PreTeXt RELAX-NG Schema

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 developers. It is essentially a contract between the two. If an author's source validates against the schema, then a developer's code 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 occurences 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.

1 Gross Structure

A PreTeXt document is always a single `mathbook` 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. (2019-02-08: the root element may be `<pretext>`, and eventually `<mathbook>` will be deprecated.)

Fragment: `gross-structure`

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

2 Document Types

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

Fragment: `document-types`

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

3 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` or a `solution-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.

Fragment: `divisions`

```
Part =
    element part {
        MetaDataShortTitle, Chapter+
```

```
    }
Chapter =
    element chapter {
        MetaDataShortTitle,
        AuthorByline*,
        Objectives?,
        (
            (
                (BlockDivision | Paragraphs | Commentary)+,
                Exercises?,
                Solutions?,
                References?
            )
        |
            (
                IntroductionDivision?,
                Section+,
                Exercises*,
                Solutions?,
                References?,
                ConclusionDivision?
            )
        ),
        Outcomes?
    }
Section =
    element section {
        MetaDataShortTitle,
        AuthorByline*,
        Objectives?,
        (
            (
                (BlockDivision | Paragraphs | Commentary)+,
                Exercises?,
                Solutions?,
                References?
            )
        |
            (
                IntroductionDivision?,
                Subsection+,
                Exercises*,
                Solutions?,
                References?,
                ConclusionDivision?
            )
        ),
        Outcomes?
    }
Subsection =
    element subsection {
        MetaDataShortTitle,
        AuthorByline*,
        Objectives?,
```

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

```
            (
                IntroductionDivision?,
                Section,
                (
                    Section |
                    Exercises |
                    References
                )*,
                ConclusionDivision?
            )
        )
    }
IndexDivision =
    element index {
        MetaDataShortTitleOptional,
        IndexList
    }
```

4 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, so should not have any numbered items ever, so the "NoNumber" provision is implicit.

Fragment: `paragraphs`

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

5 Universal Divisions

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

Fragment: `universal`

```
Exercises =
    element exercises {
        MetaDataShortTitleOptional,
        IntroductionDivision?,
        (ToDo | Exercise | ExerciseGroup)+,
        ConclusionDivision?
    }
Solutions =
    element solutions {
        MetaDataShortTitleOptional,
        attribute inline {text}?,
        attribute divisional {text}?,
        attribute project {text}?,
        IntroductionDivision?,
        ConclusionDivision?
    }
References =
    element references {
        MetaDataShortTitleOptional,
        IntroductionDivision?,
        BibliographyItem+,
        ConclusionDivision?
    }
```

6 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 constructions. 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.

Fragment: `shorttext`

```
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, not 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.

Fragment: `paragraph`

```
TextParagraph = mixed { (
    Character |
    Generator |
    Verbatim |
    Group |
    WWVariable |
    MathInline |
    Music |
    Reference |
    CodeDisplay |
    MathDisplay |
```

```
    List |
    Footnote |
    Notation |
    Index)* }
Paragraph =
    element p {
        UniqueID?,
        PermanentID?,
        TextParagraph
    }
ParagraphLined =
    element p {
        UniqueID?,
        PermanentID?,
        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 are the best way to ensure you get what you want in all possible conversions. Note that left and right braces ("curly brackets") are previously defined with the LaTeX characters.

Fragment: `delimiter-character`

```
Character =
    element lsq {empty} |
    element rsq {empty} |
    element rq {empty} |
    element lq {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 seperate 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.

Fragment: `dash-character`

```
Character |=
    element nbsp {empty} |
    element ndash {empty} |
    element mdash {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, indivisable, and content-less, like all the other characters. `fillin` is also unusual due to its allowed use within mathematics.

Fragment: `fillin-character`

```
FillIn = element fillin {attribute characters {xsd:integer}?, empty}
Character |=
    FillIn
```

We define a few characters to help with simple expressions in arithmetic. The **solidus** is slightly different from the **slash** found on a keyboard and is used for fractions and ratios. These are for simple uses in regular text, not for actual mathematics, which is described later.

Fragment: `arithmetic-character`

```
Character |=
    element solidus {empty} |
    element times {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.

Fragment: `exotic-character`

```
Character |=
    element ellipsis {empty} |
    element midpoint {empty} |
    element swungdash {empty} |
    element permille {empty} |
    element pilcrow {empty} |
    element section-mark {empty} |
    element copyright {empty} |
    element registered {empty} |
    element trademark {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 shuld be supplied as part of styling, not as part of the source language.

