callas

# pdfChip

callas pdfChip Introduction

# Table of contents

# 01

# pdfChip in a Nutshell

This short chapter will get you up to speed with what pdfChip is and it will introduce the other chapters so that you can decide whether you want to read this introduction from cover to cover or jump to specific chapters instead.

## What is it?

pdfChip is a command-line application that implements a highly-optimised HTML + CSS + JavaScript to high-quality PDF converter (with support for more advanced features such as SVG, MathML, barcodes and more). Lets look at that sentence in a little more detail…

### HTML + CSS + JavaScript

The input of pdfChip is an HTML file. As any regular HTML file used on the internet, it can use CSS (Cascading Style Sheets) to apply styling and it can use JavaScript to perform all kinds of tasks. Because pdfChip is based on the WebKit HTML rendering engine, it is familiar with modern HTML syntax and is in essence as flexible as a browser in the type of HTML it can handle.

But pdfChip also understands extensions to the HTML and CSS languages which allow it to support features that regular HTML and CSS would not have an answer for; this includes support for pagination, extended color spaces, metadata and much more.

### High-quality PDF

The output of pdfChip is a high-quality PDF document. The high-quality aspect means that:

- the PDF code is highly efficient, so that documents with many thousands of pages can be generated that are still very compact. The conversion engine is optimised so that it doesn't generate unnecessary content and includes elements which are used multiple times only once in the PDF document.
- the PDF can be generated in such a way that it is immediately usable for publishing, print production, archival and other professional uses. The ISO PDF/X and PDF/A standards are supported, custom metadata can be included, colors can be specified correctly for a wide range of applications etc…

### A Command-Line Application

pdfChip does not come with a user-interface. Instead it has to be controlled through its command-line interface (or CLI), either by using it in a terminal or command-prompt window, or by integrating it through scripting or a high-level programming language. With its lack of external dependencies, small footprint and fast operation, this makes pdfChip especially suited for integration in web-based or cloud-based applications where it can be used to turn any information into high-quality PDF for further processing.

## How does it work?

With its command-line interface, pdfChip converts an HTML file (which may reference any number of CSS and/or JavaScript files) into a PDF file. The simplest invocation of pdfChip thus becomes:

```
pdfChip <Path to HTML file> <Path to PDF file>
```

Assuming in a terminal window you are in the folder where the pdfChip application is installed and in that folder there is also a file called 'index.html', you could convert this HTML file into a PDF by issuing the command:

```
./pdfChip index.html result.pdf
```

The tutorial (introduced in the chapter "Learning pdfChip - the Tutorial") contains much more elaborate examples, but the command-line will always remain relatively simple. Of course the real power (and complexity) lies within the HTML file that is converted.

## Architecture

### Built on WebKit

Because pdfChip converts HTML into equivalent PDF documents, it is built on top of the WebKit rendering engine. WebKit is the name of the open source web browser engine, used for example by Mac OS X and Safari. Because pdfChip is built using the cross-platform QT framework, it uses the QT WebKit variant to do its work.

Normally WebKit interprets HTML and provides output based on the HTML code and any present CSS and JavaScript. In pdfChip the first part of WebKit is used (the part that interprets HTML) but the back-end is replaced by a custom PDF generator which is highly optimised to generate the minimum necessary PDF code.

### Extending WebKit

Because the HTML object model is rather limited when seen in the context of professional publishing workflows, print production workflows, or archival workflows, the WebKit implementation in pdfChip has been extended. A variety of additional HTML elements and CSS attributes have been introduced to allow generating PDF objects that cannot be described correctly with standard HTML and CSS. The chapter on “Main Features” describes many of these extensions in principal; the pdfChip Reference Manual details them.

pdfChip also provides extensions around the concept of pagination; HTML (and specifically HTML5) introduce a number of "fixed page size" features, but they are typically not very well implemented and are incomplete. pdfChip supports a much wider set of features through its HTML and CSS extentions and a number of custom JavaScript objects and functions. These too are described in somewhat more detail further in this documentation.

## Tell me more!

This pdfChip Introduction gives an overview of pdfChip on a conceptual level, refer to the list below to understand what each chapter explains. But there is also the pdfChip Reference Manual; this separate book contains all of the nitty gritty details you need to use pdfChip in your environment.

