAreas+
    },
    Hint*, Answer*, Solution*

# General feedback element
Feedback =
    element feedback {
        MetaDataTitleOptional,
        BlockSolution+
    }
# Include all exercise types in exercise and activity
Exercise |=
    element exercise {
        TrueFalse |
        MultipleChoice |
        Parsons |
        Matching |
        FreeResponse |
        Areas
    }
ProjectLike |=
    TrueFalse |
    MultipleChoice |
    Parsons |
    Matching |
    FreeResponse |
    Areas
```

40 Bibliography

This is all stop-gap and will change radically. But it seems to work for now. So these rules should not be taken as definitive, at all.

⟨64 Bibliography⟩ ≡

```
TextBib = mixed { (Character | MathInline)* }
BibliographyItem =
    element biblio {
        MetaDataTarget,
        ((
            attribute type {"raw"},
            (TextLong |
            Ibid |
            BibTitle |
            BibYear |
            BibJournal |
            BibNumber |
            BibVolume |
            BibNote)*
        ) |
```

```
        (
            attribute type {"bibtex"},
            (BibTitle |
            BibAuthor |
            BibEditor |
            BibYear |
            BibJournal |
            BibNumber |
            BibVolume |
            BibSeries |
            BibPublisher |
            BibPages |
            BibNote)*
        ))
    }
Ibid = element ibid {empty}
BibYear = element year {text}
BibJournal = element journal { TextBib }
BibNumber = element number {text}
BibVolume = element volume {text}
BibTitle = element title {TextLong}
BibNote = element note {UniqueID?, Paragraph+}
BibAuthor = element author {text}
BibEditor = element editor {text}
BibSeries = element series {text}
BibPublisher = element publisher {text}
BibPages = element pages {
    (
        attribute start {text},
        attribute end {text},
        empty
    ) |
    (
        text
    )
    }
```

41 Glossary

A <glossary> is primarly built up as a sequence of "glossary items,", using the <gi> element, by analogy with list items.

⟨65 Glossary⟩ ≡

```
GlossaryItem =
    element gi {
        MetaDataTitle,
        BlockStatementNoCaption+
    }
```

42 Examples and Questions

Expository, but with solutions, etc. (Borrows from exercises and projects.)

⟨66 Examples, and similar⟩ ≡

```
ExampleLike =
    MetaDataTitleOptional,
    (
        (BlockStatement)+ |
        (Statement, Hint*, Answer*, Solution*) |
        (IntroductionStatement?, Task+, ConclusionStatement?)
    )
Example =
    element example {ExampleLike} |
    element question {ExampleLike} |
    element problem {ExampleLike}
```

43 WeBWorK Exercises

Modified versions of various aspects to allow authoring WeBWorK exercises.
Notes:

- Statements, hints and solutions do not require at least one paragraph, so may be just a table or figure (say).
- Are static and set elements mutually exclusive?
- Can the usage part of the var element be split across math and paragraphs?

⟨67 WeBWorK⟩ ≡

```
WebWork = (WebWorkAuthored | WebWorkSource)
WebWorkSource =
    element webwork {
        attribute source {text}?,
        attribute seed {xsd:integer}?
    }
WebWorkAuthored =
    element webwork {
        UniqueID?,
        LabelID?,
        Component?,
        attribute seed {xsd:integer}?,
        attribute copy {text}?,
        element description {
            (
                TextSimple |
                SimpleLine+
            )
        }?,
        WWMacros?,
        element pg-code {text}?,
        (
            (StatementExerciseWW, HintWW?, SolutionWW?)
        |
```

```
            (IntroductionText?, TaskWW+, ConclusionText?)
        )
    }
BlockStatementWW =
            Paragraph |
            Preformatted |
            Tabular |
            ImageWW
StatementExerciseWW =
    element statement {
        (BlockStatementWW|WWInstruction)+
    }
TaskWW =
    element task {
        MetaDataTitleOptional,
        (
            (StatementExerciseWW, HintWW?, SolutionWW?) |
            (IntroductionText?, TaskWW+, ConclusionText?)
        )
    }
WWMacros =
    element pg-macros {
        element macro-file {text}+
    }
WWVariable =
    ## The WeBWorK "var" element appears in the RELAX-NG schema as a child of many elements, but
    element var {
        (attribute name {text},
        attribute evaluator {text}?,
        attribute width {text}?,
        attribute category {
                             "angle" | "decimal" | "exponent"
                           | "formula" | "fraction" | "inequality"
                           | "integer" | "interval" | "logarithm"
                           | "limit" | "number" | "point"
                           | "syntax" | "quantity" | "vector"
                           }?,
        attribute form {"popup"|"buttons"|"checkboxes"|"none"}?) |
        (attribute form {"essay"},
        attribute width {text}?)
    }
WWInstruction =
    element instruction {TextShort}
HintWW =
    element hint {
        (BlockStatementWW)+
    }
SolutionWW =
    element solution {
        (BlockStatementWW)+
    }
```

44 Literate Programming

Literate programming is a technique for documenting programs, with code **fragments** rearranged to create a syntactically correct program. A root fragment is indicated by `@filename` which could have an `@xml:id`, otherwise the `@xml:id` is required.

⟨68 Literate programming⟩ ≡

