LogicalStructure

Logical Structure

Some page-oriented output formats (like PDF, Mars and XPS) provide facilities to record the logical structure of a document. For example, it allows the formatter to express which parts of a page belong to a page header/footer and which to the main content flow. This is especially important for copy/paste operations or for text-to-speech applications. PDF 1.4 lists various reasons why a formatter should support this:

- Simple extraction of text and graphics for pasting into other applications
- Automatic reflow of text and associated graphics to fit a page of a different size than was assumed for the original layout
- Processing text for such purposes as searching, indexing, and spell-checking
- Conversion to other common file formats (such as HTML, XML, and RTF) with document structure and basic styling information preserved
- Making content accessible to the visually impaired

In terms of importance, copy/paste operations and accessibility are probably the most important aspects.

The logical structure facilities try to provide a general mechanism to describe the structure of a document to best best extent possible. Information loss during the formatting process probably cannot be avoided. For example, it will be next to impossible to map the full feature set of XSL-FO into Tagged PDF. The idea is to make a best effort to enable document conversions while losing as little information as possible.

Related Topics

In the context of logical structure and accessibility, the specification of the natural language and alternate descriptions (for example, textual descriptions of pictures) should be mentioned.

Logical Structure in PDF

PDF 1.4 has multiple chapters dedicated to this topic:

- 9.5 Marked Content
- 9.6 Logical Structure (depends on Marked Content)
- 9.7 Tagged PDF (depends on Logical Structure)

PDF/A defines the two conformance levels A and B. FOP currently supports Level B. To achieve Level A, FOP needs "full" support for Tagged PDF and natural language specification. See also PDF/A-1 conformance notes.

Initial Analysis

(based on FOP 0.95)

Support for logical structure is a major task which will have a major impact on the design of the area tree, intermediate format and renderers. The area tree currently just has a content tree per page with all visible marks plus metadata and similar features. The logical structure tree of a document will indicate that, for example, a particular paragraph is split over two pages:

Original document:

```
<fo:block font-weight="bold" role="heading1">I'm a heading</fo:block>
<fo:block role="p">This is some text that is splittable over two pages.</fo:block>
```

Area tree (figurative):

```
page-sequence
  page
    block font-weight="bold": I'm a heading
    block: This is some text that is split-
  page
    block: table over two pages.
```

Logical structure:

```
heading1 -> H1: I'm a heading (-> Page 1)
p -> P: This is some text that is splittable over two pages. (-> Pages 1 and 2)
```

Of course, this is just a very simple example (for more details, please see the PDF 1.4 spec). But it should show what the challenge is. Basically, a new tree closely following the original structure of the FO document is built and augmented with pointers into the (visible) content on the pages (coming from the layout process). Our area tree obviously cannot do that, yet. It is not interested where the individual elements on a page come from. It remains to be seen if the area tree itself could be augmented to help build the logical structure tree inside the renderer. But this smells like reconstruction of a structure we already have earlier in the process. One possibility is simply to preserve the FO tree for the rendering process. That way the renderer would have full access to all aspects for the original document. Of course, there are memory consumption implications there which means that tagged PDF should be an optionally enabled feature. As already noted in AreaTreeIntermediateXml NewDesign, support for logical structure also has implications on the new intermediate format to be designed.

But wait, what about the distinction of paragraphs and headings? FO doesn't distinguish between them. The example above uses the "role" attribute (currently not used in FOP) to indicate what kind of paragraph we're dealing with. That can be a URI (possibly indicating an RDF resource) or a string. FOP would need to know how to interpret these. These values could be different from document to document. Some kind of configuration will need to be available so the "role" values can be mapped to structure types in PDF or whatever is used in Mars or XPS. Let's also keep in mind that not all output formats will use the exact same data model to map this information. The fact that we have to rely on the "role" attribute for additional information means that FO documents lacking the role hints will produce a less-than-perfect final document. It's up to the user to supply this information.