- **pdfChip in a Nutshell**
  The chapter you are now reading
- **What pdfChip is Not**
  Knowing what pdfChip is not, is just as important as knowing what it is. This chapter explains what pdfChip should not be used for and highlights some of the limitations you may encounter.
- **The History of pdfChip**
  The origin of pdfChip explains many of the design decisions taken in the product, which in turn will let you use it more efficiently.
- **Main Features**
  A selection of the main features supported by pdfChip.
- **Learning pdfChip - the Tutorial**
  The hardest part of any new product is taking your first steps... Luckily the tutorial was designed to assist you with exactly that. This chapter introduces how you can use the tutorial and what you'll find in it.
- **pdfChip Use Cases**
  So what could pdfChip be used for? It’s far from us to limit your creativity, but this chapter lists a number of possible use cases for the product.
- **Commercial Aspects**
  Because pdfChip supports a wide range of use cases, it comes in a number of different flavors. Each has specific restrictions and this chapter explains the available flavors so that you can make an informed choice.
- **Where to Go from Here?**
  Done with this book? This chapter explains where you can find more information and how to get help if you get stuck either during your evaluation or after having purchased pdfChip.

# 02

# What pdfChip is Not

While the rest of this book focusses on what pdfChip is, this chapter explains what it is not. Because there are a number of products out there that could easily be confused with pdfChip, it's important you realise what you should and should not try to do with pdfChip. Of course your own creativity still rules, but keep this chapter in mind if you're looking at using pdfChip in your workflow.

## A Tool, not a Solution

However powerful pdfChip is, don't expect it to solve all of your workflow issues. pdfChip is a specialized tool to convert HTML files of any level of complexity into first-class PDF documents. It can be an awesome addition to your workflow (see the "Use Cases" chapter for ways in which it can help) but it will always be an addition to the workflow or the application you are building.

## Not a Report Engine

The first thing that might come to mind when thinking about pdfChip is to use it to generate simple documents such as reports or receipts. While pdfChip can definitely be used to accomplish this, keep in mind that it is not a report engine. If your documents are very simple, you might not need a specialized tool such as pdfChip to create them. And it it's really reporting you are after, keep in mind that pdfChip doesn't have customized functionality out of the box to support that.

Of course it's possible to write an HTML file which will produce nice reports including tables, lists, graphs and everything else you might want. But in the end it will probably take a lot of time to accomplish this and the cost for pdfChip and its implementation might become prohibitively high.

## Not a Web Site Conversion Tool

Yes, pdfChip takes HTML and converts it into PDF using a WebKit engine. But no, that doesn't mean it will convert arbitrary HTML or web site pages into nicely printable PDF. pdfChip was designed to take a specially crafted HTML file and provide a range of functionalities that a web site grabber / converter would not possess. If your need is to just convert existing web pages

into PDF, you'll likely find better tools out there.

# 03

# The History of pdfChip

pdfChip was a long time in the making; it originated from a range of feature requests in other callas software products such as pdfToolbox and pdfaPilot. This chapter explains briefly where the idea for a HTML to PDF conversion tool comes from and where you may have seen (parts of it) before.

## A customizable Preflight Report

For a very long time, the pdfToolbox and pdfaPilot products from callas software have allowed quality control and fixing of PDF documents for various purposes. In such a scenario you want to be able to communicate what exactly has been fixed or what errors and warnings have been found, so both products have the notion of a preflight report that details all of that information.

Preflight reports come in various flavors; some are better suited for automated processing (such as text based or XML based preflight reports), others are better suited for human consumption such as HTML or PDF preflight report. About 2005, callas software came up with three different versions of a PDF preflight report that show information either using annotations, transparent overlays or PDF layers.

In essence though, all three reports were static; customers were not able to modify the information on them nor the branding of the preflight report. Having a customisable preflight report that could be adjusted to the needs of a customer became one of the most frequent feature requests for both pdfToolbox and pdfaPilot.

## Not another report language...

The customary way to implement a custom preflight report would be to come up with some sort of "report language" or "report template system" in order to allow customers to modify the look and feel of the generated reports. The problem with this of course is that it is yet another "language" system integrators and customers must master and that such languages are typically either very complex or very limited.

Rather than going down that path, it was decided to develop an HTML template and convert that template on the fly to good PDF. HTML is a well-known and stable standard and lots of people know how to create HTML files. The tools for it are ubiquitous, and the cascading style

sheet (CSS) system provides ample branding capabilities. On top of that there are a number of very fast, stable and open source HTML engines that can be used to handle the heavy lifting around interpreting the HTML and CSS. In this case WebKit was selected as the engine.

The report engine could also take advantage of WebKit's support for Javascript and the HTML templates for the report use Javascript to integrate with the preflight engine and insert the actual results from the preflight into the generated PDF file.

This new preflight report was introduced with pdfToolbox 6 and pdfaPilot 4 and proved to be very flexible, more powerful than originally thought and very popular.

## Email Archival done Right

When pdfaPilot 5 was discussed a similar problem raised its head; the principal new feature for pdfaPilot 5 was to allow conversion of emails into archivable PDF documents. This posed two challenges:

- Different companies want different "views" or "representations" for their archived emails. Again this could have been solved with a custom "template" language, but the previously developed HTML template strategy was again easier and more flexible.
- The more severe challenge rested with the fact that archiving emails is not trivial. To be compliant with PDF/A, a whole list of demands had to be placed on the generated PDF file and the HTML to PDF engine had to flexibily support all of those. Emails also came with hyperlinks, metadata, attachements and other more advanced features that had not been needed to generate simple preflight reports.