```
Fragment =
    element fragment {
        (
            attribute xml:id {text}
        |
            (
                attribute filename {text},
                attribute xml:id {text}?
             )
        ),
        Title,
        (
            element code {text} |
            element fragref {
                attribute ref {text}
            }
        )+
    }
```

45 Frequently Used

Frequently used items, with no natural place to associate them.

⟨69 Frequent constructions⟩ ≡
⟨Attribution 70 [50]⟩
⟨Metadata 71 [51]⟩
Used on the end of prefaces to “sign” them, and on block quotes.

⟨70 Attribution⟩ ≡

```
Attribution =
    element attribution {
        (TextLong | LongLine+)
    }
```

There is a handful of elements which describe an item, but do not necessarily get processed as content. Titles are an obvious example, and index entries are another. Here we isolate a few common patterns to use for consistency throughout.

Notes:

- Language tags go on the root element to affect variants of names of objects, like theorems.
- `@permid` is part of managing editions, and is supplied by a script. You should not be adding these manually as an author. (You do want to manually author `@xml:id`.)

- The `xinlude` mechanism may pass language tags down through the root element of included files to make them universally available.
- The `xinclude` mechanism inserts a `@xml:base` attribute on the root element of an included file. So we allow this attribute on any element that allows a title.
- The `component` attribute allows versions to be controlled by a publisher file.
- These are not unordered specifications since they contain several attributes, and we enforce a `title`, `subtitle`, `<shorttitle>`, `<plaintitle>`, `creator`, `caption`, `idx` order.
- `MetaDataTarget` is for items that are targets of cross-references, but without even optional titles. Since they will be knowled, they can appear in an index. But without the potential to be titled, we do not set them up as possible root elements of a file to `xinclude`.
- `MetaDataTitle` has a required `<title>`.
- `MetaDataAltTitle` has a required `<title>`, and allows optional `<shorttitle>` and `<plaintitle>`.
- `MetaDataSubtitle` implicitly has a required `<title>`, and allows optional `<subtitle>`, `<shorttitle>` and `<plaintitle>`.
- A `<plaintitle>` means no markup whatsoever in the content, this is what "plain" means.
- `MetaDataLinedTitle` and `MetaDataLinedSubtitle` are variants of the `AltTitle` or `Subtitle` versions for use on larger divisions with `<line>` elements used to suggest line breaks in titles.
- `MetaDataCaption` implicitly has an optional title.
- Titles may contain external references (`url`) or internal cross-references (`xref`), but implementers need not make them active (i.e., they maybe text only), since titles are prone to migrating to other locations.

⟨71 Metadata⟩ ≡

```
UniqueID =
    attribute xml:id {text}
LabelID =
    attribute label {text}
PermanentID =
    attribute permid {text}
Component =
    attribute component {text}
Title =
    element title {TextLong}
LinedTitle =
    element title {LongLine+}
Subtitle =
    element subtitle {TextLong}
LinedSubtitle =
```

```
    element subtitle {LongLine+}
ShortTitle =
    element shorttitle {TextShort}
PlainTitle =
    element plaintitle {text}
Creator =
    element creator {TextShort}
XMLBase = attribute xml:base {text}
XMLLang = attribute xml:lang {text}
MetaDataTarget =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    Index*
MetaDataTitle =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title,
    Index*
MetaDataAltTitle =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title,
    ShortTitle?,
    PlainTitle?,
    Index*
MetaDataLinedTitle =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    (Title | LinedTitle),
    ShortTitle?,
    PlainTitle?,
    Index*
MetaDataSubtitle =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title,
```

