Concepts

callas pdfChip – the Foundation

For various reasons development at callas software were looking for technology that could create PDF files on the fly but did not require programming to express exactly what type of PDF was to be created (there are a number of mature, high quality libraries in the market that can already do that). An obvious approach was to use a language that is good at expressing two-dimensional static visual content. Inventing our own language was not an option (there are too many already), and some of the existing languages were not to our liking. Ultimately we found ourselves thinking about HTML 5, including CSS 3, MathML, and SVG (and possibly also JavaScript, and be it just to remain flexible in situations where something was needed that wasn't covered by HTML 5 as such). While there do exist some technologies in the market to convert HTML to PDF, each of them had some limitations we could not accept.

Because of this, development decided to create their own HTML to PDF technology - a major, non-trivial challenge! Some design decision helped us to not get lost in a sea of requirements and usage scenarios:

- callas pdfChip only creates static two-dimensional PDF content; while a future version of callas pdfChip might support video or audio streams by embedding them as video or audio annotations in PDF, callas pdfChip will never aim to replicate interactive aspects, whether encountered in the form of HTML 5 features like JavaScript, or through technologies like Flash, Silverlight on so on.
- callas pdfChip is not positioned as a technology, that out of the box converts web pages or web sites to decent PDF (though it might work well in numerous cases).
- for optimal use of callas pdfChip certain rules have to be followed (which are explained in the various chapters of this documentation).

So if it's not for converting web sites to PDF – what is it for?

callas pdfChip makes it possible to use HTML – and all the powerful features that come with it – to describe a high quality PDF file. Obviously there are a couple of aspects that can't be done well, or not at all, in HTML when it comes to defining what a PDF shall look like. We decided to work on these aspects in the following ways:

- colour: add colour related features like spot colours, and support flexible handling of colour resources, most notably ICC profiles
- advanced graphics PDF features: fully support transparency, overprint, smooth shades and so forth
- support for XMP metadata
- support for ISO standards, most notably PDF/A-1, PDF/A-2 and PDF/A-3, as well as PDF/X-1a and PDF/X-4
- pagination: as CSS 3 for Paged Media never worked out, a dual pass mode is supported allowing for limitless flexibility to include content that can only be fully known once all the page breaks have been determined
- aggregation:
  - overlay PDF pages onto pages use PDF pages as background for any object
  - overlay PDF pages onto pages
  - import PDF pages (like images), including extensive support for clipping
  - combine several HTML files into one PDF
- barcodes: callas pdfChip supports all 1D and 2D barcodes we are aware of (ca. 130 different symbologies)
- print loop: based on a custom JavaScript function provided by callas pdfChip, and in combination with suitable JavaScript scripting, enables creation of any number of PDF pages in a dynamic fashion, each with partially or completely different content

The above implies that HTML has to be written with the intended purpose of creating decent PDF from it in mind. Unless callas pdfChip is told in some fashion that a certain object is to use a spot colour, and is to be set to overprint, it won't happen. At the same time this does not preclude to write HTML that can also be used … for a web page. So while

callas pdfChip is not a general purpose web page to PDF converter, it can be immensely powerful when it comes to deriving a high quality PDF from a web page, or from a collection of web pages. In most cases callas pdfChip specific features that extend HTML 5, CSS 3 or JavaScript do not cause issues when the same HTML is served through a browser. In some cases, for example when specifying a spot colour or importing PDF pages, a fallback may have to be provided (which is a common pratice anyway in modern web programming, e.g. when following the principles of progressive enhancement).

Overall architecture of callas pdfChip

When developing callas pdfChip we did not start from scratch. There are some technologies readily available that do a great job at processing HTML 5. So we decided to pick one, and we chose WebKit as one of the two building blocks. WebKit is the engine on which the Apple Safari browser is based. As WebKit is dveloped further, callas pdfChip will be updated to inherit the WebKit enhancements.

Web browsers, and by implication WebKit, are optimised for rendering visual content on screen. Taking screen quality visual content to create PDF would leave a lot of thing to be desired if high quality PDFs are needed. Thus the part of WebKit that prepares HTML for output on a screen was replaced by a component developed by the callas software development team, internally named “cchip” (shorthand for “callas convert HTML into PDF”). cchip translates each piece of HTML content into the most suitable representation in PDF, and takes care of all the house keeping chores when writing a PDF.

Some other areas in WebKit had to be customised as well, to support callas pdfChip specific functionality, mostly to access or pass through information that is needed to write high quality PDF but might not be readily available otherwise at the time an object is to be encoded in PDF.

Performance

WebKit is an impressive technology when it comes to performance, and there is probably not much we could do to improve its performance substantially. The PDF creating module cchip though is fully under our own control. The following

top design goals have been and are at the core of the callas pdfChip development:

- create the smallest possible PDF files
- support very long / big PDF files
- create PDF files that are most efficicient when processed (for example by a PDF viewer or printer)
- do not require a lot of memory
- do not require substantially more memory for long / big documents that for short / small documents
- do not add substantial processing time on top of the time WebKit needs to process the HTML
- support current versions of Mac OS X, Microsoft Windows, and Linux
- and last but not least: it is ready when it is ready

The technology behind callas pdfChip has already been put to work before callas pdfChip was published. Since late 2013 callas pdfToolbox allows to create several types of reports based on HTML templates. Since March 2014, callas pdfaPilot can convert HTML based emails to PDF and PDF/A. All in all callas pdfChip has undergone one and half year of extensive testing before it has been shipped.