In the end the same HTML to PDF engine was used, but it of course received a significant update to allow it to support first class email to archivable PDF conversion. And that update started the thinking that this engine had a reason to exist by itself as a separate product; the birth of pdfChip.

# 04

# Main Features

This chapter does not go into technical details but presents an overview of the main features in pdfChip so that you know what is possible. Use the pdfChip Reference Manual to look up the details about those features if they would come in handy.

## HTML, CSS and Javascript

Because pdfChip is built on top of a WebKit HTML rendering engine, it supports almost all features of HTML and CSS and can take full advantage of Javascript. In general pdfChip follows all HTML rules, so you can include CSS, Javascript, images etc. just as you would while designing a web site. Except for the pdfChip specific features (customised HTML elements and CSS properties) you can even preview your HTML file in a browser or in an HTML design tool.

As with regular HTML for a web site, you can choose to insert your CSS and Javascript in your main HTML file or you can separate them in CSS and Javascript files; pdfChip will process them correctly either way.

To define items not supported by HTML or CSS, pdfChip uses custom HTML elements and custom CSS properties. Their name always begins with `cchip`, `data-cchip` or `-cchip` and they are normally ignored by a browser (the reason there are different prefixes is to keep the HTML and CSS W3C-standard compliant). The tutorial examples provide interesting techniques to use this fact so that the HTML looks one way in a browser but another way when pdfChip converts it into PDF.

## Page sizes and page boxes

While HTML normally lives in a browser, PDF is often meant to be printed. This means that the page size is very important and that additional page boxes may have to be defined; a page box is a rectangle that has special meaning and is used in a particular way by professional publishing solutions. pdfChip supports this by using the `@page` CSS rule set:

```
@page {
    size: 229mm 317mm;
    margin: 20mm;
    -cchip-trimbox: 10mm 10mm 209mm 297mm;
}
```

This example defines an A4 sized page of 20,9cm wide by 29,7cm tall (using the `cchip-trimbox` property) and provides an additional white area around that page to add information that shouldn't be printed, such as the name of the document, time and date, color bars or printer marks (using the `size` property). Check the pdfChip Reference Manual for all page box definitions.

Additionally the `margin` property is used to provide 2cm of margin inside the page - keeping away the HTML content from the edge of the page.

## Professional color

HTML and CSS provide a number of properties to define color, such as the `color` property to set the foreground property and the `background-color` property to set the background color. These properties either accept predefined names (such as "white" or "yellow") or an RGB color code.

pdfChip provides additional color definitions in order to be able to create PDF documents that will print correctly or that comply to standards. The example below shows a background and foreground color for a paragraph element.

```
p {
    background-color: -cchip-cmyk(1.0,0.0,0.0,0.0);
    color: -cchip-cmyk( 'Spot Black', 0.0, 0.0, 0.0, 1.0, 0.5 );
}
```

The background color is defined as CMYK using the `cchip-cmyk` value; given the color values provided the background color will be set to a pure cyan CMYK color. The foreground color uses a modified value that causes pdfChip to generate a spot color (or named color) called "Spot Black" and sets the background color to 50% of that spot color.

Keep in mind that both color properties will be completely ignored if you open the HTML in a browser, as the browser will not understand the `cchip-cmyk` value. It is only because pdfChip uses a customised processing engine that this works.

## Font support

CSS already allows you to specify high-quality fonts for your web pages. The only problem is that not all browsers support the same types of fonts, which often leads to very convoluted `@font-face` definitions. Because pdfChip is only dependent on WebKit, an CSS file for use by pdfChip typically requires a simple font definition, such as the following:

```
@font-face {
    font-family: 'FreeUniversal-Regular';
    src: url('../fonts/freeuniversal-regular.ttf');
    font-weight: normal;
    font-style: normal;
}
```

This specifies the location of the Free Universal font and defines how it can be used in the remainder of the CSS file. The WebKit engine supports most modern fonts (including TrueType and OpenType) and because you only need to make sure it works correctly in pdfChip, testing is much simpler too.

## Using PDF and SVG

Of course you can use various types of images, such as PNG and JPEG images in your HTML. pdfChip inserts those images in the generated PDF. If you use an image more than once, it will only be inserted once in the resulting PDF document.