```
    Subtitle?,
    ShortTitle?,
    PlainTitle?,
    Index*
MetaDataLinedSubtitle =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    (Title | LinedTitle),
    (Subtitle | LinedSubtitle)?,
    ShortTitle?,
    PlainTitle?,
    Index*
MetaDataTitleOptional =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title?,
    Index*
MetaDataAltTitleOptional =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    (Title, ShortTitle?, PlainTitle?)?,
    Index*
MetaDataTitleCreatorOptional =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title?,
    Creator?,
    Index*
MetaDataCaption =
    UniqueID?,
    LabelID?,
    PermanentID?,
    Component?,
    XMLBase?,
    XMLLang?,
    Title?,
    Caption,
    Index*
```

46 Miscellaneous

Provisional items, with uncertain futures.

⟨72 Miscellaneous or uncertain⟩ ≡

47 Organizational Devices

A **list generator** is a convenient device. It can create appendices, or smaller table-of-contents at the start of divisions.

Notation can be automatically generated. We restrict its locations to appendices.

⟨73 List generator⟩ ≡

```
ListGenerator =
    element list-of {
        attribute elements {text},
        attribute scope {text}?,
        attribute divisions {text}?,
        attribute empty {"yes" | "no"}?
    }
NotationList =
    element notation-list {empty}
```

48 Front Matter

Articles and books have material at the start, which gets organized in interesting ways. `minilicense` is very restrictive, `shortlicense` allows references (e.g. URLs). `titlepage` is like a very small database—for HTML it migrates to the top of the page for the `frontmatter`, and for LaTeX it migrates to the half-title and title pages. Since it generally makes no sense as the target of a cross-reference, `titlepage` does not allow an `@xml:id` attribute.

⟨74 Front matter⟩ ≡

```
ArticleFrontMatter =
    element frontmatter {
        MetaDataTitleOptional,
        TitlePage,
        Abstract?
    }
BookFrontMatter = element frontmatter {
        MetaDataTitleOptional,
        TitlePage,
        ColophonFront?,
        Biography*,
        Dedication?,
        Acknowledgement?,
        Preface*
    }
TitlePage =
```

```
    element titlepage {
        (
            (Author, Author*, Editor*)
            |
            (Editor, Editor*)
        ),
        Credit*,
        Date?
    }
Author =
    element author {
        element personname {TextSimple},
        element department {TextSimple | ShortLine+}?,
        element institution {TextSimple | ShortLine+}?,
        element email {text}?
    }
Editor =
    element editor {
        element personname {TextSimple},
        element department {TextSimple | ShortLine+}?,
        element institution {TextSimple | ShortLine+}?,
        element email {text}?
    }
Credit =
    element credit {
        element title {TextLong},
        Author+
    }
Date =
    element date {
        mixed {(Character | Generator)*}
    }
Abstract =
    element abstract {
        MetaDataTarget,
        BlockText+
    }
ColophonFront =
    element colophon {
        MetaDataTarget,
        element credit {
            element role {TextShort},
            element entity {TextLong}
        }*,
        element edition {text}?,
        element website {Url}?,
        element copyright {
            element year {TextShort},
            element holder {text},
            element minilicense {TextShort}?,
            element shortlicense {TextLong}?
        }?
    }
Biography =
```

```
    element biography {
        MetaDataTitleOptional,
        (BlockStatementNoCaption | ParagraphsNoNumber | Commentary)+
    }
Dedication =
    element dedication {
        MetaDataTitleOptional,
        (Paragraph|ParagraphLined)+
    }
Acknowledgement =
    element acknowledgement {
        MetaDataTitleOptional,
        (BlockStatementNoCaption | ParagraphsNoNumber | Commentary)+
    }
Preface =
    element preface {
        MetaDataTitleOptional,
        (
            (
                (BlockStatementNoCaption | ParagraphsNoNumber |
                 Commentary)+,
                Attribution*
            )
            |
            (
                (BlockStatementNoCaption | ParagraphsNoNumber |
                 Commentary)*,
                Contributors,
                (BlockStatementNoCaption | ParagraphsNoNumber |
                 Commentary)*
            )
        )
    }
```

49 Front matter (experimental)

A few simple tweaks to frontmatter elements.

We give an alternative definition of the ColophonFront to include footnotes in the shortlicense. This is the only change currently.

⟨75 Front matter (dev)⟩ ≡

```
ColophonFront |=
    element colophon {
        MetaDataTarget,
        element credit {
            element role {TextShort},
            element entity {TextLong}
        }*,
        element edition {text}?,
        element website {
            element name {TextShort},
            element address {text}
```

```
        }?,
        element copyright {
            element year {TextShort},
            element holder {text},
            element minilicense {TextShort}?,
            element shortlicense {
                TextLong &
                Footnote*
            }?
        }?
    }
```

50 Contributors

A single `contributors` element may be placed into a `preface` and is a list of `contributor`. It can be optionally preceded, or followed, by all the usual things that can go into any preface. An AuthorByline is a special instance of acknowledging a contributor on a division.

⟨76 Contributor⟩ ≡

