pdfChip

callas pdfChip

callas

Table of Contents

1. Introduction 5

1.1 pdfChip in a Nutshell 6

1.2 What pdfChip is Not 10

1.3 The History of pdfChip 12

1.4 Main Features 15

1.5 Learning pdfChip - the Tutorial 27

1.6 pdfChip Use Cases 30

1.7 License Levels 34

1.8 Handling Licensing through the License Server 37

1.9 Where to Go from Here 38

1.10 Overview of pdfChip versions in pdfToolbox 40

1.11 Overview of pdfChip versions in pdfaPilot 41

2. Reference Manual 42

2.1 How to install and run 43

2.2 Concepts 50

2.3 pdfChip specific HTML aspects 57

2.4 Using pdfChip to add barcodes and matrix codes 70

2.5 pdfChip specific CSS aspects 79

2.6 pdfChip specific JavaScript 90

2.7 pdfChip specific SVG aspects 99

2.8 Limitations and warnings 100

2.9 pdfChip CSS Feature Compatibility 102

3. What is new in 1.1 108

3.1 Support for DeviceN color spaces 109

3.2 Passing variable information to HTML template using "--import" on the command line 111

3.3 Additional info when placing PDF pages (# of pages, page geometry) 114

3.4 Creating multiple PDFs with one pdfChip command line invocation 117

4. Links between HTML files are preserved when converted into a single PDF 119

4.1 Links between HTML files are preserved when converted into a single PDF 120

5. Barcodes and matrix codes in pdfChip 121
5.1 List of supported barcodes and matrix codes 122
5.2 Extended list of parameters for the barcode object 133
5.3 How to define the size of barcode objects 161
5.4 How to define barcode objects in HTML and SVG 164
5.5 How to create and update barcode objects dynamically 170
5.6 Using gradient or image as "color" for QR code 174
5.7 Adjust page size for result PDF to size of placed PDF 179
5.8 How to create rectangular 16x48 DataMatrix Industry code 183
5.9 How to create ITF-14 barcode with bearer bars 185
6. Export InDesign files into HTML/CSS templates 187
6.1 Overview and installation 188
6.2 How does the export filter work? 192
6.3 Create a simple HTML template 195
6.4 Create an HTML template that can be used for "mail merge" style multi-page PDF generation 199
6.5 Create an HTML template for invoices 207
6.6 Autocreate Paragraph Styles from custom styling 213
7. Logging and debugging techniques 214
7.1 Extended logging capabilities: "--dump-static-html" command line parameter 215
7.2 Using "pdfChip Debug" browser plug-in for interactive debugging (1.2) 217
8. Dynamically loading resources 224
8.1 Problem description 225
8.2 Dynamic barcodes 227
8.3 Making sure the DOM is ready 229
8.4 An older solution (before pdfChip 2.6) 232
9. Optional content + Processing steps in pdfChip 236
9.1 Optional content (Layers) in pdfChip 237
10. Page size in pdfChip 245
10.1 Set page size in pdfChip 246

10.2 Very large page sizes with UserUnit 249

11. Zoom factor in pdfChip 1.4 251

11.1 Use of zoom factor for increased precision 252

12. pdfChip tips&tricks 253

12.1 Useful code snippets for defining barcode objects, using pdfChip specific CSS, importing PDF pages, setting page size and other things 254

13. pdfChip Switch app 268

13.1 Starting with the pdfChip app 269

13.2 Ways to use the app in a Switch flow 270

13.3 pdfChip app properties 272

1. Introduction

1.1 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.

1.2 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 arbi-

trary 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.

1.3 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 up-

date started the thinking that this engine had a reason to exist by itself as a separate product; the birth of pdfChip.

1.4 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 metaelement 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 MathML

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.

$$E = mc^2$$

Inserting barcodes

Barcodes have become almost omnipresent on printed material and the variety of barcodes used is staggering. Annoying-

ly barcodes are not supported in HTML; there are workarounds 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.

1.5 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 de-

tailed 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 available steps are:

- 001 Simple HTML
- 002 Page geometry boxes
- 003 Positioning
- 004 Standards compliance
- 005 Placing PDF
- 006 Colors
- 007 Metadata
- 008 JavaScript
- 009 Fonts
- 010 Barcode
- 011 Multiple pages through JavaScript
- 012 Multiple pages with advanced pagination
- 013 SVG
- 014 MathML

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.

Download ZIP archive with pdfChip tutorial files

A ZIP archive containing all the necessary files, including read me files providing an explanation for each step, can be downloaded by clicking below:

pdfChip_Tutorial_2022-11-10.zip

1.6 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!

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.

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).

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.

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.

1.7 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.
  If the allowed number of simultaneous pdfChip processes is exceeded, every additional process will remain in a "waiting" mode until another process is finnished before the conversion starts.
- **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.
  If the allowed number of pages per hour is exceeded, pdfChip will pause running and new conversions until the time interval allows the production of pages again.
- **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.
  If the allowed number of pages per document is exceeded, pdfChip will continue to produce the PDF, but every additional page over the limit will be created with an increasing delay.
- **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.

1.8 Handling Licensing through the License Server

In order to avoid having to activate a callas software product, you can use a reference to a running License Server in the call to that product. The `--licenseserver` option points the callas software product to the License Server so it can get permission to run your call. There are two License Server-based models:

Using the License Server

All commands related to the License Server are explained in the License Server manual: Using the License Server.

Using the License Server on-premise

All commands related to the License Server on-premise are explained in the License Server manual: Using the License Server on-premise.

1.9 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 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.

Technical questions

For technical questions please check the "Support" section on the callas software 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. Sometimes a short phone call or an online demo makes all the difference.

1.10 Overview of pdfChip versions in pdfToolbox

pdfToolbox	pdfChip
16.1.661	2.6.098
16.0.656	2.6.095
15.1.639	2.5.084.1
14.4.621	2.5.084
14.2.612	2.5.083
14.1.606	2.5.080
13.0.576	2.4.072
12.1.556	2.3.070
12.0.552	2.2.068
11.1.542	2.2.066
10.2.503	2.1.059
10.1.490	2.0.056
10.1.484	2.0.053
10.1.482	2.0.052

1.11 Overview of pdfChip versions in pdfaPilot

pdfToolbox	pdfChip
14.0.392	2.6.096
13.1.385	2.5.084.1
12.4.372	2.5.084
12.2.366	2.5.082
12.1.360	2.5.080
12.0.355	2.5.076
11.0.340	2.4.072
10.1.332	2.3.070
10.0.330	2.2.068
9.1.325	2.2.066
9.0.320	2.1.062
8.2.312	2.1.059
8.1.303	2.0.056

2. Reference Manual

2.1 How to install and run

pdfChip does not have a user-interface, but is used by a command-line interface (CLI). The application needs a valid activation to run. This activation is bound to the hardware from which the activation was performed.

Installing pdfChip

Available platforms for pdfChip are Windows, macOS and Linux.

You can download the latest version of pdfChip from our website by requesting a trial here: pdfchip

You can also register on our website for easy access to all pdfChip download links. Once registered and logged in, you can access the list of download links here: pdfchip

The package for Linux does not use installer software, it simply has to be unpacked within the designated folder. For example using the following command:

```
gtar zxvpf <callas pdfChip package>.tar.gz
```

Activating pdfChip

Before pdfChip can be used, the software has to be activated.

Request an activation code

Open a terminal window, change into the installation directory and type:

```
pdfChip --keycode <name> <company> trial [--aws]
```

Click to copy
OR

```
pdfChip --keycode <name> <company> <license> [--aws]
```

Click to copy

Parameters

name	name of licensee (e.g. "Registered user")
company	name of company (e.g. "User's company")
license	Licence key obtained from the registration card or the License.pdf provided by callas or the reseller.
trial	To make a request for a trial version, please use the keyword "*trial*" (for a pdfChip trial version)
aws	for installation on Amazon Web Services (EC2, using Windows or Linux 64bit)

The textual output of --keycode has to be send via e-mail to the e-mail address named in the text in order to receive an activation code from the registration server.

Activating pdfChip

After receiving the automatic email reply, the attached "Activation.pdf" has to be saved and pdfChip has to be activated. To do this, open a terminal window, change into the installation directory and type the following command:

```
pdfChip --activate  <activation file>
```

Click to copy

Parameters

activation file	Full path to Activation.pdf

A activated License.txt will then be stored in the user-preferences-folder of the current user.

If no response is received or in the event of an error, please contact support@callassoftware.com to determine the exact cause.

Please note:

It is necessary to activate the received license file to get a permanent valid license file.

The Activation.pdf (or the content of the e-mail) can only be used for activation for 48 hours.
After this time frame, a new Activation.pdf has to be requested from the activation server.

Deactivating pdfChip

As the activation (and the resulting license file) is bound to the hardware, it is necessary to deactivate a license on one machine before a activation can take place on a different machine.

Request the current activation code

Open a terminal window, change into the installation directory and type:

```
pdfChip --status
```

Click to copy

Deactivate using the activation code

Copy the activation code which is listed within the output (24 alphanumeric characters).

```
pdfChip --deactivate <activation code>
```

Click to copy

Confirm the question (Do you want to proceed (y/n)?) by typing "y". The pdfChip activation will be removed from the system.

Using pdfChip

The command-line interface of pdfChip converts an HTML file into a PDF. Referenced images, CSS and JavaScripts will be included in the created PDF.

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

On Mac OS X and Linux, the command in the terminal window should look like:

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

Using Windows, the command would be:

```
pdfChip.exe index.html result.pdf
```

The names of the input HTML file and the output PDF are totally free, you can use whatever works in your environment.

Return codes

All return codes below 100 indicate a successful operation, a code starting with 100 indicates an error.

Return code	Description
0	Successful operation

Return codes indicating an error	
Return code	Description
20	An unknown error has occurred.
21	Unable to write pdf file, file access error.
22	Undefined Mediabox, minimum page size should be 3 by 3 units.
23	An error has occurred in flate encoder.
24	An error has occurred in flate decoder.
25	An error has occurred in lzw decoder.
26	An error has occurred in ASCII-85 decoder.
27	An error has occurred in ASCII-Hex decoder.
28	An error has occurred in RunLength decoder.
29	An error has occurred in prediction decoder.
30	An error has occurred in dct (jpeg-image) decoder.
31	File offset is greater than 9999999999 bytes. PDF only allows offsets up to 10 decimal digits.
32	The required resource name is too large, too much resources are required.
33	A spot color space has a wrong alternate color space type.
34	A path length entry is out of range.
35	There are too many PDF files open.
36	Unable to open the PDF file for import.
37	PDF file is not supported, it might be encrypted.
38	Unable to parse pdf file, syntax error found.
39	The page could not be found.
40	A given parameter is invalid or wrong.

Return codes indicating an error	
Return code	Description
41	The given object number is invalid
42	A inline-image contains an ID without a BI.
43	A content stream contains illegal commands.
44	A resource could not be found.
45	A given resource is wrong.
46	A given function object is wrong.
47	Unable to read font resource.
48	Unable to find glyphs in current font.
49	An error has occured in XMP Metadata function.
50	Invalid context (e.g. a context is popped wich never was pushed)
51	Page without underlay or overlay reservation.
52	The keycode you have entered is invalid.
53	The keycode you have entered has expired.
54	Unable to draw barcode.
55	Unable to open icc profile from file path.
56	Unable to read all required encryption parameter.
57	The page limit is exhausted.
58	Too many parallel processes.
59	The requested barcode is not supported.
60	Unable to read all linearized hint data.
100	Failed loading HTML page.

Return codes indicating an error	
Return code	**Description**
101	Invalid command line argument.
102	Unknown HTML contents.
103	Remote ICC profiles are not supported.
104	WebKit error or JavaScript error.
105	DeviceN is specified incorrectly in CSS (i.e. wrong number of colors)
106	JavaScript extension 'cchip' is not supported

If you are experiencing a return code, which is not listed above (or in the output of the call: "pdfChip --status"), please contact us. Please provide the respective files then, they are necessary to further analyse the issue.

2.2 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.

2.3 pdfChip specific HTML aspects

In pdfChip most valid HTML tags can be used. Due to the big amount of available tags and and an even bigger number of possible combinations, some of them might result in an unexpected result. Due to the different needs for formatting content on a page with a fixed size than for a website (which shall be properly displayed on every output device) some formatting tags don't make sense.

This chapter contains some details of some special HTML features which have been added to achieve some special needs to be able to use PDFs (and not only images) as well as adding XMP Metadata, including PDF Standards identifier, adding an OutputIntent or attaching (embedding) files to the created PDF document. Please refer to the CSS chapter for details regarding layout.

Use PDF as image format

pdfChip allows the usage of PDF pages as source for image tags. Since PDFs can contain more than one page, a syntax for selecting the page to be placed has been added to the HTML syntax.

The PDF that is positioned will not become rasterized, but rather the original PDF content is merged with the generated PDF document.

Also Adobe Illustrator (.ai) files can be used in the same way like PDF files. Only the PDF representation of the file will be used then, all internal Illustrator information stored in the file will be discarded and not be a part of the new PDF file.

URL syntax for PDF pages

The URL for PDF supports the following features:

```
<URL>#page=<PAGE-NUM>&box=<BOXNAME>&boxadj=<LEFT>,<TOP>,<RIGHT>,<BOTTOM>
```

- <URL>: the url to a PDF file
- <PAGE-NUM>: the page number (one based)

- <BOXNAME>: specify the page box used for placement: trim, crop, media, bleed, art. Default: CropBox
- <LEFT>,<TOP>,<RIGHT>,<BOTTOM>: adjustment for the page box. Positive values will extend the selected page box. Default: 0
  Values can be specified in 'mm', 'pt', 'cm', 'pc', 'in' units. Default unit is 'pt.
- Note:
  If the page=<PAGE-NUM> part is missing the first page from the PDF referenced by URL is used for placement.

Example

Place the first page of "sample.pdf"

```
<img src="sample.pdf">
```

Places the second page of sample.pdf

```
<img src="sample.pdf#page=2">
```

Supported tags HTML and CSS properties

- HTML Tags:
  - **<img** src="sample.pdf#page=2">
- CSS properties
  - **background:url**("sample.pdf#page=2")
  - **background-image:url**("sample.pdf#page=2")

Use Adobe Illustrator files (.ai) as image format

Also Adobe Illustrator (.ai) files can be used in the same way as PDF files. Only the PDF representation of the file will be used in this case, all internal Illustrator information stored in the file will be discarded and will not be included in the PDF file generated by pdfChip.

Support for image file formats

pdfChip supports the following image file formats:

- GIF
- PNG
- JPEG, JPG
- TIFF, TIF
- PSD

and also:

- SVG

For the image file formats, pdfChip passes the image data, including masks, alpha channels and ICC profiles, through to the PDF data. By doing so, the image resolution, width and height of image, Bits per color component, color model, transparency, and ICC profile are fully maintained.

For PSD and TIF with Photoshop information the following information is maintained additionally:

- Clipping path (if present)
- Photoshop layers are converted to spot channels
- XMP metadata (if present)

SVG gets converted directly to PDF data structures (for details see pdfChip specific SVG aspects).

Create File Attachment annotations

File attachments can be created by using <a> link tags with pdfChip custom attributes.

A file attachment annotation is created if the <a> tag contains the following attributes:

- href: must be present, content is ignored
- data-cchip-embed: Path to file to embed

Optional attributes:

- data-cchip-mimetype: MIME type of attachment (required for PDF/A-3)
- data-cchip-desc: Desription for attachment
- data-cchip-relationship: The AFRelationShip entry (“Source”, “Data”, “Alternative”, “Supplement”, "Unspecified"; required for PDF/A-3)
- data-cchip-bookmark: Title of optional bookmark entry
- data-cchip-bm-path: Optional path into bookmark tree

The <a> can't be empty and must include some visual content.

Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8"/>
    <title>PDF with File Attachment annotation</title>
    <style>
         ...
    </style>
  </head>

  <body>
        <p>This is a PDF, which contains an embedded text file.<br>It is linked
with the text below:</p>
        <a
          href
                            data-cchip-embed="external/Hello.txt"
                            data-cchip-relationship="Supplement"
                            data-cchip-desc="Embedded text file"
                            data-cchip-mimetype="text/plain">
                              This is the File Attachment annotation.
                  </a>
  </body>

</html>
```

Click to copy

pdfChip-annotation_attachments.zip

Embedding files as attachments into the generated PDF

Files can be embedded as a file attachment to the PDF by specifying a <link> tag with rel attribute with value "cchip-

embedded-file". The href attribute of the link tag must point to a file.

- rel: Value must be "cchip-embedded-file"
- href: Path to file to embed

Optional attributes:

- data-cchip-relationship: The AFRelationShip entry (“Source”, “Data”, “Alternative”, “Supplement”, "Unspecified"; required for PDF/A-3)
- data-cchip-filename: Will be set to actual file name if not specified, otherwise it will be used as file name for the embedded file
- data-cchip-mimetype: MIME type of attachment (required for PDF/A-3)
- data-cchip-desc: Desription for attachment

Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8"/>
    <title>PDF with embedded file</title>
    <link
            rel="cchip-embedded-file"
        href="<Path to file to embed>"
        data-cchip-relationship="<type of relationship, e.g. 'Supplement'>"
        data-cchip-desc="Some description"
        data-cchip-mimetype="<Type of embedded file, e.g. 'text/xml'>"
        data-cchip-filename="<define a custome file name, e.g. 'source.xml'>"
   />
   </head>
  <body>
    ...
  </body>
</html>
```

Click to copy

pdfChip-file-attachments.zip

Add XMP Metadata

pdfChip allows the creation of XMP Metadata by using custom properties in <meta> tags inside <head>.

A <meta> tag is used for XMP metadata creation only if it contains all of the following attributes:

- property
- content
- data-cchip-xmp-ns
- data-cchip-xmp-prefix
- data-cchip-xmp-property
- data-cchip-xmp-type

The 'property' attribute

The contents of this attribute is actually not used for XMP creation, but according to the HTML specification it has to be present.

The 'content' attribute

The contents of this attribute will be used as XMP property value.

The 'data-cchip-xmp-ns' attribute

The cchip_xmp_ns attribute specifies the XMP namespace URI for the property.

The 'data-cchip-xmp-prefix' attribute

The cchip_xmp_prefix attribute specifies the preferred prefix for the XMP namespace URI of the property.

The 'data-cchip-xmp-property' attribute

The cchip_xmp_property attribute specifies the XMP property name.

The 'data-cchip-xmp-type' attribute

The cchip_xmp_type attribute specifies the XMP property value type.

Supported values (case insensitive):

- langAlt: Creates a language alternative. Currently only the creation of the x-default entry is supported.

- seq: Ordered list of simple types
- bag: Unordered list of simple types
- seqstruct: Ordered list of structured types
- bagstruct: Unordered list of structured types

All other types are treated as simple XMP value types (e.g. Text, Date, …).

Arrays of simple types

The *seq* and *bag* property types create a new array if not already present and add the value to this array.

Arrays of structs

The seqstruct and bagstruct property types create a new array if not already present and add the struct value to this array. For specifying the namespace URI and prefix for the struct additional properties must be present in the <meta> tag:

- data-cchip-xmp-struct-ns
- data-cchip-xmp-struct-prefix

Struct members can be specified by the XMP Toolkit subpath syntax:

```
"History[1]/stEvt:when"
```

Examples

Adding the "dc:title" property

This example adds a language alternative for the dc:title property.

```
<html>
    <head>
        <meta
            property="Subject"
            content="ccmip test (Iñtërnâtiônàlizætiøn)"
```

```
            data-cchip-xmp-ns="http://purl.org/dc/elements/1.1/"
            data-cchip-xmp-prefix="dc"
            data-cchip-xmp-property="title"
            data-cchip-xmp-type="langAlt"
        >
    </head>
</html>
```

Adding a "xmpMM::History" property

This example adds a sequence of struct 'ResourceEvent'

```
<!-- Create a xmpMM:History Sequence of struct stEvt::ResourceEvent -->
<meta property="" content="Thursday, 06 August 2015 09:45 PM"
    data-cchip-xmp-ns="http://ns.adobe.com/xap/1.0/mm/"
    data-cchip-xmp-prefix="xmpMM"
    data-cchip-xmp-property="History"
    data-cchip-xmp-type="SeqStruct"
    data-cchip-xmp-struct-name="ResourceEvent"
    data-cchip-xmp-struct-ns="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#"
    data-cchip-xmp-struct-prefix="stEvt"
>
<!-- Add an entry to the xmpMM:History sequence  -->
<meta property="" content="2013-09-06T16:01:13.000Z"
    data-cchip-xmp-ns="http://ns.adobe.com/xap/1.0/mm/"
    data-cchip-xmp-prefix="xmpMM"
    data-cchip-xmp-property="History[1]/stEvt:when"
    data-cchip-xmp-type="Date"
>
<meta property="" content="email_sent"
    data-cchip-xmp-ns="http://ns.adobe.com/xap/1.0/mm/"
    data-cchip-xmp-prefix="xmpMM"
    data-cchip-xmp-property="History[1]/stEvt:action"
    data-cchip-xmp-type="Text"
>
<meta property="" content="Zeitpunkt des Versands des Originals"
    data-cchip-xmp-ns="http://ns.adobe.com/xap/1.0/mm/"
    data-cchip-xmp-prefix="xmpMM"
    data-cchip-xmp-property="History[1]/stEvt:parameters"
    data-cchip-xmp-type="Text"
>
<meta property="" content="Microsoft Office Outlook 12.0"
```

```
    data-cchip-xmp-ns="http://ns.adobe.com/xap/1.0/mm/"
    data-cchip-xmp-prefix="xmpMM"
    data-cchip-xmp-property="History[1]/stEvt:softwareAgent"
    data-cchip-xmp-type="Text"
>
```

Create PDF Standards Identifier

pdfChip allows the creation of PDF documents that pretend compliancy to several PDF standards. There is no guarantee that the files are really compliant since no compliancy check is performed after creation of the PDF document. A <meta> tag is used for triggering the insertion of XMP metadata and Document Info entries for the following PDF standards:

PDF/A

If one of the PDF/A meta tags is present an XMP PDF/A Extension Schema will be created if necessary.

- <meta property="cchip_pdfa" content="PDF/A-1a">
- <meta property="cchip-pdfa" content="PDF/A-1b">
- <meta property="cchip-pdfa" content="PDF/A-2a">
- <meta property="cchip-pdfa" content="PDF/A-2u">
- <meta property="cchip-pdfa" content="PDF/A-2b">
- <meta property="cchip-pdfa" content="PDF/A-3a">
- <meta property="cchip-pdfa" content="PDF/A-3u">
- <meta property="cchip-pdfa" content="PDF/A-3b">

PDF/X

- <meta property="cchip-pdfx" content="PDF/X-1A">
- <meta property="cchip-pdfx" content="PDF/X-3">
- <meta property="cchip-pdfx" content="PDF/X-4">

PDF/E

- <meta property="cchip-pdfe" content="PDF/E-1">

PDF/VT

PDF/VT also sets PDF/X-4

- <meta property="cchip-pdfvt" content="PDF/VT-1">
- <meta property="cchip-pdfvt" content="PDF/VT-2">

PDF/UA

- <meta property="cchip-pdfua" content="PDF/UA-1">

Add Output Intents

Output Intents can be included by specifying an <link> tag with rel attribute with value "cchip-outputintent". The href attribute of the link tag must point to a PDF file that contains at least one Output Intent. pdfChip will parse the PDF file and extract the first Output Intent.

- <link rel="cchip-outputintent" href="./templates/outputintent.pdf"/>

It will insert one Output Intent for every standard requested as described in "Create PDF Standards Identifier" if needed as well. All Output Intents will point to the same ICC profile.

- <meta property="cchip-pdfx" ... > will result in /GTS_PDFX
- <meta property="cchip-pdfa" ... > will result in /GTS_PDFA1
- <meta property="cchip-pdfe" ... > will result in /GTS_PDFE

How to handle parts in separate HTML files

In practice, different parts of a planned document may be contained in a number of HTML files, which are using links between each other to jump between them. As a result pdfChip has to differ between external and internal cross references. It has to include and to adjust the links of those documents, which shall become part of the generated document and leave external links unchanged.

To achieve this, all (references) HTML files, which shall be included in the document have to be added to the CLI call:

```
pdfChip {path to cover/cover.html} {path to first chapter/first.html} {path to second chapter/second.html} ...
```

If an HTML contains a link (<a href="...">) and this link points to one of the input HTML files, this link will become a link annotation, otherwise it will be kept as is and this will become an URI action for an external resource. The HTML input files can be named identically.

- If an HTML link has a href attribute and does not contain a fragment identifier ('#'), the first page of the linked document will be addressed
- If a HTML-link has a href attribute and does contain a fragment identifier ('#'), the substring following the # character will be addressed and used as the ID

Defining the transparency blend space

Setting the blend space can be critical to ensuring consistent rendering results. The blend space for the PDF document to be created can be defined by means of the "cchip-transparency-blendspace" value assigned to the 'rel' attribute inside a link tag in the head section of the HTML document. The actual blend space can be defined as follows:

- data-param (required); can have one of the following values:
  - DeviceCMYK
  - DeviceRGB
  - DeviceGray
  - ICC
- href Either contains path to an ICC profile (only Gray, RGB and CMYK allowed) or is an empty string; only gets used if data-param = "ICC"

Whenever a transparency groups gets created, the following rules apply:

- When a "cchip-transparency-blendspace" 'rel' entry in the head exists:
  - Colorspace defined in data-param = ... (i.e. DeviceCMYK, DeviceRGB, DeviceGray or an ICC profile) will be used.

- If no such entry exists
  - If an OutputIntent is defined (e.g. per <meta name="cchip_pdfx" content="PDF/X-1a">), and the colorspace defined as destination is CMYK, DeviceCMYK will be used as transparency blendspace.
  - If the OutputIntent defines a RGB or Gray colorspace as destination, the respective destination ICC profile will be used.
  - If no OutputIntent is defined, the transparency blendspace will be set to DeviceCMYK

Examples

With referenced ICC profile

```
<html>
    <head>
        ...
        <link
                rel="cchip-transparency-blendspace"
                data-param = "ICC"
                href="./path/to/some/icc-profile.icc"
        />
        ...
    </head>
    <body>
        ...
    </body>
```

Without referenced ICC profile

```
<html>
    <head>
        ...
        <link
                rel="cchip-transparency-blendspace"
                data-param = "DeviceCMYK"
                href=""
        />
```

```
    ...
</head>
<body>
    ...
</body>
```

2.4 Using pdfChip to add barcodes and matrix codes

Although HTML doesn't support barcode generation beyond the usage of barcode fonts, pdfChip offers the possibility to add barcodes directly. The barcode functionality in callas pdfChip is based on the barcode generator TBarCode from TEC-IT Datenverarbeitung GmbH (www.tec-it.com). For extensive information about the various types of barcodes, please read the "Barcode Reference" (download link can be found below).

Portions of this article and the Barcode Reference offered below for download are Copyright TEC-IT Datenverarbeitung GmbH, Steyr/Austria, www.tec-it.com.

Barcode_Reference_EN_2015-10-30-1.pdf

Quick lookup of supported barcodes

Running pdfChip with the parameter ./pdfChip --list-barcodes on the command line:

```
./pdfChip --list-barcodes
```

will output a list of all the barcodes supported by pdfChip. When using the TYPE value to request a barcode through the <param> entry in a barcode <object> make sure to copy everything between the quotes, leaving space characters intact.

Example output from using ./pdfChip --list-barcodes:

```
ID  TYPE                        Data
 1  "Code 11"                   "123456"
 2  "Code 2 of 5 Standard"      "123456"
 3  "Code 2 of 5 Interleaved"   "123456"
 4  "Code 2 of 5 IATA"          "123456"
 5  "Code 2 of 5 Matrix"        "123456"
 6  "Code 2 of 5 DataLogic"     "123456"
```

```
7  "Code 2 of 5 Industry"       "123456"
8  "Code 39"  "ABCDEF"
...
```

How to specify barcodes

Embedding happens using an <object> tag that has to be formatted as follows:

```
<object type="application/barcode">
    <param name="type" value="➔insert name of desired barcode type">
    <param name="data" value="➔insert value to be encoded by the barcode">
</object>
```

No barcode validation takes place, so a wrong value (e.g. incorrect checksum) for the data will result in an invalid barcode (for some barcodes the checksum will be computed automatically if left out from the provided value). Size and layout of the barcode can be adjusted using the usual HTML or CSS parameters. In addition the size and appearance of barcodes can also be controlled by using one or several of the optional parameters described below.

To view the supported types of barcodes, please go to.

Meaning of values provided under "Data"

The values provided in the "Data" column roughly indicate what kind of data can be represented by the respective barcode type. For full details please see the Barcode Reference in the Annex of the Reference Manual.

- ABCabc: lower and uppercase characters and digits; may also support punctuation characters or even arbitray binary data
- ABCDEF: uppercase characters and digits; may also support some punctuation characters
- 123456: digits only; various rules about maximum number of digits and constraints on some of digits may apply

Optional parameters

pdfChip supports various additional optional parameters that provide more fine grained control over the size, appearance and other aspects of 1D and 2D codes. Not all parameter are meaningful for all types of 1D and 2D codes – for full details see the Barcode Reference in the Annex of this Reference Manual.

An example for using optional parameters for the generation of an "EAN 13" code is shown below:

```
<object type="application/barcode">
        <param name="data" value="123456789012">
        <param name="type" value="EAN 13">
        <param name="modulewidth" value="0.33mm">
        <param name="barwidthreduction" value="10%">
        <param name="textplacement" value="none">
</object>
```

String formatting

Provides control over how strings are formatted. For more details see section **4.6 Format** in the **Barcode Reference**.

Usage:

```
<param name="format" value="A##B###C&">
```

Module width

Provides control over the Module width. For more details see section **4.2 Module Width** in the **Barcode Reference**.

Usage:

```
<param name="modulewidth" value="0.33mm">
<!-- **-1**, units: mm, ", mils, **pixel**-->
```

Horizontal resolution

Providing the horizontal resolution triggers an optimisation of the module width for best possible consistency of bars and gaps in the barcode and thus the barcode readability. For more details see section **4.2 Module Width** in the **Barcode Reference**.

Usage:

```
<param name="hres" value="600">
<!-- **-1**-->
```

Vertical resolution

Providing the vertical resolution triggers an optimisation of the module width for best possible consistency of bars and gaps in the barcode and thus the barcode readability. For more details see section **4.2 Module Width** in the **Barcode Reference**.

Usage:

```
<param name="vres" value="600">
<!-- **-1**-->
```

Text placement

Provides control over the positioning of the human readable text relative to the barcode proper. Applies only to 1D codes.

Usage:

```
<param name="textplacement" value="none"><!-- above, **below**, none-->
```

Text distance

Provides control over the distance of the human readable text from the barcode proper. Applies only to 1D codes.

Usage:

```
<param name="textdistance" value="0.5mm"><!-- **-1**, units: mm, ", mils, **pix-
el**-->
```

Bearer bars

Provides control over the presence and position of bearer bars. For more details see section **3.3 Barcode Glossary, Bearer Bars**, **6.1.43 ITF-14** and **6.1.66 UPC SCS** (Shipping Container Symbols)in the **Barcode Reference**.

Usage:

```
<param name="bearerbars" value="topbottom"><!-- **none**, top, bottom, topbottom-->
```

Bearer width

Provides control over the width of bearer bars. For more details see section **3.3 Barcode Glossary, Bearer Bars**, **6.1.43 ITF-14** and **6.1.66 UPC SCS** (Shipping Container Symbols) in the **Barcode Reference**.

Usage:

```
<param name="bearerwidth" value="0.5mm"><!-- **-1**, units: mm, ", mils, **pixel**-
->
```

Notch height

Provides control over the notch height. For certain types of barcodes like e.g. "EAN 13", some of the bars are typically longer than the rest of the bars. This parameter provides control over by how much they will be longer.

Usage:

```
<param name="notchheight" value="0.5mm"><!-- **-1**, units: mm, ", mils, **pixel**-
->
```

Bar width reduction (BWR)

Provides control over the bar width reduction. For more details see section **4.3 Bar Width Reduction** (Pixel Shaving) in the **Barcode Reference**.

Usage:

```
<param name="barwidthreduction" value="1%"><!-- **0**, units: %, mm, ", mils,
**pixel**-->
```

Quiet zone left

Provides control over the quiet zone on the left. For more details see section **4.4 Quiet Zone** in the **Barcode Reference**.