Browsers can include SVG (Scalable Vector Graphics) files just as they can images; the following example includes an SVG image and works correctly in all browsers **and** in pdfChip.

```
<img src="./images/penguin.svg"/>
```

You can also embed the SVG code immediately into your HTML file and this too is supported by browsers and pdfChip alike. The following example inserts a five-pointed star using SVG.

```
<svg width="100mm" height="100mm">
    <polygon points="100,10 40,198 190,
                      78 10,78 160,198"
             style="fill:lime;stroke:purple;
                    stroke-width:5;
                    fill-rule:nonzero;"/>
</svg>
```

It's important to note that pdfChip doesn't rasterize the SVG; it isn't converted into an image. Instead it is inserted in the PDF so that there is no quality loss, even if the PDF is afterwards scaled up or printed on a high-resolution output device.

But pdfChip goes further than supporting regular images and SVG files; it also supports the insertion of PDF documents directly. Look at the following HTML fragment:

```
<img src="./images/callas-logo.pdf#page=2"/>
```

A regular browser will not display this image, because it doesn't support PDF documents as the source for images. pdfChip does, and for this example will insert the second page (`page=2`) of the given PDF document into the result PDF.

Again no rasterisation takes place - even better - the PDF file is taken as is and inserted into the result PDF with as little changes as possible. This means that pdfChip can easily be used to accomplish impositioning for example (a process where a large sheet is filled with pages from an input PDF so that it can be printed and afterwards cut and folded to a magazine or newspaper for example). But even for less advanced workflows, it means that a resolution independent graphic (a PDF) can be used instead of a plain image.

## ISO compliant PDF

Over the years the ISO (International Standards Organisation) developed a number of important standards around PDF; the two most important once are:

- **PDF/X**: a standard to allow optimal file exchange in graphic arts workflows and,
- **PDF/A**: a standard to allow long-term (50 years or more) PDF file archival.

pdfChip supports both standards through custom HTML elements. Consider the following example HTML:

```
<meta property="cchip-pdfx" content="PDF/X-1a">
<link rel="cchip-outputintent"
      href="./templates/outputintent.pdf"/>
```

The custom `meta` element with its name set to "cchip_pdfx" instructs pdfChip that the PDF it outputs should have the correct PDF/X identification tags inserted. The `content` attibute is set to "PDF/X-1a" which identifies the PDF/X version further as PDF/X-1a, currently the most commonly version of that standard.

The `link` element is also important here; PDF/X files need to contain an output intent and the link element points to a PDF document that contains the output intent we want for our resulting file (a template file if you want). pdfChip will parse the PDF file that is pointed to ("outputintent.pdf" in our example) and copy its output intent into the PDF file is generates.

pdfChip supports more standards; you can find the full list and instructions in the pdfChip Reference Manual. Beware of a potential pitfall however: when pdfChip sees these instructions, it merely inserts the correct standard tags to identify the file it generates as a standards-compliant file. It is still your responsability to ensure that all content in the generated PDF conforms to that standard!

## Inserting custom metadata

Metadata is often very important in document workflows and PDF uses XMP (Extensible Metadata Platform) to carry metadata inside the PDF document. Because metadata is so important, pdfChip has a way to insert it into the resulting PDF document.

```
<meta property="" content="callas documentation"
            data-cchip-xmp-ns="http://www.gwg.org/jt/xmlns/"
            data-cchip-xmp-prefix="gwg-at"
            data-cchip-xmp-property="Publication"
            data-cchip-xmp-type="Text">
```

This custom `meta`element inserts an XMP tag called "gwg-at:Publication" which is of type "Text" and has the value "callas documentation". The prefix links to a namespace defined as "http://www.gwg.org/jt/xmlns/".

It's important to note that some standards (such as PDF/A) require that every piece of metadata inserted in a PDF document is also clearly defined by a metadata definition and pdfChip will correctly insert that information as well. The pdfChip Reference Manual has more details on the subject.

## Support for JavaScript

Perhaps the most powerful aspect of pdfChip and its WebKit foundation is that Javascript is fully supported, and that you can use it just as you would in a browser environment. WebKit really does behave like a browser in almost every aspect and that means you can include Javascript functions to examine and change the HTML DOM (Document Object Model) for example. This you can use Javascript to change properties of elements in your HTML file or to insert completely new elements altogether.

You can insert `script` tags in your HTML file or - just as you're used to on a web site perhaps - you can link to separate script files. Script files you have written or that you downloaded from the Internet. In some of the tutorial examples you'll see JQuery used to manipulate HTML elements and insert new elements. You'll see such advanced scripting functionality come back again as we discuss supporting MathML in the following section.

In the tutorial samples JQuery was downloaded and included in the sample's file structure. However, you can also refer to online Javascript; just be careful if the Javascript calls you make are asynchronuous, pdfChip provides support functions to make sure this works well during conversion.

Javascript also allows implementing scenarios where a lot of external data (data coming from a database for example) needs to be integrated. While your Javascript functionality will not be able to extract data from the database directly, there are way to connect to a URL and gather data (using proxy classes written in another scripting language such as PHP to interrogate the database and return the information requested as XML) and there are ways to read for example CSV files. Together with the possibilities to easily create as many pages as you want during conversion of PDF, this is ideal for many variable data or transactional printing workflows.