```
Contributor =
    element contributor {
        MetaDataTarget,
        element personname {TextSimple},
        element department {TextSimple}?,
        element institution {TextSimple}?,
        element location {TextSimple}?,
        element email {text}?
    }
Contributors =
    element contributors {
        Contributor+
    }
AuthorByline =
    element author {(TextSimple|Xref)}
```

51 Back Matter

Articles and books have material at the end, structured as a sequence of `appendix`. A `solutions` division should be numbered and rendered as if it was one of the `appendix`, and so can mix-in in any order.

⟨77 Back matter⟩ ≡

```
ArticleBackMatter =
    element backmatter {
        MetaDataTitleOptional,
        (ArticleAppendix|Solutions)*,
        References?,
        IndexDivision?,
        ColophonBack?
```

```
    }
BookBackMatter =
    element backmatter {
        MetaDataTitleOptional,
        (BookAppendix|Solutions)*,
        References?,
        IndexDivision?,
        ColophonBack?
    }
ColophonBack =
    element colophon {
        MetaDataTarget,
        (BlockText | SideBySideNoCaption | SideBySideGroupNoCaption)+
    }
```

52 Document Information

The `docinfo` section is like a small database for the document.

⟨78 Document information⟩ ≡

```
DocInfo =
    element docinfo {
        XMLBase?,
        XMLLang?,
        Configuration+
    }
```

⟨Brand logo 79 [58]⟩
⟨Preambles 80 [59]⟩
⟨LaTeX macros 81 [59]⟩
⟨Cross-reference text style 82 [59]⟩
⟨Project initialism 83 [59]⟩
⟨Feedback link 84 [59]⟩
⟨Element renaming 85 [59]⟩
⟨Image archives 86 [60]⟩
⟨Author biographies 87 [60]⟩
⟨Numbering of part divisions 88 [60]⟩
A nice icon near the top of an electronic version is a nice touch, and can link back to a project landing page.

⟨79 Brand logo⟩ ≡

```
Configuration |=
    element brandlogo {
        attribute url {text}?,
        attribute source {text}
    }
```

We add some items which will become parts of preambles to support math in LaTeX syntax, `<latex-image>`, and `<asymptote>`. LaTeX packages, and their cousins, MathJax extensions, can be specified to support mathematics elements (`<m>` and friends). Images specified by LaTeX or Asymptote syntax sometimes need extra information in their preambles.

⟨80 Preambles⟩ ≡

```
Configuration |=
        element math-package {
            attribute latex-name {text},
            attribute mathjax-name {text}
        }*
Configuration |=
    element latex-image-preamble {text}
Configuration |=
    element asymptote-preamble {text}
```

Macros for LaTeX are shared across implementations. This should move under some general LaTeX section, the name is too vague.

⟨81 LaTeX macros⟩ ≡

```
Configuration |=
    element macros {text}
```

The style of text used in a cross-reference (the `xref` element) is contained in the source and uses the same per-item choices.

⟨82 Cross-reference text style⟩ ≡

```
Configuration |=
    element
        cross-references {
            attribute text { XrefTextStyle }
        }
```

An initialism is a useful short version of a book title.

⟨83 Project initialism⟩ ≡

```
Configuration |=
    element initialism {text}
```

Online versions can request feedback via a URL for some form. Maybe this should really be an `href` for consistency. There should be a device to provide text to go with the link.

⟨84 Feedback link⟩ ≡

```
Configuration |=
    element feedback {
        element url {text}
    }
```

Some elements can be renamed. This should be a rare event. Since the content of this element can (optionally) be specified in different languages, the `@xml:lang` attribute is appropriate.[1]

⟨85 Element renaming⟩ ≡

[1] https://www.w3.org/International/questions/qa-when-xmllang

```
Configuration |=
    element rename {
        attribute element {text},
        attribute xml:lang {text}?,
        text
    }
```

Image archives have some global specification. The `from` attribute gives a root for only working on a subtree of the document. The content is a comma-separated list of file extensions.

⟨86 Image archives⟩ ≡

```
Configuration |=
    element images {
        element archive {
            attribute from {text}?,
            text
        }+
    }
```

An author biography (or several) might be a paragraph or two each, or each one might be several pages. This style can be controlled.

⟨87 Author biographies⟩ ≡

```
Configuration |=
    element author-biographies {
        attribute length {"short" | "long"}
    }
```

Many aspects of numbering are configurable. These choices affect the numbers printed, and so are an author's decision, and hence run with the source.

⟨88 Numbering of part divisions⟩ ≡