Reading through the PDF specification, some other details can be identified that have to be looked into when implementing support for logical structure:

- Hyphen glyphs added by the layout engine have to be marked as such (0x002D != 0x00AD). Split words have to be exported combined when doing copy/paste. (PDF 1.4, 9.7.2, page 617)
- Abbreviations and synonyms: These can be expanded in PDF (for text-to-speech applications) but FO doesn't provide a mechanism. This might have to be solved through an extension on an fo:inline or fo:wrapper.
- The natural reading language has to be specifiable not only on the document level but also in line.
- Handling footnotes might be particularly tricky because pointers outside the normal flow have to be done.
- Artifacts (footnote rules, cut marks etc.) need to be marked as such.

Implementation Levels

There's no hard line what it means to support tagged PDF. The PDF/A-1 specification says: "Writers of conforming files should attempt to capture a document's logical structure hierarchy to the finest granularity possible, ..." That means preserving tables, lists, links etc.

Leaving PDF/A-1a support aside, this could be reduced to just preserving the text as such and the natural reading order. But that would technically not be "Tagged PDF" anymore. That's just the "9.6 Logical Structure" feature in PDF 1.4. It could probably 💡 be implemented without major changes in the area tree.

The structures of tagged PDF are extensible, meaning that it is possible to map a large quantity of information from XSL-FO. A 100% round-trip is probably not realistic, though. But from this it is possible to split two additional implementation levels. Either only some reasonable set of information is mapped or otherwise the largest possible set is implemented (trying to capture most of XSL-FO's capabilities). It can be worthwhile to look at the "standard attribute owners" that Adobe defined. They define a vocabulary of attributes to map certain feature sets.

So, let's try to assign identifiers to the various implementation levels:

- **Level 1:** Minimal logical structure (only helps for copy/paste, minimal accessibility support (together with language indicators) and indexing applications, doesn't enable PDF/A-1a conformance)
- **Level 2:** Minimal tagged PDF (enables PDF/A-1a, limited conversion fidelity)
- **Level 3:** Extensively tagged PDF (adds enhanced conversion fidelity, limited round-trips)

At this time it is unknown if Level 1 would meet the requirements established by various laws in some countries (most notably the USA).

Deciding what to implement

It is very important to know what exactly the goals are if the right approach is to be chosen (if one wants to minimize the investment). If the goal is accessibility, the whole thing goes in a somewhat different direction than when document conversions are the ultimate goal.

The first implementation was done for Accessibility - see PDF_Accessibility

Miscellaneous

An idea from an interested party has been brought up that using tagged PDF it might be possible to reconstruct the original XML file (not XSL-FO!) that the PDF was constructed from. However, it would make much more external information necessary to try to preserve the structure of the original document which can be considerably different from the XSL-FO document generated from it. This sounds rather unrealistic at this time. It is better to find a way to provide access to the original XML document for the case where access to the original data is required. Round-trips in this context are extremely difficult.

Tasks

DRAFT! BRAIN-STORMING!

Level 1:

- Add inline-level language indicators.
- Generate IDs for all formatting objects which don't have an ID so the contents split over various areas can be collected.
- Add is-first and is-last trait on areas so the full original block can be reconstructed.
- Add infrastructure in the PDF library so the structure information can be generated.
- Verify that soft and hard hyphens are distinguished in the area tree and the renderers.

- Add code in the renderers that can assemble the structure tree from the area tree. Note: This will only be an approximation. (This is a throw-away part when Level 2 is implemented!)

Level 2:

- Everything from Level 2 except the structure tree builder/guesser.
- Decide on access to FO tree in renderers or build-up of a separate customized data structure.
    - From there, drill further down, keeping in mind that the intermediate format has to support all that. The FO tree could be re-serialized as an annotated XSL-FO document. Note: a redesigned intermediate format will make this task a lot different!
- Add support for interpreting the "role" attribute. Possibly define an external XML with additional information.

Level 3:

- Just some work supporting additional attributes. Probably done on-demand. No architectural changes expected compared to level 2.