Towards a Universally Editable Portable Document Format

Tamir Hassan
Round-Trip PDF Solutions
Vienna, Austria
tamir@roundtrippdf.com

ABSTRACT

PDF is the established format for the exchange of final-form print-oriented documents on the Web, and for a good reason: it is the only format that guarantees the preservation of layout across different platforms, systems and viewing devices. Its main disadvantage, however, is that a document, once converted to PDF, is very difficult to edit. As of today (2018), there is still no universal format for the exchange of editable formatted text documents on the Web; users can only exchange the application's source files, which do not benefit from the robustness and portability of PDF.

This position paper describes how we can engineer such an editable format based on some of the principles of PDF. We begin by analysing the current status quo, and provide a summary of current approaches for editing existing PDFs, other relevant document formats, and ways to embed the document's structure into the PDF itself. We then ask ourselves what it really means for a formatted document to be editable, and discuss the related problem of enabling WYSIWYG direct manipulation even in cases where layout is usually computed or optimized using offline or batch methods (as is common with long-form documents).

After defining our goals, we propose a framework for creating such editable portable documents and present a prototype tool that demonstrates our initial steps and serves as a proof of concept. We conclude by providing a roadmap for future work.

CCS CONCEPTS

• **Applied computing** → **Document preparation**; • **Human-centered computing** → *Interaction techniques*;

KEYWORDS

Document formats, PDF, layout optimization, interactive editing

ACM Reference Format:
Tamir Hassan. 2018. Towards a Universally Editable Portable Document Format. In *DocEng '18: ACM Symposium on Document Engineering 2018, August 28–31, 2018, Halifax, NS, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209280.3229083

DocEng '18, August 28–31, 2018, Halifax, NS, Canada

ACM 978-1-4503-5769-2/18/08…$15.00
https://doi.org/10.1145/3209280.3229083

1 INTRODUCTION

PDFs are unique in their ability to preserve the visual presentation of a document across different platforms, systems and viewing devices. This is thanks to PDF's roots in PostScript, a printer language on which it is based. The widespread availability of print functions in software at the time (PDF was introduced in 1993) meant that PDF was the "lowest common denominator" among documents from different sources; any document that could also be printed could be effortlessly converted to PDF.

However, this is also PDF's main drawback, and this remains true to this day: the resulting document is essentially a vector graphic, and is difficult to edit, at least in such a meaningful way that would have been possible by the authoring application, such as a word processor. Users wishing to exchange editable documents are therefore still forced to use their native application's format, and do not benefit from the robustness, openness and portability that people have now come to expect from a modern document format.

This paper proposes a framework to tackle this problem, working towards a open, *universal* format that can be edited by a variety of applications conforming to a commonly agreed specification. The following subsection begins by introducing the current state of the art: previous approaches to editing PDFs, relevant developments in the PDF format, and discusses the challenges of a universally editable format with reference to current editable document formats. We conclude the introduction by defining the problem more precisely, stating the goals that this new format should attain.

Section 2 presents the proposed framework and the underlying technologies. Section 3 presents our initial experiments using a publishing tool developed as a proof-of-concept and Section 4 discusses further issues that need to be addressed. Finally, Section 5 provides a roadmap for the future directions of this work.

1.1 Editability of current PDFs

A born-digital PDF that has been generated directly from the source application, such as a word processor or DTP package, is essentially a vector graphic, lacking (sufficient) logical structure information about its content. A number of products exist in the marketplace for editing such PDF files, but they are usually limited to simple "touch up" operations, such as adding or removing words and graphic objects; some more advanced products, such as FoxIt PhantomPDF, also support limited reflowing of the text by attempting to detect its structure, but this does not always work predictably.

Currently, editing a previously generated PDF is generally seen as a last resort, and is only performed if the source file or application is unavailable. In contrast, when opening the source file in its native application, such as a word processor, the application's internal model stores the content in a more structured way, enabling text to reflow when editing, so that the layout automatically adjusts

and items to move to the next page if necessary. This structure is (mostly) lost when the document is converted to PDF.

If the document's logical structure is known in sufficient detail, this internal model can be rebuilt, enabling the document to be edited in a similar way. However, each application has its own internal model, designed specifically for the way that it works; the way the users interact with it and the types of document that it generates. There are many features that are common across a wide range of mostly-text documents from different sources, such as bulleted and numbered lists, tables etc., and one of the main challenges in designing a universal format is to determine which core features will be present in the generic document model.

1.2 Other notable developments in PDF

Since its initial release in 1993, there have been several developments to PDF to make it more intelligent and versatile, but these have all stopped short of the goal of making the files editable. **Tagged PDF** was introduced in version 1.6 of the specification (2004), where extra metadata tags can be added to the content stream to denote the logical structure of the text. These tags make it possible to perform a basic reflow for small screen devices, remove linebreaks when text is copied out of the PDF and are a core component of accessible PDFs for users of assistive technology such as screenreaders. In 2017, the **responsive PDF** project was publicly announced, which will extend the tagging concept to make it possible to automatically derive a full HTML representation of the content, which can be presented in a responsive way using web technologies.