Fragment: `icon-character`

```
Character |=
    element icon {
        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.

Fragment: `music-character`

```
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})*
        }*
    }
```

Fragment: `character`

```
<code: dash-character  10>
<code: fillin-character  10>
<code: delimiter-character  9>
<code: arithmetic-character  10>
<code: exotic-character  10>
<code: icon-character  10>
<code: music-character  11>
```

There are empty elements to generate certain items, like the date, or names of commonly referenced tools, such as PreTeXt itself. These include some com-

mon **Latin abbreviations**, for the purpose of handling the periods properly in conversions to LaTeX.

Fragment: `generator`

```
Generator =
    element today {empty} |
    element timeofday {empty} |
    element tex {empty} |
    element latex {empty} |
    element pretext {empty} |
    element webwork {empty} |
    element ad {empty} |
    element am {empty} |
    element bc {empty} |
    element circa {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 large class of similary 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-emptieness here might get a bit messy, so a Schematron warning could be a good alternative.

Fragment: `siunit`

```
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.

Fragment: `verbatim`

```
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 pronouncable as words (e.g.SCUBA).

Fragment: `abbreviation-group`

```
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.

Fragment: `delimiter-group`

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

Fragment: `highlight-group`

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

Fragment: `editing-group`

```
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.

Fragment: `xml-syntax-group`

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

An empty taxon will match either version.

Fragment: scientific-name-group

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

Fragment: group
<code: abbreviation-group 13>
<code: delimiter-group 13>
<code: highlight-group 13>
<code: editing-group 13>
<code: xml-syntax-group 13>
<code: scientific-name-group 14>

7 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.

Fragment: mathematics

```
MathInline =
    element m {
        mixed {(FillIn | WWVariable)*}
    }
MathRow =
    element mrow {
        MetaDataTarget,
        (
            attribute number {"yes" | "no"}
            |
            attribute tag {"star" | "dstar" | "tstar" |
                           "dagger" | "ddagger" | "tdagger" |
```

```
                        "hash" | "dhash" | "thash" |
                        "maltese" | "dmaltese" | "tmaltese" }
        )?,
        attribute break {"yes" | "no"}?,
        mixed {(Xref | FillIn | WWVariable)*}
    }
MathIntertext = element intertext {TextLong}
MathDisplay =
    element me {
        mixed {(FillIn | WWVariable)*}
    }
|
    element men {
        MetaDataTarget,
        mixed {(FillIn | 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)*
    }
```

8 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. A `todo` is text that is never printed in your output, unless you explicitly request that it be printed or inventoried. You can put it anywhere you might put a paragraph, as a peer, but do not use it within a paragraph (or other places where you are authoring text, it can gum up processing).

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 proofs, hints, 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. The `NoCaption` variant prohibits all numbered items, but allows placement of items like `image`. It also does not include `Sage`.

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.

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.

`demonstration` is slated for removal or an overhaul, and so is in the Bad Bank. Avoid using them for now.

Fragment: `block`

```
BlockText =
    Paragraph | BlockQuote | Preformatted | ToDo
BlockStatementNoCaption =
    BlockText | Aside |
    SideBySideNoCaption | SideBySideGroupNoCaption
BlockStatement =
    BlockText |
    Figure | Aside |
    SideBySide | SideBySideGroup | Sage
BlockDivision =
    BlockStatement |
    Remark | Computation | Theorem | Proof | Definition |
    Axiom | Example | Exercise | Project |
    Poem | Assemblage | ListGenerator |
    Demonstration
```

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 statment 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.

Fragment: `block-component`

```
Prelude =
    element prelude {BlockText+}
Interlude =
    element interlude {BlockText+}
Postlude =
    element postlude {BlockText+}
Statement =
    element statement {
        BlockStatement+
```

```
    }
Hint =
    element hint {
        MetaDataTitleOptional,
        BlockStatement+
    }
Answer =
    element answer {
        MetaDataTitleOptional,
        BlockStatement+
    }
Solution =
    element solution {
        MetaDataTitleOptional,
        BlockStatement+
    }
```

9 Introductions and Conclusions

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.