A word on…

HTML 5 comes as a pack of technologies – CSS 3, MathML, SVG, and JavaScript. All of these are supported by WebKit and thus by callas pdfChip. While it's easy to see in which ways CSS 3 is relevant, it might be less obvious for the other components.

… CSS 3

There are some very important aspects about CSS 3 that one must understand when relying on it: CSS 3 is not one specification; instead it is a group of related specifications. CSS 3 is not “frozen”; instead, new modules can be added at any time. CSS 3 is not necessarily fully supported by any existing implementation; some modules are possibly not supported at all (because they are still too new), others are only supported to a very limited degree (because it is either “not so important” to developers or their market, or maybe to “costly” to implement fully. All this applies to callas pdfChip as

well. An excellent source to find out whether a given CSS 3 feature can be used in callas pdfChip – have a look at the "Can I Use" website at http://caniuse.com/ and check the information about support of a given feature in Apple Safari.

… MathML

Anybody looking at the creation of text books or scientific publications, will be happy to know that MathML can be used in callas pdfChip. Some limitations do apply though:

- MathML (currently at version 3) comes in two flavors: content MathML and presentation MathML. There is hardly any support for content MathML in today's browsers, and everybody – users of MathML in general as much as developers of MathML supporting technology – seem to focus on just presentation MathML.
- Support for presentation MathML in WebKit is not perfect, certain more complex aspects of MathML are just not working in WebKit – unless one adds MathJAX to the equation (pun intended): MathJAX is an open source, free of charge JavaScript library that turbo charges WebKit (or other browsers/web engines), and achieves almost perfect support for presentation MathML (and on the side also allows for use of ASCIIMath, TeX, or LaTeX based representations of mathematical expressions).

… SVG

SVG and PDF share the same imaging concepts, and most of the SVG syntax has direct equivalents with syntax in PDF. This is very handy when one wishes to have maximum control over how content is encoded into a PDF page. SVG does not paginate well – in this regard it is similar to an image.

Note: Where a single page PDF is to be created, SVG files can also be processed directly by callas pdfChip.

… JavaScript

In its early days JavaScript inside HTML content has mostly been used for creation of effects. Over time it became a full fledged programming language, even supporting object ori-

ented programming. Todays rich interactive websites are not thinkable without JavaScript. And driven by the interest in making websites more interesting and interactive, the developers behind the JavaScript engine in WebKit have invested a lot of effort in making it highly performant.

This can be taken advantage of in callas pdfChip. Whether information is to be retrieved from whatever web service, or whether decision about the content to be encoded is to be made on the basis of whatever source of data – it can be done, and it can be done very efficiently. In addition, callas pdfChip can be extended, by using a suitable JavaScript library. For example, the hyphenation support in WebKit is not very good. This can be remedied by using a JavaScript library like the Hunspell based “hyphenator.js” library. Also, in a number of cases where WebKit does not support a recently introduced CSS 3 feature yet, in many cases a so called “polyfill” is available that just fills such a gap and makes WebKit – and thus callas pdfChip – behave as if it supported that feature.

Single pass processing

Unless advanced pagination requirements are to be addressed, the default operating mode, Single Pass, will be fully sufficient. The underlying concept is simple: callas pdfChip processes the incoming HTML file (which implies execution of JavaScript used by the file obviously) and converts all visual content, as well as applicable metadata, to PDF syntax. This resulting PDF syntax is wrapped up in a compact PDF file.

callas pdfChip in many regards behaves like a web browser, thus it is absolutely adequate to use URLs the same way as they are used on HTML pages, It is not a prerequisite that all of the referenced resources exist locally on the machine where callas pdfChip is running. That said – as resolving links can fail in a browser if the respective web server or web services is not reachable or not available, so it can fail in callas pdfChip. In addition, accessing a resource on the local machine or in the local area network tends to work faster than doing the same over the internet.

When making use of JavaScript, it is important to understand that in principle callas pdfChip works in synchronous mode. Where JavaScript is used in an asynchronous fashion. Special

precautions have to be taken into account – make sure to read and understand the section on “pdfChip specific JavaScript aspects”.

Multiple pass processing

Everyone looking at pagination functionality in HTML 5 will end up looking at the CSS 3 Paged Media module. Some will already by disappointed by the limitations in the Paged Media module, like lack of internal styling inside running headers or footers. Disappointment will grow substantially once one finds out that most non-trivial features in the Paged Media module are hardly implemented in any of the leading browsers or web engines.

We felt the same disappointment, and decided to give up on CSS 3 Paged Media and instead choose a different, conceptually pretty simple approach: process the HTML file more than once, remember relevant information from the first processing round and make use of it in following processing rounds. Obvious candidates for this technique are total number of pages (adding text such as “Page 5 out of 12”), or the text of the current (for a given page) section headings for use in running headers and footers.

callas pdfChip collects and then makes available such information between passes. In addition, based on custom JavaScript calls, additional information can be collected during a pass and provided for processing by a subsequent pass. This can become suitable for the creation of fully dynamic table of contents (even for several HTML files converted to a single aggregated PDF file), including correct page numbers and links. The same applies to cross references, lists of figures or indexes.