Usage:

```
<param name="quietzoneleft" value="0.5"><!-- **0**-->
```

Quiet zone right

Provides control over the quiet zone on the right. For more details see section **4.4 Quiet Zone** in the **Barcode Reference**.

Usage:

```
<param name="quietzoneright" value="0.5"><!-- **0**-->
```

Quiet zone top

Provides control over the quiet zone at the top. For more details see section **4.4 Quiet Zone** in the **Barcode Reference**.

Usage:

```
<param name="quietzonetop" value="0.5"><!-- **0**-->
```

Quiet zone bottom

Provides control over the quiet zone at the bottom. For more details see section **4.4 Quiet Zone** in the **Barcode Reference**.

Usage:

```
<param name="quietzonebottom" value="0.5"><!-- **0**-->
```

Quiet zone unit

Provides control over the unit used for controlling the quiet zone. For more details see section **4.4 Quiet Zone** in the **Barcode Reference**.

Usage:

```
<param name="quietzoneunit" value="X"><!-- **X**, mm, ", mils, pixel. X: multiples
of module width-->
```

Escaping

For some types of 1D and 2D codes it is possible to encode binary data. Where such binary data includes non-printable characters, such characters need to be provided in an escaped fashion. Several escaping mechanisms can be used. For example, "\h0A" represents the hexadecimal value of "0x0A". Es the escaping mechanisms make use of the backslash character, any occurrence of the actual backslash character must be written as a double backslash ("\\") to avoid unwanted un-escaping.

Escape sequence	Description	Valid for Barcode Symbology
\a	Bell (alert)	All
\b	Backspace	All
\f	Form feed	All

\n	New Line	All
\r	Carriage Return	All
\t	Horizontal Tab	All
\v	Vertical Tab	All
\\	The backslash \ itself	All
\0	Zero Byte (if subsequent char is non-numeric); Available in TBarCode V10+	All
\0ooo	ASCII-character in octal notation: ooo … up to 3 octal digits (0..7); First digit is always zero.	All
\ddd	ASCII-character in decimal notation: ddd … up to decimal digits (0..9); First digit must not be zero.	All
\xhh	For encoding bytes or ASCII-characters in hexadecimal notation; hh … hexadecimal digits (0..F)	All
\Crrggbb	Color selection	See Pharmacode
\Ce	Reset the color to default	See Pharmacode
\F	FNC1 (Function Number Character 1) used as field separator	GS-128, Codablock-F – MicroPDF417: a special FNC1 codeword is inserted when using emulation mode for GS1-128 or Code-128 – Data Matrix: a special FNC1 codeword is inserted
\F	Inserts a Gs (Group Separator) or ASCII 1DHex. Don't encode the \x1d directly!	PDF417, MaxiCode and in QR-Code – QR-Code: When using format UCC/EAN/GS1 Gs is inserted in Byte Mode, a % is inserted in alphanumeric mode.
\Ennnnnn	Extended Channel Interpretation (ECI). nnnnnn … 6 digit ECI number with leading zeros. Used for defining the character set (code page) for subsequent encoded data – see C.1 ECI	MaxiCode, Data Matrix, QR-Code, PDF417, MicroPDF417, Aztec Code
\EB, \EE	Special ECI identifiers for nesting ECIs. \EB (ECI Begin) opens a nesting level, \EE	QR-Code

	(ECI End) closes it.	
\G	Global Language Identifier (GLI), similar to ECI (see \E).	PDF417
\S	Symbol separator character for C128 emulation	
\	Function sequence. Currently FNC1, FNC2, FNC3, FNC4 are implemented. \ is equal to \F.	
\210	FNC1	Code128, GS1-128, Codablock-F
\211	FNC2	Code128, GS1-128, Codablock-F
\212	FNC3	Code128, GS1-128, Codablock-F
\213	FNC4	Code128, GS1-128, Codablock-F
\x11	DC1	Code93, Code93Ext
\x12	DC2	Code93, Code93Ext
\x13	DC3	Code93, Code93Ext
\x14	DC4	Code93, Code93Ext
\x1e	Rs (Record Separator), ASCII 1EHex	PDF417, QR-Code, Data Matrix, MaxiCode (Mode 3,4 SCM)
\x1d	Gs (Group Separator), ASCII 1DHex	PDF417, QR-Code, Data Matrix, MaxiCode (Mode 3,4 SCM)
\x04	Eot (End of Transmission), ASCII 04Hex	PDF417, QR-Code, Data Matrix, MaxiCode (Mode 3,4 SCM)

2.5 pdfChip specific CSS aspects

In pdfChip, (almost) all valid CSS3 properties can be used. On top of that, pdfChip implements a range of additional CSS properties, mainly in order to address certain requirements of the graphic arts industry. This chapter describes these custom CSS properties as well as other properties that are useful in the context of pdfChip.

Page geometry boxes

PDF page geometry boxes can be specified inside the CSS @page{} rule. The following custom CSS properties are available:

```
-cchip-mediabox
-cchip-cropbox
-cchip-bleedbox
-cchip-trimbox
-cchip-artbox
```

Page geometry boxes are defined in PDF coordinates: 0/0 is left bottom of the page and Y goes up (rather than in screen coordinates where center is left top and Y goes down). Each of the page geometry box properties takes four values: The fist two define the coordinates of the lower left corner, the third the width and the forth the height of the box.

If the Mediabox is not explicitly specified it will be set to the bounding box of all specified geometry boxes. If none of the page geometry boxes is present the size entry will be used, if that is not present the default size, A4 (297 x 210 mm), will be used.

Example for a typical A4 page:

```
@page {
    size: 230mm 317mm;
    -cchip-mediabox: 0mm 0mm 230mm 317mm;
    -cchip-cropbox: 0mm 0mm 230mm 317mm;
    -cchip-bleedbox: 7mm 7mm 216mm 303mm;
    -cchip-trimbox: 10mm 10mm 210mm 297mm;
        }
```

If a pdfChip-page geometry box property is set, then:

- the appropriate page geometry box is present in the output PDF
- the appropriate value is available is JavaScript 'page' object

In order to use page geometry boxes in JavaScript the syntax is

- cchip.pages[i].artbox
- cchip.pages[i].trimbox
- cchip.pages[i].bleedbox
- cchip.pages[i].cropbox
- cchip.pages[i].mediabox

E.g. in order to check if the BleedBox is set on the first page:

```
if (cchip.pages[1].bleedbox) {
        ...do something with bleedbox...
        }
```

Force Mediabox and CropBox to be equal and sit at the origin (0,0)

Due to various constraints and limitations, it is often necessary to make the pagesize larger than actually needed and at the same time use a CropBox to crop it to the actually intended displayed/rendered size. This leads to PDFs where the page as displayed/rendered does not sit at the origin but rather away from it (i.e. where the lower left of the CropBox sits). An additional parameter controls whether or not Mediabox and/or Cropbox sit at the origin.

```
-cchip-media-and-crop-box-at-origin: off | on  (off is default value)
```

Click to copy

If '-cchip-mediabox' is present the '-cchip-media-and-crop-box-at-origin' entry is ignored. It will, however, overwrite a '-cchip-cropbox' entry, so that the Cropbox will start at the origin even if the '-cchip-cropbox' specifies otherwise.

Rotating page content

In pdfChip, you can use all CSS positioning properties. This includes properties for rotating page content which are not supported by all web browsers and are therefore not commonly used. For this reason they are listed here.

- -webkit-transform: Sets the rotation factor
- -webkit-transform-origin: Defines the origin for rotation

It is useful to combine these properties with other positioning properties in order to set the origin accordingly.

Example for rotating content 45 degrees counterclockwise with an origin at 20mm / 100 mm (from top of the page).

```
.rotated-45 {
    position: absolute;
    left: 20mm; bottom: 100mm;
    -webkit-transform: rotate(-45deg);
    -webkit-transform-origin: left bottom;
        }
```

Since HTML thinks in 96 dpi, some transformations in HTML cause displacement of content for which users can:

- The Origin in HTML is always at the top left, the Origin for CropBox etc. is at the bottom left (in terms of page size)
- Always make the pagesize/MediaBox a little bit bigger than the page should be at the end, especially in the vertical direction (at least a few mm, rather more), but also in the horizontal direction (here a few points would be enough, but in case of doubt just add 10mm)
- Set the actual page size via the CropBox
- The CropBox should be in the upper left corner and TrimBox equal to the CropBox
- Example for a desired page format of DIN A4 (210x297mm):

- @page {
  /* Size of the page, equivalent to MediaBox, origin always at 0/0 */
  /* 10mm wider and 100m higher than actually needed */
  size: 220mm 397mm;
  /* push the cropbox up 100mm from below, so that it closes at the top */
  -chip-cropbox: 0mm 100mm 210mm 297mm;
  /* trimbox equal to the cropbox, or at least within the cropbox */
  -chip-trim box: 0mm 100mm 210mm 297mm;
  }

Page breaks

Another type of CSS property that is especially useful in pdfChip is related to setting or avoiding page breaks, because page breaks naturally play a much more important role in PDF creation than in the design of web pages.

- page-break-after
- page-break-before
- page-break-inside

Below, the most important values for each of these properties are listed

Value name	Result	Applicable in
auto	Default. Automatic page breaks	page-break-after, page-break-before, page-break-inside
always	Always insert a page break	page-break-after, page-break-before
avoid	Avoid page break (if possible)	page-break-after, page-break-before,

		page-break-inside
left	Insert page breaks so that the next page is formatted as a left page	page-break-after, page-break-before
right	Insert page breaks so that the next page is formatted as a right page	page-break-after, page-break-before

The example below inserts a page break before the next element.

```
<p style="page-break-after: always" />
```

Theoretically, a 'page break' should work as suggested above and simply create another PDF page when deployed. However, this CSS does not always work reliably in the web engines we tested and also not in the webkit we use. In the HTML world (unlike with PDF), page breaks are not the core of the requirements.

To avoid this, the template can be adjusted or the vertical distance between the source pages can be increased, so that the content is mounted on the next page.

Defining colors for print - CMYK, spot or ICC based color

Device color spaces

CSS Property	Value Range	Resulting Color Space
-cchip-gray(g)	g: 0.0 … 1.0	DeviceGray
-cchip-rgb(r,g,b)	rgb: 0.0 … 1.0	DeviceRGB

CSS Property	Value Range	Resulting Color Space
-cchip-cmyk(c,m,y,k)	cmyk: 0.0 … 1.0	DeviceCMYK

Device independent color spaces (ICC based and Lab)

CSS Property	Value Range	Resulting Color Space
-cchip-icc-gray('ICCPATH', g)	g: 0.0 … 1.0	ICC based Gray
-cchip-icc-rgb('ICCPATH', r,g,b)	rgb: 0.0 ... 1.0	ICC based RGB
-cchip-icc-cmyk('ICCPATH', c,m,y,k)	cmyk: 0.0 … 1.0	ICC based CMYK
-cchip-lab(l,a,b)	l: 0.0 … 100.0 ab: -128.0 … +127.0	Lab
-cchip-lab(l,a,b,[D50|D65])	l: 0.0 … 100.0 ab: -128.0 … +127.0 white point: D50 or D65 (default)	Lab
-cchip-icc-lab('ICCPATH', l,a,b)	"l: 0.0 … 100.0 ab: -128.0 … +127.0"	ICC based Lab

With 'ICCPATH' path to a local ICC profile.

Spot color (with Alternate color defintions using device dependent or device independent color spaces)

CSS Property	Value Range	Resulting Color Space
-cchip-gray('NAME',g [, tint])	g: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate DeviceGray
-cchip-icc-gray('ICCPATH', 'NAME',g [, tint])	g: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate ICC based Gray

CSS Property	Value Range	Resulting Color Space
-cchip-rgb('NAME',r,g,b [, tint])	rgb: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate DeviceRGB
-cchip-icc-rgb('ICCPATH', 'NAME',r,g,b [, tint])	rgb: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate ICC based RGB
-cchip-cmyk('NAME',c,m,y,k [, tint])	cmyk: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate DeviceCMYK
-cchip-icc-cmyk('ICCPATH', 'NAME',c,m,y,k [, tint])	cmyk: 0.0 … 1.0, tint 0 … 1.0	Spot color NAME, Alternate ICC based CMYK
-cchip-lab('NAME',l,a,b [, tint])	l: 0.0 … 100.0 ab: -128.0 … +127.0, tint 0 … 1.0	Spot color NAME, Alternate Lab
-cchip-lab('NAME',l,a,b [,D50|D65][, tint])	l: 0.0 … 100.0 ab: -128.0 … +127.0 white point: D50 or D65 (default) tint 0 … 1.0	Spot color NAME, Alternate Lab
-cchip-icc-lab('ICCPATH', 'NAME',l,a,b [, tint])	l: 0.0 … 100.0 ab: -128.0 … +127.0, tint 0 … 1.0	Spot color NAME, Alternate ICC based Lab

With 'ICCPATH' path to a local ICC profile. Profiles have to be accessible in the file system, it is e.g. not possible to derive them via http.

In order to define colors in a way that a regular Browser will be able to display a color the definitions can be combined with regular HTML/CSS color definitions as shown below.

Example that defines a background color as spot color with the name "Spot" using an alternate color in ICC based CMYK C=0% M=80% Y=80% K=0% and ISO Coated v2 as source color space. The spot color is used with a tint value of 75%.

```
.background-spot_orange-ICCbasedcmyk {
                background-color: orange;
                background-color: -cchip-icc-cmyk('./ISO Coated v2 (ECI).icc',
                'Orange',0.0,0.8,0.8,0.0, 0.75)
```

```
                    }
```

DeviceN color spaces

Defining DeviceN color spaces is a bit more complex than using other color spaces, which comes from the fact that DeviceN in principle is a (multi-component) color space conceptually defined in two steps:

- First, one or several separation color space have to be established: often spot color colorants, but also process color colorants (one or even several None components could be possible).
- Second, these separation color space are defined in a certain order and form within the DeviceN color space
- In addition, the DeviceN color space itself also needs an alternate colorspace definition

```
@-cchip-devicen{
-cchip-devicen-name: "test-colorspace-name";
-cchip-components: -cchip-cmyk('Cyan'   ,1   ,0   ,0   ,0  )
                          -cchip-cmyk('Magenta',0   ,1   ,0   ,0  )
                          -cchip-cmyk('Yellow' ,0   ,0   ,1   ,0  )
                          -cchip-cmyk('Black'  ,0   ,0   ,0   ,1  )
                          -cchip-cmyk('Fifth colorant'  ,0.5 ,0.5 ,0.5 ,0  )
                          -cchip-cmyk('Sixth colorant'  ,0   ,0.5 ,0.5 ,0.2)
}
```

The alternates of the components can become defined in all supported spot color definitions (see chapter above).

A mixture of different alternate color spaces may become converted to CMYK-only alternate values.

Using and mixing different channels is similar to the usage of common color spaces, for example:

```
.devicen.c1  { background-color: -cchip-devicen('test-colorspace-
name',1,0,0,0,0,0);  }
 .devicen.c2  { background-color: -cchip-devicen('test-colorspace-
name',0,1,0,0,0,0);  }
    .devicen.c3  { background-color: -cchip-devicen('test-colorspace-name',0,0,0,0.
25,1,0);  }
```

Limitations

- pdfChip colors are implemented only for CSS/HTML but not for JavaScript. The following JavaScript is **not possible** for pdfChip colors:
- note.style.color = "rgb(155, 102, 102)"
- In some situations colors are converted to Device RGB:
  - Rasterization.
  - Colors are accessed via JavaScript. E.g. if “mydiv.style.backgroundColor” in JavaScript it would be output as RGB even if it has accurately been defined as CMYK via '-cchip-cmyk' in CSS.
  - DeviceN is not supported inside rasterized content.

Extended Graphics State parameters

Special pdfChip parameters

CSS Property	Value Range	Default value
-cchip-flatness-tolerance	>= 0.0	1.0
-cchip-smoothness-tolerance	0.0 … 1.0	-1.0 *)
-cchip-text-knockout	0, 1	0
-cchip-overprint	0, 1	0
-cchip-overprint-mode	0, 1	0
-cchip-stroke-adjustment	0, 1	0
-cchip-rendering-intent	absolute-colorimetric, relative-colorimetric, perceptual, saturation	relative-colorimetric
-cchip-black-point-compensation	On, Off, Default	Default

*) Special value -1.0 for pdfChip-smoothness-tolerance means “nothing was set in CSS and pdfChip should use it's own default"

Example that switches overprint and overprint mode ON and sets the rendering intent to "saturation" for a color.

```
.background-spot_orange-ICCbasedcmyk {
    -cchip-overprint: 1;
    -cchip-overprint-mode: 1;
    -cchip-rendering-intent: absolute-colorimetric;
    background-color: orange;
    background-color: -cchip-icc-cmyk('./ISO Coated v2 (ECI).icc',
    'Orange',0.0,0.8,0.8,0.0, 0.75);
}
```

Transparency

The CSS3 property "opacity" can be used in order to define transparent PDF objects.

CSS Property	Value Range	Default value
opacity	0.0 ... 1.0	1.0

E.g. style="opacity: 0.5" sets opacity to 50%, the ca value in the result PDF's Extended Graphic State is thereby set to 0.5.

PDF as image in background

A PDF might be used as the background "image" inside of the background property in the same way as in HTML in the img tag. The PDF objects of the background "image" will show up in the destination PDF as page objects (not rasterized).

Please go to the chapter "pdfChip specific HTML aspects" for further information about selecting a PDF page or clipping a PDF page.

Setting the text rendering mode

The PDF specification allows several "Rendering mode" for text objects. Depending on the defined mode, some text might be invisible (but can be searched and copied, e.g. in scanned and OCR-ed documents), be painted with a stroke, used as a clipping path and much more.

This feature is available since callas pdfChip 2.1.061

```
-cchip-text-rendering-mode: 0;
```

0	Fill text (Default)
1	Stroke text
2	Fill, then stroke text
3	Neither fill nor stroke text
4	Fill text and add to path for clipping
5	Stroke text and add to path for clipping
6	Fill, then stroke text and add to path for clipping
7	Add text to path for clipping

2.6 pdfChip specific 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 oriented 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 to make it very 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. This chapter contains full information on the specific JavaScript functionality added by pdfChip and how you can take advantage of it.

"Normal" HTML JavaScript

Because pdfChip is based on the WebKit engine (since pdfChip version 2.6.092 similar to Safari 16), it fully supports - even advanced - JavaScript. Anything that works in a normal browser will also work during a conversion with pdfChip. Of course there are features that are offered by the browser itself (such as the "Window" object) that won't work in pdfChip because there is no such object during the conversion pdfChip does. To find out if a JavaScript feature is supported in Safari 16.0, you can visit caniuse.com.

The following are a few popular JavaScript libraries that have been tested using pdfKit. This doesn't mean that you are limited to those; it simply shows off some of the possibilities available to you.

- **jQuery:** a small, lightweight and versatile JavaScript library that is mainly interesting in a pdfChip context for its HTML dom traversal and manipulation API.
- **MathJax:** a very complete and easy to use JavaScript library to render formulas in MathML.
- **Hypenathor:** a hyphenation library that can supplement the lack of (good) hyphenation in standard CSS.

- **Polyfill libraries:** are JavaScript libraries used to implement specific CSS features not or not very well implemented by browsers. Many such polyfill libraries exist to plug holes that exist in WebKit for specific advanced CSS features.

Modifying the print loop

The purpose of pdfChip is to convert HTML into good PDF; often use cases will need to modify the given HTML template and alter the appearance of a single page or multiple pages throughout the generated PDF document. To support this pdfChip implements a number of custom Javascript functions and objects that are introduced in this section. Full information about the functions and objects used is available in the following sections.

Use in one-pass conversions

pdfChip defines a printLoop and printPages function to let you take full control over how and when pages are output. This lets you modify (for example) a single-page HTML template and output as many pages as you want:

```
function cchipPrintLoop() {

        for (var theIndex = 0; theIndex < 10; theIndex++) {

            $('#test').text('penguins');
            cchip.printPages();
        }
    }
```

As soon as you include a JavaScript file into your HTML template that defines the above printLoop function, pdfChip will automatically execute it for you. This simple example function interates 10 times; each time it modifies a paragraph using a jQuery statement and then uses cchip.printPages to convert the HTML template as it is at that point in time to PDF pages.

When using cchipPrintLoop in this fashion, you still only end up with one output PDF file, even if you call printPages multi-

ple times. pdfChip always appends the output from printPages to the same (single) output PDF file.

Use in multiple-pass conversions

When using overlays or underlays, the same technique is still usable. Of course uderlays and overlays have a different HTML template and thus will also use different JavaScript files, which allows giving an overlay or underlay an adjusted print loop:

```
function cchipPrintLoop() {

        for (var theIndex = 0; theIndex < cchip.pages.length; theIndex++) {

            $('#test').text('penguins');
            cchip.printPages();
        }
    }
```

The above example for an under- or overlay is virtually identical to the one-pass example with one important change. The number of iterations is now determined by cchip.pages.length. This cchip object is added by pdfChip to give you access to information from the main HTML template. In this example it's used to generate an under- or overlay with the same number of pages as what was generated by the conversion of the original HTML template.

Reference

This section contains reference information for all pdfChip specific JavaScript functions and objects.

cchipPrintLoop

```
function cchipPrintLoop()
```

If the HTML document contains a printLoop function (either embedded in the HTML file or in a separately included JavaScript file), this modifies how pdfChip generates its out-

put PDF file. No PDF creation is done automatically, instead pdfChip relies on the printPages function to be used to output any PDF pages as necessary.

This means that the body of the printLoop function should be used to alter the HTML template as necessary and that the modified HTML DOM should be output by invoking the printPages function. Note that printPages can be invoked multiple times and if so that the result of these multiple invokations will be merged into one output PDF file.

Example:

```
function cchipPrintLoop() {

        for (var theIndex = 0; theIndex < 10; theIndex++) {

            $('#test').text('penguins');
            cchip.printPages();
        }
    }
```

'cchipPrintLoop' is not guaranteed to wait for resources added dynamically by Javascript code.

This can be handled:

- automatically using 'cchip.onPrintReady' method,
- manually using 'cchip.beginPrinting' and 'cchip.endPrinting' methods.
- using await/async JS functionality - more details here.

cchip

During conversion of the main HTML file the cchip object is extended by properties that hold information about the converted document. This information can be used from within the HTML template for an overlay or underlay.

cchip.printPages

```
function cchip.printPages()
```

Outputs the current HTML DOM to the PDF output file. Can be invoked multiple times, but can only be invoked from the body of the printLoop function.

Example:

```
function cchipPrintLoop() {

        cchip.printPages();

    }
```

If used with no parameters the current HTML DOM will be output. The number of pages can be limited by specifying the required number as parameter to the cchipPrintLoop. In this case only one page will be output:

```
function cchipPrintLoop() {

        cchip.printPages(1);
    }
```

cchip.beginPrinting(), cchip.endPrinting()

If 'cchip.beginPrinting' is called, conversion is not finished until matching 'cchip.endPrinting' is called.

If 'cchip.beginPrinting' is called multiple times then 'cchip.endPrinting' should be called multiple times as well.

If 'cchip.beginPrinting' or 'cchip.onPrintReady' is not called conversion is finished just after 'cchipPrintLoop()' method is executed.

Please note that these methods should be used if printing should happen after some specific JS event. Example: If MathJax is used, printing should happen only after MathJax finished all its work. This can be done in the following way:

```
function doPrinting() {
    cchip.printPages();
    cchip.endPrinting();
}
function cchipPrintLoop() {
    cchip.beginPrinting();
    MathJax.Hub.Queue(doPrinting);
}
```

Without 'begin/endPrinting()" calls pdfChip will exit before 'doPrinting' method executed and no output PDF will be created.

cchip.onPrintReady(callback)

cchip.onPrintReady(f) installs a callback function f() that is called when the DOM is ready for printing, e.g. all images are loaded. The normal way to use this function is to first manipulate the DOM, then call cchip.onPrintReady(f) that calls f() when the DOM is ready and exit the cchipPrintLoop(). The function f() must call cchip.printPages() in order to actually create PDF pages from the DOM and initiate further DOM manipulations and printing if required.

The following example illustrates how this function can be used.

```
<html>
<head>
<script>
function cchipPrintLoop(){
var img = document.getElementById("myimg");
img.src = "files/image.jpg";
cchip.onPrintReady( cchip.printPages );
}
</script>
</head>
<body>
<img id="myimg" src="files/2.png">
</body>
</html>
```

The cchipPrintLoop() function is used to place an image (image. jpg) into the DOM. Instead of directly calling cchip.printPages it calls the cchip.onPrintReady function that installs

cchip.printPages as a callback function which makes sure that it will only be used after all images have been loaded.

cchip.dumpStaticHtml()

cchip.dumpStaticHtml() function writes current HTML DOM state to HTML file.
HTML <script> tags are removed during write process. Example, for input "in.html" and output "out.pdf" the following code:

```
function cchipPrintLoop()
{
    cchip.dumpStaticHtml();
    cchip.printPages();
}
```

will produce dump HTML file on the path "dump-static-html-2020-05-15--19-24-15/in-000-out-000-js-000.html"

cchip.log

```
function cchip.log( inTextToLog )
```

This function logs any string pass to it to stderr during conversion of the HTML template.

Example:

```
function cchipPrintLoop() {

    cchip.log("Printing first page of DOM);
    cchip.printPages(1);

  }
```

cchip.urls

An array containing the URLs of all HTML files being converted. Overlays and underlays are not included here. If pdfChip is called with a single HTML file, this list will contain only one

element; if pdfChip receives multiple HTML files on its command-line, all of the main HTML files will be available in this list.

cchip.overlays

An array containing the URLs for all overlay HTML files used during the conversion.

cchip.underlays

An array containing the URLs for all underlay HTML files used during the conversion.

cchip.versionString

Version of the pdfChip executable, e.g. "2.2.066". For 64bit application " (x64)" string appended to version, e.g. "2.2.066 (x64)".

cchip.pages

An array containing information about the individual pages resulting from the conversion of the main HTML template into a PDF document. The different properties of the page elements in this array contain information about the pages. Specifically the following properties can be used:

- number
  The (zero-based) page number of the page.
- mediabox
  Information on the mediabox for the page using a height, width, bottom and left property. All properties are expressed in points.
- cropbox
  Information on the cropbox for the page using a bottom, left, top and right property. All properties are expressed in points.
- trimbox
  Information on the trimbox for the page using a bottom,

left, top and right property. All properties are expressed in points.

- **bleedbox**
  Information on the bleedbox for the page using a bottom, left, top and right property. All properties are expressed in points.
- **margins**
  Information on the margins for the page using a bottom, left, top and right property. All properties are expressed in points.
- **h**
  An array with information for the content (text) of the currently active headers for this page. Because the array is zero based, cchip.pages[theIndex].h[0] returns the content of the current h1 header level.

Example:

```
for ( var i=0; i < cchip.pages.length; ++i ) {
  var page = cchip.pages[i];
  if (page.cropbox)
      cchip.log( "cropbox: " + page.cropbox.left + ' ' + page.cropbox.bottom + ' '
+ page.cropbox.width + ' ' + page.cropbox.height )
}
```

2.7 pdfChip specific SVG aspects

In pdfChip SVG objects are supported in the same way as they work in Webkit as well.

For using pdfChip specific colors, "fill" and "stroke" SVG attributes as well via corresponding "fill" and "stroke" CSS properties can be used.

Example

pdfChip adds some custom fuctionality to the HTML syntax like placing PDFs in image tags or adding XML Metadata to the resulting PDF.

```
<div>
    <svg height=100 width=100>
        <ellipse cx="35" cy="25" rx="27" ry="20"
            fill="-cchip-cmyk('yellow',0,0,1,0,0.9)"
            stroke="-cchip-cmyk(1,0,0,0)">
    </svg>
</div>
```

2.8 Limitations and warnings

In CSS 3 but not (well) supported in pdfChip

While in pdfChip almost all valid CSS3 properties can be used, it does not make sense for some of them. It is obvious that this applies to all dynamic page content like animations.

CSS 3 properties for dynamic page content will have not effect in pdfChip

- Transitions
- Animations
- User-Interface properties
- Aural Style Sheets (text to speech, sound synthesis)

Columns

The CSS 3 properties for columns: column-count, column-gap and column-rule are currenty not supported.

The much more powerfull CSS Regions module should be used instead. The CSS Regions module allows content from one or more elements to flow through one or more boxes.

The CSS 3 Paged Media Module

The Paged Media Module is currently not supported by pdfChip (except for defining page sizes usign the "@page Rule"), nor would that be the case for most of the current browser versions.

The Paged Media Module specifies how pages are generated. It has functionality for page size, margins, orientation, headers and footers, enables page numbering and running headers or footers.

Although the Paged Media Module is not supported it is possible with pdfChip to achieve whatever (in theory) would be possible with this module:

- To define page sizes use the @page rule (the only Paged Media Module feature supported in pdfChip).
- Advanced functionality for adding page numbers, running headers and footers the pdfChip overlays should be used, possibly in combination with the pdfChip Dual Pass operating mode.
- It is even possible to define page paramters that are specific to the print process (page geometry boxes) using special pdfChip custom CSS properties.

In MathML 3 but not (well) supported in pdfChip

callas pdfChip is based on WebKit, and WebKit's support for MathML 3 is seriously limited. Unless extra steps are taken, callas pdfChip will not do a good job when converting non-trivial MathML to PDF.

To overcome this limitation, use MathJax, a JavaScript library that extends WebKit (as much as most other web engines and browsers) such that presentation MathML is supported almost completely (see "Supported MathML commands" for information about the limitations of MathJax when processing MathML).

Please also keep in mind, that support for "Content MathML" is in essence seriously limited (or "experimental"). While Content MathML is semantically richer than presentation it – going back to its nature – provides much less control over how a formula is presented than Presentation MathML. Thus it comes at no surprise that whenever specifics of how a formula is presented are important, anybody is turning to Presentation MathML anyway, so lack of support for Content MathML usually is not an issue for when creating PDF from HTML 5 and MathML 3.

2.9 pdfChip CSS Feature Compatibility

Valid for pdfChip based on 2.6.XXX and higher (e.g. 2.6.092)

Feature	Status
Display: Inline-Block	Supported
Display: Table	Supported
Display: None	Supported
Position: Relative	Supported
Position: Absolute	Supported
Position: Fixed	Supported
Position: Sticky	Supported
Flexbox	Supported
Flex Direction	Supported
Flex Wrap	Supported
Justify Content	Supported
Align Items	Supported
Align Content	Supported
Flex Grow	Supported
Flex Shrink	Supported
Flex Basis	Supported
Order	Supported
CSS Grid	Supported
Grid Template Columns	Supported
Grid Template Rows	Supported

Valid for pdfChip based on 2.5.XXX (e.g. 2.5.080)

Feature	Status
Display: Inline-Block	Supported
Display: Table	Supported
Display: None	Supported
Position: Relative	Supported
Position: Absolute	Supported
Position: Fixed	Supported
Position: Sticky	Not Supported
Flexbox	Supported
Flex Direction	Supported
Flex Wrap	Supported
Justify Content	Supported
Align Items	Supported
Align Content	Supported
Flex Grow	Supported
Flex Shrink	Supported
Flex Basis	Supported
Order	Supported
CSS Grid	Not Supported
Grid Template Columns	Not Supported
Grid Template Rows	Not Supported

3. What is new in 1.1

3.1 Support for DeviceN color spaces

pdfChip CSS rule "-cchip-devicen"

The new pdfChip CSS rule "-**cchip-devicen**" makes it possible to define DeviceN color spaces.

The "-**cchip-devicen**" rule consists of two main parts:

- define the name by which the DeviceN colorspace is known inside the CSS styles:
  -cchip-devicen-name: "test-colorspace-name";
- define the components for the DeviceN colorspace, using a Separation colorspace definition for each such component:
  -cchip-cmyk('Pink', 0.0, 1.0, 0.0, 0.0)

Any alternate space – such as DeviceCMYK, ICC based RGB or Lab, etc. may be used, but it is recommended to use the same alternate space for all components in a given DeviceN colorspace, as otherwise the DeviceN colorspace appearance on devices that do not support all of its colorants, its presentation may only be a rough approximation.