Fragment: `introduction-conclusion`

```
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+
    }
```

10 References

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

Fragment: `reference`

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

Fragment: `footnote`

```
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.

Fragment: `index`

```
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}
```

11 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 objectivbes, outcomes, and standards in a variety of ways.

Fragment: `objective-outcome`

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

12 Block Quotes

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

Fragment: `blockquote`

```
BlockQuote =
    element blockquote {
        MetaDataTitleOptional,
        Paragraph+,
        Attribution?
    }
Line =
    element line {TextShort}
```

13 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.

Fragment: `verbatimdisplay`

```
CodeLine =
    element cline {text}
CodeDisplay =
    element cd {
        attribute latexsep {text}?,
        (text | CodeLine+)
    }
Preformatted =
    element pre {
        text | CodeLine+
    }
Console =
    element console {
        PermanentID?,
        (
            element prompt {text}?,
            element input {text}?,
            element output {text}?
        )+
    }
Program =
    element program {
        PermanentID?,
        attribute language {text}?,
        attribute interactive {"pythontutor"}?,
        element input {text}
    }
```

14 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.

Fragment: `list`

```
List =
    element ol {
        PermanentID?,
        attribute cols {"2"|"3"|"4"|"5"|"6"}?,
        attribute label {text}?,
        element li {
            MetaDataTarget,
            (TextParagraph | BlockStatement+)
        }+
    }
|
    element ul {
        PermanentID?,
        attribute cols {"2"|"3"|"4"|"5"|"6"}?,
        attribute label {"disc" | "circle" | "square" | ""}?,
        element li {
            PermanentID?,
            (TextParagraph | BlockStatement+)
        }+
    }
|
    element dl {
        PermanentID?,
        attribute width {"narrow" | "medium" | "wide"}?,
        element li {
            MetaDataTitle,
            BlockStatement+
        }+
    }
```

15 Definitions

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

Fragment: `definition-like`

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

16 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 prrof 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`.

Fragment: `theorem-like`

```
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}
```

17 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).

Fragment: `axiom-like`

```
AxiomLike =
    MetaDataTitleCreatorOptional,
    Statement
```

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

18 Projects and Activities

A favorite of Inquiry-Based Learning textbooks. Numbered independently. Possibly structured with `task`.

Fragment: `project-like`

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

19 Remarks and Other Comments

Really simple blocks, they do not have much structure, and so are just runs of paragraphs.

Fragment: `remark-like`

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

20 Computations and Technology

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

Fragment: `computation-like`

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

21 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.

Fragment: `aside`

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

22 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.

Fragment: `assemblage`

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

23 Figures, Tables, Listings and Named Lists

These are containers that carry titles, captions, and numbers and need to be filled with other (indivisable) items. They have a mandatory `caption` (which can have no text, but will still produce a numbered caption), and may have a `title`, which could more appropriately be called a **heading**. These are also called **captioned items**.

Fragment: `table-figure`

```
Caption =
    element caption {TextLong}
Figure =
    element figure {
        MetaDataCaption,
        (
            Image |
            SideBySide |
            SideBySideGroup |
            Video |
            MuseScore
        )
    }
|
    element table {
        MetaDataCaption,
        Tabular
    }
|
    element listing {
        MetaDataCaption,
        (
            Program |
            Console
        )
    }
|
    element list {
        MetaDataCaption,
        IntroductionText?,
        List,
```

```
        ConclusionText?
    }
```

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

Fragment: `tabular`

```
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 |
            Line+ |
            Paragraph+
        )
    }
TableRow =
    element row {
        AlignmentHorizontal?,
        AlignmentVertical?,
        BorderBottom?,
        BorderLeft?,
        TableCell+
    }
TableColumn =
    element col {
        AlignmentHorizontal?,
        BorderTop?,
        BorderRight?,
        attribute width {text}?
    }
Tabular =
    element tabular {
        PermanentID?,
        AlignmentHorizontal?,
        AlignmentVertical?,
        BorderTop?,
        BorderBottom?,
```

```
        BorderLeft?,
        BorderRight?,
        TableColumn*,
        TableRow+
    }
```

24 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.

For WeBWorK the NoCaption variant allows `Tabular` and `ImageWW`. This may change if these two items are liberated from `<sidebyside>`. Presently, there should only be a single item inside a `sidebyside` inside a `webwork`, but we do not have a rule for that.

Fragment: `sidebyside`

```
Stack =
    element stack {
        (
            Tabular |
            Image |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List
        )+
    }
SidebySideAttributes =
    PermanentID?,
    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 |
            ImageWW |
            Video |
            Program |
            Console |
            Paragraph |
            Preformatted |
            List |
            Stack
        )+
    }
SideBySideGroup =
    element sbsgroup {
        SidebySideAttributes,
        SideBySide+
    }
SideBySideGroupNoCaption =
    element sbsgroup {
        SidebySideAttributes,
        SideBySideNoCaption+
    }
```

25 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.

Fragment: `image`