PDF has always been a container format, allowing embedding of data in other formats, which are normally ignored by most viewing applications. Probably the best examples of such **hybrid PDFs** are those created by Open- and LibreOffice, which simply include the document's ODF source in one of the streams. Unfortunately, this feature was not very well advertised (and not activated by default); and without a suitable file extension such as, for example, `.odf.pdf`, nobody would be aware that they were actually viewing an editable, hybrid document. Technically, the hybrid ODF/PDF document is equivalent to having two separate files, as the PDF representation is still totally independent from the ODF source. Thus, if the file is opened for editing on another system, the robustness guaranteed by PDF is lost.

1.3 Other editable formats

Applications commonly used to author complete documents include word processors, desktop publishing packages, drawing programs and even presentation software. They are all largely based on a page or *canvas* metaphor, which allows a certain amount of direct, *explicit* manipulation of the layout. In fact, the underlying document model in all these programs is very similar; only the tools that are offered to the user have a different emphasis (which has led to the current situation where users often have to deal with several different user interfaces for performing the same task).

Text content, on the other hand, is *implicitly* laid out: the content, combined with the typesetting parameters, ultimately determine its form and how much vertical space it will take up. Thus, editing the text can lead to more substantial changes in layout if larger objects are placed in sequence to follow the text, or are somehow anchored to parts of the text instead of to the page.

There are also a number of non-WYSIWYG typesetting systems for formatting content, where the content's structure is typically explicitly defined, and its presentation is *implicitly* determined by its structure and other typesetting commands. LaTeX is probably the best known system in academia, but there are also many formatting systems for specific cases that take structured content as input, usually in XML. But the best example of this approach is actually the Web itself; the physical presentation of websites is still dependent on various external factors such as the size of the browser window; basic HTML does not lend itself well to absolute positioning of objects. And despite continuous development by standards bodies, browser manufacturers and other stakeholders, it is still very difficult to guarantee that a document will display identically across different browsers.

1.4 Problem definition

The previous subsections have summarized the variety of formats for text-oriented documents and their development over the past years. It is important that a next-generation format for editable document exchange supports these new developments, in order to serve its users well for the foreseeable future. We propose that such a format should:

- be universal: an open specification enables creation and editing via a variety of applications, just like PDF does today;
- be portable: the document should be self-contained and be engineered in such a way that missing data (e.g. fonts) do not affect layout;
- support editing via a graphical WYSIWYG interface as well as structured markup;
- enable layout to be defined by both explicit, direct manipulation operations and implicit, batch or optimization procedures, thus combining the benefits of both approaches (see Section 2.2);
- store the content in a well-structured form, enabling the markup to be easily extended by semantic annotation if required.

It is worth noting that this paper concentrates almost exclusively on text, as text is the most common type of content across various document formats, and hence lends itself well to being handled in a universal way. Other content types are discussed in Section 4.1.

2 KEY TECHNOLOGIES

2.1 Standardized layout primitives

There are two main ways that layouts are described in WYSIWYG document authoring applications. The first method, used by word processors, assumes a content frame taking up the whole page (excluding margins), which is filled in sequence from top to bottom. With this type of layout, adding or removing content always causes the layout to be automatically updated based on simple, predictable rules; if content is inserted, the remaining content moves down. If the page is full, a new page is created.

In desktop publishing software, text is usually placed into frames, which are manually defined and can be freely positioned on the page. In contrast to word processing software, the layout usually requires manual adjustment after content is added. Such a model is more flexible and allows arbitrary layouts to be designed, but needs to be adjusted manually when content is added or removed.

The current proof-of-concept enables the author to choose to use either of these two models for laying out the page. In fact, the first model is extended to allow for any number of nested X-Y divisions. This way, multi-column layouts and even more complex pages (such as this paper) can be created, while still benefiting from automatic layout adjustment when content is edited.

2.2 Two-step layout optimization

Non-WYSIWYG publishing systems, such as LaTeX, have remained popular for long-form content such as scientific articles. This is due to the system automatically taking care of the layout (at least in theory), leaving the author to concentrate on writing the content. At compile time, a number of optimization routines are carried out to perform pagination, place figures in appropriate places, update the table of contents and references, etc. In contrast to the WYSIWYG approaches described above, due to the global nature of the layout optimization being used, even carrying out minor edits to the text can often lead to unexpected, larger changes occurring, such as a figure jumping to another page.