## Beautiful formulas with MatML

In some workflows it is important to be able to include nicely formatted mathematical formulas in the generated PDF document (think about textbooks for example). HTML has the possibility to define formulas by using MathML. The following is a MathML representation of probably the most famous formula of all times, thanks to Albert Einstein:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <mrow>
        <mi>E</mi>
        <mo>=</mo>
            <mrow mathcolor='#cc0000'>
                <mi>m</mi>
                <mo> </mo>
                <msup><mi>c</mi><mn>2</mn></msup>
            </mrow>
    </mrow>
</math>
```

Converting this MathML into a beautiful formula can be done in a number of different ways; the tutorial shows how to use the MathJax Javascript library to accomplish this.

$$E = m\ c^2$$

## Inserting barcodes

Barcodes have become almost omnipresent on printed material and the variety of barcodes used is staggering. Annoyingly barcodes are not supported in HTML; there are work-arounds through the use of barcode fonts, but these sometimes lack quality and are limited in the types of barcode they can represent. There is no good solution for 2D barcodes such as QR codes just to name one.

pdfChip itself **does** support barcodes, through the use of the barcode generator TBarCode from TEC-IT Datenverarbeitung GmbH (www.tec-it.com). Just about any barcode you can think of is supported by inserting a custom `object` in the HTML file as such:

```
<object class="barcode" type="application/barcode"
        style="width:30mm; height:30mm;">
    <param name="type" value="QR-Code">
    <param name="data" value="http://www.callassoftware.com">
</object>
```

To use this functionality, you must have an `object` element in your HTML file and its `type` must be set to "application/barcode". The different `param` nodes of this object then provide the necessary input for the barcode generator, most importantly the type of barcode you want to insert and its value. pdfChip would convert the above example in the following QR-code, linking to the callas software web site:

The pdfChip Reference Manual provides full information on all of the supported barcode types and what their parameters should be. It is very important to stress however that pdfChip does

**no** barcode validation, so the parameters you specify should be correct and suitable for the type of barcode you want. If not, pdfChip will return an error or create an incorrect barcode.

## Generating multiple pages

How can you generate multiple 'copies' of your HTML content? If you have a business card layout in HTML, or a form letter... how can you generate a PDF file with thousands of pages, where each page has been tweaked (for example to change names, or addresses or background images or...)?

pdfChip supports this through the use of a predefined Javascript function called `cchipPrintLoop()`. If you define this function in your HTML file or in one of the Javascript files included in your HTML file, it will be called automatically by pdfChip. In it you can setup a loop that modifies the HTML DOM (replacing place holder elements with data you load from a CSV file perhaps) and then calls the `cchip.printPages` function. This is a member function of the `cchip` object and it outputs your HTML file in the state it is at that moment and inserts the generated PDF into the output PDF. You can call `cchip.printPages` multiples times and each time the generated PDF pages will be added to your output. A simple example could look like this:

```
function cchipPrintLoop() {
    for (var i=0; i < 10; i++) {
        /* Modify HTML DOM here */
        cchip.printPages();
    }
}
```

In this example, the HTML DOM isn't actually modified (there's just the comment explaining where you could do this) so the output PDF will consist of 10 identical copies, all concatenated together into your output PDF document. The tutorial contains a few examples of more complex setups where you can see how this could be used to create variable data type documents for example.

Remark that the generated PDF in this example isn't necessarily 10 pages long! If you have an HTML file which converts into a multiple page document, you'll get 10 multiple page PDF files concatenated together. So if your HTML generates a two-page letter, the resulting PDF if you use the above print loop function will be 10 times 2 pages, or 20 pages.

## Advanced pagination

Different than the previous section, advanced pagination comes into play not if you want multiple copies of the same document, but if you have long document which paginates into multiple pages. Think about a book for example: very long HTML that generates a PDF file with potentially hundreds of pages.

The problem with such files is how to add features such as running headers or page numbers, and pdfChip has special support for such environments through something called overlays and underlays. How does this work?

### The problem with pagination

The problem with pagination is that you cannot place page numbers in your original HTML file for example, because you do not yet know how the content will be paginated. And it's hard to predict (and guessing is never a good strategy) where an advanced layout engine such as WebKit will break content into pages.

What you need to overcome this is a sort of two-stage process, where your HTML file would be divided into pages and where you then get the possibility to add additional content to your document. And that is exactly what pdfChip allows, it actually even has a three-stage process.

## Multiple processing steps

In the first chapter of this book, the command-line for pdfChip was introduced as:

```
pdfChip <Path to HTML file> <Path to PDF file>
```

This command-line provides the simple one-stage conversion process that is also used in most tutorial examples. But the command-line allows additional arguments like this:

```
pdfChip <Path to HTML file>
        --underlay=<Path to underlay HTML file>
        --overlay=<Path to overlay HTML file>
        <Path to PDF file>
```

We still start with the main HTML file. This is the HTML that contains the content we want to convert into a PDF file. But this is followed by an `--underlay` and/or `--overlay` command (both are completely optional). If one of these arguments is present, pdfChip does a second and/or third processing step.

First the main HTML file is converted into PDF; after this the pagination is done. The HTML has been converted using the WebKit layout engine and it is now known how exactly the document is going to be converted into PDF pages. The additional passes for the underlay or overlay can use this information to their advantage. When all conversions are done, the underlay PDF document is inserted into the output PDF document; all of its content is inserted underneath the content that is already there (hence the name underlay). The same happens with the PDF generated by the conversion of the overlay HTML but this content obviously is added on top of the output PDF.

## The cchip object

During the first pass, pdfChip stores a lot of information about the document in the `cchip` object and the print loop of the underlay or overlay HTML file can use this (we already mentioned the `cchip` object when introducing multi-page PDF generation earlier. Consider this simple example of an overlay print loop

```
function cchipPrintLoop() {
    for (var i=0; i < cchip.pages.length; i++) {
        $('#overlay-pagelabel p:first').text("page " + (i+1));
        cchip.printPages();
    }
}
```

Our overlay HTML is a very simple one-page file for this example. The print loop queries the `cchip` object to figure out how many pages resulted from paginating the main HTML file. Then it generates the same amount of pages, but each time there is a JQuery expression to change the page number (an object in the overlay HTML identified by the ID "overlay-pagelabel") to the correct value. The result is a paginated file that gets the page numbers neatly added in the second pass pdfChip makes.

# Limitations

While pdfChip is very similar to a browser and while WebKit gives it a lot of flexibility and power, there are still a few limitations you should keep in mind.

## Columns

The CSS properties to generate multiple columns are not supported by pdfChip. Basically pdfChip behaves like the printable version of such content which normally always has one column. Specifically this means that you should not rely on the `column-count`, `column-gap` and `column-rule` properties.

There is a potential work-around through the CSS regions concept, even though this is not an integral part of the CSS standard yet. But WebKit supports it and it is a very powerful layout technology.

## Canvas

The HTML5 canvas is an HTML element that allows drawing graphics on the fly somewhere in an HTML page. It's a powerful technique but you should not, or only after lots of testing, use it in combination with pdfChip. The reason mainly has to do with how the canvas is converted into PDF and most of the time that will be through rasterisation. This means you end up with a PDF document that contains a rasterised version of your canvas content which is typically not what you want.

In most cases look at SVG as a more powerful technique to include arbitrary drawing in our HTML file and maintain it while converting to PDF.

# 05

# Learning pdfChip - the Tutorial

The hardest part of any new product is taking your first steps... Luckily the tutorial was designed to assist you with exactly that. This chapter introduces you to the tutorial and what you'll find in it.

## Folder structure and conventions

The tutorial consists of a number of numbered folders - when new tutorial examples are devised, they will simply be added to the end of the list. Each of these folders is self-contained and is an HTML template together with all of the additional files it needs for pdfChip to be able to convert it into a PDF document. The naming convention has been kept as similar as possible:

- The HTML file to convert is always called `index.html`. In those few examples where there are multiple HTML files a suffix number has been added (index2, index3...)
- If the example needs fonts, they are inside of the `fonts` sub folder.
- If the example needs images, they are inside of the `images` sub folder.
- If the example needs scripts, they are inside of the `scripts` sub folder.
- If the example needs CSS files, they are inside of the `styles` sub folder.
- The `templates` folder contains other PDF files from which pdfChip can copy data (see the section "ISO compliant PDF" in the previous chapter or the tutorial example on "Creating standards-compliant PDF" for more information).

Each tutorial example folder also contains a read me PDF document with more information on what the tutorial step wants to clarify and how to use it.

## Building tutorial examples

For all tutorial examples start with reading the read me PDF file located in the tutorial example folder. If there are any special conversion remarks for that example, they will be detailed there. Most tutorial examples though can simply be converted using the simple pdfChip command-

line syntax; if you are using a terminal or command-prompt window and you change directories into the directory of the tutorial example, the command:

```
<Path to pdfChip>/pdfChip index.html result.pdf
```

will convert the HTML file into a PDF file in the same folder, named `result.pdf`

## The list with tutorial steps

There are tutorial examples for all main features of pdfChip. To see what is available simply go through the sub folders of the tutorial folder. The documentation file always indicates what the topic for that tutorial example is.

The tutorial was designed to take you from the very simple to the more complex subjects. As such it may be worthwhile to simply go through all tutorial examples in the order they are presented. If you are very familiar with HTML and CSS, some of these steps may be obvious as they reiterate HTML or CSS concepts in the light of using pdfChip.

# 06

# pdfChip Use Cases

There are a number of obvious use cases and workflows where pdfChip can easily add value; this chapter lists the most important of those. Use these as inspiration if you want but don't let it stop your creativity to find other, novel, uses!

## Imposition workflows

The fact that pdfChip is capable of 'importing' PDF files into the final output PDF it creates, makes it ideal for workflows where impositioning or any other composition of PDF documents needs to be done. In such workflows it is of primary importance that the imposition or composition process doesn't break the PDF files that are combined and pdfChip is an excellent tool to ensure that.

On top of that the support for SVG should be taken into account here as well, SVG is ideal to add additional information to imposed sheets, whether they are printer or registration marks, color bars or various bits of text. Together with the built-in support for barcodes that provides an excellent environment to decorate the final PDF with all necessary information.

## Creating downloadable web content

In many cases web sites contain or generate information that at some point needs to be printed or delivered in printable format to a visitor of that site; PDF obviously is the ideal medium for that. The fact that pdfChip consumes an HTML template is ideal in many ways:

- Simply changing the CSS file is sometimes enough already to provide a nice, printable view on the information on the site. pdfChip can then be used to convert that printable view into a perfect PDF document.
- If the information comes straight out of a database, the HTML template can use Javascript to read the correct data and modify the HTML template.
- Because pdfChip can produce standards-compliant PDF/X or PDF/A documents, it is ideal for documents that need to go through a publishing or archival workflow afterwards.
- pdfChip works with HTML templates; the skills to edit such templates are wide-spread and the templates are easy to maintain, fit into a version control workflow and share.

pdfChip is also a self-contained solution, it does not need other libraries or tools (such as Adobe InDesign Server for example) to fill the templates with actual data.

The environments where this could be used are very diverse; it could range from generating sales literature on the fly (and customised for the web site visitor) to the generation of receipts, tickets, real estate property information, recipes... Anything that is normally shown on the web site itself but at times needs to be available in a printable form as well.

## Magazine and Newspaper publishing

In many cases magazine and newspaper publishing workflows entail a fair amount of composition, ranging from adding advertisements, to titles, page numbers or barcodes. The same advantages for imposition workflows play here again, PDF files (with ads for example) can be added without any loss of quality or fear that the placed ad might be changed and the HTML templates with its Javascript support are extremely flexible when it comes to adding additional content to the publications.

In some workflows (think of specialty magazines or newspapers for example) the complete template could be done in HTML; with the support for CSS regions in pdfChip that certainly becomes a possibility. In that case pdfChip provides a lightweight and very flexible publication composition engine.

## Online editing

More and more web sites provide the possibility to select a template for a publication and then customise it in more or less complex ways - ranging from adding text to changing fonts, colors and images. The challenge is to provide an online tool which lets the user make choices and then provide an instant customised PDF of the final piece.

Because pdfChip uses HTML templates, the editing part could be built simply using those HTML templates and pdfChip could be used in the background to convert the template into PDF. pdfChip is fast and uses little resources, so it could be called while editing to provide instant feedback. When using the PDF to give feedback to the customer, there are no suprises at the end of the cycle (no differences between the preview of the template and the actual printed final piece).

## Book publishing

The challenge of book printing (and even more so for on-demand book printing) is to take lots of relatively simple but structured content, and produce a relatively long, nice-looking, PDF. The advanced pagination features of pdfChip come in to play to deal with page numbers, running headers and so on.

Because pdfChip can convert multiple HTML files into a single PDF file, it's ideal in environments where a book is delivered in pieces (one HTML file per chapter for example). The Javascript support even lets the template pull the information out of an XML file making links with a database easy.

## Variable data PDF creation and transactional printing

In these workflows the HTML template is typically relatively simple. The challenge is to produce a thousand, or a million pieces that are all customised and to do so in a speedy way while

producing an optimised PDF document. pdfChip can easily read data from XML or CSV and can repeatedly output the same page or pages into a PDF document, where each copy is customised. Because this customisation uses Javascript it's fast and very flexible; it allows customisation of text and graphics with ease.

Creating long documents is a stronghold of pdfChip, its processing time scales up nicely with longer documents and the PDF documents it creates are optimised as much as possible. Images that appear multiple times for example, are included in the generated PDF document only once which helps keep the generated PDF file as small as possible.

Finally, in variable data and transactional workflows, the ability to include barcodes used for marketing purposes (such as QR codes) or administrative purposes (such as data matrices) is of great benefit.