In order to use such a pdfChip DeviceN colorspace definition, the property -cchip-devicen is used, with a reference to the DeviceN colorspace name, and the necessary tint values for each of its components:

- color: -cchip-devicen('tritone-devicen', 1.0, 0.2, 0.6);

The code below shows a complete example:

```
@-cchip-devicen{
        -cchip-devicen-name: 'six-colorant-space';
        -cchip-components:
                -cchip-cmyk('Cyan', 1.0, 0.0, 0.0, 0.0)
                -cchip-cmyk('Pink', 0.0, 1.0, 0.0, 0.0)
                -cchip-cmyk('Yellow', 0.0 ,0.0 ,1.0 ,0.0)
                -cchip-cmyk('Anthracite', 0.0, 0.0, 0.0, 1,0)
                -cchip-cmyk('Orange', 0.5, 0.5, 0.5, 0.0)
                -cchip-cmyk('Green', 0.0, 0.5, 0.5, 0.2);
}

.six-color {
```

```
    color: -cchip-devicen('six-colorant-space', 1.0, 0.2, 0.1, 0.0, 0.28, 0.02);
}
```

Click to copy

3.2 Passing variable information to HTML template using "--import" on the command line

As of pdfChip 1.1, it is possible to pass through variable information from the command line into the pdfChip specific "cchip" user data object. The idea is to associate a key on the command line with a file containing a JSON expression. This implies that a file with such JSON expression must exist before its content can be handed over as a command line parameter.

The "--import" command line parameter

Command line argument

```
--import=<key>:<file.json>
```

or

```
--import=<key>:<file.js>
```

The parameter --import is optional. If present it must also provide a <key> and point to a file <file.json> containing a JSON expression.

- <key> must be a string that can be used as a valid key in JavaScript
- <file.js> resp. <file.json> must be an absolute or relative file path.

Several instances of the --import parameter may be present in a command line call, each creating their own data object.

JSON expression or Javascript variable

The variable information may be provided as a JavaScript file or a JSON file:

- Where a JavaScript file `<file.js>` is provided it must contain a single variable assignment. The JavaScript variable name will be ignored, and only the JSON expression is used.
- Where a JSON file `<file.json>` is provided it must contain single JSON expression.

Using the variable information in JavaScript

The variable information provided in <file.js> or <file.json> will be associated – using the <key> value – with the pdfChip specific data object cchip.user.

Example

- pdfChip command line call:

```
pdfChip ./in.html --import=addresses:./address-list.json  ./out.pdf
```

- content of the variable information as a JSON file address-list.json:

```
{
        contacts [
                {
                        "fullname" : "John Doe",
                        "zip" : "12345",
                        "city" : "Big Town"
                },
                {
                        "fullname" : "Mary Miller",
                        "zip" : "54321",
                        "city" : "Small Town"
                },
                ...
        ]
}
```

- content of the variable information as a JavaScript file address-list.js

```
var some_variable_name = {
```

```
        contacts [
                {
                        "fullname" : "John Doe",
                        "zip" : "12345",
                        "city" : "Big Town"
                },
                {
                        "fullname" : "Mary Miller",
                        "zip" : "54321",
                        "city" : "Small Town"
                },
                ...
        ]
}
```

- retrieving data from the addresses JSON expression

```
var first_city = cchip.user.addresses.contacts[0].city
```

3.3 Additional info when placing PDF pages (# of pages, page geometry)

pdfChip 1.1 introduces the possibility to determine information about imported PDF pages, namely:

- number of pages of the PDF file from which a page has been imported (see `cchip.getPDFPageCount(obj)` below)
- complete page geometry information for each of the pages (see `cchip.getPDFPageBox(obj, box)` below)

This capability makes it possible to find out essential information about PDF pages to be imported without having to preprocess such PDF file or without having to have prior knowledge about such a PDF file. It is nonetheless required to first import at least one page – if in doubt, import the first page – of the resp. PDF. Using JavaScript it is then possible to adjust the settings for the imported PDF page, or to import additional PDF pages and set or adjust their page geometry dependent settings as needed.

cchip.getPDFPageCount(obj)

`cchip.getPDFPageCount(obj)` returns the number of pages in the PDF file, of which a page has been referenced in the `src` attribute of an `<image>` element. The parameter `obj` must be a reference to that `<image>` element. One way to retrieve that reference to the `<image>` element is the use of a function call such as `document.getElementById("my_pdf_id")`, where `"my_pdf_id"` is the ID property of that element.

Example for cchip.getPDFPageCount(obj)

The example shown below simply logs the number of page in the imported PDF to the console (which will show up on stdout) by using cchip.getPDFPageCount(obj).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
```

```
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>
<title>Retrieve number of pages in imported PDF</title>
<script>
    function cchipPrintLoop(){
        var obj = document.getElementById("my_pdf_id");
        cchip.log( '# of pages: ' + cchip.getPDFPageCount(obj) );
        cchip.printPages();
    }
</script>
</head>
<body>
    <div>
        <img id="my_pdf_id" src='sample.pdf#page=1' height="100%" width="100%" />
    </div>
</body>
</html>
```

Output (on stdout) from running the above example with an imported PDF file of 17 pages:

```
JsCChipObject::log: # of pages: 17
```

cchip.getPDFPageBox(obj, box)

`cchip.getPDFPageBox(obj, box)` returns the width and height for the PDF page referenced in the `src` attribute of an `<image>` element. The parameter `obj` must be a reference to that `<image>` element. One way to retrieve that reference to the `<image>` element is the use of a function call such as `document.getElementById("my_pdf_id")`, where `"my_pdf_id"` is the ID property of that element.

Example for cchip.getPDFPageBox(obj, box)

The example shown below illustrates the use of `cchip.getPDFPageBox(obj,box)`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=UTF-8" http-equiv="Content-Type"/>
<title>Retrieving number of pages for an imported PDF</title>
<script>
    function cchipPrintLoop(){
      var obj = document.getElementById("my_pdf_id");
      function printBox( box ) {
          cchip.log( box + " box: " + JSON.stringify( cchip.getPDFPageBox(obj,
box) ) );
      }
      printBox("trim");
      printBox("crop");
      printBox("bleed");
      printBox("media");
      printBox("art");
      cchip.printPages();
    }
</script>
</head>
<body>
        <div>
      <img id="my_pdf_id" src='example.pdf#page=7' height="100%" width="100%" />
        </div>
</body>
```

Output (on stdout) from running the above example:

```
JsCChipObject::log: trim box: {"height":467.7445983886719,"width":381.
3650207519531}
JsCChipObject::log: crop box: {"height":472.8420104980469,"width":385.918212890625}
JsCChipObject::log: bleed box: {"height":450,"width":375}
JsCChipObject::log: media box: {"height":487.5,"width":390}
JsCChipObject::log: art box: {"height":472.8420104980469,"width":385.918212890625}
```

3.4 Creating multiple PDFs with one pdfChip command line invocation

pdfChip 1.1 introduces a very simple mechanism to create more than one PDF file from a single command line invocation of pdfChip:

- `cchip.setOutputPdf()` creates and opens a new PDF file, PDF pages created after this function call will be created in the 'new' PDF
- `cchip.closeOutputPdf()` closes the current PDF output file; it should be called each time no more pages are to be added to the current output PDF. Any output PDF files not closed during processing by pdfChip will be closed automatically once all processing has been completed. As on most operating systems the number of open file handles is limited, care should be taken not to leave output PDF files unnecessarily open.

When used to create just one PDF file per pdfChip command line invocation, there is no need to use either of these two function calls.

cchip.setOutputPdf()

```
cchip.setOutputPdf(<filename>)
```

Creates and opens a new output PDF file. Any PDF pages created after calling this function will be created in the new output PDF file `<filename>`. The folder of the newly cerated output PDF file will be the same as the parent folder for the output PDF file passed as the respective argument on the command line.

The `<filename>` parameter contains the name of the output PDF file to be created and opened. The file name must comply with the rules for file names of the file system on which it is to be created.

cchip.closeOutputPdf()

```
cchip.closeOutputPdf()
```

Closes the current output PDF file. A new output PDF file must be created and opened after issuing this function call, unless no further PDF pages are being created.

Example

In the following example, each time cchip.printPages() is called, a new output PDF file is created and opened, filled with pages by the cchip.printPages() call, and is then closed (to avoid running out of file handles in cases where many thousands of PDF file get created by pdfChip during a single command line invocation - like would be typical when creating delivery notes, invoices, bank statements or similar high volume PDF documents). The file names to be used are retrieved by a custom function getPdfFileName() that retrieves it from an existing list of file names – but any other method could be used equally well.

```
function createNextPDF()
{
    cchip.setOutputPdf( getPdfFileName(filename_list, seq_nr)  );
    cchip.printPages();
    cchip.closeOutputPdf()
}
```

Click to copy

4. Links between HTML files are preserved when converted into a single PDF

4.1 Links between HTML files are preserved when converted into a single PDF

The example creates a single PDF out of 9 different pieces (index.html files). The table of contents on the second "page" of the first input index.html (0000 - Cover and TOC) has links to all other input files. These remain working in the result PDF.

pdfChip_Introduction_Example-1.zip

The sample has a build script in the respective folder that allows for creating a result PDF from the single source files on a Unix/Linux based computer. The following should work on Windows with a proper path to pdfChip and when executed from the root folder of the example.

```
"C:\Program folder\callas pdfChip\pdfChip" --overlay=".\_resources\html\overlay.
html" --underlay=".\_resources\html\underlay.html" ".\0000 - Cover and TOC\index.
html" ".\0100 - pdfChip in a Nutshell\index.html" ".\0200 - What pdfChip is not\in-
dex.html" ".\0300 - The History of pdfChip\index.html" ".\0400 - Main Features\in-
dex.html" ".\0500 - Learning pdfChip - the Tutorial\index.html" ".\0600 - pdfChip
Use Cases\index.html" ".\0700 - Licensing and Flavors\index.html" ".\0800 - Where
to Go from Here\index.html" ".\pdfChip Introduction.pdf"
```

Click to copy

pdfChip takes more than one input file and merges them into a single output file (if not otherwise directed e.g. in the cchip.PrintLoop).

Beginning with pdfChip 1.2 links between several input HTML files are transferred into PDF links. In the example that results in a Table of Contents page that accurately links to the respective chapters in the manual.

Please note that this is an old version of the pdfChip manual, so it should only be used for testing pdfChip and not for reading about pdfChip. E.g. the old manual does not have any information that "Links between HTML files are sustained when converted into a single PDF"...

5. Barcodes and matrix codes in pdfChip

5.1 List of supported barcodes and matrix codes

The barcodes and matrix codes listed in the table below are supported by pdfChip.

For a more detailed introduction to barcode objects, please have a look at the pdfChip Reference article about barcode objects. For more information about barcodes and matrix codes in general please download the "Barcode Reference Manual":

Barcode_Reference_EN_2018-06-11.pdf

List of barcodes and matrix codes supported by pdfChip

1	Code 11	123456	<param name="type" value="Code 11">
2	Code 2 of 5 Standard	123456	<param name="type" value="Code 2 of 5 Standard">
3	Code 2 of 5 Interleaved	123456	<param name="type" value="Code 2 of 5 Interleaved">
4	Code 2 of 5 IATA	123456	<param name="type" value="Code 2 of 5 IATA">
5	Code 2 of 5 Matrix	123456	<param name="type" value="Code 2 of 5 Matrix">
6	Code 2 of 5 DataLogic	123456	<param name="type" value="Code 2 of 5 DataLogic">
7	Code 2 of 5 In-	123456	<param name="type"

	dustry		value="Code 2 of 5 Industry">
8	Code 39	ABCDEF	<param name="type" value="Code 39">
9	Code 39 Full ASCII	ABCabc	<param name="type" value="Code 39 Full ASCII">
10	EAN 8	12345670	<param name="type" value="EAN 8">
11	EAN 8 + 2 Digits	1234567012	<param name="type" value="EAN 8 + 2 Digits">
12	EAN 8 + 5 Digits	1234567012345	<param name="type" value="EAN 8 + 5 Digits">
13	EAN 13	1234567890128	<param name="type" value="EAN 13">
14	EAN 13 + 2 Digits	123456789012812	<param name="type" value="EAN 13 + 2 Digits">
15	EAN 13 + 5 Digits	123456789012812345	<param name="type" value="EAN 13 + 5 Digits">
16	EAN/UCC 128	ABCabc	<param name="type" value="EAN/UCC 128">
17	UPC 12	123456789012	<param name="type" value="UPC 12">
18	Codabar 2 Widths	A123456A	<param name="type" value="Codabar 2 Widths">
20	Code 128	ABCabc	<param name="type" value="Code 128">
21	DP Leitcode	012345678	<param name="type" value="DP Leitcode">

22	DP Identcode	012345678	<param name="type" value="DP Identcode">
23	ISBN 13 + 5 Digits	978-1-23456-789-712345	<param name="type" value="ISBN 13 + 5 Digits">
24	ISMN	979-0-1234-5678-5	<param name="type" value="ISMN">
25	Code 93	ABCDEF	<param name="type" value="Code 93">
26	ISSN	9771234567898	<param name="type" value="ISSN">
27	ISSN + 2 Digits	977123456789812	<param name="type" value="ISSN + 2 Digits">
28	Flattermarken	123456	<param name="type" value="Flattermarken">
29	GS1 DataBar (RSS-14)	00614141999996	<param name="type" value="GS1 DataBar (RSS-14)">
30	GS1 DataBar Limited (RSS)	00614141999996	<param name="type" value="GS1 DataBar Limited (RSS)">
31	GS1 DataBar Expanded (RSS)	0100614141999996	<param name="type" value="GS1 DataBar Expanded (RSS)">
32	Telepen Alpha	ABCabc	<param name="type" value="Telepen Alpha">
33	UCC 128	ABCabc	<param name="type" value="UCC 128">
34	UPC A	123456789012	<param name="type" value="UPC A">
35	UPC A + 2 Digits	12345678901212	<param name="type" value="UPC A + 2 Digits">
36	UPC A + 5 Dig-	12345678901212345	<param name="type"

	its		value="UPC A + 5 Digits">
37	UPC E	12345670	<param name="type" value="UPC E">
38	UPC E + 2 Digits	1234567012	<param name="type" value="UPC E + 2 Digits">
39	UPC E + 5 Digits	1234567012345	<param name="type" value="UPC E + 5 Digits">
40	USPS PostNet 5 (ZIP)	12345	<param name="type" value="USPS PostNet 5 (ZIP)">
41	USPS PostNet 6 (ZIP+cd)	123455	<param name="type" value="USPS PostNet 6 (ZIP+cd)">
42	USPS PostNet 9 (ZIP+4)	123456789	<param name="type" value="USPS PostNet 9 (ZIP+4)">
43	USPS PostNet 10 (ZIP+4+cd)	1234567895	<param name="type" value="USPS PostNet 10 (ZIP+4+cd)">
44	USPS PostNet 11 (ZIP+4+2)	12345678901	<param name="type" value="USPS PostNet 11 (ZIP+4+2)">
45	USPS PostNet 12 (ZIP+4+2+cd)	123456789014	<param name="type" value="USPS PostNet 12 (ZIP+4+2+cd)">
46	Plessey	123456	<param name="type" value="Plessey">
47	MSI	123456	<param name="type" value="MSI">
48	SSCC 18	012345678901234560	<param name="type" value="SSCC 18">
50	LOGMARS	ABCDEF	<param name="type"

			value="LOGMARS">
51	Pharmacode One-Track	123456	<param name="type" value="Pharmacode One-Track">
52	PZN7	1234562	<param name="type" value="PZN7">
53	Pharmacode Two-Track	123456	<param name="type" value="Pharmacode Two-Track">
54	Brazilian CEP-Net	12345678	<param name="type" value="Brazilian CEP-Net">
55	PDF417	ABCabc	<param name="type" value="PDF417">
56	PDF417 Truncated	ABCabc	<param name="type" value="PDF417 Truncated">
57	MaxiCode	ABCabc	<param name="type" value="MaxiCode">
58	QR-Code	ABCabc	<param name="type" value="QR-Code">
59	Code 128 Subset A	ABCabc	<param name="type" value="Code 128 Subset A">
60	Code 128 Subset B	ABCabc	<param name="type" value="Code 128 Subset B">
61	Code 128 Subset C	ABCabc	<param name="type" value="Code 128 Subset C">
62	Code 93 Full ASCII	ABCabc	<param name="type" value="Code 93 Full ASCII">
63	Australian Post Custom	12345678	<param name="type" value="Australian Post

			Custom">
64	Australian Post Custom2	12345678ABab	<param name="type" value="Australian Post Custom2">
65	Australian Post Custom3	12345678ABCabc	<param name="type" value="Australian Post Custom3">
66	Australian Post Reply Paid	12345678	<param name="type" value="Australian Post Reply Paid">
67	Australian Post Routing	12345678	<param name="type" value="Australian Post Routing">
68	Australian Post Redirect	12345678	<param name="type" value="Australian Post Redirect">
69	ISBN 13	978-1-23456-789-7	<param name="type" value="ISBN 13">
70	Royal Mail 4 State (RM4SCC)	ABCDEF1234	<param name="type" value="Royal Mail 4 State (RM4SCC)">
71	Data Matrix	ABCabc	<param name="type" value="Data Matrix">
72	EAN 14 (GTIN 14)	00614141999996	<param name="type" value="EAN 14 (GTIN 14)">
73	VIN / FIN	VB1YYY1JX3M386752	<param name="type" value="VIN / FIN">
74	Codablock-F	ABCabc	<param name="type" value="Codablock-F">
75	NVE 18	012345678901234560	<param name="type" value="NVE 18">
76	Japanese Postal	1234567	<param name="type" value="Japanese Postal">

77	Korean Postal Authority	123456	<param name="type" value="Korean Postal Authority">
78	GS1 DataBar Truncated (RSS)	00614141999996	<param name="type" value="GS1 DataBar Truncated (RSS)">
79	GS1 DataBar Stacked (RSS)	00614141999996	<param name="type" value="GS1 DataBar Stacked (RSS)">
80	GS1 DataBar Stacked Omnidir (RSS)	00614141999996	<param name="type" value="GS1 DataBar Stacked Omnidir (RSS)">
81	GS1 DataBar Expanded Stacked (RSS)	0100614141999996	<param name="type" value="GS1 DataBar Expanded Stacked (RSS)">
82	PLANET 12 digit	123456789014	<param name="type" value="PLANET 12 digit">
83	PLANET 14 digit	12345678901239	<param name="type" value="PLANET 14 digit">
84	Micro PDF417	ABCabc	<param name="type" value="Micro PDF417">
85	USPS Intelligent Mail Barcode (IM)	12345678901234567890	<param name="type" value="USPS Intelligent Mail Barcode (IM)">
86	Plessey Bidirectional	123456	<param name="type" value="Plessey Bidirectional">
87	Telepen	123456	<param name="type" value="Telepen">
88	GS1 128 (EAN/UCC 128)	01090999995432171512052110Abc123	<param name="type" value="GS1 128 (EAN/UCC 128)">

89	ITF 14 (GTIN 14)	00614141999996	<param name="type" value="ITF 14 (GTIN 14)">
90	KIX	AaBbCcDdEe	<param name="type" value="KIX">
91	Code 32	012345676	<param name="type" value="Code 32">
92	Aztec Code	ABCabc	<param name="type" value="Aztec Code">
93	DAFT Code	DAFT	<param name="type" value="DAFT Code">
94	Italian Postal 2 of 5	123456789012	<param name="type" value="Italian Postal 2 of 5">
96	DPD	00071106016325329483751792 76	<param name="type" value="DPD">
97	Micro QR-Code	ABCDEF	<param name="type" value="Micro QR-Code">
98	HIBC LIC 128	+A99912345/9901510X3	<param name="type" value="HIBC LIC 128">
99	HIBC LIC 39	+A99912345/9901510X3	<param name="type" value="HIBC LIC 39">
100	HIBC PAS 128	+/EAH783/Z34H159	<param name="type" value="HIBC PAS 128">
101	HIBC PAS 39	+/EAH783/Z34H159	<param name="type" value="HIBC PAS 39">
102	HIBC LIC Data Matrix	+A99912345/9901510X3	<param name="type" value="HIBC LIC Data Matrix">
103	HIBC PAS Data Matrix	+/EAH783/Z34H159	<param name="type" value="HIBC PAS Data Matrix">
104	HIBC LIC QR-	+A99912345/9901510X3	<param name="type"

	Code		value="HIBC LIC QR-Code">
105	HIBC PAS QR-Code	+/EAH783/Z34H159	<param name="type" value="HIBC PAS QR-Code">
105	HIBC PAS Aztec Code	+/EAH783/Z34H159	<param name="type" value="HIBC PAS Aztec Code">
106	HIBC LIC PDF417	+A99912345/9901510X3	<param name="type" value="HIBC LIC PDF417">
107	HIBC PAS PDF417	+/EAH783/Z34H159	<param name="type" value="HIBC PAS PDF417">
108	HIBC LIC Micro PDF417	+A99912345/9901510X3	<param name="type" value="HIBC LIC Micro PDF417">
109	HIBC PAS Micro PDF417	+/EAH783/Z34H159	<param name="type" value="HIBC PAS Micro PDF417">
110	HIBC LIC Codablock-F	+A99912345/9901510X3	<param name="type" value="HIBC LIC Codablock-F">
111	HIBC PAS Codablock-F	+/EAH783/Z34H159	<param name="type" value="HIBC PAS Codablock-F">
112	QR-Code 2005	ABCabc	<param name="type" value="QR-Code 2005">
113	PZN8	12345678	<param name="type" value="PZN8">
115	DotCode	ABCabc	<param name="type" value="DotCode">
116	Han Xin Code	ABCabc	<param name="type" value="Han Xin Code">
117	USPS Intelli-	9102805213683062522920	<param name="type"

	gent Mail Package (IMpb)		value="USPS Intelligent Mail Package (IMpb)">
118	Swedish Postal Shipment Item ID	EM100027995SE	<param name="type" value="Swedish Postal Shipment Item ID">
119	Royal Mail Mailmark® 2D Barcode	JGB 012100123412345678AB19XY1A 0 ABCDEFGHIJ1234567890ABCDEFGHIJ1234567890A	<param name="type" value="Royal Mail CMDM Mailmark">
120	UPU S10 – Generic Postal Code	EM100027995SE	<param name="type" value="UPU S10">
121	Royal Mail Mailmark® 4-state Barcode	41038422416563762EF61AH8T	<param name="type" value="Royal Mail Mailmark 4-state">
124	HIBC LIC Aztec Code	+A99912345/$$52001510X3	<param name="type" value="HIBC LIC Aztec Code">
124	Pharmacy Product Number Code (PPN Code)	9N110375286414	<param name="type" value="PPN (Pharmacy Product Number)">
125	NTIN Code	04150123456782	<param name="type" value="NTIN (Data Matrix)">
127	GS1 Data Matrix	010909999954321715120521 10Abc123	<param name="type" value=""GS1 Data Matrix">
128	GS1 QR Code	010909999954321715120521 10Abc123	<param name="type" value=""GS1 QR Code">

Note:
The ID listed in the table above is based on the ID assigned internally to the respective barcode by the used library. They may therefore be not unique.

Equivalent barcodes

Some barcodes are known by regionally different names:

Name used in barcode library	Alternative names
EAN 13	JAN / Japanese Article Number
Pharmacode One-Track	LAETUS-Code
EAN/UCC 128	GS1-128 EAN-128 UCC-128
Code 2 of 5 Interleaved	I-2/5 Code 25
Royal Mail 4 State (RM4SCC)	Singapore Post 4-State Customer Code (Sin-Post)

5.2 Extended list of parameters for the barcode object

As of version 1.2, pdfChip supports a substantially extended list of parameters for the barcode object. Many of these parameters are symbology specific, and a good understanding of how a given barcode or matrix code works is required in order to successfully use the parameters.

Whenever uncertain which parameters to use, and how to use them, please contact the callas software support team at support@callassoftware.com.

Parameter	Value	Descri
activetextindex	number	Sets th The us rently That n tIndex ject #1 ner. Note: By def You ca that o
autocorrect	either "true" or "false"	Sets a If set t the inp - Code

Parameter	Value	Descri
		acters - ISBN the ba - Code data d - Code and al - GS1- (the FI Note:
aztec_enforcebinaryencoding	either "true" or "false"	Deterr mode Note: bols th
aztec_errorcorrection	number	Sets th (from Note: 3 addi
aztec_format_format	List of possible values: "Default" (default value) or "UCCEAN" or "Industry"	Specif
aztec_format_specifier	text string	Specif try for
aztec_runemode	either "true" or "false"	Deterr mode.

Parameter	Value	Descri
		Note: A tion. A
aztec_size	List of possible values:, "Default" (used as default), "15x15", "19x19", "23x23", "27x27", "31x31", "37x37", "41x41", "45x45", "49x49", "53x53", "57x57", "61x61", "67x67", "71x71", "75x75", "79x79", "83x83", "87x87", "91x91", "95x95", "101x101", "105x105", "109x109", "113x113", "117x117", "121x121", "125x125", "131x131", "135x135", "139x139", "143x143", "147x147", "151x151", "19x19_Rd", "23x23_Rd", "27x27_Rd" and "Rune"	Sets A Specif colum size is compu Note: for rea
barshape_shape	List of possible values: "Default", "Rectangle", "Ellipse", "BigEllipse", "RoundedRectangle", "Image"	Sets th Attent ly bar dange A char advert readal Note:
barwidthreduction	number with unit, allowed units are "mm", "cm", "m", "in", "ft", "pt", "pc" and "%"	Sets b Specif in the This p such p the ink settin an sui tings b paper,

Parameter	Value	Descri
		the th tion va
bearerbars	List of possible values: "none" (used as default), "topbottom", "top", "bottom" and "Rectangle"	Sets th By def bearer tom" c for the The ve et zon essary Remar bearer the pr also e proba Bearer (ITF-14 drawn synchr bars. See th
bearerwidth	number with unit, allowed units are "mm", "cm", "m", "in", "ft", "pt" and "pc"	Sets th mm]. By def bearer tom" c for the The ve et zon

Parameter	Value	Descri
		essary
cbf_columns	number	Adjust Specif Codab numb
cbf_format	List of possible values: "Default" (used as default) and "UC-CEAN"	Sets C Specif EAN o
cbf_rowheight	number	Sets th Specif value i accord sure th ping.
cbf_rows	number	Adjust Specif ablock of row
cbf_rowseparatorheight	number	Sets th Specif value i calcul gle). If to avo

Parameter	Value	Descri
cdmethod	List of possible values: "None" (used as default), "Standard", "Mod10", "Mod43", "2Mod47", "DPLeit", "DPIdent", "1Code11", "2Code11", "USPSPostnet", "MSI1", "MSI2", "Plessey", "EAN8", "EAN13", "UPCA", "UPCE", "EAN128", "Code128", "RM4SCC", "PZN", "Mod11W7", "EAN14", "Mod10Kor", "Mod10Pla", "Mod10ItlPst25", "Mod36", "Mod16", "Mod10Luhn", "VIN", "Mod10LuhnRev", "Mod23PPSN", "Mod10IMPackage", "Mod11W10" or "SwedishPostal"	Choos The m the ad user-fr each b codes metho ogies Code- code E Attent or reco is gene charac er that adapte
codepage	List of possible values: "Custom", "ANSI", "Windows1252", "Latin_I", "ASCI-IExt_437", "UTF8", "Korean", "Japan-ese_Shift_JIS", "Sim-plified_Chinese", "Trad_Chinese_Big5", "ANSI_Cyrillic", "KOI8_R", "GB18030", "MAC_Roman", "ISO_8859_1", "ISO_8859_2", "ISO_8859_3", "ISO_8859_4", "ISO_8859_5", "ISO_8859_6", "ISO_8859_7", "ISO_8859_8", "ISO_8859_9",	Prede If the e the va ed coc the co The ite ten us When barcoc Symbo For ex MicroP QR-Co All oth ANSI.

Parameter	Value	Descri
	"ISO_8859_10", "ISO_8859_11", "ISO_8859_12", "ISO_8859_13", "ISO_8859_14", "ISO_8859_15", "ISO_8859_16", "UTF16LE", "UTF16BE", "Default"	
compression	List of possible values: "None" (used as default), "Deflate", "GZip", "ZLib"	Sets c
displaytext	text string	Sets th humar Note: able te (only i The cu tactive
dm_enforcebinaryencoding	either "true" or "false	Deterr mode Note: bols th
dm_format	List of possible values: "Default" (default value) or "UC-CEAN", "Industry", "Macro05", "Macro06", "Reader" and "Post-Matrix"	Specif Specif encod
dm_rectangular	either "true" or "false"	Switch

Parameter	Value	Descri
		Per de Chang
dm_size	List of possible values: "Default" (used as default), "10x10", "12x12", "14x14", "16x16", "18x18", "20x20", "22x22", "24x24", "26x26", "32x32", "36x36", "40x40", "44x44", "48x48", "52x52", "64x64", "72x72", "80x80", "88x88", "96x96", "104x104", "120x120", "132x132", "144x144", "8x18", "8x32", "12x26", "12x36", "16x36" or "16x48"	Sets D Specif colum square select autom
dotcode_enforcebinaryencoding	either "true" or "false"	Deterr mode Note: bols th
dotcode_format_format	List of possible values: "Auto", "Generic", "GS1", "Industry", "Macro05", "Macro06", "Macro12", "MacroCustom", "Reader"	Specif Specif
dotcode_format_specifier	text string	Forma letter
dotcode_mask	List of possible values: "Default" (used as default), "0", "1", "2" and "3"	Sets Q Specif to ach is com - if the mask.
dotcode_printdirection	List of possible values: "DontCare" (used as default), "OptimizeHorizontal" and "OptimizeVertical"	Sets th optim

Parameter	Value	Descri
		Specif care" i
dotcode_size_mode	List of possible values: "Default", "RatioWidthHeight", "Fixed-Width", "FixedHeight"	Specif Specif fault".
dotcode_size_size	text string	Specif Depen should "Defau "Ratio the rat are va "Fixed width "Fixed height
encodingmode	List of possible values: "CodePage" (used as default), "Low-Byte", "ByteStream", "BYTE_HILO", "Hexadecimal"	Sets th The da data. D gle cha "enco Manua
format	text string	Sets fo

Parameter	Value	Descri
		The fo encod eratio switch Please
hanxin_eclevel	List of possible values: "L1" (used as default), "L2", "L3" and "L4"	Sets H Sets H correc
hanxin_enforcebinaryencoding	either "true" or "false"	Deterr mode Note: bols th
hanxin_mask	List of possible values: "Default" (used as default), "0", "1", "2" and "3"	Sets H Specif Code t mask i sumin define
hanxin_version	List of possible values: "Default" (used as default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81",	Specif Specif numb sions" the sy

Parameter	Value	Descri
	"82", "83" and "84"	
hres	number	Sets u Sets o lation able d resolu Note: bar co
maxi_mode	number	Specif Specif Mode Mode Code Mode (Posta Mode quenc Mode rectio
maxi_scm_countrycode	text string	Count (SCM). Set Ma

Parameter	Value	Descri
		For a c Barcod
maxi_scm_postalcode	text string	The Po tured other Set Ma For a c Barcod
maxi_scm_serviceclass	text string	Servic the mo Set Ma For a c Barcod
maxi_undercut	number	Under Specif This va (which value, cur.