```
Image = ImageRaster | ImageCode
ImageRaster =
    element image {
        PermanentID?,
        attribute width {text}?,
        attribute archive {text}?,
        element description {TextShort}?,
        attribute source {text}
    }
ImageCode =
    element image {
        UniqueID?,
        PermanentID?,
        attribute width {text}?,
        attribute archive {text}?,
        element description {TextShort}?,
        (
            element latex-image {text} |
            element asymptote {text} |
            element sageplot {text}
        )
    }
ImageWW =
    element image {
        attribute pg-name {text}?,
        element description {(TextShort | WWVariable)*}?
    }
```

26 Sage Code

Sage is integral.

Fragment: `sage`

```
Sage = element sage {
    PermanentID?,
    attribute doctest {text}?,
    attribute tolerance {text}?,
    attribute language {text}?,
    attribute type {text}?,
    (element input {text}, element output {text}?)?
}
```

27 Interactive Elements

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

Fragment: `interactive`

```
MuseScore =
    element score {
        attribute musescoreuser {text},
```

```
        attribute musescore {text}
    }
```

28 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 `mbx` 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.

Fragment: `audiovideo`

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

29 Poetry

Poems!

Fragment: `poetry`

```
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
    }
```

30 Exercises

Inline, divisional, and WeBWorK. Exercises use ordered lists for parts of their statement (something that may need to be revisited).

Fragment: `exercise`

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

31 Bibliography

This is all stop-gap and will change radically. But it seems to work for now.

Fragment: bibliography

```
BibliographyItem =
    element biblio {
        MetaDataTarget,
        attribute type {"raw"}?,
        (TextLong |
        Ibid |
        BibTitle |
        BibYear |
        BibJournal |
        BibNumber |
        BibVolume |
        BibNote)*
    }
Ibid = element ibid {empty}
BibYear = element year {text}
BibJournal = element journal {text}
BibNumber = element number {text}
BibVolume = element volume {text}
BibTitle = element title {TextLong}
BibNote = element note {UniqueID?, Paragraph+}
```

32 Examples and Questions

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

Fragment: example-like

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

33 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?

Fragment: webwork

```
WebWork = (WebWorkAuthored | WebWorkSource)
WebWorkSource =
    element webwork {
        attribute source {text}?,
        attribute seed {xsd:integer}?
    }
WebWorkAuthored =
    element webwork {
        MetaDataTitleOptional,
        attribute seed {xsd:integer}?,
        WWMacros?,
        WWSetup?,
        (
            (StatementExerciseWW, HintWW?, SolutionWW?)
        |
            element stage {
                Title?,
                StatementExerciseWW,
                HintWW?,
                SolutionWW?
            }+
        )
    }
StatementExerciseWW =
    element statement {
        (
            Paragraph |
            WWInstruction |
            Preformatted |
            SideBySideNoCaption
        )+
    }
WWMacros =
    element pg-macros {
        element macro-file {text}+
    }
WWSetup =
    element setup {
        element pg-code {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"|"none"}?) |
```

```
        (attribute form {"essay"},
        attribute width {text}?)
    }
WWInstruction =
    element instruction {TextShort}
HintWW =
    element hint {
        (
            Paragraph |
            Preformatted |
            SideBySideNoCaption
        )+
    }
SolutionWW =
    element solution {
        (
            Paragraph |
            Preformatted |
            SideBySideNoCaption
        )+
    }
```

34 Frequently Used

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

Fragment: `frequently-used`
<code: statements 34>
<code: attribution 34>
<code: metadata 35>

Fragment: `statements`
Used on the end of prefaces to "sign" them, and on block quotes.

Fragment: `attribution`

```
Attribution =
    element attribution {
        (TextShort | Line+)
    }
```

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 thse manually as an author. (You do want to manually author `@xml:id`.)
- The `xinlude` mechanism may pass language tgs 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.
- These are not unordered specifications since they contain several attributes, and we enforce a `title`, `subtitle`, `<shorttitle>`, `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>`, and allows an optional `<shorttitle>`.
- `MetaDataSubtitle` implicitly has a required `<title>`, and allows optional `<subtitle>` and `<shorttitle>`.
- `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.

Fragment: `metadata`

```
UniqueID =
    attribute xml:id {text}
PermanentID =
    attribute permid {text}
Title =
    element title {TextLong}
Subtitle =
    element subtitle {TextLong}
ShortTitle =
    element shorttitle {TextShort}
Creator =
    element creator {TextShort}
XMLBase = attribute xml:base {text}
XMLLang = attribute xml:lang {text}
MetaDataTarget =
    UniqueID?,
    PermanentID?,
    Index*
MetaDataTitle =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    Title,
    Index*
MetaDataShortTitle =
    UniqueID?,
    PermanentID?,
    XMLBase?,
```