```
Configuration |=
    element numbering {
        element division {
            attribute part {"decorative" | "structural"}
        }?
    }
```

53 Document Information (experimental)

We extend the `docinfo` to include new elements. We add each to the `Configuration` group.

A textbook can have a blurb (roughly what you would expect on the back of the book), and optionally a `@shelf` that tells Runestone how to categorize the book.

⟨89 Blurb⟩ ≡

```
Configuration |=
    element blurb {
        attribute shelf {text},
        text
    }
```

⟨90 Document ID⟩ ≡

```
Configuration |=
    element document-id {
        attribute edition {text}?,
        text
    }
```

Now we collect these to add to the dev schema.

⟨91 Experimental Document Info⟩ ≡
⟨Blurb 89 [60]⟩
⟨Document ID 90 [61]⟩

54 Hierarchical Structure

We collect all the specifications, roughly in a top-down order, so the generated schema files have a rational ordering to them, even if the order presented here is different.

⟨92 Hierarchical Structure⟩ ≡
Root of file: `pretext.rnc`

```
grammar {
```

⟨Start elements 1 [2]⟩
⟨Gross structure 2 [2]⟩
⟨Document types 3 [3]⟩
⟨Divisions 4 [4]⟩
⟨Front matter 74 [54]⟩
⟨Back matter 77 [57]⟩
⟨Paragraphs division 5 [6]⟩
⟨Specialized divisions 6 [7]⟩
⟨Blocks 32 [22]⟩
⟨Common components of blocks 33 [23]⟩
⟨Introductions, conclusions, headnotes 34 [24]⟩
⟨Objectives and outcomes 39 [26]⟩
⟨Block quotes 40 [27]⟩
⟨Verbatim displays 41 [27]⟩
⟨Lists 42 [28]⟩
⟨Definitions 43 [29]⟩
⟨Theorems, and similar 44 [29]⟩
⟨Axioms, and similar 46 [30]⟩
⟨Examples, and similar 66 [48]⟩
⟨Projects, and similar 47 [31]⟩
⟨Remarks, and similar 48 [31]⟩
⟨Computation, and similar 49 [32]⟩

⟨Asides, and similar 50 [32]⟩
⟨Assemblages 51 [32]⟩
⟨Captioned and titled displays 52 [33]⟩
⟨Side-by-side layouts 55 [35]⟩
⟨Images 56 [36]⟩
⟨Tabular display 53 [33]⟩
⟨Sage code 57 [38]⟩
⟨Interactives 58 [38]⟩
⟨Video and audio 60 [41]⟩
⟨Exercises 62 [42]⟩
⟨Poems 61 [42]⟩
⟨Bibliography 64 [46]⟩
⟨Glossary 65 [47]⟩
⟨Contributor 76 [57]⟩
⟨WeBWorK 67 [48]⟩
⟨Literate programming 68 [50]⟩
⟨Miscellaneous or uncertain 72 [54]⟩
⟨Frequent constructions 69 [50]⟩
⟨Paragraphs 10 [13]⟩
⟨Running text 9 [13]⟩
⟨Footnotes 36 [25]⟩
⟨Index entries 37 [25]⟩
⟨Cross-references 35 [24]⟩
⟨Mathematics 30 [20]⟩
⟨Verbatim text 22 [18]⟩
⟨Text groups 29 [20]⟩
⟨Text generators 19 [17]⟩
⟨Fill-in blank character 20 [18]⟩
⟨SI units 21 [18]⟩
⟨Characters 18 [17]⟩
⟨List generator 73 [54]⟩
⟨Document information 78 [58]⟩

```
}
```

55 Development Schema

Here we collect all fragments that are still experimental and put them in a rnc file that includes the stable schema.

⟨93 Development Schema⟩ ≡
Root of file: `pretext-dev.rnc`

```
namespace xhtml = "http://www.w3.org/1999/xhtml"
  grammar {

  include "pretext.rnc"
```

⟨Interactives 59 [39]⟩
⟨Front matter (dev) 75 [56]⟩
⟨Experimental Document Info 91 [61]⟩
⟨Proofs, and similar 45 [30]⟩
⟨Figure (experimental) 54 [35]⟩

⟨Worksheets (experimental) 8 [8]⟩
⟨Solutions (experimental) 7 [8]⟩
⟨References (experimental) 38 [26]⟩
⟨Mathematics (experimental) 31 [21]⟩
⟨Exercises (experimental) 63 [43]⟩

}

A Fragments

(Continued on next page)

(Continued on next page)