Parameter	Value	Descri
		Note:
maxi_usepreamble_date	text string	Specif - the c cally i encod Note: true! Activa Specif coded vant i additi param
maxi_usepreamble_use	either "true" or "false"	Use pr Activa Specif coded vant i additi param
modulewidth	number with unit, allowed units are "mm", "cm", "m", "in", "ft", "pt" and "pc"	Sets a of 0.0

Parameter	Value	Descri
		The m ment (specifi ly (def object encod fore of fluenc the an symbo mm.
mpdf417mode	List of possible values: "Default" (used as default), "EAN128", "C128Std", "C128FNC2", "EAN128Lk", "05Macro", "06Macro", "CCA", "CCB" and "Binary"	Sets M This o most c scann check input mode smalle
mpdf417version	List of possible values:, "Default", "1x11", "1x14", "1x17", "1x20", "1x24", "1x28", "2x8", "2x11", "2x14", "2x17", "2x20", "2x23", "2x26", "3x6", "3x8", "3x10", "3x12", "3x15", "3x20", "3x26", "3x32", "3x38", "3x44", "4x4", "4x6", "4x8", "4x10", "4x12", "4x15", "4x20", "4x26", "4x32", "4x38" and "4x44"	Sets M The M and da the sy "4x44" set exp alyzing matica
mqr_mask	List of possible values: "Default" (used as default), "0", "1", "2", "3" and "4"	Sets M Specif Code t mask i the en

Parameter	Value	Descri
		mask.
mqr_version	List of possible values: "Default" (used as default), "0", "1", "2", "3" and "4"	Specif Specif numb sions" the sy
notchheight	number with unit, allowed units are "mm", "cm", "m", "in", "ft", "pt" and "pc"	Sets th es). The te some a little these cally. bars. Note: lation. exceed Thoug ed to
options	text string	Sets th The op lowed charac The fo A nam be giv

Parameter	Value	Descri
		As Sim Simpl caped are tru e.g.: N NAME_ As Tex Text va tain sp that o e.g.: N NAME_ As Co Comp plex va ues. e.g.: N BER=1 The fo DRAW_ PostSc DRAW_ mode CHECK BCSet CHECK

Parameter	Value	Descri
		wheth check DATA_ cape s DATA_ data (s MQR_ BCSet_ MQR_ Set_M QrCod Set_Q QrCod Set_Q QrCod applic QrCod level (s QrCod Set_Q QrCod compa e_QRM QrCod the QR DataM Set_D DataM bol (se DataM

Parameter	Value	Descri
		BCSet_ DataM the Da DataM coding DataM native DataM scanne HanXi Set_H HanXi encod HanXi el (see HanXi Set_H DotCo Size (s DotCo tion (s tion). DotCo encod DotCo DotCo DotCo Set_D EanUp (see B

Parameter	Value	Descri
		EanUp (see Fo EanUp out inc Valid v [bool] [enum accord [string [int] (i [byte]
pdf417_addressee	text string	Sets M Specif Option er bloc
pdf417_checksum	number	Sets M Sets 1 x12 + x Option er bloc
pdf417_columns	number	Sets th

Parameter	Value	Descri
		Sets th fixed v tomat
pdf417_eclevel	number	Sets P Set th Possib tion (r error c
pdf417_encodingmode	List of possible values: "Default" (used as default) and "Binary"	Sets P Sets th the ge binary the be
pdf417_fileid	text string	Sets M The Fi data c File-ID index. Macro File-ID
pdf417_filename	text string	Sets M Specif

Parameter	Value	Descri
		Optior er bloc
pdf417_filesize	number	Sets M Specif length Optior er bloc
pdf417_rowcolratio	text string	Sets P Sets th Does c "pdf41 row:co
pdf417_rowheight	number	Sets P Sets th mm]. I
pdf417_rows	number	Sets n Sets th not se the de
pdf417_segcount	number	Sets M Specif

Parameter	Value	Descri
		Optior er bloc
pdf417_segindex	number	Sets M Index with 1 chain. Macro File ID PDF41 This m symbo The sc contro struct functi Note: ner, th
pdf417_seglast	either "true" or "false"	Marks PDF41 Marks symbo PDF41
pdf417_sender	text string	Sets M

Parameter	Value	Descri
		Specif Optior er bloc
pdf417_timestamp	number	Sets M Specif 1, 197 Optior er bloc
qr_eclevel	List of possible values:, "Low" (used as default), "Medium", "Quartil" and "High"	Sets Q Sets Q error c
qr_fmtappindicator	text string	Set QR try" fo Is used cation put str
qr_format	List of possible values: "Default" (default value) or "UCCEAN" or "Industry"	Specif Specif encod
qr_kanjichinesecompaction	List of possible values: "Default" (used as default), "None", "Kanji" and "Chinese"	Enable tion.

Parameter	Value	Descri
		Enable ters in Unico page s mode JIS X 0 Chines acters 97-001 to GB/ Note: charac ent de
qr_mask	List of possible values: "Default" (used as default), "0", "1", "2", "3", "4", "5", "6" and "7"	Sets Q Specif to ach is com the en mask.
qr_version	List of possible values:, "Default" (used as default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39" and "40"	Specif Specif of diffe no fixe bol is
quietzonebottom	number	Sets th Specif units. left of

Parameter	Value	Descri
		Note: the qu "narro
quietzoneleft	number	Sets th units. Specif units. left of Note: the qu "narro
quietzoneright	number	Sets th units. Specif units. left of Note: the qu "narro
quietzonetop	number	Sets th Specif units. left of

Parameter	Value	Descri
		Note: the qu "narro
quietzoneunit	List of possible values: "X", "mm", "cm", "m", "in", "ft", "pt" and "pc" ("X" is the current "modulewidth")	The ur
ratio	text string	Set pri Specif the sir "1B:2E most r space. values It is ex
rotation	List of possible values: "0" (used as default), "90", "180", "270"	Sets th Rotate
rss_segmperrow	number	Sets d a fixed The va words Values encod ues ar numb
swap_foreground_background	either "true" or "false" (used as default)	Swaps

Parameter	Value	Descri
		Using (or the the ba tween white. age or
textalignment	List of possible values: "Default" (used as default), "Left", "Right", "Center"	Sets th the hu The de Note: ing "a
textdistance	number with unit, allowed units are "mm", "cm", "m", "in", "ft", "pt" and "pc"	Sets th barcod The de be ove Note: text bu
textplacement	List of possible values: "below" (used as default), "none" and "above"	Define has te
vres	number	Sets u Sets o lation able d resolu

Parameter	Value	Descri
		Note: T bar co

5.3 How to define the size of barcode objects

Barcodes and matrix codes are created through "barcode objects", which are a pdfChip specific custom type of <object>.

Such objects can be used in HYML content, inside SVG objects and inside SVG files (as of pdfChip 1.2, SVG files can be converted directly to PDF, without having to embed them in HTML).

Barcode object example

An example for the syntax of a barcode object is shown below:

```
<object type="application/barcode">
    <param name="data" value="123456789012">
    <param name="type" value="EAN 13">
    <param name="modulewidth" value="0.33mm">
    <param name="barwidthreduction" value="10%">
    <param name="textplacement" value="none">
</object>
```

Defining the size of a barcode or matrix code

In many usage scenarios, the exact size of a barcode or matrix code is important. There are several ways to define this size:

- by providing a value for the "modulewidth" parameter (i.e. the width of smallest element in a barcode or the width or height of the smallest element in a matrix code)
- by providing "width" and "height" attributes in the <object> tag

The "width" and "height" attributes in the <object> tag values take precedence over the "modulewidth" parameter. Where it is necessary to define a certain for a barcode or matrix code but still let the "modulewidth" drive the actual size of the barcode or matrix code, "width" and "height" attributes should be present for the parent of the <object> tag.

Using "modulewidth" to define the size of an EAN code

```
        <div>
                <object type="application/barcode" style="background-color: #eff;">
                        <param name="data" value="123456789012">
                        <param name="type" value="EAN 13">
                        <param name="modulewidth" value="0.33mm">
                        <param name="barwidthreduction" value="10%">
                        <param name="quietzoneleft" value="10">
                        <param name="quietzoneright" value="10">
                        <param name="quietzoneunit" val-
ue="X">
                </object>
        </div>
```

Using "width" style attribute to define the size of an EAN code

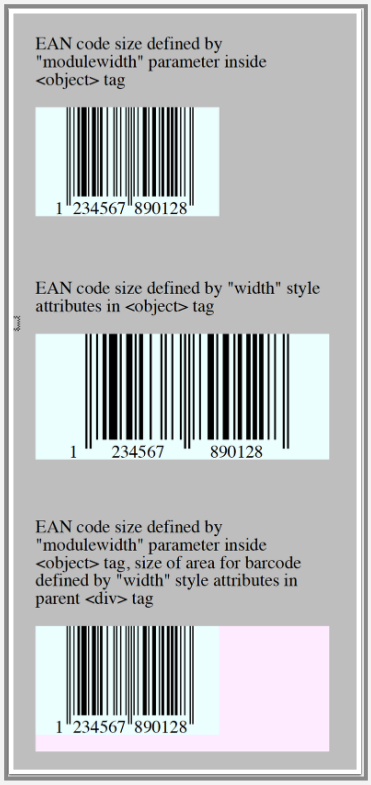
```
        <div style="width: 50mm; height: 30mm; background-color: #fef">
                <object type="application/barcode" style="width: 70mm; height:
30mm; background-color: #eff;">
                        <param name="data" value="123456789012">
                        <param name="type" value="EAN 13">
                        <param name="modulewidth" value="0.33mm">
                        <param name="barwidthreduction" value="10%">
                        <param name="quietzoneleft" value="10">
                        <param name="quietzoneright" value="10">
                        <param name="quietzoneunit" val-
ue="X">
                </object>
        </div>
```

Using "modulewidth" to define the size of an EAN code, while creating an area for the EAN code by setting the size of the parent

```
        <div style="width: 70mm; height: 30mm; background-color: #fef">
```

```
                <object type="application/barcode" style="background-color: #eff;">
                        <param name="data" value="123456789012">
                        <param name="type" value="EAN 13">
                        <param name="modulewidth" value="0.33mm">
                        <param name="barwidthreduction" value="10%">
                        <param name="quietzoneleft" value="10">
                        <param name="quietzoneright" value="10">
                        <param name="quietzoneunit" val-
ue="X">
                </object>
        </div>
```

The example below shows the effect of the methods described above. An HTML file with the examples is attached below the figure.



defining_size_of_barcode.html

5.4 How to define barcode objects in HTML and SVG

Barcodes and matrix codes are created through "barcode objects", which are a pdfChip specific custom type of <object>.

Such objects can be used in HTML content, inside SVG objects and inside SVG files (as of pdfChip 1.2, SVG files can be converted directly to PDF, without having to embed them in HTML).

Please note that when viewing an HTML file or an SVG file which contain a pdfChip barcode object, the browser will typically show a "missing plug-in" sign where the barcode or matrix code would be present, as pdfChip barcode objects are only supported for processing by pdfChip.

Defining a barcode object in HTML

An example for the syntax of a barcode object inside HTML is shown below.

Please note that the background-color and color attributes are used for demonstration purposes only. Normally barcodes and matrix codes would be printed in black (or a very dark color) on white (or a very light background).

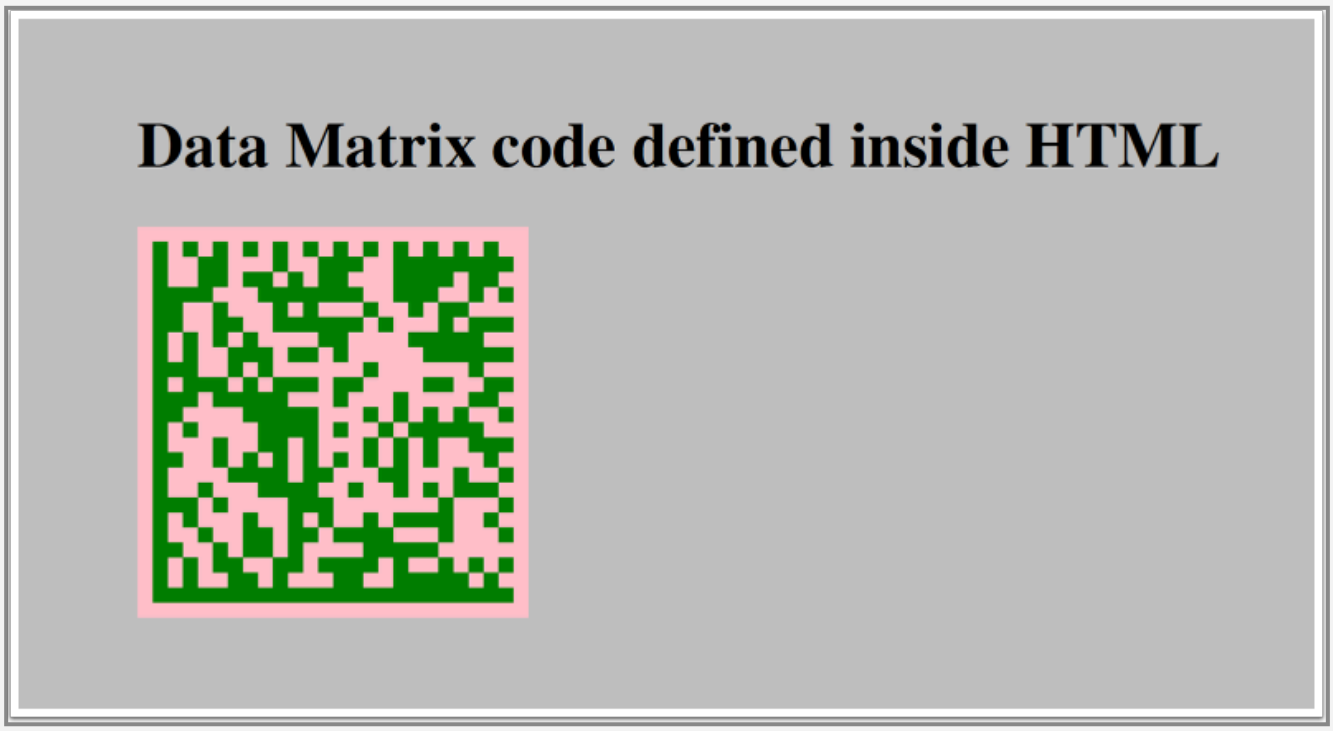
Example

The example code below – which can also be found in the downladable HTML file "Data_Matrix_code_defined_inside_HTML.html" – creates a DataMatrix code by means of a barcode object inside HTML. The output is also shown below (the background color "pink" is only used for demonstration purposes. Normally using a white background is preferred).

```
<object xmlns="http://www.w3.org/1999/xhtml"
                type="application/barcode"
                style="margin: 0; padding: 0; background-color:pink; color:
green;">
        <param name="type" value="Data Matrix">
        <param id="data" name="data" value="This Data Matrix code was created
```

```
through a barcode object inside HTML">
        <param name="modulewidth" value="2mm">
        <param name="quietzoneleft" value="1">
        <param name="quietzoneright" value="1">
        <param name="quietzonetop" value="1">
        <param name="quietzonebottom" value="1">
        <param name="quietzoneunit" value="X">
</object>
```

Data_Matrix_code_defined_inside_HTML.html



Defining a barcode object inside an SVG object

An example for the syntax of a barcode object inside an SVG object inside HTML is shown below.

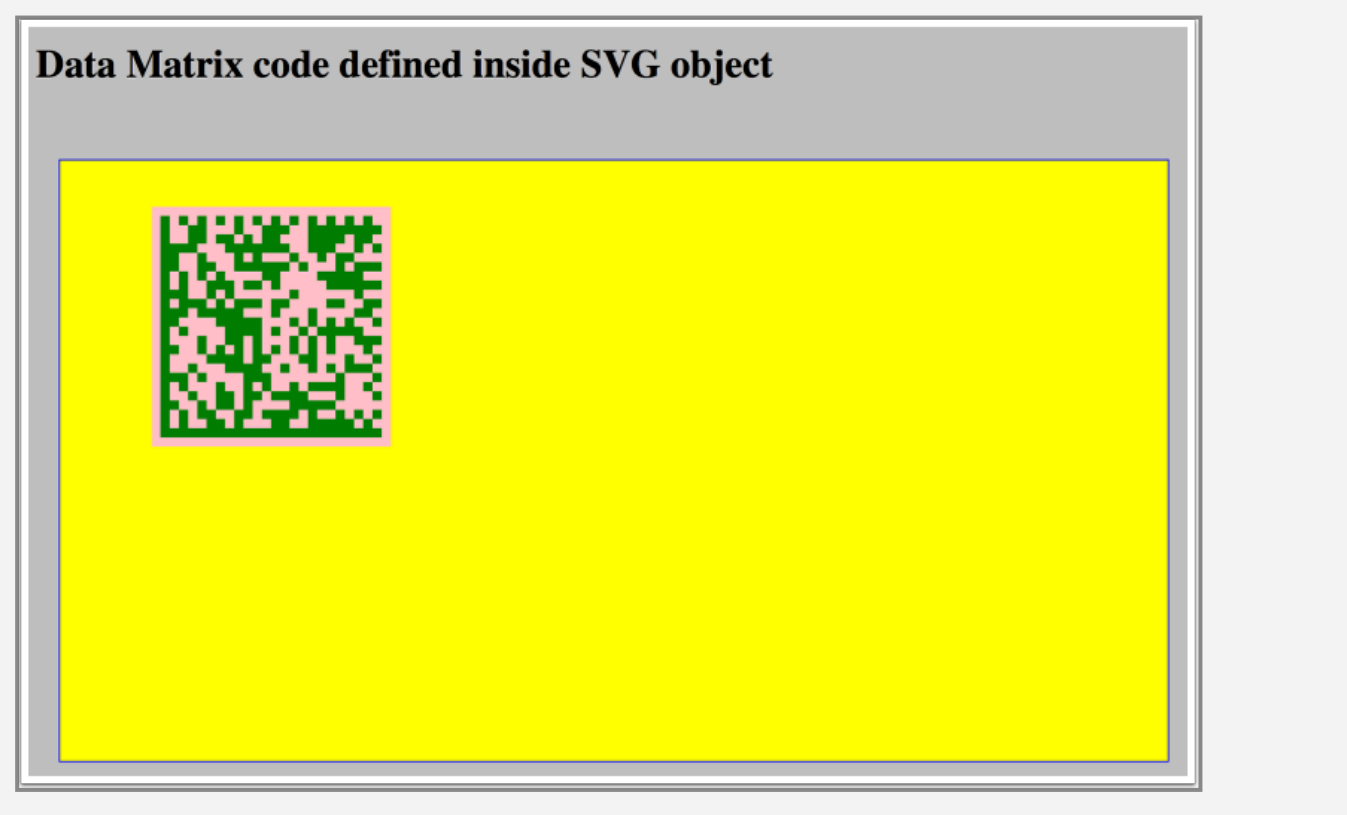
Please note that the background-color and color attributes are used for demonstration purposes only. Normally barcodes and matrix codes would be printed in black (or a very dark color) on white (or a very light background).

Example

The example code below – which can also be found in the downladable HTML file "Data_Matrix_code_defined_inside_SVG_object.html" – creates a DataMatrix code by means of a barcode object inside SVG which in turn is contained as an SVG object inside HTML. The output is also shown below (the background color "pink" is only used for demonstration purposes. Normally using a white background is preferred).

```
<h1>Data Matrix code defined inside SVG object</h1>
<svg width="250mm" height="150mm" xmlns="http://www.w3.org/2000/svg">
  <rect x="5mm" y="10mm" width="240mm" height="130mm" stroke="blue" fill="yellow">
  </rect>
  <foreignObject x="25mm" y="20mm" width="120mm" height="120mm">
        <object xmlns="http://www.w3.org/1999/xhtml"
                type="application/barcode"
                style="margin: 0; padding: 0; background-color:pink; color:
green;">
                        <param name="type" value="Data Matrix">
                        <param id="data" name="data" value="This Data Matrix code
was created through SVG">
                        <param name="modulewidth" value="2mm">
                        <param name="quietzoneleft" value="1">
                        <param name="quietzoneright" value="1">
                        <param name="quietzonetop" value="1">
                        <param name="quietzonebottom" value="1">
                        <param name="quietzoneunit" value="X">
        </object>
  </foreignObject>
</svg>
```

Data_Matrix_code_defined_inside_SVG_object.html



Defining a barcode object inside an SVG file

An example for the syntax of a barcode object inside an SVG file – which can be processed directly by pdfChip – is shown below.

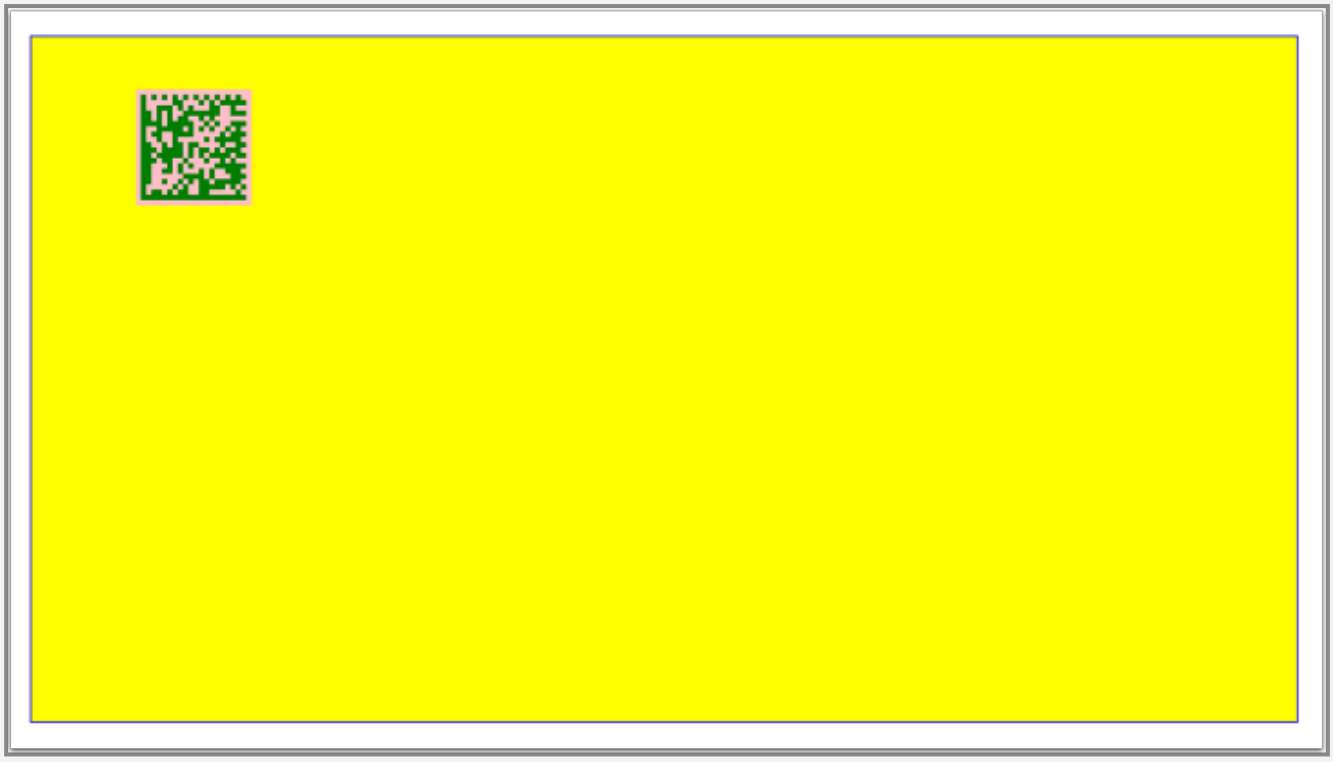
Please note that the background-color and color attributes are used for demonstration purposes only. Normally barcodes and matrix codes would be printed in black (or a very dark color) on white (or a very light background).

Example

The example code below – which can also be found in the downladable SVG file "Data_Matrix_code_defined_inside_SVG_file.svg" – creates a DataMatrix code by means of a barcode object inside an SVG file. The output is also shown below (the background color "pink" is only used for demonstration purposes. Normally using a white background is preferred).

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     width="250mm" height="150mm">
        <style type="text/css">
                @page {
                        /* Size of the page, equivalent to MediaBox, origin always
at 0/0 */
                        size: 307mm 310mm; /* slightly larger than target format */
                        -cchip-cropbox: 0mm 100mm 297mm 210mm;
                        /* crops @page-size (MediaBox) to the actually
                        intended page size for display and printing */
                }
        </style>
        <defs>        </defs>
        <rect x="5mm" y="10mm" width="240mm" height="130mm" stroke="blue"
fill="yellow">
        </rect>
        <foreignObject x="25mm" y="20mm" width="80mm" height="60mm">
                <object xmlns="http://www.w3.org/1999/xhtml"
                                type="application/barcode"
                                style="margin: 0; padding: 0; background-col-
or:pink; color: green;">
                        <param name="type" value="Data Matrix">
</param>
                        <param name="data" value="Created from SVG file."> </param>
                        <param name="modulewidth" value="1mm">
</param>
                        <param name="quietzoneleft" value="1">
</param>
                        <param name="quietzoneright" value="1">
</param>
                        <param name="quietzonetop" value="1">
        </param>
                        <param name="quietzonebottom" value="1">
</param>
                        <param name="quietzoneunit" value="X">
        </param>
                </object>
        </foreignObject>
</svg>
```

Data_Matrix_code_defined_inside_SVG_file_v2.svg



5.5 How to create and update barcode objects dynamically

As barcode objects are custom objects implemented by pdfChip, and are not regular parts of the DOM, when dynamically creating or updating barcode objects, it is necessary to force an update of a barcode object after it has been created or modified. The method to enforce such updating is surprisingly simple: the (modified) barcode is virtually assigned to itself...

The example illustrated in this article explains how this can be achieved, by means of a simple HTML file, with some JavaScript included in its <head> tag, that creates several pages, each with on QR-Code on it, and the text that has been encoded in the QR-Code in plain text below the QR-Code.

```
        <body style="padding: 20mm; ">
                <p>Dynamically creating/updating a QR-Code <br/>(requires use of
cchipPrintLoop() / JavaScript)<p>
                <object id='id_barcode' type='application/barcode'>
                        <param name='type' value='QR-Code'/>
                        <!--  the 'data' entry wil be dynamically populated by the
JavaScript above -->
                        <param id='id_barcodedata' name='data' value='Hello
World!' />
                        <param name='modulewidth' value='2mm'>
                        <param name="quietzoneleft" value="1">
                        <param name="quietzoneright" value="1">
                        <param name="quietzonetop" value="1">
                        <param name="quietzonebottom" value="1">
                        <param name="quietzoneunit" value="X">
                </object>
                <p id="id_barcodecontent" style="color: #888">placeholder for the
data encoded in the QR-Code</p>
        </body>
```

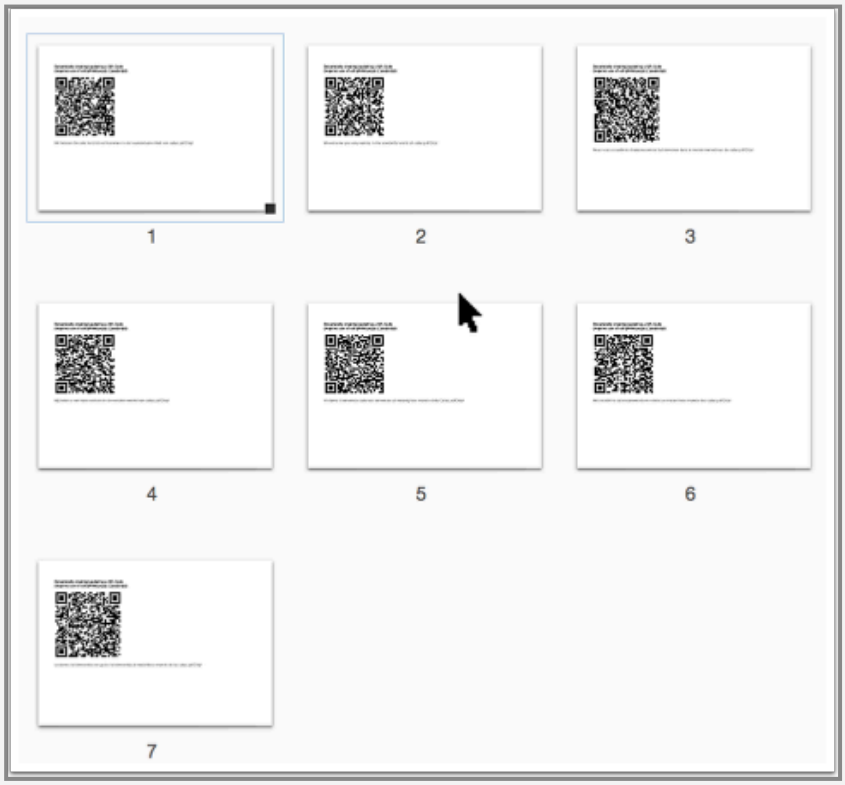
Two parts of the content shown above (in dark blue bolded text) are filled repeatedly by the JavaScript below.

- first, the parameter "daat" inside the barcode object (identified by "id_barcodedata") is set to the new value (taken from an array of values)
- next, an update is forced by assigning the modified set of parameters to the barcode object (essentially simply by setting "b.innerHTML = b.innerHTML")
- last but not least, the text below the QR-Code, identified by "id_barcodecontent" is also modified; a forced update is not necessary, as the <p> tag and its content are regular parts of the DOM – any changes to it take effect immediately.