```
    XMLLang?,
    Title,
    ShortTitle?,
    Index*
MetaDataSubtitle =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    Title,
    Subtitle?,
    ShortTitle?,
    Index*
MetaDataTitleOptional =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    Title?,
    Index*
MetaDataShortTitleOptional =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    (Title, ShortTitle?)?,
    Index*
MetaDataTitleCreatorOptional =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    Title?,
    Creator?,
    Index*
MetaDataCaption =
    UniqueID?,
    PermanentID?,
    XMLBase?,
    XMLLang?,
    Title?,
    Caption,
    Index*
```

35 Miscellaneous

Provisional items, with uncertain futures.

Fragment: `miscellaneous`

```
ToDo = element todo {text}
```

36 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.

Fragment: `listgenerator`

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

37 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, `titlepagfe` does not allow an `@xml:id` attribute.

Fragment: `frontmatter`

```
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 | Line+}?,
        element institution {TextSimple | Line+}?,
        element email {text}?
    }
Editor =
    element editor {
        element personname {TextSimple},
        element department {TextSimple | Line+}?,
        element institution {TextSimple | Line+}?,
        element email {text}?
    }
Credit =
    element credit {
        element title {TextLong},
        Author+
    }
Date =
    element date {
        mixed {(Character | Generator)*}
    }
Abstract =
    element abstract {
        MetaDataTitleOptional,
        BlockText+
    }
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}?
        }?
    }
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)*
            )
        )
    }
```

38 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.

Fragment: `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)}
```

39 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.

Fragment: `backmatter`

```
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)+
    }
```

40 Document Information

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

Fragment: `docinfo`

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

```
<code: addins  42>
<code: numbering  43>
```
A nice icon near the top of an electronic version is a nice touch, and can link back to a project landing page.

Fragment: `brandlogo`

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

We add some items to the preamble for LaTeX. A `package` needs to have an identical implementation, and be of the same name, as a version that exists for MathJax.

Fragment: `latexpreamble`

```
Configuration |=
    element latex-preamble {
        element package {text}*
    }
```

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

Fragment: `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.

Fragment: `cross-references`

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

Images specified by LaTeX syntax sometimes need extra information in their preambles.

Fragment: `lateximagepreamble`

```
Configuration |=
    element latex-image-preamble {text}
```

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

Fragment: `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.

Fragment: `feedback`

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

Some elements can be renamed. This should be a rare event.

Fragment: `rename`

```
Configuration |=
    element rename {
        attribute element {text},
        attribute 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.

Fragment: `imagearchive`

```
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 severalpages. This style can be controlled.

Fragment: `authorbiographylength`

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

Analytics can be built into electronic versions by providing identifying account information.

Fragment: `addins`

```
Configuration |=
    element search {
        element google {
            element cx {text}
        }
    }
|
    element analytics {
        (
            element google {
                element tracking {text}
            }
        |
            element statcounter {
                element project {text},
                element security {text}
            }
        )+
    }
```

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.

Fragment: `numbering`

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

41 Hierachical 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.

Begin File: `pretext.rnc`

```
grammar {

<code: gross-structure  2>
<code: document-types  2>
<code: divisions  3>
<code: frontmatter  37>
<code: backmatter  40>
<code: paragraphs  6>
<code: universal  7>
<code: block  16>
<code: block-component  16>
<code: introduction-conclusion  17>
<code: objective-outcome  19>
<code: blockquote  20>
```

```
<code: verbatimdisplay  20>
<code: list  21>
<code: definition-like  21>
<code: theorem-like  22>
<code: axiom-like  22>
<code: example-like  32>
<code: project-like  23>
<code: remark-like  23>
<code: computation-like  24>
<code: aside  24>
<code: assemblage  25>
<code: table-figure  25>
<code: sidebyside  27>
<code: image  28>
<code: tabular  26>
<code: sage  29>
<code: interactive  29>
<code: audiovideo  30>
<code: exercise  31>
<code: poetry  30>
<code: bibliography  32>
<code: contributor  39>
<code: webwork  32>
<code: miscellaneous  36>
<code: frequently-used  34>
<code: paragraph  8>
<code: shorttext  8>
<code: footnote  18>
<code: index  19>
<code: reference  18>
<code: mathematics  14>
<code: verbatim  12>
<code: group  14>
<code: generator  12>
<code: siunit  12>
<code: character  11>
<code: listgenerator  37>
<code: bad-bank  44>
<code: docinfo  40>


}
```

42 Bad Bank

Fragment: bad-bank

```
Demonstration = element demonstration {
    Title,
    Paragraph,
    Sage
}
```

Index