This makes true WYSIWYG editing very difficult, not only due to these unexpected changes, but also due to the computational power required to constantly recompute the optimum layout. Furthermore, the lack of direct manipulation of the objects also makes it difficult to make any adjustments to the layout after it has been computed. These disadvantages are often the sources of much frustration for users of such systems, yet there appears to be surprisingly little research being done to solve this problem.

We propose a two-step layout optimization strategy to make our framework amenable to long-form documents, allowing them to be edited in a WYSIWYG fashion. When the content is updated, the layout is adjusted according to the principles in Section 2.1. Local optimizations, such as line breaks and balanced columns, are also carried out, as these are unlikely to distract the author or take too long to process. Larger optimizations, such as the repositioning of figures, are performed at a later stage at the author's convenience. As soon as the relevant constraints are no longer satisfied (e.g. the figure is no longer placed on or after its first calling position in the text), the system should warn the user in an appropriate way (e.g. by using a red highlight or other visual marking in the case of a GUI) and advise that a recomputation should be carried out when the user is ready.

As this recomputation is completely separate from the interactive editing process, it could also be carried out by an external program or web service if, for example, the user has particular requirements for auto-layouting that are not met by the editor. For example, [Mit16, Mit17] present significant improvements to the greedy optimization algorithms for pagination and float placement that are included in current LaTeX distributions. Alternatively, for some use cases, it may be preferable to use local optimization followed by manual adjustment instead of global optimization.

Given that the underlying source of the document is still text, it remains just as easy to edit the text (and even the layout) using a text editor, making this format inherently suitable for collaborative low-bandwidth, offline-first applications, allowing the use of standardized versioning systems such as Subversion and Git.

2.3 Robust micro-typography

Text typesetting is variable by nature, and the precise placement of characters may vary slightly according to the algorithm that is being used to set the text. Some of these variations may be unintentional (i.e. due to bugs); for example, kerning pairs in a font might be ignored or some rounding errors might occur in the calculation of glyph widths; other more advanced algorithms might purposefully carry out optimizations to counteract certain optical illusions. Such minor shifts are not normally noticeable to the untrained eye, but can accumulate, causing changes in line breaking and other unexpected layout changes in the document.

In order to maintain the robustness of PDF, we need to continue storing the precise positions of each glyph. To facilitate editability, the original content needs to be stored, together with all relevant settings, such as the font information, hyphenation, any manual spacing adjustments, etc. It is then up to each individual application to respect these settings and enable the text to be edited in a conservative way, preserving the line breaks for parts of the text that are unchanged. In previous work [HH15] we have shown that there is an inherent flexibility of 5–10% when setting text.

Of course, for an editable document, it is virtually a prerequisite that all fonts are fully embedded (i.e. no subsetting). PDF includes mechanisms for both partial and "full" embedding of fonts, although subset fonts are usually used in final-form documents, as they lead to smaller files and have fewer licensing issues. However, even fully embedded fonts are usually missing kerning information, which needs to be embedded separately, in order to ensure high quality typography when editing the document. A discussion on the licensing issues of font embedding for editability is presented in Section 4.2.

3 EXPERIMENTS AND PROOF OF CONCEPT

We have developed an open-source typesetting tool, Pint, to test this concept, which allows the creation and modification of editable formatted documents from the command line from XML input. Pint ("Pint Is Not TeX") currently supports the following features:

- X-Y and frame-based layouts
- support for multiple columns and column balancing
- Knuth-Plass line-break optimization
- stylesheet-based control of heading and paragraph styles
- addition of figure floats

Pint is written in Java and uses the PDFBox library for PDF generation. The roadmap for future development includes support for better float management (including resizing of figures), tables and citation/reference management.

A document can be authored by creating three XML files: the marked-up content, the abstract layout description and a stylesheet. Optionally, additional files, such as fonts and images, may be included. When Pint is first run on these source files, as well as generating the PDF, it generates an additional XML *physical layout file*, which stores the positions of all the layout frames, enabling non-destructive editing.

For the purpose of our proof-of-concept, edits to the XML files are seen as being equivalent to the edits that would be carried out using a WYSIWYG interface. This includes making changes to the content, as well as changing the coordinates of objects positioned using the frame model, etc. (which corresponds to direct

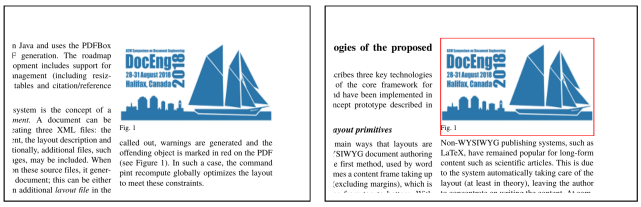

Figure 1: After adding text, the figure's placement no longer satisfies the given constraints, generating a warning marking (right)

manipulation). After each modification, the `pint update` command can be run to carry out all automatic updates that would occur in the layout structure and local optimizations. If any defined constraints are not met, for example the content overruns a frame or a figure float is placed before the page on which it is called out, warnings are generated and the offending object is marked with a red outline on the PDF (see Figure 1). In such a case, the command `pint recompute` globally optimizes the layout to meet these constraints.