```
                <script>
                        // the data variable contains the entries to be used for
dynamically populating teh QR-Codes
                        var data =
                        [
                                  { data: "Wir heissen Sie sehr herzlich willkom-
men in der wunderbaren Welt von callas pdfChip!" }
                                , { data: "We welcome you very warmly in the won-
derful world of callas pdfChip!" }
                                , { data: "Nous vous accueillons chaleureusement
la bienvenue dans le monde merveilleux de callas pdfChip!" }
                                , { data: "Wij heten u van harte welkom in de won-
dere wereld van callas pdfChip!" }
                                , { data: "Vi diamo il benvenuto caloroso benvenu-
to al meraviglioso mondo della Callas pdfChip!"}
                                , { data: "Nós recebê-lo calorosamente bem-vindos
ao maravilhoso mundo dos callas pdfChip!"}
                                , { data: "Le damos la bienvenida con gusto la bi-
envenida al maravilloso mundo de las calas pdfChip!"}
                        ];
                        // - sets new value for the QR Code data
                        // - forces an update by inserting  the parameter entries -
 i.e. the children
                        //   of the barcode object - into the barcode object itself
                        function setBarcode(d) {
                                document.getElementById('id_barcodedata').setAt-
tribute('value',d.data);
                                //force update for parameters of the barcode ob-
ject:
                                var b = document.getElementById('id_barcode');
                                b.innerHTML = b.innerHTML;
                                // also set the text below the QR-Code to the same
```

```
value
                                var p  = document.getElementById('id_barcodecon-
tent');
                                p.innerHTML = d.data;
                        }

                        function printRemainingBarcodes(i) {
                                cchip.printPages(1);

                                if (++i < data.length) {
                                        setBarcode(data[i]);
                                        cchip.onPrintReady( () => { printRemaining-
Barcodes(i); } );
                                }
                        }

                        // cchipPrintLoop() is a pdfChip specific function that
makes it possible to dynamically generate pages
                        function cchipPrintLoop(){
                                setBarcode(data[0]);
                                cchip.onPrintReady( () => { printRemainingBar-
codes(0); } );
                        }
                </script>
```

Seven pages are created by the sample HTML file provided at the bottom of this article:



The contents of one of the generated pages / QR-Codes:

An example HTML file is provided below for download:

update_barcode_objects_dynamically.html.zip

5.6 Using gradient or image as "color" for QR code

Using the "swap_foreground_background" parameter for a barcode object, it is possible to apply arbitrary visual content as the "color" for a barcode or matrix code. "swap_foreground_background" essentiallly swaps the foreground (CSS "color" property) and the background (CSS "background" property) color. While in CSS for the foreground color only regular or custom pdfChip color definitions can be used, the possibilities for "background" are much richer. Here it is possible to define gradients or images.

The following is just one example of taking advantage of the "swap_foreground_background" parameter. The HTML source file can be found for download at the bottom of this article.

Regular "black on white" QR Code

```
<object class="barcode_object"
                type="application/barcode"
                style = "
                        color: black;
                        background-color: #ddd;
                "
>
        <param name="type" value="QR-Code">
        <param name="modulewidth" value="1mm">
        <param name="data" value="www.callassoftware.com">
        <param name="quietzoneleft" value="1">
        <param name="quietzoneright" value="1">
        <param name="quietzonetop" value="1">
        <param name="quietzonebottom" value="1">
        <param name="quietzoneunit" value="X">
</object>
```

Defining a gradient as the "color" for a QR Code

```
        <p>
                <object class="barcode_object"
                                type="application/barcode"
                                style = "
                                        color: #ddd;
                                        background: linear-gradient(135deg, fire-
brick,
                                                red, orange, orange, green, blue,
indigo, violet);
                                "
                >
                        <param name="type" value="QR-Code">
                        <param name="modulewidth" value="1mm">
                        <param name="data" value="www.callassoftware.com">
                        <param name="swap_foreground_background" val-
ue="true">                        <param name="quietzoneleft" value="1">
                        <param name="quietzoneright" value="1">
                        <param name="quietzonetop" value="1">
                        <param name="quietzonebottom" value="1">
                        <param name="quietzoneunit" value="X">
                </object>
        </p>
```

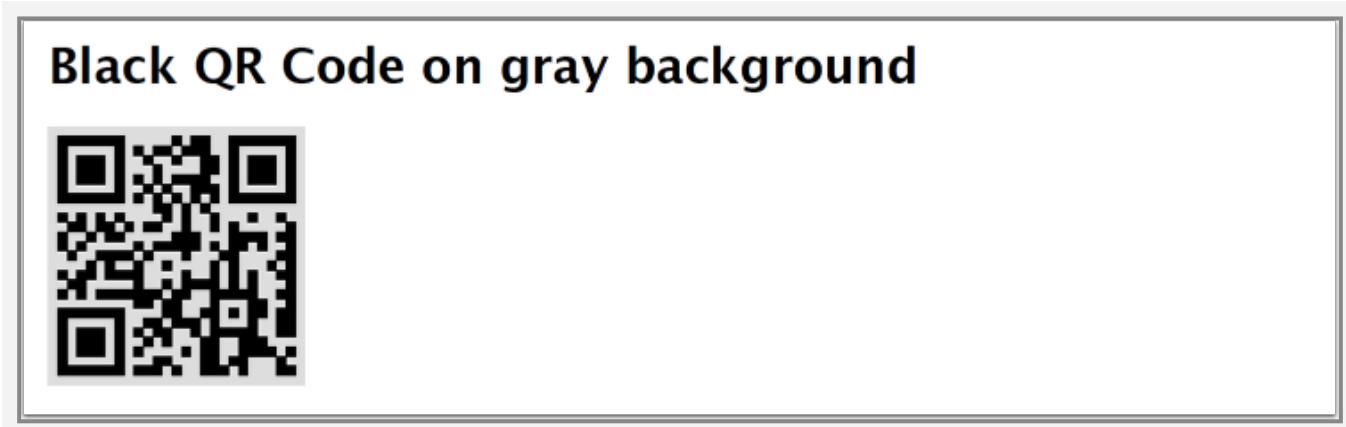
Defining an image as the "color" for a QR Code

```
                <object class="barcode_object"
                                type="application/barcode"
                                style = "
                                        color: #ddd;
                                        background: url('img/wood.jpg') ;
                                        background-size: 96pt 96pt;
                                "
                >
                        <param name="type" value="QR-Code">
                        <param name="modulewidth" value="1mm">
                        <param name="data" value="www.callassoftware.com">
                        <param name="swap_foreground_background" value="true">
```

```
                <param name="quietzoneleft" value="1">
                <param name="quietzoneright" value="1">
                <param name="quietzonetop" value="1">
                <param name="quietzonebottom" value="1">
                <param name="quietzoneunit" value="X">
        </object>
```

Defining an image based pattern as the "color" for cells in a QR Code

```
        <object class="barcode_object"
                type="application/barcode"
                style = "
                        color: #ddd;
                        background: url('img/icon_pdfchip.svg') ;
                        background-size: 6pt 6pt;
                        background-repeat: repeat;
                "
        >
                <param name="type" value="QR-Code">
                <param name="modulewidth" value="6pt">
                <param name="data" value="www.callassoftware.com">
                <param name="swap_foreground_background" value="true">
                <param name="quietzoneleft" value="1">
                <param name="quietzoneright" value="1">
                <param name="quietzonetop" value="1">
                <param name="quietzonebottom" value="1">
                <param name="quietzoneunit" value="X">
        </object>
```

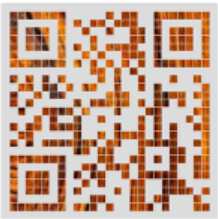


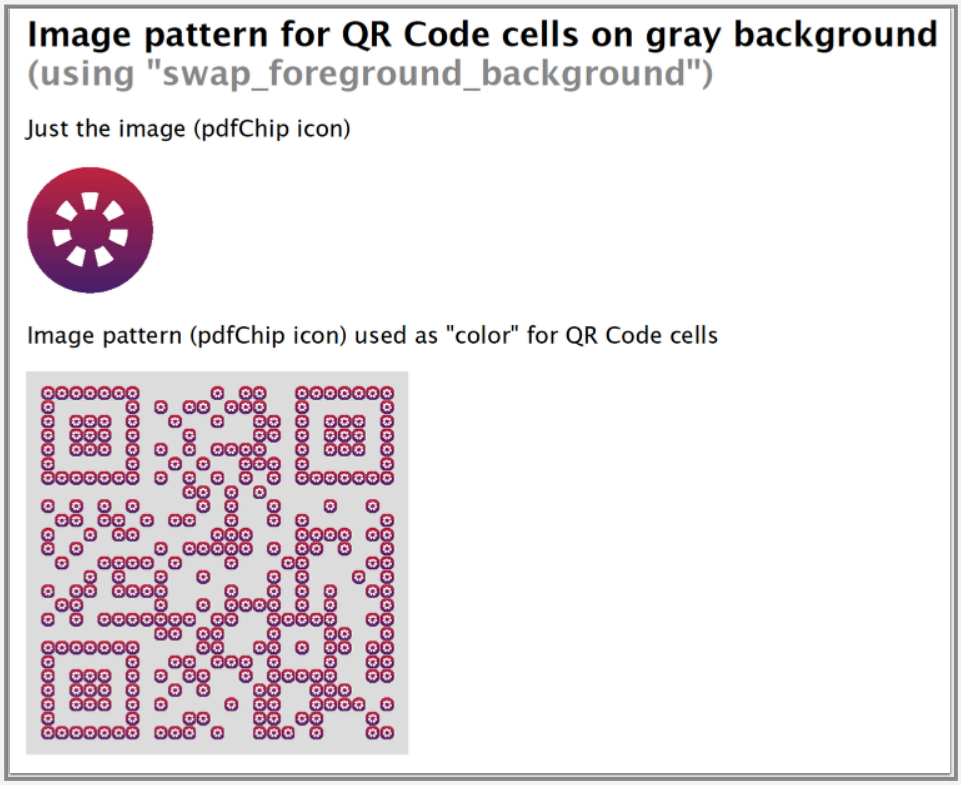
Gradient colored QR Code on gray background (using "swap_foreground_background")

"Image colored" QR Code on gray background (using "swap_foreground_background")

Just the image (wooden surface)

Image (wooden surface) used as "color" for QR Code





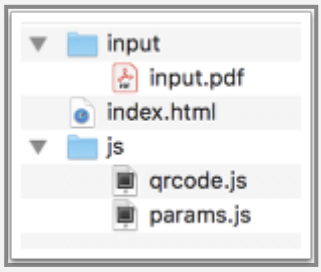
For the HTML source code please download the ZIP file below:

Using_gradient_or_image_as_"color"_for_QR_co-2.zip

5.7 Adjust page size for result PDF to size of placed PDF

Sometimes the dimensions of the result PDF have to be adjusted to an object. This, for instance, is required when the purpose is to create variants from an input PDF where each of the variants has some additional content. In this article we describe how to create variants from an incoming PDF where each variant has an individual QR code. The trick is how to adjust the size of the result PDF variants to the size of the incoming PDF.

The example archive has just 4 files:



- **input.pdf** is the placed PDF and can be replaced with any other PDF
- **index.html** is a very small HTML template
- **params.js** defines some static parameters for the QR code and the total number of pages
- **qrcode.js** has the logic that adjusts the page size, places the QR codes and creates pages

Adjust_size_to_placed_PDF.zip

index.html

The HTML template is very simple.

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8"/>
    <title>QRcode variants</title>
    <script src="js/params.js" type="text/javascript"> </script>
    <script src="js/qrcode.js" type="text/javascript"> </script>
  </head>
  <body>
    <img id="inputPDF" src="./input/input.pdf">
    <div id="QR_Code" style="position: absolute; left: 10mm; top: 10mm"></div>
  </body>
</html>
```

1. Two JavaScript files are referenced
2. The PDF file is placed
3. A DIV container is created for the QR code

params.js

```
var params = {
    'jobID': '01234567890123456789',
    'customerID': '01234567890123456789',
    'addvalues': '00',
    'numberofpages': 100
    }
```

The params.js JavaScript just has a few parameters for the QR code. It can be overwritten for every job if needed and that is the reason why it is separate from the other JavaScript.

1. jobID, customerID and addvalues are static values that are the same for all pages of the result PDF
2. numberofpages defines the required number of different page variants form the input PDF

qrcode.js

This JavaScript file has all logic of this example. It consists of two functions.

```
function adjustDocumentSizeToHtmlElement( inElementID ) {

    // Get the width and height of the element in points
1   var obj = document.getElementById(inElementID);
    var theElementWidth = obj.offsetWidth * 0.75;
    var theElementHeight = obj.offsetHeight * 0.75;
    // Create style element for page size also adding a margin and use the pdfChip cropbox feature
2   var theMargin = 500;
    var style = document.createElement('style');
    style.innerHTML = '@page {size: ' + (theElementWidth + theMargin) + 'pt ' + (theElementHeight
3       + theMargin) + 'pt; ' + '-cchip-cropbox: 0 ' + theMargin + 'pt ' + theElementWidth + 'pt '
        +theElementHeight + 'pt;' + '}';
    document.head.appendChild(style);
}
```

The first function "adjustDocumentSizeToHtmlElement" adjusts the size of the HTML template to the size of the placed PDF

1. The placed PDF was contained in an img tag with the id "inputPDF". This will be used when adjustDocumentSizeToHtmlElement is called as the "inElementID".
   The dimensions of this object are determined and written into two varialbes.
2. A margin is defined to use --cchip-cropbox to make the result page 500pt bigger than the required size and to then crop it to that required size.
3. A style element for the page size @page is created and appended and the required size is written into this @page element. In addition the --cchip-cropbox is defined.

```
function cchipPrintLoop() {
1   adjustDocumentSizeToHtmlElement("inputPDF");
2   for (var i = 0; i < params.numberofpages; i++) {
        // Build the QR value
3       var pageID = i;
        var QRValue = params.jobID + params.customerID + pageID + params.addvalues;
        // Create the QR object in HTML
4       document.getElementById("QR_Code").innerHTML = '<object class="qrcode"
            type="application/barcode" style="width:30mm; height:30mm; background-color:#FFFFFF;
            background-color:-cchip-cmyk(0,0,0,0); color:#000000; color:-cchip-cmyk(0,0,0,1)"> <
            param name="type" value="QR-Code"> <param name="data" value="' + QRValue + '"><param
            name="modulewidth" value="1mm"> <param name="qr_version" value="4"></object>';
        // Create one page instance
5       cchip.printPages();

};
```

The second function is the cchipPrintLoop function that allows for creating pages. In this function:

1. The first function adjustDocumentSizeToHtmlElement is called with "inputPDF" as parameter (that defines the object from which the page size has to be derived as described above).
2. A loop is started for the number of required page variants
3. The QR code value is concatenated from the static values in params.js and the counter that is individual for each page variant. In this example the counter is just made part of the QR code value.
4. The QR code object is created and written into the HTML object with id "QR_Code".
5. A page is created via cchip.printPages.

5.8 How to create rectangular 16x48 DataMatrix Industry code

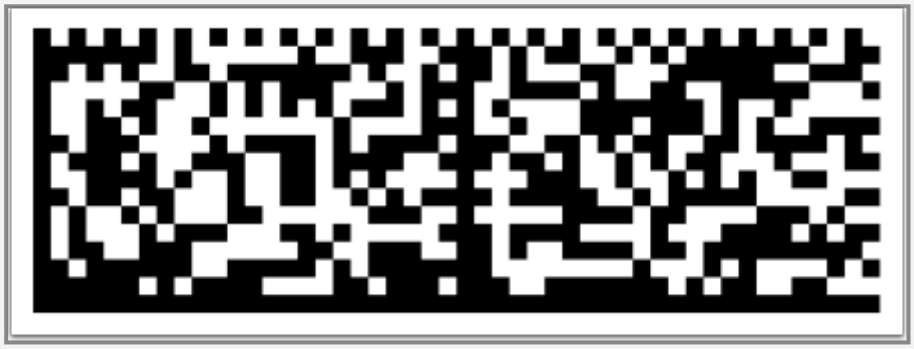
pdfChip supports over 100 symbologies (types of barcodes and matrix codes). In addition, a set of about 100 specific barcode object parameters – plus the extra list of special options in the "options" parameter – make it possible to create many more specialized subtypes of barcodes.

This article illustrates how to create a DataMatrix code of type "Industry", with a rectangular size of "16x48" cells.

The special parameters necessary to create a DataMatrix code of type "Industry" with a rectangular size of 16x48 cells are:

- **dm_format**: defines "Industry" as the format to be used here
- **dm_size**: sets the desired size, in this case 16 by 48 cells
- **dm_rectangular**: sets the form of the DataMatrix code to be rectangular

```
<object type="application/barcode" >
        <param name="type" value="Data Matrix">
        <param name="data" value="Put actual data here">
        <param name="modulewidth" value="0.25577mm">
        <param name="dm_format" value="Industry">
        <param name="dm_size" value="16x48">
        <param name="dm_rectangular" value="true">
</object>
```



Note: The actual xqsize in this case is very small (ca. 12mm x 4mm).

The attachment below contains an HTML source file for this rectangular DataMatrix Industry code.

DataMatrix_Industry__rectangular__16x48.html

5.9 How to create ITF-14 barcode with bearer bars

The ITF-14 barcode is one of the very few barcodes for which use of so called bearer bars are defined.

Normally, bearer bars are only created at the top and the bottom of a barcode. In cases where bearer bars are created on all four sides, it is important to provide space for the bearer bars to the left and to the right of the barcode in the form of a properly sized quiet zones. Using a value of zero for the quiet zone will lead to an error when creating a barcode.

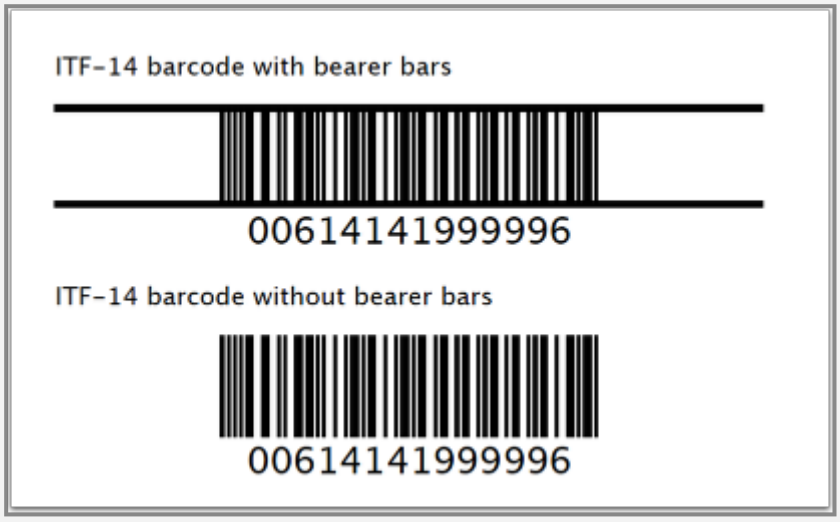
An HTML file with the code used for this example is available for download:

ITF-14_with_bearer_bars-1.html

Relevant HTML source code for the barcode object:

```
<object type="application/barcode" style="height: 0.5in">
        <param name="type" value="ITF 14 (GTIN 14)">
        <param name="modulewidth" value="0.254mm">
        <param name="data" id="id_barcodevalue" value="00614141999996">
        <param name="quietzoneleft" value="1">
        <param name="quietzoneright" value="1">
        <param name="quietzoneunit" value="in">
        <param name="bearerbars"  value="topbottom">
        <param name="bearerwidth"  value="0.01in">
</object>
```

PDF output from example file



6. Export InDesign files into HTML/CSS templates

6.1 Overview and installation

Beginning with version 1.2 pdfChip comes with an Adobe InDesign export filter that allows users to create their templates in InDesign. InDesign is the most commonly used layout tool with many features for the creation of print content.

It is easily possible to export an InDesign file to HTML with InDesign itself, however, this export will create rather big and unstructured files with various elements that are not required if the HTML is going to be converted to PDF. In addition it lacks of support for the pdfChip specific contents, e.g. of CMYK or ICCbased colors.

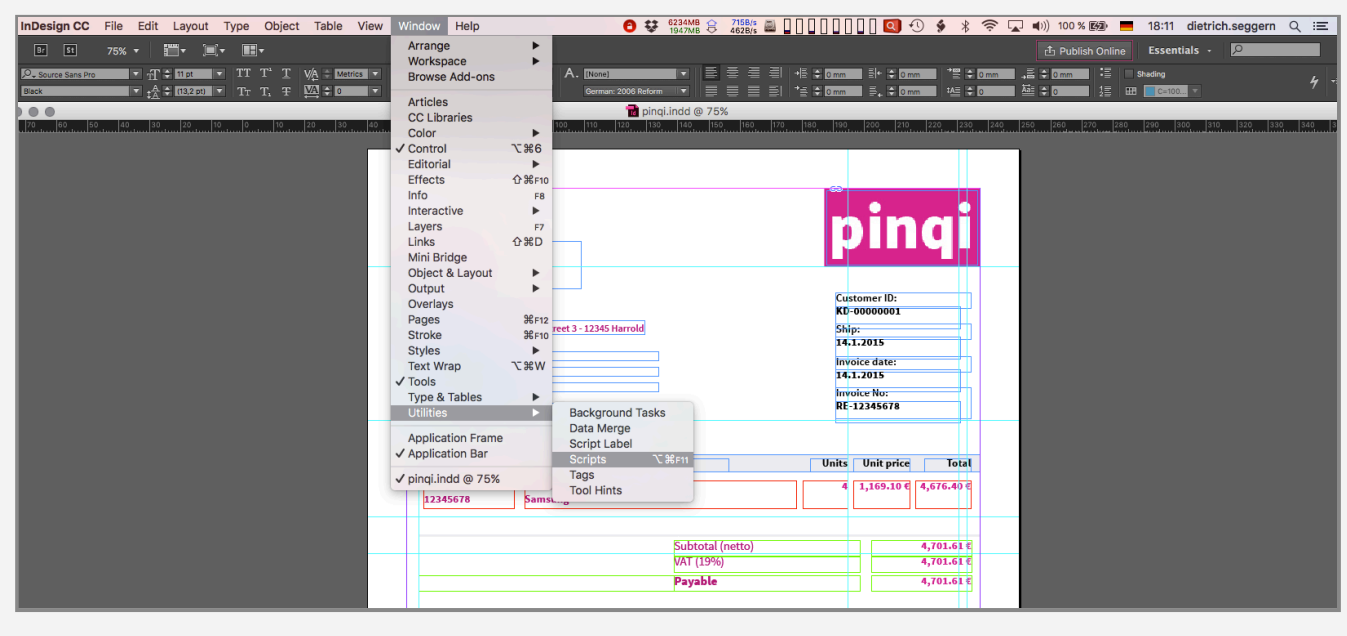
The export2pdfChip export filter will:

- export all fonts that are used in the InDesign file into the HTML template (TrueType and OpenType are supported),
- create CSS colors for all InDesign color swatches that are used in the InDesignFile (including CMYK); for color spaces that are not supported in HTML it will in addition create an alternate color that will be used by an HTML renderer, e.g. a web browser,
- create @page rule entries for the page geometry of the InDesign file including TrimBox and BleedBox, when the HTML is converted to PDF the result will have as many pages as the InDesign file
- will take over most important text formatting rules from Paragraph and Character Styles,
- create some basic vector objects from any such objects in the InDesign file.

The main purpose of the export filter is to make it possible to use the layout features of InDesign for the creation of HTML templates for pdfChip, the main purpose is not to create an as good as possible HTML rendition from any InDesign file.

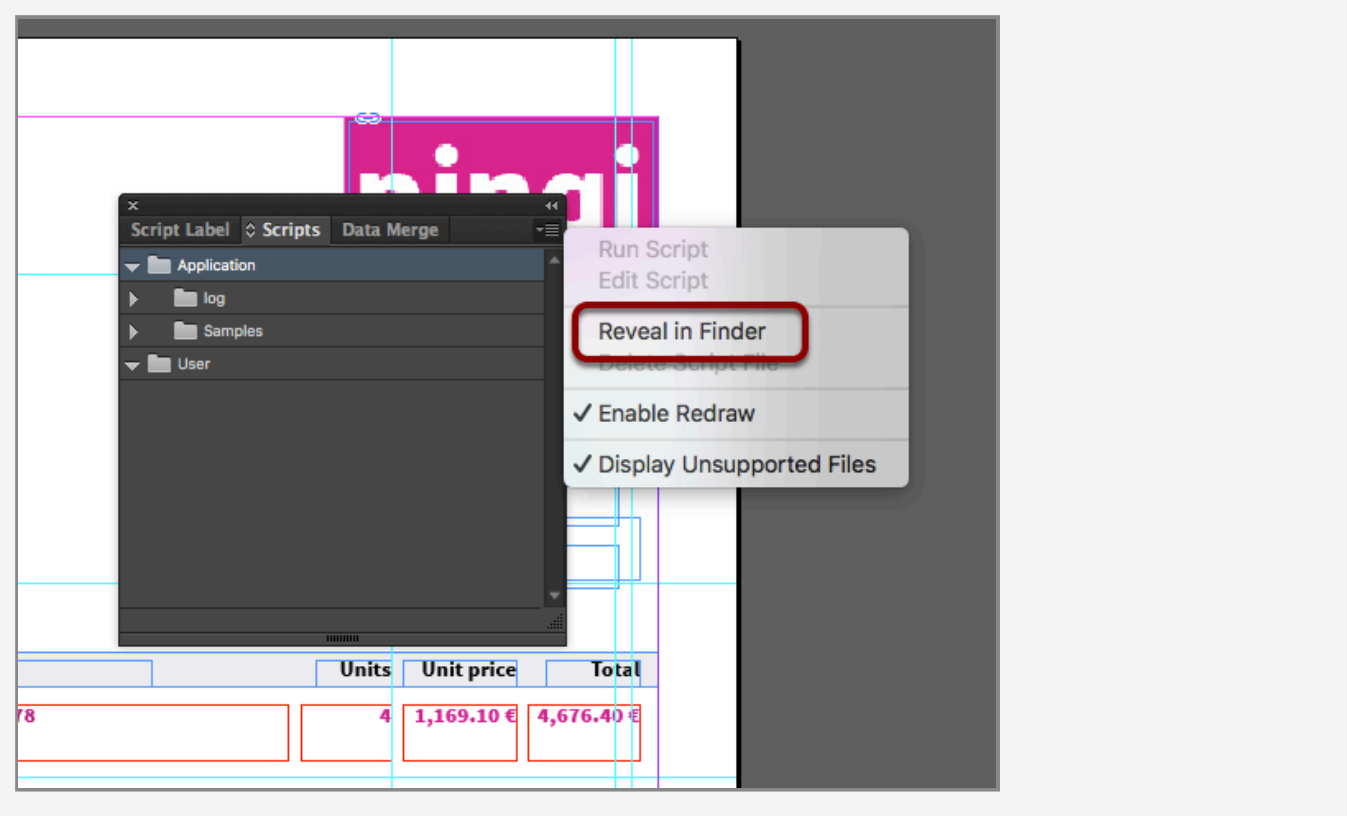
Installation

Open the Scripts Utilities in Indesign



For installation in InDesign go to Window, Utilities and select Scripts. This will open the Scripts palette.

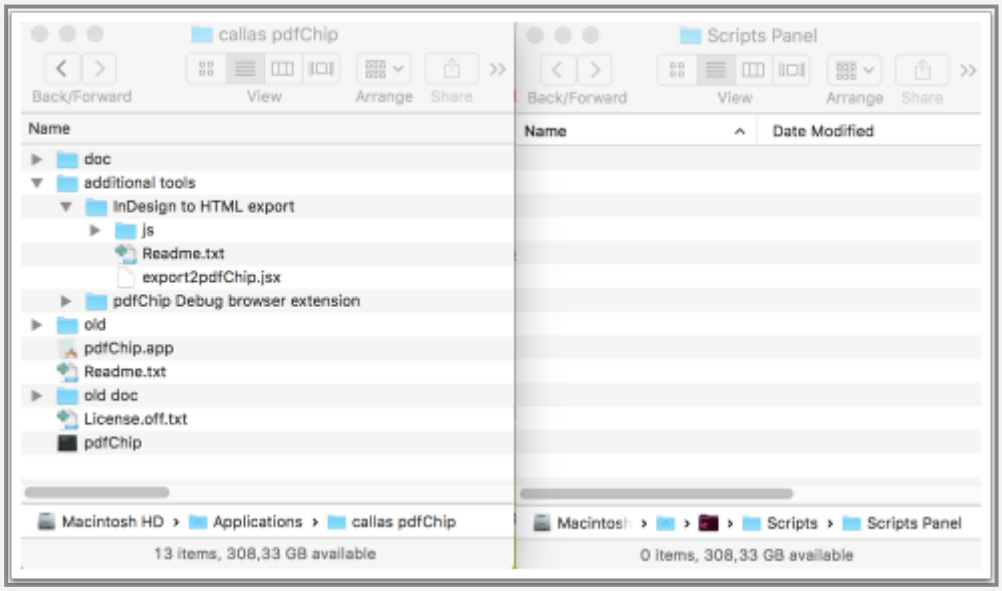
Scripts palette



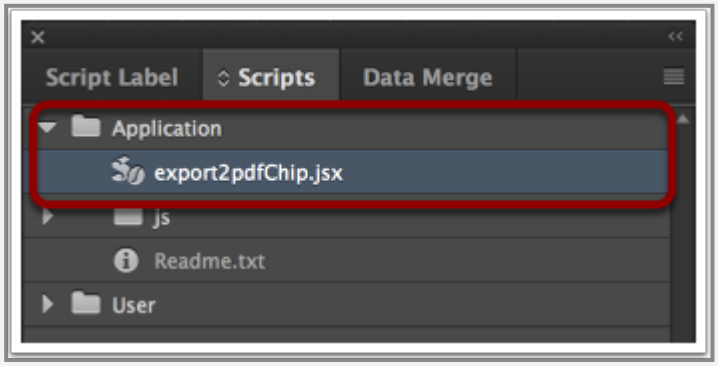
The Scripts palette shows scripts that are available for the user or for the application (all users on the computer). Pick the one that is appropriate in your case and open the options menu and select "Reveal in Finder".

Install the export filter

Install the script by copying all content from the pdfChip program folder "additional tools/InDesign to HTML export" into the Scripts Panel folder in the Scripts folder.



The script and a "js" folder are now listed in the Scripts palette



The script "export2pdfChip.jsx" is the actual export script. It can be used by double clicking it there in the Scripts palette.

6.2 How does the export filter work?

In order for an InDesign file to be properly exported all text should be formatted using paragraph or character styles. Only then it is possible to translate all formatting into CSS that is used for formatting HTML.

There is a helper script "AutoCreate_ParagraphStyles.jsx" that allows you to automatically convert all custom text formatting or character styles into paragraph styles in InDesign. However, if you want to be in control of what shows up in CSS styles in your HTML template you will have do that your self.

Exporting InDesign into pdfChip templates

The InDesign export filter is started by double clicking it in the Scripts palette in InDesign. It creates a new folder next to the InDesign file with the name: <InDesign file name>"_htmlOut". If there is already a folder with this name an error message is displayed before it is overwritten. The new folder has the index.html template and further content in subfolders that it references:

- fonts
- images
- js
- styles

A few files are always created by the export filter: The index.html template itself and a corresponding CSS file: styles/style.css. The js folder contains the Hypenator JavaScript library that is by default used for all exported text to make sure that it is hyphenated.

Basically all text and vector elements will go into the index.html and the style.css has all formatting: text sizes, positioning, colors, images, fonts etc. etc. This will, however, for text only work if it`s formatting uses InDesign paragraph stylesheets and not "custom formatting". See "Auto convert custom formatting or character styles into paragraph styles" how you can do that.

The *fonts* folder has all fonts that have been used in the InDesign file. Currently TrueType and OpenType fonts are supported, since only those work safely in pdfChip. If other font formats are used in the InDesign file an message shows up during export. For each font a @font-face entry is created in styles/style.css.

The *images* folder has copies of all images that are referenced from the InDesign file. The images in this folder are now referenced from styles/style.css. This works for properly for jpg, png or other formats that are supported in HTML. If an image e.g. uses TIFF the HTML will have a broken link (and the PDF created from it will be empty).

Multi page files

The HTML template will consist of a single HTML "page" even if you have several pages in your InDesign file. But this single page will have all content of all InDesign pages. The CSS page size definitions will make sure that when pdfChip converts this template into a PDF file that you will have as many PDF pages as your original InDesign file had.

Auto convert text using custom formatting or character styles into paragraph styles

In order to properly convert text formatting all such formatting should take place using either paragraph or character styles in InDesign. The helper script "AutoCreate_ParagraphStyles" creates a generic paragraph style for each text that is not already assigned to such a style. The generic paragraph styles gets the name "AutoStyle"<No>, where <No> is a number.

If there already is a paragraph style with the name, an error message appears. This will e.g. usually be the case when you would run the script twice on the same InDesign file. If you need to do that because you have inserted new text, you will have to rename as many existing paragraph styles beginning with AutoStyle as you need for the new text.

Hyphenation

If you are using text boxes in InDesign all text will automatically be hyphenated. Only if you have disabled hyphenation in the respective Paragraph Style in InDesign this will not be the case in the result HTML and PDF.

The language settings from InDesign Paragraph Styles are also taken over, so if you are marking a paragraph as English and another as German, these settings will be taken over into the HTML template. This does, however, not work for language settings in Character Styles.

In order to achieve that the InDesign Export automatically inserts and activates a JavaScript library "Hyphenator" into each HTML template that it creates.

Further information about this great library that has been written by Mathias Nater, Zürich (mathiasnater at gmail dot com) is available via this link: https://github.com/mnater/Hyphenator.

Hyphenation results may differ from the results of InDesign since both engines are using different dictionaries and rules.

6.3 Create a simple HTML template

The InDesign Ticket Example

Attached is an InDesign file as an example for a Ticket.

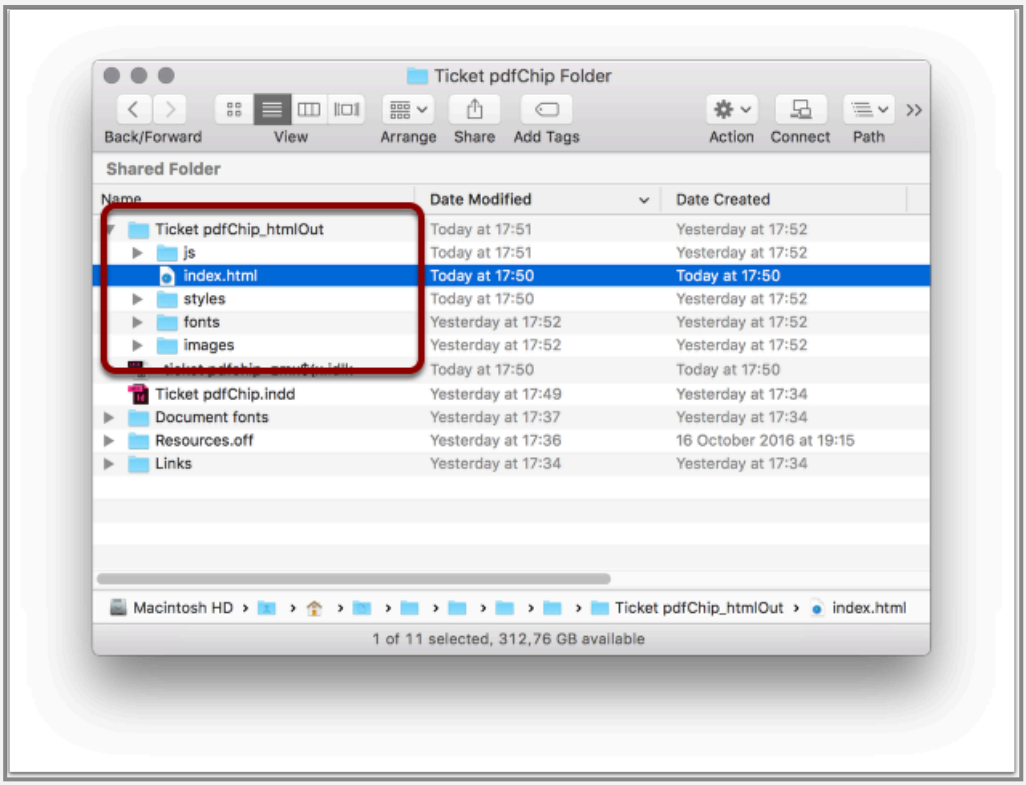
The InDesign Tickt Example has some text using a free font "Free Universal", some images and a color bar at the bottom in a spot color. All other colors are defined in CMYK.