# 07

# License Levels

Because pdfChip supports a wide range of use cases, it comes in a number of different license levels. Each has specific restrictions and this chapter explains the available license levels so that you can make an informed choice.

## Server licensing and activation

pdfChip is always sold per server, whether that server is real or virtual. The license you acquire is for a particular operating system and can be used to activate one (1) server. Activation requires an exchange of information with the callas activation server, which can happen over the Internet (if the server running pdfChip has access to the Internet) or through email.

As all server-level software from callas software, pdfChip comes with an SMA (or Software Maintenance Agreement). The SMA is obligatory for the first year and optional afterwards. With the SMA comes priority support, and free updates and upgrades. The cost of the SMA is 20% of the product price yearly.

## pdfChip license levels

Because of the diversity of environments where pdfChip can be used, the product is sold in a number of different license levels. Different license levels come at a different price point, but also carry restrictions as explained in this section.

### Upgrading between license levels

We understand your needs can evolve over time, and pdfChip can evolve with you. To upgrade from a lower to a higher license level of pdfChip, the cost is the difference in price between those two levels. If you have an active SMA, the difference in price for the SMA will be added to the upgrade price as well.

### Restrictions

This is the explanation of the different restrictions used to diversify the different pdfChip license levels:

- **Parallel processes**

Determines how many conversions from HTML to PDF can be run in parallel. Each conversion runs on a separate processor or processor core (in a different process); your hardware needs to be able to support the number of parallel processes your license allows for optimal performance.

- **Pages per hour**
  The number of pages per hour that pdfChip can generate; this is the number of pages in output PDF files, not the number of HTML files converted.
- **Barcode support**
  The number of barcodes supported by pdfChip; some flavors support only a reduced set of barcodes.
- **Number of pages per document**
  Determines the maximum number of pages that can be output for a single PDF by pdfChip. While this is defined as a hard number in the overview table below, pdfChip will actually allow some overrun (at the expense of slower processing) to provide you with flexibility in your workflow.
- **Advanced pagination**
  Whether or not pdfChip supports to multi-pass advanced pagination features.
- **Tagged PDF**
  Whether or not pdfChip supports creation of Tagged PDF (necessary for some archival workflows and for accessible documents).

### Flavor overview

The following table lists the different pdfChip flavors and the restrictions they implement.

| Feature | S | M | L | XL |
|---|---|---|---|---|
| Parallel processes | 1 | 4 | 8 | Unlimited |
| Number of pages per hour | 1000 | 5000 | 25000 | Unlimited |
| Barcode support | EAN, UPC, ISBN, Code39, Code128, QR | All | All | All |
| Number of pages per document | 25 | 250 | 1500 | Unlimited |
| Advanced pagination | No | Yes | Yes | Yes |

For more information, or exact pricing for pdfChip license levels, see the next section.

## Getting help on pricing

If you have questions on what license level of pdfChip would be the best in your environment, if you want more information on the exact restrictions in a specific license level or if you have questions on licensing conditions of pdfChip, contact callas software directly using the info@callassoftware.com email address or contact Four Pees using info@fourpees.com.

Four Pees is the worldwide distributor for all callas software products and can provide licensing information, put you in contact with a local system integrator or reseller who can help you in your own language or help with any integration or training needs you encounter with pdfChip.

# 08

# Where to Go from Here

You're at the end of this book, but of course there is much more information about pdfChip. This chapter points the way to additional help available to you.

## pdfChip Reference Manual

This pdfChip Introduction book explains what is possible with pdfChip but not how you can accomplish these things; the pdfChip Reference Manual contains all technical details you need to implement pdfChip in your workflow or application. The reference manual is included in the pdfChip package together with this book; if you don't find it or want to make sure you have the latest version, visit the "Downloads" section on the callas software web site.

## The callas software web site

The callas software web site contains trial downloads for all callas products, including pdfChip. The trial will allow you to test pdfChip on your own HTML files in your own environment so that you are certain it is a good fit with what you are looking for.

On the same web site you'll also find more documentation, frequently asked questions and various tutorials. Of course all commercial information about pdfChip and the other callas products is also available.

## Commercial questions

If you have questions on what flavor of pdfChip would be the best in your environment, if you want more information on the exact restrictions in a specific flavor or if you have questions on licensing conditions of pdfChip, contact callas software directly using the info@callassoftware.com email address or contact Four Pees using info@fourpees.com.

Four Pees is the worldwide distributor for all callas software products and can provide licensing information, put you in contact with a local system integrator or reseller who can help you in your own languaged or help with any integration or training needs you encounter with pdfChip.

## Technical questions

For technical questions please check the "Support" section on the callas software web site or on the Four Pees web site. You'll find an extensive knowledge base with frequently asked questions, tips & tricks, tutorials and recordings of webinars explaining various technical aspects of pdfChip.

If that doesn't resolve your question, or if you want to talk to someone about your particular project or integration needs, send an email to support@callassoftware.com or support@fourpees.com. Sometimes a short phone call or a online demo makes all the difference.