These XML files enable offline, low-bandwidth editing by any text or XML editor and can easily be integrated into a versioning system such as Subversion or Git. For exchanging the document with other users, a self-contained hybrid PDF with these files embedded (e.g. as PDF/A-3) is a much better option, and this feature is currently planned for the next release of Pint.

The camera-ready version of this paper was formatted with Pint. The astute reader may notice the lack of typographical nuances common to other papers in these Proceedings, such as kerning and ligatures, which have not been implemented yet. The source and resulting layout files have been embedded in the submitted PDF using Adobe Acrobat, and it remains to be seen whether they will survive the transition to the ACM Digital Library.

4 DISCUSSION

PDF has always been very similar to a vector format, and it has long been possible to import PDFs into drawing applications such as Adobe Illustrator to edit them that way. The framework presented in this paper has shown how this editability can be extended to the text content of a PDF, making the sorts of edit operations possible that would normally require a word processor, typesetting system, or similar software. Compared to simply distributing the source files (or hybrid PDFs with embedded source files), our approach guarantees openness and portability; the layout is always preserved and the document can be edited in a non-destructive way. The framework also addresses the disparities between layout editing via direct manipulation and ex post facto or batch optimization, enabling the benefits of both approaches to be combined.

4.1 Other types of content

Print-oriented documents are not limited to simply drawings, bitmaps and text; they often include other types of generated content, such as charts and diagrams, which are generated by software using completely different internal models. In order to remain editable, the source information and/or data needs to be embedded in the file. A system similar to OLE from the 1990s would enable such content to be edited if appropriate software is available.

It is feasible to extend the logical markup required for editability to include semantic information about the document. In a related project, we are working on embedding such information in scientific articles corresponding to the JATS-XML standard. The ability to embed arbitrary types of content is also likely to be of particular interest to authors of data-driven documents in certain fields.

4.2 Font licensing issues

Fonts, or at least their digital implementations, have been traditionally subject to copyright, and their complete inclusion within a PDF file is equivalent to distributing the font files themselves, enabling their reuse for completely different purposes outside the scope for which they were licensed. In order to make it possible to use the fonts for creating PDFs, most foundries only allow their fonts to be embedded using subsetting, i.e. including only the characters that actually occur in the document.

Subset fonts, however, make editing much more difficult as the font must be substituted with an alternative if unavailable characters are used. In order to ensure that the replacement characters fit their original space, PDF provides for the possibility to synthesize fonts to match the original font's metrics (i.e. character widths, etc.). However, in such cases, metrics are not even available for characters that have not been used.

In order to ensure proper editability of text whose font has been (temporarily) substituted, metric information must be embedded for all characters of the font, and all kerning pairs, etc., even if they have not been used. It is not clear whether embedding this information from a commercial font might violate its copyright (as OpenType fonts are much more like programs in this respect, it probably would, at least for such fonts), but it is conceivable that font foundries would be amenable to permitting embedding of their fonts in such a way, as this information is still insufficient to recreate the font for unlicensed reuse.

Thanks to initiatives such as Google Fonts, there are now a much larger variety of free fonts available, which are also of sufficient quality and can be used for a wide variety of documents. Many of the remaining cases are likely to be corporate fonts, and by embedding their complete metric information, such documents could be edited by collaborators outside of the organization using a substitute font, whilst ensuring that the content will fit correctly when the correct font is reapplied for printing.

5 THE ROAD AHEAD

The introduction of an editable, portable document format has the potential to improve the way we communicate and share documents in a wide variety of fields, from scholarly publishing to finance to graphic design, leading to significant increases in efficiency. In order to communicate the ideas and encourage further discussion about the topic, a website is now available at `https://editablepdf.org`, which contains links to the Pint software, sample files, and will soon contain all current file format definitions. It is hoped that this initiative will encourage other stakeholders and members of the document engineering community to participate in shaping the future.

REFERENCES

[HH15] Tamir Hassan and Andrew Hunter. Knuth-Plass revisited: Flexible line-breaking for automatic document layout. In *DocEng 2015: Proceedings of the 15th ACM Symposium on Document Engineering*, 2015.

[Mit16] Frank Mittelbach. A general framework for globally optimized pagination. In *DocEng 2016: Proceedings of the 16th ACM Symposium on Document Engineering*, 2016.

[Mit17] Frank Mittelbach. Effective floating strategies. In *DocEng 2017: Proceedings of the 17th ACM Symposium on Document Engineering*, 2017.