You should download unpack and open the file in order to follow the steps of this article.

InDesign2HTML4pdfChip_1_Ticket.zip

Export the InDesign file into a pdfChip HTML template using the Scripts palette

The "export2pdfChip.jsx" script will create a folder next to the InDesign file

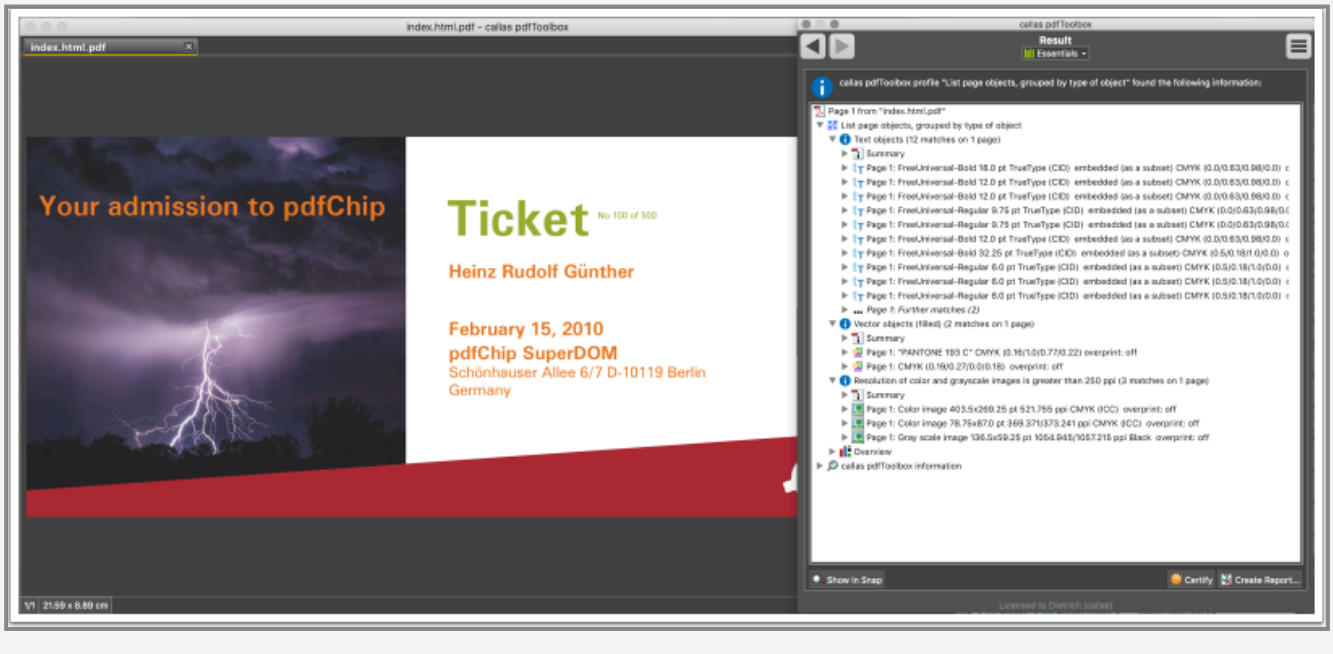


This new folder contains the new HTML template for pdfChip.

In order to create a PDF file from it you can now convert it with pdfChip (on command line or integrated into your environment or by using an editor with a build interface like Sublime).

You may now open the result in pdfToolbox or any other PDF viewer

If you then analyse the PDF, e.g. with pdfToolbox' "List page objects by type of object" profile, you will see that the export has accurately taken all CMYK colors, fonts, images etc. into the HTML template from where pdfChip has then created a printable PDF file:



Fonts are embedded, vector graphics are using CMYK or spot color and images are using ICCbased CMYK.

6.4 Create an HTML template that can be used for "mail merge" style multi-page PDF generation

In this article we are using the same InDesign file that is used in "Create a simple HTML template".

InDesign2HTML4pdfChip_Ticket-1.zip

Marking variable text in InDesign

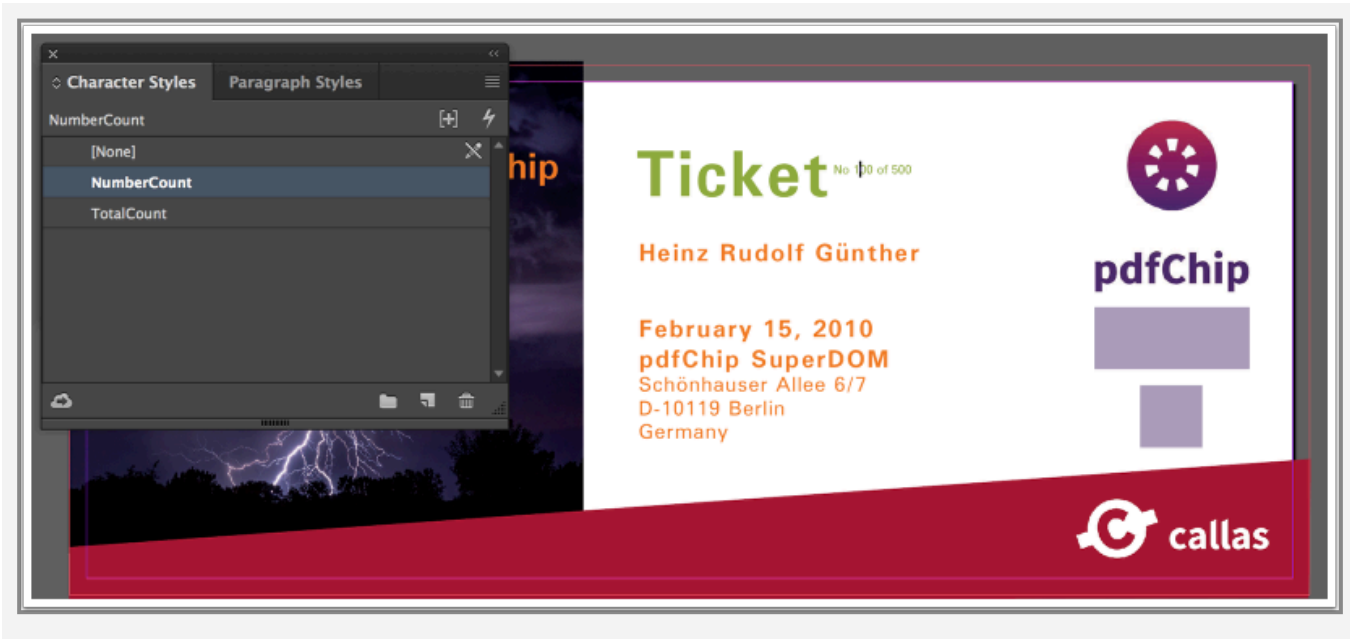
Since the export filter uses CSS IDs and Classes for all objects that it creates and since you can modify or replace any object in a HTML template that can be identified by means of such an ID or a Class you can basically modify everything in the HTML template for individualisation. However, this would be difficult to maintain, because if you would create a new object in a later version of your InDesign file the export script may change the order in the CSS so that all objects will get different names and you would have to modify your JavaScript.

Therefore it is much better to use a persistent identification means which can be found in the InDesign "Script Labels". Script Labels are text strings that are translated by the export script into CSS classes. These classes can then be used in the JavaScript to identify the variable objects and to replace or modify them during PDF page creation in pdfChip. These Script Labels will alway be assigned to "their" objects whatever you do with the rest of the InDesign file.

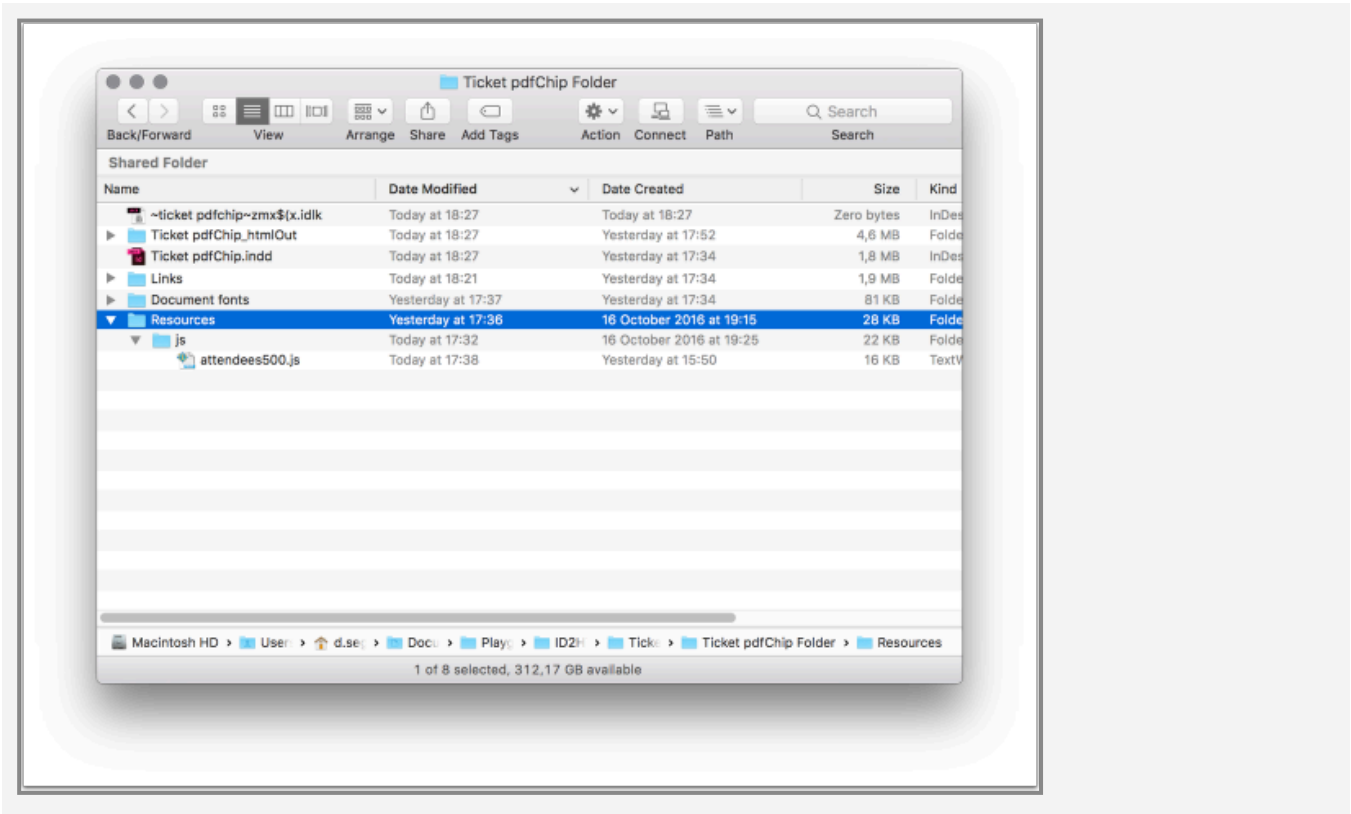
The "Script Label" palette in InDesign is usually next to the Scripts palette in the same window.

If you select the name in the Ticket example you will see that it has the Script Label "attendeename". Similarily the violet boxes below the pdfChip icon have "Barcode" and "QR_Code".

In addition we want to print the current number on each of the individual tickets. Therefore the InDesign file has a text "No 100 of 500" where 100 and 500 are obviously placeholders for the actual numbers. Since these are only part of the same text it is not possible to assign them different Script Labels. You could now either modify the whole text or - as we have done in this example - use a Character Style that is used nowhere else in the document. We are using "NumberCount" and "TotalCount".



Before we start the export...



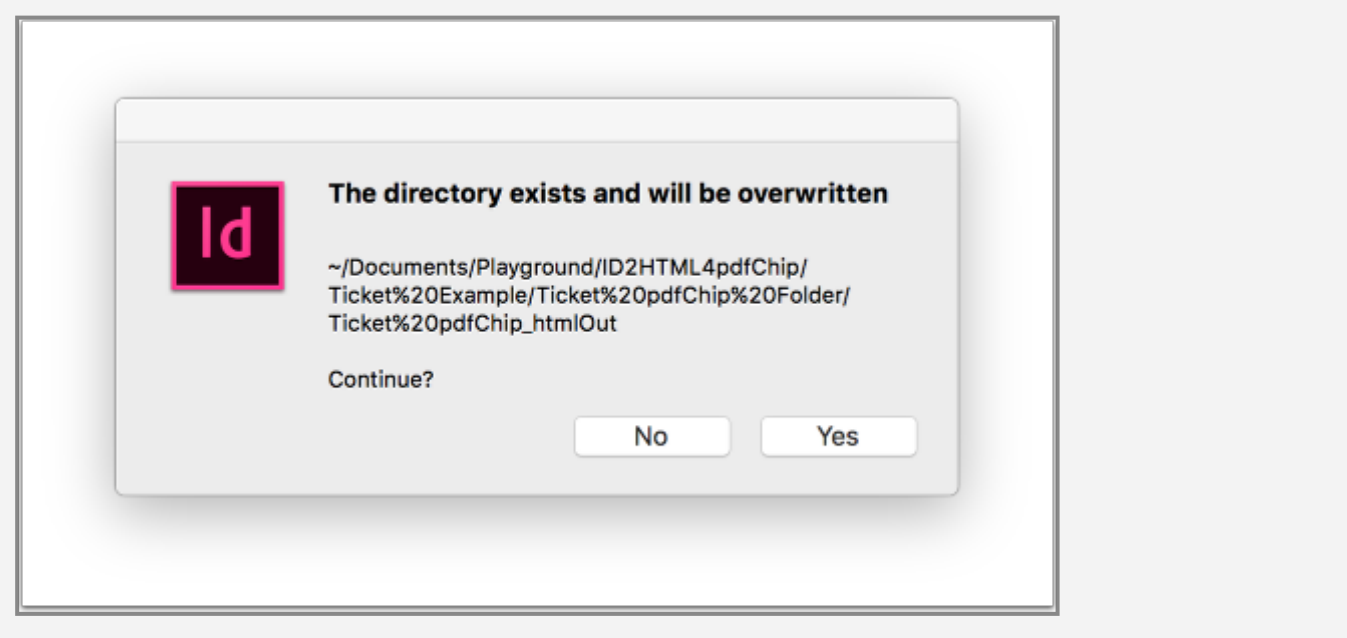
The InDesign folder contains a subfolder "Resources". (If you are using the download from the "Create a simple HTML tem-

plate" example it will have "Resources.off" instead. In that case just remove the ".off".)

The export script will copy all contents from this folder into the folder of the HTML template. In addition if the Resources folder contains a "js" folder it will automatically create references to the JavaScript files in this folder in the index.html file that it creates.

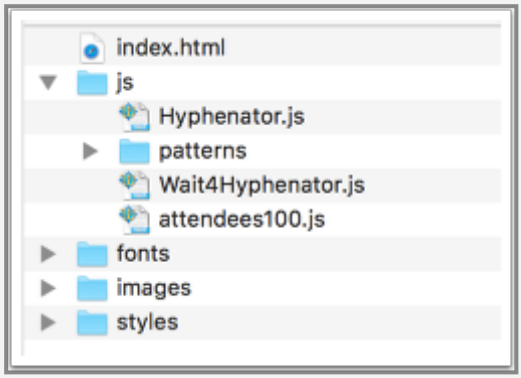
So, if you create a JavaScript that creates individual instances from your template you wil only have to put it into this place in order to attach it to any HTML template that the export script creates.

If you have exported the InDesign file before you will see a message that the previous result will be overwrittten:

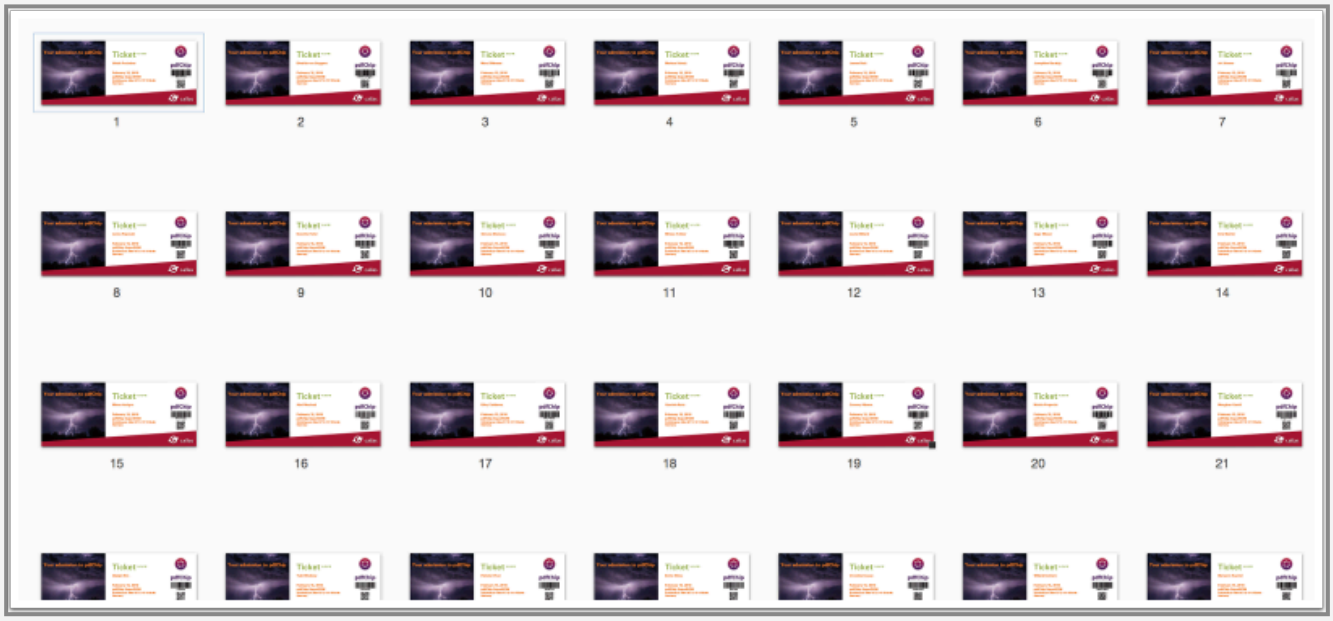


You may click "Yes" to confirm or rename the previous result if you want to keep it.

The mail merge result



The js folder has the Hyphenator.js file and an attendees100.js file. If you now convert the index.html into PDF using pdfChip you will create a 100 pages PDF file with individual tickets for exactly 100 names. Each page has a barcode and a QR code that encodes the name of the ticket owner. And finally each ticket says which number it is out of the 100 tickets that we have created.



How did that work?

As said before we are using CSS Classes and IDs to identify objects in the HTML file.

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8"/>
    <title>Ticket pdfChip.indd</title>
    <link rel="stylesheet" href="styles/style.css" type="text/css"/>
    <script src="js/attendees100.js" type="text/javascript"> </script>
    <script src="js/Hyphenator.js" type="text/javascript"> </script>
    <script type="text/javascript">Hyphenator.config({displaytogglebox : false });Hyphenator.run();</script>
  </head>
  <body onload="undefined">
    <div id="page_0" class="page">
      <div id="rectangle_u0" class="rectangle page_0 QR_Code Layer1"> </div>
      <div id="rectangle_u1" class="rectangle page_0 Barcode Layer1"> </div>
      <div id="rectangle_u3" class="rectangle page_0 Layer1"> </div>
      <div id="rectangle_u4" class="rectangle page_0 Layer1 image_u1"> </div>
      <div id="rectangle_u5" class="rectangle page_0 Layer1 image_u2"> </div>
      <div id="textframe_u0" class="textframe page_0 Layer1"> </div>
      <div id="textframe_u1" class="textframe page_0 Layer1">

        <p class="p_Numbers donthyphenate" lang="en-us">
          No
          <span class="span_NumberCount">100</span>
          of
          <span class="span_TotalCount">500</span>
        </p>
      </div>
      <div id="textframe_u2" class="textframe page_0 Layer1">

        <p class="p_Ticket donthyphenate" lang="en-us">Ticket</p>
      </div>
```

Above entries are the ones for the QR Code, the Barcode and the two entries for the numbers.

```
        <span class="span_TotalCount">500</span>
      </p>
    </div>
    <div id="textframe_u2" class="textframe page_0 Layer1">

      <p class="p_Ticket donthyphenate" lang="en-us">Ticket</p>
    </div>
    <div id="textframe_u3" class="textframe page_0 attendeename Layer1">

      <p class="p_Customarea donthyphenate" lang="en-us">Heinz Rudolf Günther</p>
    </div>
    <div id="textframe_u4" class="textframe page_0 Layer1">

      <p class="p_Customarea donthyphenate" lang="en-us">pdfChip SuperDOM</p>
      <p class="p_Address donthyphenate" lang="en-us">Schönhauser Allee 6/7 D-10119 Berlin</p>
      <p class="p_Address donthyphenate" lang="en-us">Germany</p>
    </div>
    <div id="textframe_u5" class="textframe page_0 Layer1">
```

Further down in the not so long file you will see the entry for the actual name. In fact this name will not occur on any of the tickets that we have created. It is only a placeholder that is being replaced by the names that are taken from the JavaScript file.

The "attendees100.js" file

```
var attendees = [
{"Name":"Ulrich Frotscher"},
{"Name":"Dietrich von Seggern"},
{"Name":"Marc Dillmann"},
{"Name":"Markus Schulz"},
{"Name":"James Butt"},
{"Name":"Josephine Darakjy"},
{"Name":"Art Venere"},
{"Name":"Lenna Paprocki"},
{"Name":"Donette Foller"},
{"Name":"Simona Morasca"},
{"Name":"Mitsue Tollner"},
{"Name":"Leota Dilliard"},
{"Name":"Sage Wieser"},
{"Name":"Kris Marrier"},
{"Name":"Minna Amigon"},
{"Name":"Abel Maclead"},
```

The JavaScript file starts with 100 name entries.

```
function cchipPrintLoop() {
    for (var i = 0; i < attendees.length; i++) {
        document.getElementsByClassName("attendeename")[0].innerHTML =
            '<p class="p_Customarea donthyphenate" lang="en-us">' + attendees[i].Name + '</p>';
        document.getElementsByClassName("span_NumberCount")[0].innerHTML = i + 1;
        document.getElementsByClassName("span_TotalCount")[0].innerHTML = attendees.length;


        document.getElementsByClassName("Barcode")[0].innerHTML = '<object class="qrcode" type="application/barcode"
            style="width:1.09in; height:.435in; background-color:#FFFFFF; background-color:-cchip-cmyk(0,0,0,0); color:#000000;
            color:-cchip-cmyk(0,0,0,1); font-size:9px"> <param name="type" value="Code 128"> <param name="data" value="" +
            attendees[i].Name + '"> </object>';

        document.getElementsByClassName("QR_Code")[0].innerHTML = '<object class="qrcode" type="application/barcode" style="width:
            .435in; height:.435in; background-color:#FFFFFF; background-color:-cchip-cmyk(0,0,0,0); color:#000000; color:-cchip-
            cmyk(0,0,0,1)"> <param name="type" value="QR-Code"> <param name="data" value="" + attendees[i].Name + '"> </object>';
        cchip.printPages();
    };
};
```

The names are followed by a single function "cchipPrint-Loop". This is a basic pdfChip function that allows you to take control of when a PDF page is created. You can now use JavaScript within this loop to modify the internal representation of the HTML file, the DOM (Document Object Model) and ask pdfChip to create a new page whenever you have done so by using the cchip.printPages() call. Modifying the DOM with JavaScript is standard web technology that is used on millions of web pages every second of the day.

Each of the calls beginning with "document.getElementsBy-ClassName" identifies one of the objects using either the Script Labels that we have used "attendeename", "Barcode" and "QR_Code" or by using their Character Style names "span_NumberCount" and "span_TotalCount" (or rather what the export script has made of them by prefixing them with span_).

The code that is put into the "innerHTML" of each of these objects is HTML code into which the current object from the list of names is placed to create individual tickets. For the Barcode and the QR_Code again pdfChip specific code is used. This is explained other parts of this pdfChip manual, you should go there if you want to know more about that part of the story.

6.5 Create an HTML template for invoices

This article explains a rather complex project that shows how a HTML template for invoicing can be created in InDesign, exported into a pdfChip template using the export filter and then turned into invoices.

Attached is an InDesign file as an example for an invoice. You should download unpack and open the file in order to follow the steps of this article.

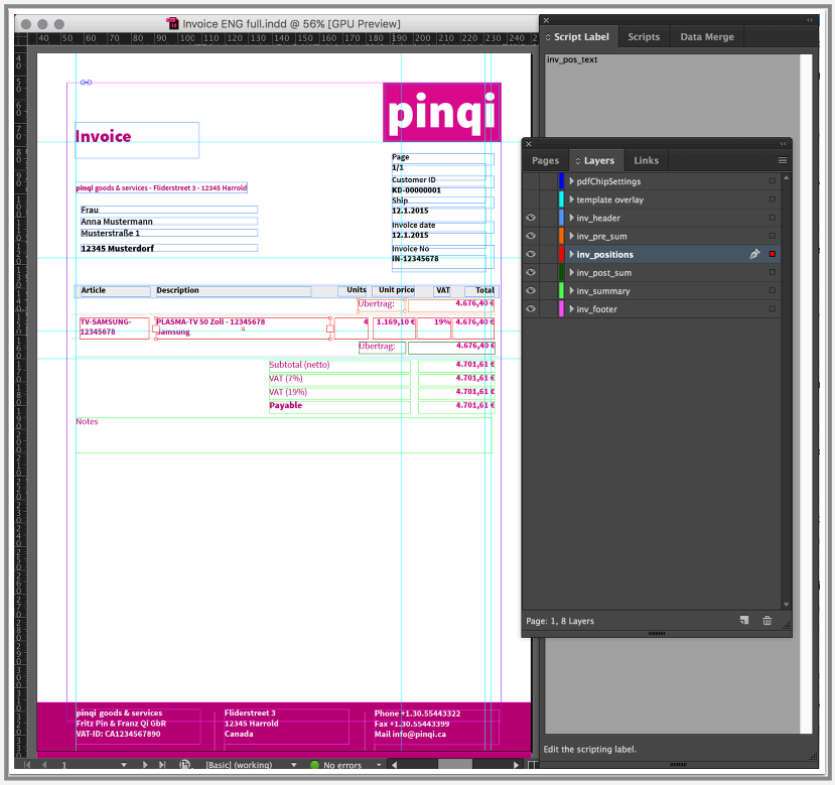
This project is much more advanced than the other ones in which the InDesign Export is explained. The reason is that invoices have rather difficult requirements:

- repeating fields for each of the line items on the invoice
- varying number of pages, depending on the number of line items
- first, last and middle pages may have different layouts
- at the end of each page a subtotals block and at the end of the invoice a totals block is required
- page breaks need to be determined based on the height of all of the line items on the respective page minus the height of the subtotals block

So, if you are not yet familiar with how the InDesign Export works you should first read the other articles in this chapter.

As always when it is about creating various instances from a template we need a data source that has the variable data. In this example we are using XML files in the ZUGFeRD format. (ZUGFeRD is a German standard for electronic invoices and a ZUGFeRD invoice consists of a PDF/A-3 container into which an XML invoice is embedded.) This example reads data from such ZUGFeRD invoices, puts it into the corresponding objects in the template and creates a single PDF file for each invoice. Finally the XML is embedded into the PDF and the file is saved according to the PDF/A-3 standard. So a full ZUGFeRD invoice is created.

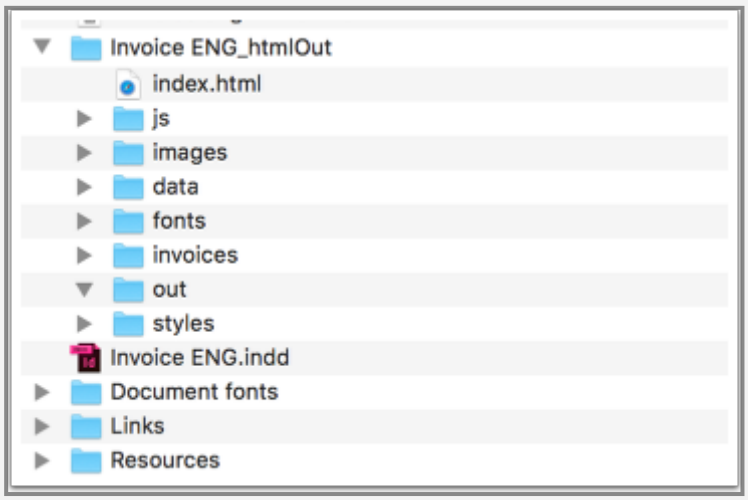
The InDesign file



The InDesign file uses layers and groups to mark content that belongs to each other like the different kinds of data in a line. Script Labels are used to mark objects so that a JavaScript can then later pick up that information to modify the respective content. The file has three pages: First and last for the corresponding pages of the PDF invoice and a middle page that is the template for all other pages.

pinqi_invoice_template.zip

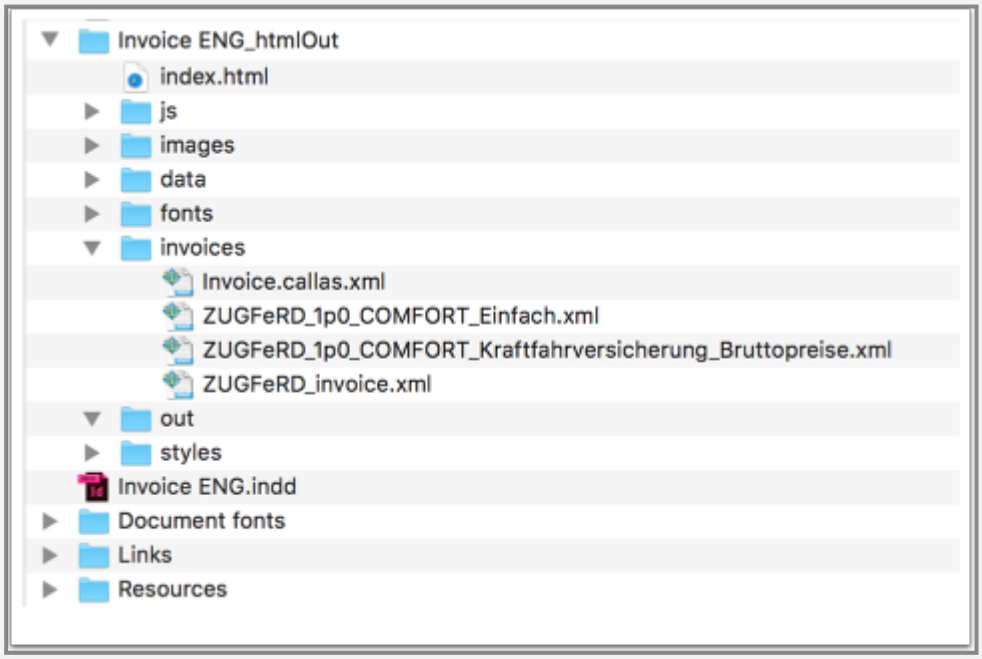
HTML template created via the export2pdfChip.jsx plug-in



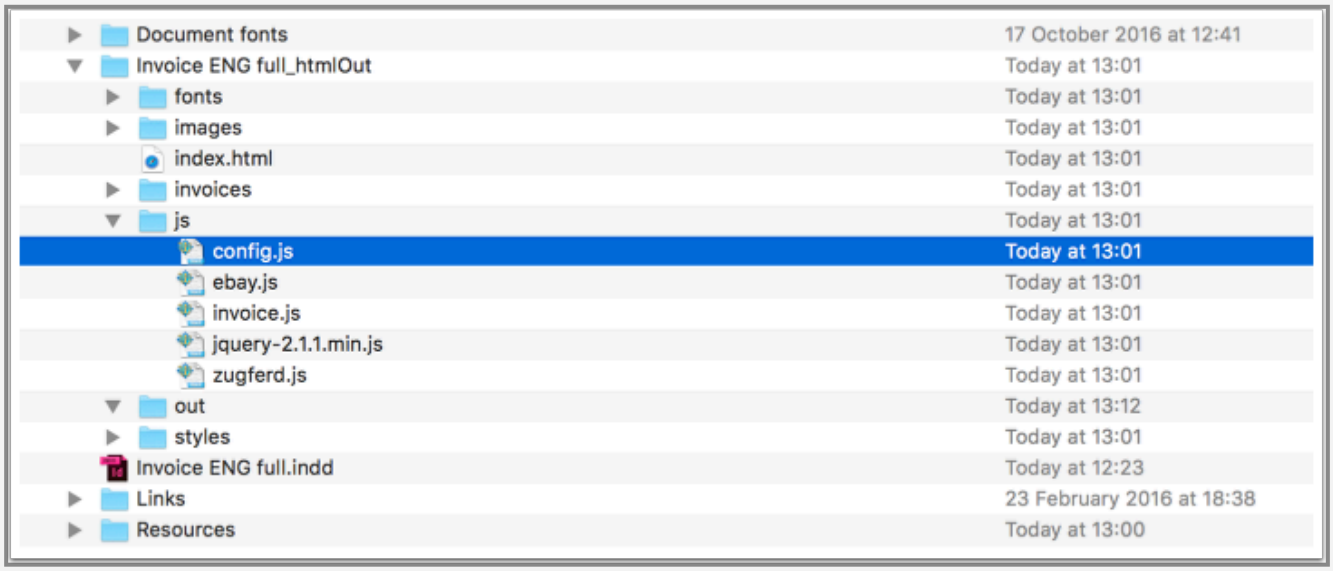
All content from the Resources folder is copied into the template and the index.html has references to all JavaScripts in the js folder.

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8"/>
    <title>Invoice ENG.indd</title>
    <link rel="stylesheet" href="styles/style.css" type="text/css"/>
    <script src="js/Hyphenator.js" type="text/javascript"> </script>
    <script src="js/Wait4Hyphenator.js" type="text/javascript"> </script>
    <script src="js/CVS" type="text/javascript"> </script>
    <script src="js/config.js" type="text/javascript"> </script>
    <script src="js/invoice.js" type="text/javascript"> </script>
    <script src="js/jquery-2.1.1.min.js" type="text/javascript"> </script>
    <script src="js/zugferd.js" type="text/javascript"> </script>
  </head>
  <body onload="onLoad()">
    <div id="page_0" class="page">
```

The invoices folder contains sample XML files for ZUGFeRD invoices: A few examples from the ZUGFeRD standard specification and a very long example with callas products on it.

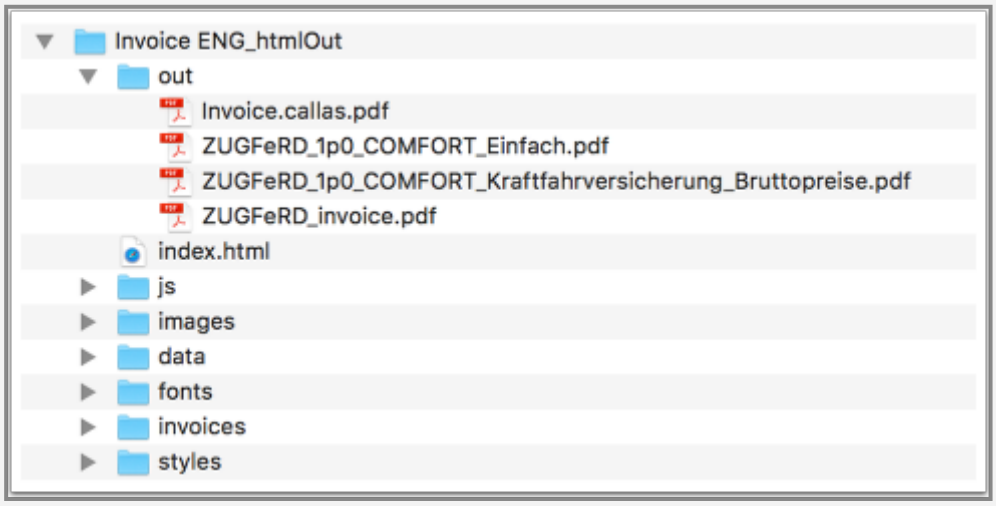


If you want to modify any of the XML files or add new ones you will have to also add them to the config.js file.



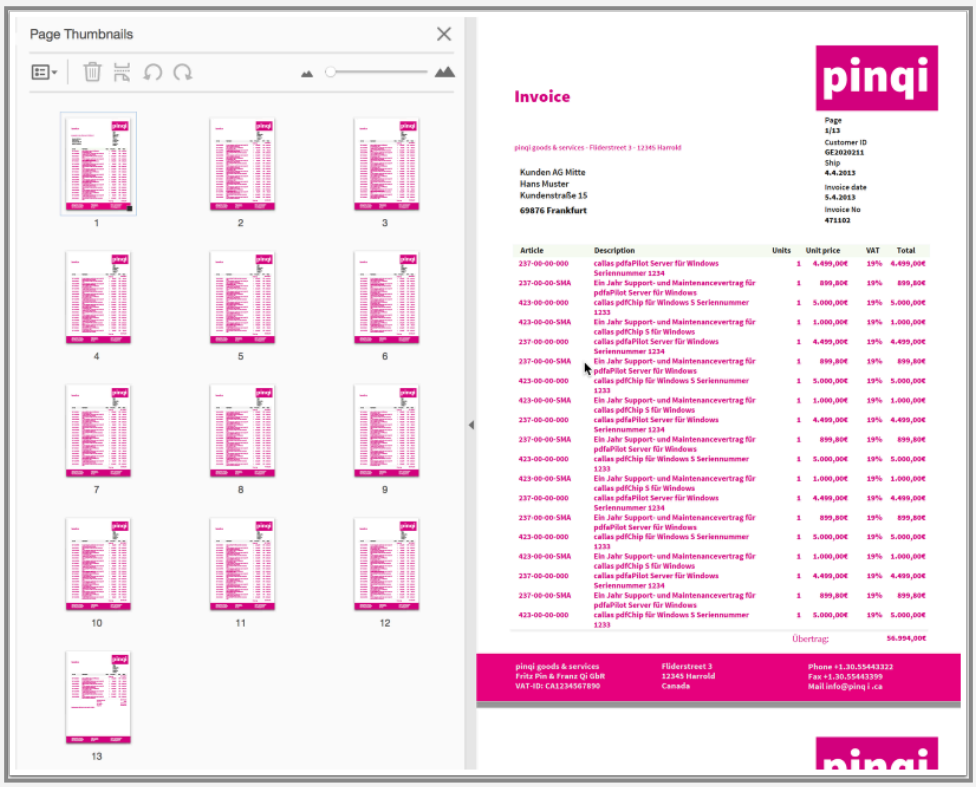
```
var config =
{
   settings:
   {
      individualPDFs  : false, // if true or source="zugferd", each invoice is create in a seperate PDF file.
      source          : "zugferd",//"zugferd", // data source: or "zugferd", else static data is used
      zugferd:
      {
         invoice_xml: ["./invoices/ZUGFeRD_invoice.xml"
                      ,"./invoices/ZUGFeRD_1p0_COMFORT_Einfach.xml"
                      ,"./invoices/ZUGFeRD_1p0_COMFORT_Kraftfahrversicherung_Bruttopreise.xml"
                      ,"./invoices/Invoice.callas.xml"
                      ], // Array of ZUGFeRD invoices to process
      },
```

The invoice PDF files are as usual created by processing the index.html with pdfChip



Each PDF has it's XML file embedded.

The invoice "invoice.callas.pdf" consists of 13 pages and the first and last pages are different from each other and from all middle pages as required in above "specification".



6.6 Autocreate Paragraph Styles from custom styling

If you have an InDesign document that has lots of formatted text but no Paragraph Styles it is sometimes cumbersome to create those. However, if you want to use that document as the basis for a pdfChip HTML template you would need Paragraph Styles for all text formatting so that corresponding CSS entries can be created from them.

This helper script allows you to automatically create Paragraph Styles for all such text which is not already using Paragraph Styles. It will create generic names for any new Paragraph Styles: AutoStyle1, AutoStyle2, ... If the InDesign document already has Paragraph Styles with these names an error message will show up. That means, if you want to use it for a document for which you have previously done that already, you will have to either rename any such generic names or delete them beforehand.

AutoCreate_ParagraphStyles.jsx

For installation in InDesign go to Window, Utilities and select Scripts. This will open the Scripts palette.

The Scripts palette shows scripts that are available for the user or for the application (all users on the computer). Pick the one that is appropriate in your case and open the options menu and select "Reveal in Finder". Then put the script into that folder. After that it can be used by clicking it in the Scripts palette.

7. Logging and debugging techniques

7.1 Extended logging capabilities: "--dump-static-html" command line parameter

A new command line parameter "--**dump-static-html**" has been introduced for pdfChip. It makes it possible to create HTML files representing DOM snapshots – equivalent to what pdfChip turns into PDF pages for each cchip.printPages() call. If cchip.printPages() is not used, this parameter will create one HTML file representing the DOM that is the basis for all the PDF pages created.

The "--dump-static-html" command line parameter

Command line argument.

```
--dump-static-html[=<folder name>]
```

If the --dump-static-html option is specified then the current DOM state is written to an HTML file on each cchip.print-Pages() call.

The parameter =<folder name> is optional and defaults to dump-static-html. Its value is used as a prefix (to which a time stamp will get appended) for the names of the folders which contain the HTML file(s) with the dumped DOM.

Destination folder

The <folder name> parameter specifies a prefix for the name (not the path!) of the destination folder.

The folder is always created in the same folder as the output PDF file(s).

A time stamp is appended to destination folder name in the following manner:

```
<folder name>-yyyy-mm-dd–hh-mm-ss
```

Output file names format

Names of the HTML output files are constructed as follows:

```
“<html-file-name>-<num-html>-<pdf-file-name>-<num-pdf>.html”
```

where

- <num-html> is a counter for each occurrence of the ‘cchip.printPages’ call executed by the input HTML file with the name <html-file-name>. The `<num-html>` counter is reset to 0 for each input HTML file that is converted - which will only happen if more than one input HTML file has been passed to a pdfChip command line call.
- `<num-pdf>` is a counter for of ‘cchip.printPages’ call per output PDF, where that output PDF has the name `<pdf-file-name>`. The `<num-pdf>` counter is reset to 0 for each output PDF file that gets created - which will only happen if more in any of the input HTML files the function `cchip.setOutputPdf()` is called to start creating pages in a new PDF output file.

Example

For the following command line

```
pdfChip.exe --dump-static-html=dump index.html out.pdf
```

let “index.html” file contain two cchip.printPages() calls. Then during processing by pdfChip on Septermber 23, 2016, at 9:19:30 AM two static HTML files will get created in a folder called dump-2016-09-23—09-19-30:

- *“dump-2016-09-23—09-19-30/index-000-out-000.html”*
- *“dump-2016-09-23—09-19-30/index-001-out-001.html”*

7.2 Using "pdfChip Debug" browser plug-in for interactive debugging (1.2)

pdfChip Debug is a browser extension for Google Chrome (supported on Mac OS X and Microsoft Windows). It emulates almost all of pdfChip specific functionality, mostly functions and data provided by the cchip.* object.

Without such emulation, it may be impossible or of limited use to run and inspect an HTML file written specifically for use of pdfChip features in a browser, as a browser will not normally implement or otherwise provide functionality that is specific to pdfChip.

This article describes how to install the pdfChip Debug browser extension and how to make use of it.

Security advice

In order for the pdfChip browser extension to work the settings in the browser (Google Chrome) have to be adjusted in a couple of way, in order for pdfChip Debug to be able to do its work:

- enable Developer mode
- allow access to local files
- turn pdfChip Debug's active mode on

The first two increase the risk of «bad things happening» (i.e. all kinds of «cyber attacks»), especially when opening or browsing files or URLs that you do not know or that possibly can't or should not be trusted. **Thus, use these settings with caution!**

Also, in order to make transparent what happens – or could happen – when installing and using pdfChip Debug, it is provided as an unpacked extension, so its source code is fully accessible.

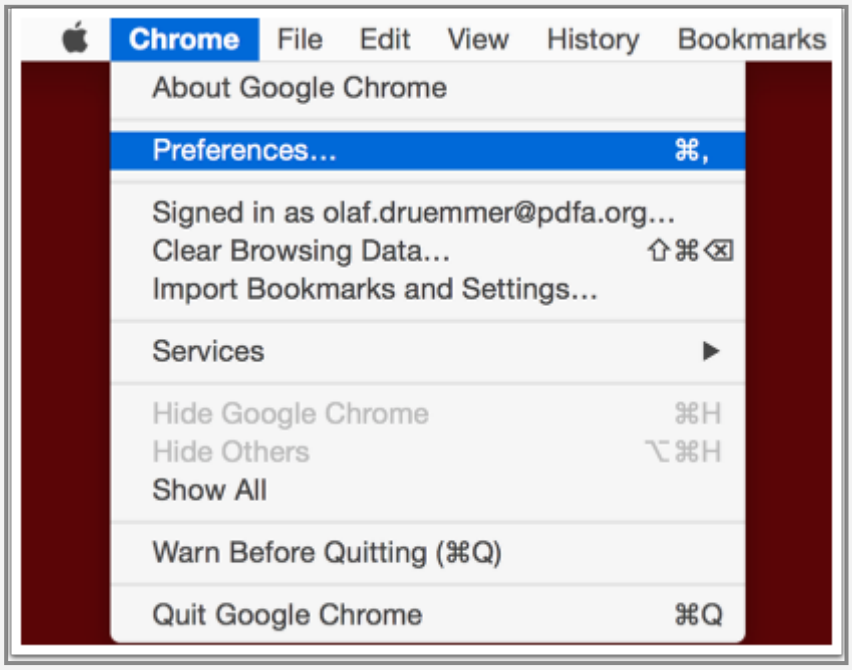
It is recommended to disable *Developer mode* and *Access to local files* when not doing development work or when browsing the internet.

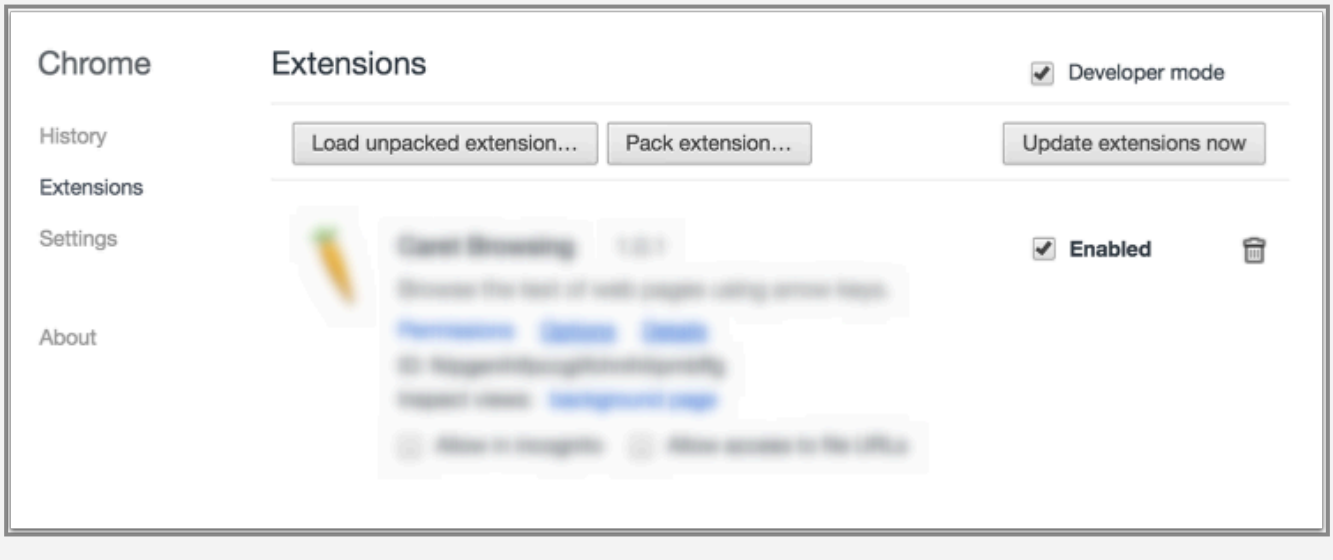
Installing pdfChip Debug

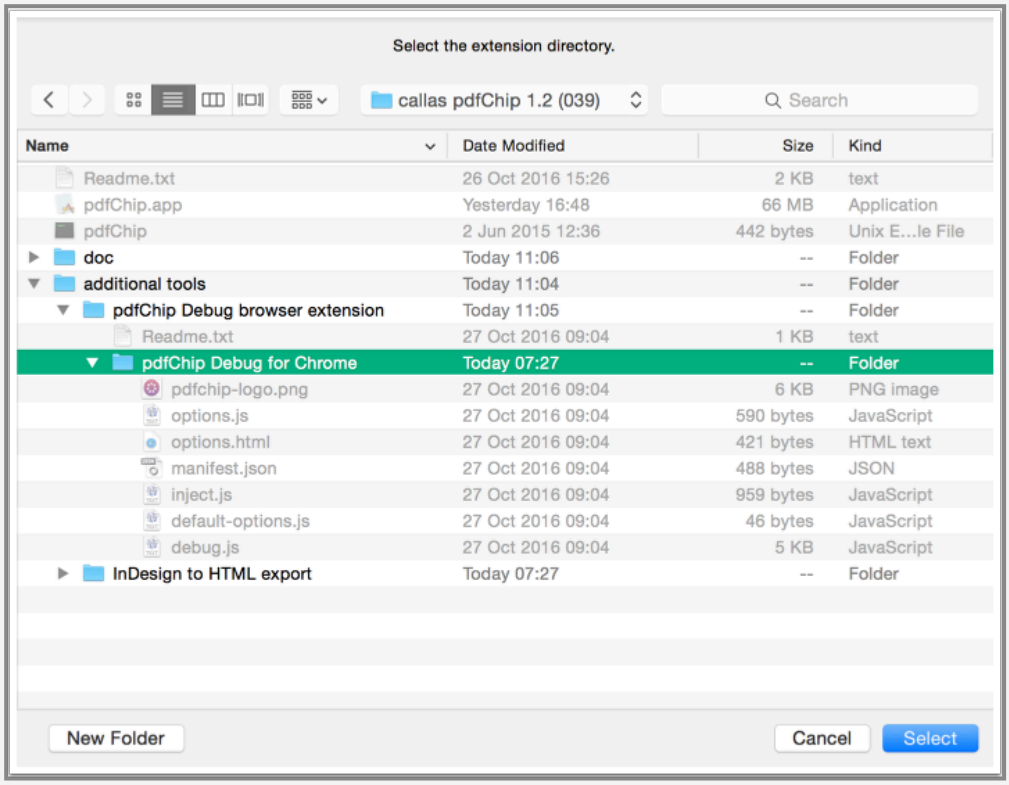
1. in Google Chrome, go to "Preferences"
2. switch to "Extensions" pane
3. locate and select "pdfChip Debug for Chrome" folder on your file system
4. press "Select" button

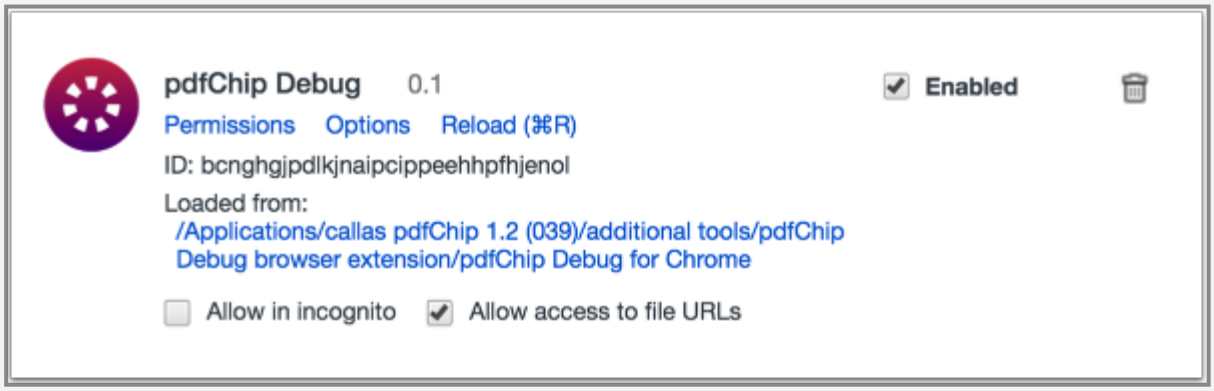
By default, pdfChip Debug will now be **enabled** and will have **access to the local file system**.

pdfChip debug can be disabled or removed from Google Chrome at any time.




Chrome
History
Extensions
Settings
About
Extensions
Developer mode
Load unpacked extension...
Pack extension...
Update extensions now
Enabled


Select the extension directory.
callas pdfChip 1.2 (039)
Search
Name
Date Modified
Size
Kind
Readme.txt 26 Oct 2016 15:26 2 KB text
pdfChip.app Yesterday 16:48 66 MB Application
pdfChip 2 Jun 2015 12:36 442 bytes Unix E...le File
doc Today 11:06 -- Folder
additional tools Today 11:04 -- Folder
pdfChip Debug browser extension Today 11:05 -- Folder
Readme.txt 27 Oct 2016 09:04 1 KB text
pdfChip Debug for Chrome Today 07:27 -- Folder
pdfchip-logo.png 27 Oct 2016 09:04 6 KB PNG image
options.js 27 Oct 2016 09:04 590 bytes JavaScript
options.html 27 Oct 2016 09:04 421 bytes HTML text
manifest.json 27 Oct 2016 09:04 488 bytes JSON
inject.js 27 Oct 2016 09:04 959 bytes JavaScript
default-options.js 27 Oct 2016 09:04 46 bytes JavaScript
debug.js 27 Oct 2016 09:04 5 KB JavaScript
InDesign to HTML export Today 07:27 -- Folder
New Folder
Cancel
Select

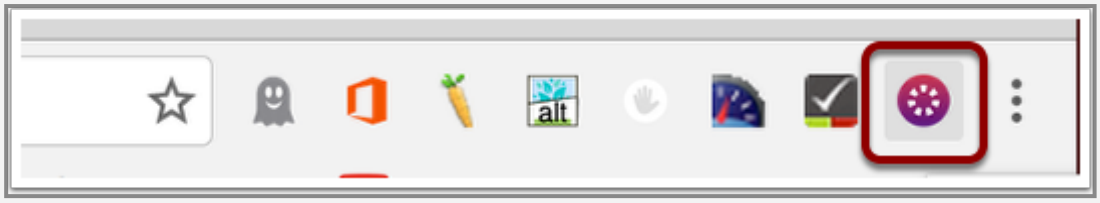


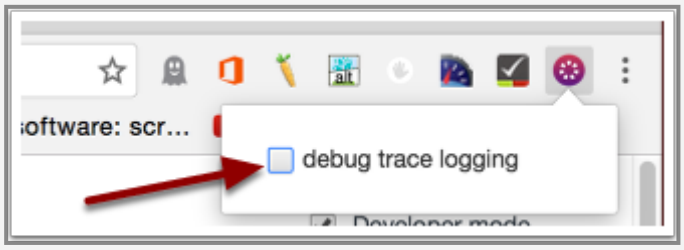
Activating pdfChip Debug

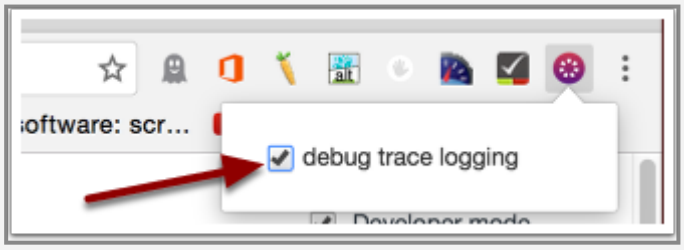
After installing pdfChip Debug, a pdfChip icon will be shown in the Google Chrome toolbar.

In order to activate pdfChip Debug – such that its emulation of pdfChip functionality actually kicks on – click in that icon, and tick the checkbox in order to activate "debug trace logging".

Do not forget to disable again once done inspecting and debugging your pdfChip projects.

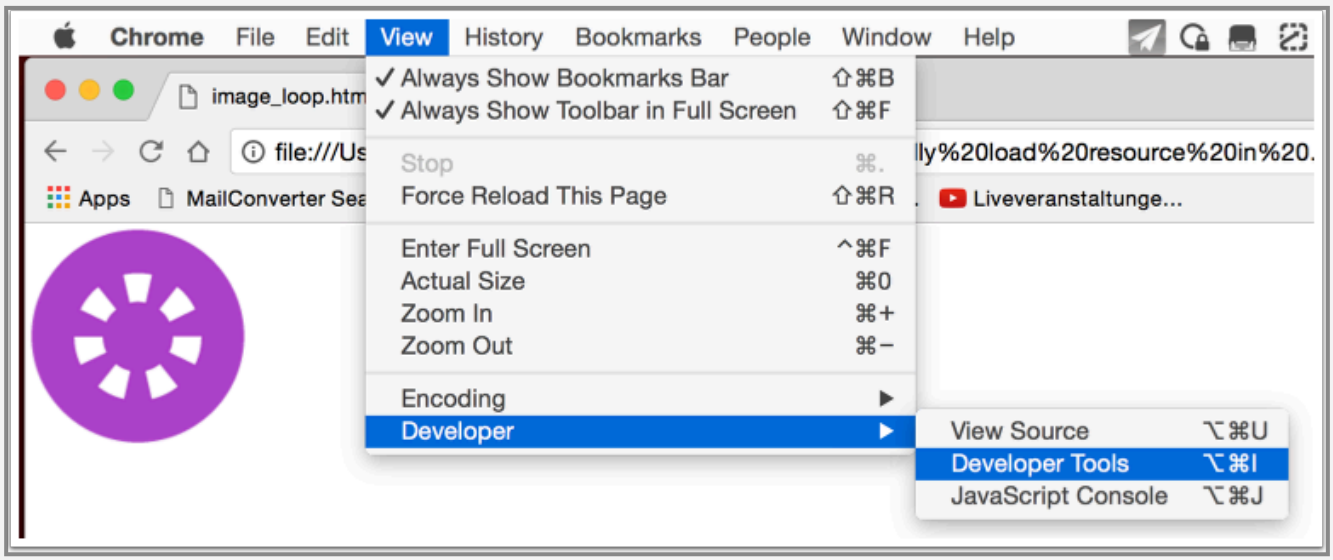






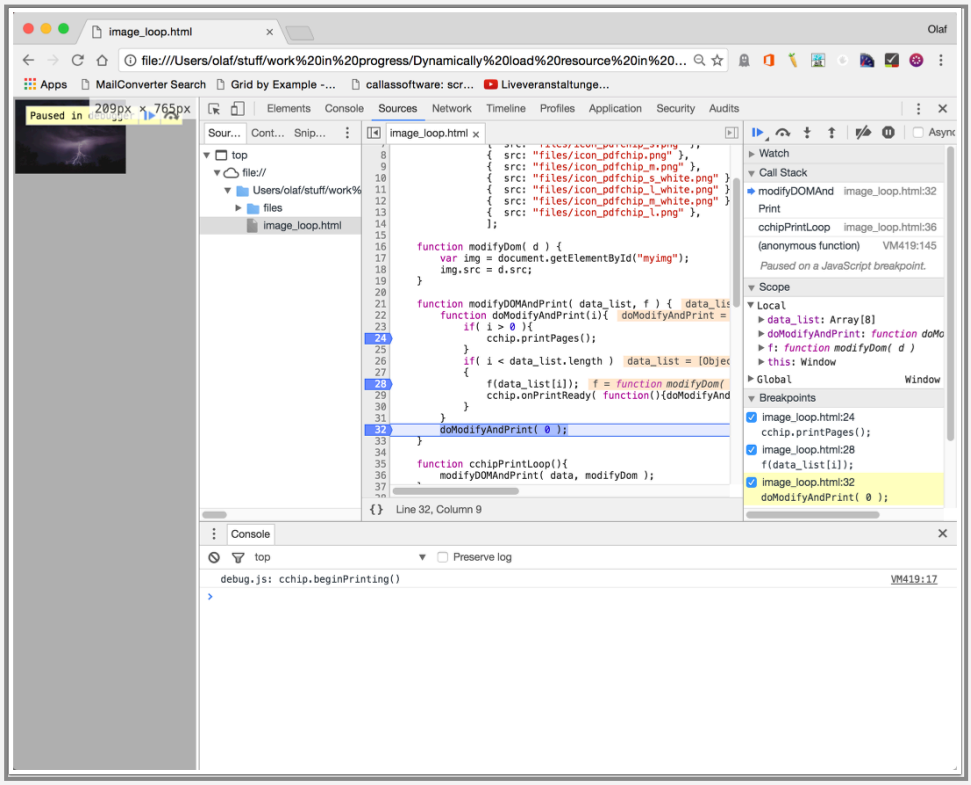
Initiate a debugging session

With a suitable HTML file loaded – whether from the local file system or from a URL – simply launch the "Developer Tools" in Google Chrome.



Google Chrome Developer Tools

The Google Chrome Developer Tools provide a convenient and powerful environment for inspecting HTML projects. Due to the pdfChip Debug provided emulation of pdfChip specific functionality and data objects it is possible to step through pdfChip specific JavaScript routines, set breakpoints, inspect variables, or interact with the HTML project over the console.


image_loop.html
file:///Users/olaf/stuff/work%20in%20progress/Dynamically%20load%20resource%20in%20...
Apps
MailConverter Search
Grid by Example -...
callassoftware: scr...
Liveveranstaltunge...
Paused in debugger
209px x 765px
Elements
Console
Sources
Network
Timeline
Profiles
Application
Security
Audits
Sour...
Cont...
Snip...
top
file://
Users/olaf/stuff/work%
files
image_loop.html
image_loop.html
function modifyDom(d) {
var img = document.getElementById("myimg");
img.src = d.src;
}
function modifyDOMAndPrint(data_list, f) {
function doModifyAndPrint(i){
if(i > 0){
cchip.printPages();
}
if(i < data_list.length)
{
f(data_list[i]);
cchip.onPrintReady(function(){doModifyAnd
}
}
doModifyAndPrint(0);
}
function cchipPrintLoop(){
modifyDOMAndPrint(data, modifyDom);
Line 32, Column 9
Watch
Call Stack
modifyDOMAnd image_loop.html:32
Print
cchipPrintLoop image_loop.html:36
(anonymous function) VM419:145
Paused on a JavaScript breakpoint.
Scope
Local
data_list: Array[8]
doModifyAndPrint: function doMo
f: function modifyDom(d)
this: Window
Global Window
Breakpoints
image_loop.html:24
cchip.printPages();
image_loop.html:28
f(data_list[i]);
image_loop.html:32
doModifyAndPrint(0);
Console
top
Preserve log
debug.js: cchip.beginPrinting()
VM419:17

cchip.dumpStaticHtml
__defineGetter__
__defineSetter__
__lookupGetter__
__lookupSetter__
beginPrinting
closeOutputPdf
constructor
customData
dumpStaticHtml
endPrinting
getPDFPageBox
getPDFPageCount
hasOwnProperty
isPrototypeOf
log
onPrintReady
overlays
pages
printPages
privateSetPluginOptions
propertyIsEnumerable
setOutputPdf
toLocaleString
toString
underlays
urls
valueOf
versionString

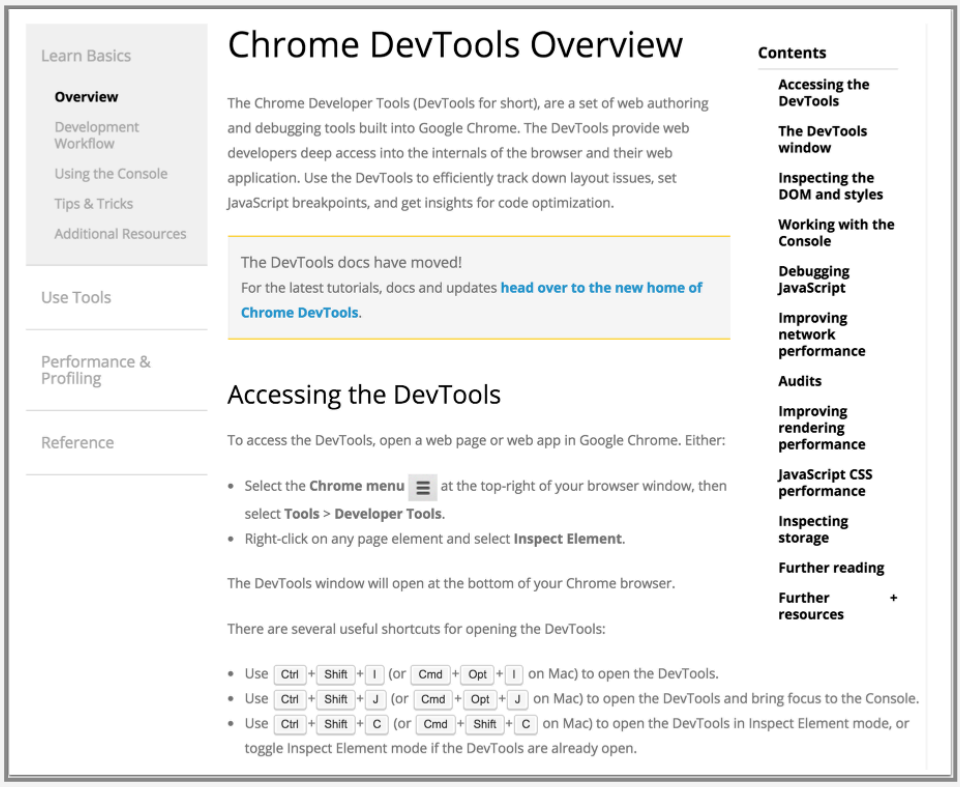
```
> cchip.dumpStaticHtml()
  debug.js: dumpStaticHtml:
  <html><head>

  </head>

  <body>
      <img id="myimg" style="width:100mm" src="files/icon_pdfchip_l.png">


  </body></html>
```

Maximizing use of Google Chrome Developer Tools for pdfChip projects

Google provides excellent resources about how to use their Google Chrome Developer Tools – reading is absolutely recommend, simple check out:

https://developer.chrome.com/devtools

Start page of Google Chrome Developer Tools documentation:



Learn Basics
- Overview
- Development Workflow
- Using the Console
- Tips & Tricks
- Additional Resources

Use Tools

Performance & Profiling

Reference

Chrome DevTools Overview

The Chrome Developer Tools (DevTools for short), are a set of web authoring and debugging tools built into Google Chrome. The DevTools provide web developers deep access into the internals of the browser and their web application. Use the DevTools to efficiently track down layout issues, set JavaScript breakpoints, and get insights for code optimization.

The DevTools docs have moved!
For the latest tutorials, docs and updates **head over to the new home of Chrome DevTools**.

Accessing the DevTools

To access the DevTools, open a web page or web app in Google Chrome. Either:

- Select the **Chrome menu** ≡ at the top-right of your browser window, then select **Tools** > **Developer Tools**.
- Right-click on any page element and select **Inspect Element**.

The DevTools window will open at the bottom of your Chrome browser.

There are several useful shortcuts for opening the DevTools:

- Use Ctrl + Shift + I (or Cmd + Opt + I on Mac) to open the DevTools.
- Use Ctrl + Shift + J (or Cmd + Opt + J on Mac) to open the DevTools and bring focus to the Console.
- Use Ctrl + Shift + C (or Cmd + Shift + C on Mac) to open the DevTools in Inspect Element mode, or toggle Inspect Element mode if the DevTools are already open.

Contents
- Accessing the DevTools
- The DevTools window
- Inspecting the DOM and styles
- Working with the Console
- Debugging JavaScript
- Improving network performance
- Audits
- Improving rendering performance
- JavaScript CSS performance
- Inspecting storage
- Further reading
- Further resources +

8. Dynamically loading resources

8.1 Problem description

pdfChip converts the HTML DOM into a PDF document. In order to do so, it uses the WebKit engine. As this engine has been developed for use online, and thus high-latency environments (environments where all resources aren't necessarily quickly available), it does what is called 'delay-loading'. This means that it 'finalizes' the DOM, while in the background waiting until resources are downloaded. Once they are available, the DOM is updated on-the-fly.

This works particularly well for websites; you don't want to stare at a blank window until every last image has been downloaded for example. It works less good for some pdfChip templates.

The problem caused by cchip.printPages

The JavaScript function:

```
cchip.printPages()
```

Click to copy

causes pdfChip to take the HTML DOM as it is at that specific moment, and convert it into PDF. This can interfere with the delay-loading principle described above. If certain resources have not yet been loaded at that moment in time, they are not available for the conversion to PDF. This may mean that certain elements are missing from the generated PDF file.

This problem is mostly visible with images, and barcodes generated by pdfChip.

HTML objects with parameters

There is a similar, though different, issue with HTML objects that have parameters, when those parameters are dynamically changed in the DOM. This is of primary importance for barcodes when it comes to pdfChip. This chapter also describes a solution for this issue.

Differences between pdfChip versions

To compound problems, there is a difference in how this can be solved depending on the specific version of pdfChip (or pdfToolbox which also uses the pdfChip engine for certain functionality). This chapter focuses on solutions for:

- pdfChip versions of 2.6 and higher
- pdfToolbox versions of 16 and higher

The last article in this chapter describes a (less powerful and more complex) technique that can be used in older versions of the applications. You are highly advised to upgrade to at least the versions mentioned above if you want to dynamically load resources.

8.2 Dynamic barcodes

pdfChip includes a barcode library that allows creating high-quality barcodes by inserting special HTML objects into the DOM. Those can be static, but they can also be dynamically created. Often, the easiest solution is to have a 'template' barcode object in the HTML, and then modifying its parameters using JavaScript.

The HTML would look like this:

```
<object id="barcodeObject" class="qrcode" type="application/barcode"
        style="width:15mm; height:15mm;">

        <param name="type" value="QR-Code">
        <param id="barcodeValue" name="data" value="Sample Da-
ta">
</object>
```

Click to copy

This HTML instructs pdfChip to create a QR code, with the data that is encoded set to "Sample Data". The following JavaScript then modifies the data for the QR code.

```
document.getElementById('barcodeValue').value = "Penguins are great!";
```

Click to copy

The function '`getElementById`' finds the correct object in the HTML DOM. Then, its value is modified to the value we really want to have in the generated PDF. There is, however, a problem with that.

Modifying parameters of objects

The problem is that WebKit doesn't realize something changed to the DOM, when you use the above JavaScript code. So in your final PDF, you'll end up with the default value ('Sample data'), rather than what you set in JavaScript.

To fix that, you need to jolt WebKit a bit so it understands something changed and uses the new parameters. You can do this quite simply with the following line of JavaScript:

```
document.getElementById('barcodeObject').innerHTML = document.getElementById('bar-
codeObject').innerHTML;
```

Click to copy

This code looks weird, because it sets the HTML content of the 'barcodeObject' object, to the *same thing*! But that's intentional; this is sufficient to make WebKit realize something changed to the DOM and for it to pick up the new values you set up using JavaScript.

Multiple barcode parameter changes

If you want to modify a barcode multiple times (for example in a loop) and then call '`printPages`' to output more pages to the output PDF, you have to use this trick each time. The sequence should be:

- Modify barcode parameters.
- Assign `innerHTML` trick to jolt WebKit into action.
- Use '`printPages`' to output additional page(s).

8.3 Making sure the DOM is ready

This page is valid only for pdfChip 2.6 and up, or pdfToolbox 16 and up.

As explained before, the main problem with dynamically altering the HTML DOM, is that not all resources may be available when you're telling pdfChip to create PDF content. This is mostly visible for content that the WebKit engine builds dynamically. For pdfChip, this includes loading images from either online, or on-site sources (from a URL, or from a file), and creating barcodes using the built-in barcode engine.

It's important to realize that this problem is a timing issue. As such, it may not *always* occur in a template. You could have a template with 2 barcodes, where one loads correctly while the other one doesn't load at all.

It's also important to realize that an image not appearing in the final PDF, for example, might not be caused by this issue at all. You may have an error in the image's 'scr' parameter, the placement of the image may be wrong... Dynamic resource loading is just one of the issues that may cause this.

If the WebKit engine not being ready is the issue, how do we fix that? Luckily this is quite easy. The code snippets below show the `cchipPrintLoop` function before and after fixing the issue.

Fixing the print loop

Before

A typical print loop function may look like this:

```
function cchipPrintLoop() {
```

```
        // Modify the src of an image dynamically
        document.getElementById('myImage').src = './images/penguin.jpg';

        // Asking pdfChip to print
        cchip.printPages();
}
```

Click to copy

The code defines the '`cchipPrintLoop`' function that is called automatically by pdfChip. This function proceeds to:

- Get the image element by using its unique ID as set in the HTML,
- Change the '`src`' attribute to the path of the image we want to use,
- Tell pdfChip we are ready and want to convert the DOM to PDF.

After

To fix the dynamic resource loading problem, modify the code to this:

```
async function cchipPrintLoop() {

        // Modify the src of an image dynamically
        document.getElementById('myImage').src = './images/penguin.jpg';

        // Asking pdfChip to print
        await cchip.printReadyPromise();
        cchip.printPages();
}
```

Click to copy

Two changes have been done in this code, and they are both necessary:

- The '`printReadyPromise`' function is called, *after* the DOM was modified and *before* pdfChip is instructed to create PDF.
- Because this function is called using the '`await`' key-word in front of it, there needs to be an '`async`' key-

word before the function definition for the print loop function.

The async / await paradigm in JavaScript is one possible way to deal with asynchronous JavaScript code and essentially convert it to synchronous code. If you don't understand what this means or how it is used here, just copy the syntax as stated above.

When dealing with more complex HTML templates, it is a good idea to ensure you understand the principles of asynchronous JavaScript code, and the importance of the async / await paradigm.

8.4 An older solution (before pdfChip 2.6)

This article is valid for all pdChip versions, but a better solution is available if you are using pdfChip 2.6 or higher. The full description of that is in the previous article here.

A way to resolve the problem of dynamic image loading is the cchip.onPrintReady() function that is built into pdfChip.

The cchip.onPrintReady() function

cchip.onPrintReady(f) installs a callback function f() that is called when the DOM is ready for printing, e.g. all images are loaded. The normal way to use this function is to first manipulate the DOM, then call cchip.onPrintReady(f) that calls f() when the DOM is ready and exit the cchipPrintLoop(). The function f() must call cchip.printPages() in order to actually create PDF pages from the DOM and initiate further DOM manipulations and printing if required.

The following example illustrates how this function can be used.

```
<html>
    <head>
        <script>
            function cchipPrintLoop(){
                var img = document.getElementById("myimg");
                img.src = "files/image.jpg";
                cchip.onPrintReady( cchip.printPages );
            }
        </script>
        </head>
        <body>
            <img id="myimg" src="files/2.png">
        </body>
</html>
```

Click to copy

The cchipPrintLoop() function is used to place an image (image.jpg) into the DOM. Instead of directly calling cchip.printPages it calls the cchip.onPrintReady function that installs cchip.printPages as a callback function which makes sure that it will only be used after all images have been loaded.

Single_image_load.zip

Dynamically load images

cchip.onPrintReady has, however, one severe problem: It cannot be used in a loop, so that iterating over a number of images would not work. Instead a more complex way has to be used.

```
<html>
        <head>
                <script type="text/javascript">

                           function setImage(i) {
                                if ( i != 1 ) {
                                        cchip.printPages()
                                }
                                if (i<9) {
                                        var img = document.getElementById("myimg");
                                        img.src = "files/" + i + ".png";
                                        cchip.onPrintReady(function () {setIm-
age(i+1)} );
                                }
                        }
                        function cchipPrintLoop() {
                                setImage(1);
                        }

            </script>
        </head>
        <body>
            <img id="myimg" style="width:100mm" src="files/2.png">
        </body>
</html>
```

Click to copy

In this example the cchip.PrintLoop calls a setImage function with a parameter 1. For this first time (i=1) cchip.printPages is not called but the image object on the page is replaced with 1.png and then cchip.onPrintReady calls this function again for the next page. Since cchip.onPrintReady waits until all images are loaded this will only take place after the image is in the DOM and then cchip.printPages will be executed as the first statement in the next setImage run.

Dynmically_load_images.zip

Dynamically load images using an array

The previous example is not ideal because image names need to have numbers to address them. The example below uses an array instead that lists all image paths, so that images may have arbitrary names.

It is even possible to use this example and put arbitrary objects into the "data" array and they will be placed into the corresponding CSS ID. Only the type of object identified by the ID has to work with the entries of the data array; in this example it has to be an image.

Dynamically_load_images_from_an_array.zip

Dynamically load images using cchipUtils.js

This example uses a utility JavaScript that you may use in your HTML instead. That allows you to get rid of the rather complex code inside of your own JavaScript but to only reference the utility script instead. The utility script provides a function "cchip.modifyAndPrintDom" that can be used inside of a cchipPrintLoop.

```
<html>
```

```
        <head>
            <script src="js/cchipUtils.js"></script>
            <script type="text/javascript">
                    function setImageArg( urlString ) {
                        document.getElementById("myimg").src = urlString;
                    }
                    function cchipPrintLoop() {
                            cchip.modifyAndPrintDom( setImageArg,
                                    ["files/icon_pdfchip_l_white.png"
                                ,"files/icon_pdfchip_s.png"
                                ,"files/icon_pdfchip.png"
                                ,"files/icon_pdfchip_m.png"
                                ,"files/icon_pdfchip_s_white.png"
                                ,"files/icon_pdfchip_l_white.png"
                                ,"files/icon_pdfchip_m_white.png"
                                ,"files/icon_pdfchip_l.png"] );
                        }
                </script>
        </head>
        <body >
                <img id="myimg" style="width:100mm" src="files/image.jpg">
        </body>
</html>
```

Click to copy

cchip.modifyAndPrintDom takes the name of a function that modifies the DOM as it's first parameter and an array of objects as it's second parameter. The array of objects should contain all objects that are to be feeded into the modifying function for modifying the DOM. The modifying function, in this example "setImageArg", has to have a parameter as placeholder for the new content. The function will be called for each item in the array with the respective item as parameter.

The main purpose of cchip.modifyAndPrintDom is to make sure that the DOM is fully updated with any modifications before it creates a PDF output from the current state of the DOM.

Dynamically_load_images_using_cchipUtils.zip

9. Optional content + Processing steps in pdfChip

9.1 Optional content (Layers) in pdfChip

Specifying optional content (commonly named as "Layers") in CSS is done in two steps:

1. define optional content rules, for optional content groups (OCGs), optional content group nodes (OCG nodes) and/or optional content membership dictionaries (OCMDs) by using pdfChip specific syntax
2. set style attributes for HTML elements or through CSS classes and IDs via names defined in the optional content rules

A simple example is shown here:

```
<!DOCTYPE html>
<html lang="en">
        <head>
                <meta charset="UTF-8">
                <title> Optional content example </title>
                <style>
                        @-cchip-pdf-ocg-node{
                                        -cchip-pdf-ocg-display-name: "Show/hide
paragraphs";
                                        -cchip-pdf-ocg-name: "paragraphs";
                        }
                        p {
                                -cchip-pdf-ocg: 'paragraphs';
                        }
                </style>
        </head>
        <body>
                <h1>This is a heading. </h1>
                <p>This is some text in a paragraph. It is only shown if visibili-
ty for the layer "Show/hide paragraphs" is turned on.
        </body>
</html>
```

Click to copy

In the attached ZIP archive two examples are included, one very simple example, and a second example demonstrating handling of OCGs in imported PDF pages. At least pdfChip 1.4

(December 2017) is required to process the optional content related features.

Optional_content_with_pdfChip_(2_examples).zip

Optional content rules

Optional content groups: @-cchip-pdf-ocg

An optional content group rule establishes an *optional content group* (often referred to as *layer*).

```
@-cchip-pdf-ocg {
        /* attributes: */
        ...
}
```

Click to copy

The attributes shown in the table below are available to define properties of an optional content group:

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
-cchip-pdf-ocg-display-name	**Name** entry in an Optional Content Group Dictionary (ISO 32000-1, Table 98)	*(optional)* a string providing the display name for the OCG
-cchip-pdf-ocg-name	*no equivalent in PDF syntax; only used for associating styles with OCG rules*	*(required)* a string providing the name for the OCG to be used in CSS style attributes
-cchip-pdf-ocg-visibility	inserts reference to the OCG in the **ON** array of the default Optional Content Configuration DIctionary	(optional; default: *on*) a value setting the visibility of the OCG; possible values are *on* and *off*; the default value is *on*

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
	(OCCD) (ISO 32000-1, Table 101)	
-cchip-pdf-ocg-parent-name	*no equivalent in PDF syntax; only used in optional content rules for associating an OCG with a parent OCG*	*(optional)* a string providing the name of the OCG or OCG node that is the parent of this OCG; this requires that a separate OCG or OCG node is defined that serves as a parent for this OCG; this feature makes it possible to define nested lists of OCGs that can be displayed as a hierarchy in a user interface.
-cchip-pdf-ocg-rbgroup	**RBGroups** entry in default Optional Content Configuration Dictionary (ISO 3200-1, Table 101)	*(optional)* a positive integer (i.e. 1 or greater) identifying the radio button group to which this OCG belongs. All OCGs that have the same *-cchip-pdf-ocg-rbgroup* number will belong to the same radio button group. A value of -1 indicates that the OCG does not belong to any radio button group; the default value is *-1*
-cchip-pdf-ocg-gts-procsteps-group	**GTS_ProcStepsGroup** entry in a **GTS_Metadata** dictionary (ISO 19593-1, Table 2)	*(optional)* a string providing the metadata entry for the OCG's processing steps *group* as defined in ISO 19593-1, "Graphic technology — Use of PDF to associate processing steps and content data – Part 1: Processing steps for packaging and labels"; see below for a list of pre-defined values
-cchip-pdf-ocg-gts-procstep-stype	**GTS_ProcStepsType** entry in a **GTS_Metadata** dictionary (ISO 19593-1, Table 2)	*(optional)* a string providing the metadata entry for the OCG's processing steps *step* as defined in ISO 19593-1, "Graphic technology — Use of PDF to associate processing steps and content data – Part 1: Processing steps for packaging and labels"; see below for a list of pre-defined values

An OCG will only be created in the resulting PDF, if there is actually content using it. The presence of an @-cchip-pdf-ocg alone is not sufficient.

Example for a @-cchip-pdf-ocg rule:

```
@-cchip-pdf-ocg{
        -cchip-pdf-ocg-display-name: "Show / hide this optional content"
```

```
        -cchip-pdf-ocg-name: "example-ocg";
        -cchip-pdf-ocg-visibility: on;
}
```

Click to copy

Optional content group nodes: @-cchip-pdf-ocg-node

An optional content group node rule establishes an *optional content group node.* Such a node is not in itself associated with any optional content, but serves as a node in the hierarchical display of optional content groups in a user interface.

```
@-cchip-pdf-ocg-node {
        /* attributes: */
        ...
}
```

Click to copy

The attributes shown in the table below are available to define properties of an optional content group node:

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
-cchip-pdf-ocg-display-name	optional first entry (of type text string) in an **Order** array (or sub-array) in the default Optional Content Configuration Dictionary (ISO 32000-1, Table 101)	*(optional)* a string providing the display name for the OCG node
-cchip-pdf-ocg-name	*no equivalent in PDF syntax; only used for associating styles with OCG node rules*	*(required)* a string providing the name for the OCG node to be used in CSS style attributes
-cchip-pdf-ocg-parent-name	*no equivalent in PDF syntax; only used in OCG node rules for*	*(optional)* a string providing the name of the OCG that is the parent of this OCG node; this requires that an OCG is defined that serves as a parent for this OCG

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
	associating an OCG node with a parent OCG	node; this feature makes it possible to define nested lists of OCGs that can be displayed as a hierarchy in a user interface.

Example for a @-cchip-pdf-ocg-node rule:

```
@-cchip-pdf-ocg-node{
        -cchip-pdf-ocg-display-name: "Example of a node (without OCG)";
        -cchip-pdf-ocg-name: "example-ocg-node-name";
}
```

Click to copy

Optional content membership dictionaries: @-cchip-pdf-ocmd

Optional Content Membership Dictionaries (OCMDs) make it possible to go beyond associating some content with a specific OCG. Using OCMDs it is possible to associate content (and its visibility) with the visibility of OCGs. A simple use would be to make some content visible if at least one of three OCGs is visible, and to turn it off if all three OCGs are off.

```
@-cchip-pdf-ocmd {
        /* attributes: */
        ...
}
```

Click to copy

The attributes shown in the table below are available to define properties of an optional content group node:

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
-cchip-pdf-	P entry in an Option-	*(optional)* a value defining the visibility policy for con-

Attribute	Matching optional content property in PDF 1.7 (ISO 32000-1)	Description
ocmd-policy	al Content Membership Dictionary (ISO 32000-1, Table 99)	tent belonging to this OCMD; possible values are *anyon*, *anyoff*, *allon*, *allof*, the default is *anyon.*
-cchip-pdf-ocmd-name	*no equivalent in PDF syntax; only used for associating styles with OCMD rules*	*(required)* a string providing the name for the OCMD to be used in CSS style attributes
-cchip-pdf-ocg-list	OCGs entry in an Optional Content Membership Dictionary (ISO 32000-1, Table 99)	*(optional)* a list of strings providing the names of the OCG that are associated with this OCMD

```
@-cchip-pdf-ocmd {
        -cchip-pdf-ocmd-policy: allon; /* allon, alloff, anyon(default), anyoff */
        -cchip-pdf-ocmd-name: "example-ocg-group";
        -cchip-pdf-ocg-list: "example-ocg-1" "example-ocg-2";/* space-separated
list of strings */
}
```

Click to copy

Hierachies of OCGs

Hierarchies of OCGs are defined bottom up by having the child in a hierarchy reference its parent node by means of the -cchip-pdf-ocg-parent-name in an @-cchip-pdf-ocg or @-cchip-ocg-node rule. The following examples demonstrates a simply hierarchy with one top most OCG and two childs, with the second child having a child of its own.

```
/*parent node*/
@-cchip-pdf-ocg{
        -cchip-pdf-ocg-display-name: "Parent OCG entry";
        -cchip-pdf-ocg-name: "example-ocg-parent";
}
```

```
/*first child node*/
@-cchip-pdf-ocg{
        -cchip-pdf-ocg-display-name: "Child #1";
        -cchip-pdf-ocg-name: "example-ocg-child-1";
        -cchip-pdf-ocg-parent-name: "example-ocg-parent";
}
/*second child node*/
@-cchip-pdf-ocg{
        -cchip-pdf-ocg-display-name: "Child #2";
        -cchip-pdf-ocg-name: "example-ocg-child-2";
        -cchip-pdf-ocg-parent-name: "example-ocg-parent";
}
/*child of child node #2*/
@-cchip-pdf-ocg{
        -cchip-pdf-ocg-display-name: "Child of child #2";
        -cchip-pdf-ocg-name: "example-ocg-child-of-child-2";
        -cchip-pdf-ocg-parent-name: "example-ocg-child-2";
}
```

Click to copy

Importing PDF pages that already contain optional content

When importing PDF pages that already contain optional content, one of the following will happen:

- for any OCG in the imported PDF page for which there is also an OCG rule with the same display name in the PDF created by pdfChip, the properties defined in the OCG rule will override the properties defined in the imported PDF page; for example, if the visibility for an OCG in the imported PDF page is set to true, whereas it is set to false in the OCG rule, the visibility for this OCG will be false for the PDF created by pdfChip.
- if for an OCG in an imported PDF page there is not yet already an OCG rule with the same display name in the PDF created by pdfChip, the definition of the OCG in the imported PDF page will be copied into the PDF created by pdfChip. As a result, the properties for the OCG as they exist in the imported PDF page will then also exist in the same manner in the PDF created by pdfChip, excluding properties that define a relationship with other OCGs. For

example, the visibility of the OCG will be the same in the PDF created by pdfChip but if it belonged to a radio button group, or has a parent OCG node, neither of these two will be created in the PDF created by pdfChip. Furthermore, when two PDF pages are imported that each contain an OCG with the same display name, but with differing properties, the properties of the PDF page imported first will prevail.

10. Page size in pdfChip

10.1 Set page size in pdfChip

The PDF page size can be specified inside the CSS `@page{}` rule. The `size` CSS descriptor, used inside the `@page` rule, sets the dimensions and orientation of the box which is used to represent a page. If no `@page` rule is defined A4 will be used as default.

CSS Syntax for @page

The size can be defined by a keyword (e.g. A4) or by absolute dimensions. Below are some examples of @page rules to set the page size in pdfChip.

1 value: height = width

```
@page {
  size: 6in;
}


@page {
  size: 50mm;
}
```

Click to copy

2 values: width then height

```
@page {
  size: 4in 6in;
}


@page {
  size: 50mm 150mm;
}
```

Click to copy

Keyword values for absolute size

Any of the following values: A4, A5, A3, B5, B4, letter (8.5in x 11in), legal (8.5in x 14in), ledger (11in x 17in)

```
@page {
  size: A4;
}


@page {
  size: letter;
}
```

Click to copy

Combination of size and orientation

```
@page {
  size: A4 landscape;
}

@page {
  size: B5 portrait;
}
```

Click to copy

Setting the page size on the CLI

The earliest version with full support for `--force-page-size` and `--default-page-size` is pdfChip 2.6.094.

To set the page size of a PDF on the command line, one of the following arguments can be used:

Set the default page size

If no page size is specified in the CSS `@page{}` rule, A4 will be used as default size. To change the default value for the page size to a specified value, the following command line argument can be used:

```
--default-page-size="CSS syntax for @page"

Example:

--default-page-size="letter"
```

Click to copy

Override the page size set in @pages

If a page size is set in the CSS `@page{}` rule but needs to be overridden, the command-line argument `--force-page-size` can be used (this will set the page size even if `@page` rule does specify a page size):

```
--force-page-size="CSS syntax for @page"

Example:

--force-page-size="A4 landscape"
```

Click to copy

10.2 Very large page sizes with UserUnit

Previous versions of pdfChip were limited to page sizes of 200 inch by 200 inch (or 5.08m by 5.08m or 14400pt by 14400pt). Starting with pdfChip 1.4, it is now possible to create page sizes beyond that limit.
Internally this is possible by taking advantage of the "UserUnit" property in the PDF syntax. A UserUnit is essentially a scaling factor, and by its default value is 1. Setting it for example to 10 multiplies all dimensional values – in the page geometry boxes and also in all positioning values for objects in the page description - and thus enable a page size of 2000 inch by 2000 inch (or 50.8m by 50.8m). For detailed information about the UserUnit entry in the PDF syntax, please see ISO 32000-1, 8.3.2.3 User Space.

When not explicitly set, the UserUnit entry will be set by pdfChip to a suitable value on a page by page basis. For page sizes up to 200 inch by 200 inch, it will not be set explicitly and thus default to 1. For any page size beyond 200 inch by 200 inch, pdfChip will set the smallest possible UserUnit value that accommodates the page size and can be divided by 10.

Setting UserUnit entry explicitly

As UserUnit entries are associated with pages in PDF syntax, the @page rule in provides the place in CSS to define the desired UserUnit entry.

Various approaches are supported:

- `-cchip-force-userunit` simply enforces the indicated value, regardless of the page size chosen; it is important to make sure that the chosen page size can be accommodated based on the UserUnit value defined; for example a value of 2 implies that page sizes must not be larger then 11.6m by 11.6m (i.e. 2 times 5.08m by 2 times 5.08m)
- `-cchip-userunit-increment` will only lead to a UserUnit value other than the default value 1 if necessary (because the page size is larger than 200 inch by 200 inch

/ 5.08m by 5.08m); in that case a UserUnit value is chosen automatically that is a multiple of the value provided for -cchip-userunit-increment. For example, if the page size is 60m by 60m (i.e. larger than 50.8m by 50.8, which is the maximum at a UserUnit value of 10), 20 will be used as the UserUnit value.

Only one of these two properties should be used for any given @page rule.
pdfChip will use the values that are in effect during the cchip.printPages() call.

Example for enforcing a specific UserUnit value:

```
@page {
    -cchip-force-userunit: 2;
    -cchip-bleedbox: 10px 10px 480px 580px;
    -cchip-artbox: 20px 10px 480px 580px;
  }
```

Click to copy
Example for enforcing a certain multiple of automatically determined UserUnit values:

```
@page {
    -cchip-userunit-increment: 10;
    -cchip-bleedbox: 10px 10px 480px 580px;
    -cchip-artbox: 20px 10px 480px 580px;
  }
```

Click to copy

Sample file

UserUnits_20170122.zip

11. Zoom factor in pdfChip 1.4

11.1 Use of zoom factor for increased precision

By default, the internal precision in pdfChip for rendering HTML content is 1/97 inch. By implication this means that the position objects may be off by up to half of 1/96 inch, and dimensions, such as font size, may be off by the same amount. In many cases this does not have any negative impact. Where a higher precision is required, it can be set by using the --zoom-factor command line parameter.

Zoom factor command line parameter

The `--zoom-factor` command line parameter expects an integer as its value. The integer represent the factor by which the precision shall be multiplied. For example, a value of 10 leads to a precision of 1/960 inch (ca. 0,0264583 mm).

Example:

```
pdfChip --zoom-factor=10 input.html output.pdf
```

Click to copy

12. pdfChip tips&tricks

12.1 Useful code snippets for defining barcode objects, using pdfChip specific CSS, importing PDF pages, setting page size and other things

The content offered below implements a very simple idea but could turn out to be real time saver for users creating their own HTML, JavaScript and CSS to be processed by pdfChip: it compiles a bunch of code snippets that deal with pdfChip specific ways of using spot colors, defining barcode objects, or making use of custom pdfChip specific CSS, JavaScript functions, and so on.

It will be still be necessary to read the documentation, and to know what one is doing - but once one gets the hang of it, the only piece missing often is: exactly which syntax has to be used to do this... Just keep the snippets provided below (both in text form as well as a downloadable file) open in your text editor, and help yourself whenever there is the need.

pdfchip_Self-Service-Code-Snippets_2018-09-07.txt

Page size and geometry

```
@page {
    /* define ArtBox, TrimBox, BleedBox, CropBox as needed; optional */
    /* valid units are px, pt, in, mm */
    /* Note: the use of ArtBox is not recommended */
    -cchip-artbox: 10mm 110mm 210mm 297mm;
    -cchip-trimbox: 10mm 110mm 210mm 297mm;
    -cchip-bleedbox: 7mm 107mm 216mm 303mm;
    -cchip-cropbox: 0mm 100mm 230mm 317mm;
    /* Size of the page, equivalent to MediaBox, origin always at 0/0 */
    size: 230mm 417mm;
}
In order to work with page geometry boxes in JavaScript the syntax is:
    cchip.pages[i].artbox
```

```
            cchip.pages[i].bleedbox
            cchip.pages[i].trimbox
            cchip.pages[i].cropbox
        e.g.
        if (cchip.pages[1].bleedbox) {
            ...do something with bleedbox...
            }
```

Click to copy

Defining fonts

```
        /*
            Note: for reliable results
            - associate one font file with one font-family name
            - avoid any automatic or heuristic mapping of fonts or font faces
        */
        @font-face {
            font-family: 'SourceCodePro-Regular';
            src: url('./fonts/SourceCodePro-Regular.otf');
        }
        @font-face {
            font-family: 'OpenSans-CondLight';
            src: url('./fonts/OpenSans-CondLight.ttf');
        }
```

Click to copy

Placing PDF pages

```
The URL for PDF supports the following features:
   <URL>#page=<PAGE-NUM>&box=<BOXNAME>&boxadj=<LEFT>,<TOP>,<RIGHT>,<BOTTOM>

Place the first page of "sample.pdf"
   <img src="sample.pdf">

Places the second page of sample.pdf
   <img src="sample.pdf#page=2">

Places the second page of sample.pdf, imported page area is based on its TrimBox
```

```
   <img src="sample.pdf#page=2&box=trim">

Places the second page of sample.pdf, imported page area is based on its TrimBox,
slightly enlarged on left and right sides by 3mm
   <img src="sample.pdf#page=2&box=trim&boxadj=3mm,0mm,3mm,0mm">

Import PDF pages ... :
   in HTML:
   <img src="sample.pdf#page=2">
   in CSS:
   background:url("sample.pdf#page=2")
   background-image:url("sample.pdf#page=2")
```

Click to copy

Rotation and other transforms

```
        .rotated-45 {
            position: absolute;
            left: 20mm; bottom: 100mm;
            -webkit-transform: rotate(-45deg);
            -webkit-transform-origin: left bottom;
        }
/* recommended resource:
https://developer.mozilla.org/en-US/docs/Web/CSS/transform
*/
```

Click to copy

Setting pdfChip color definitions

```
.pdfchip_colordefinitions {
    /*
        DeviceCMYK examples
        provide 4 values in the range 0.0..1.0
    */
    color: cyan;
    color: -cchip-cmyk(1.0,0.0,0.0,0.0);
    color: magenta;
    color: -cchip-cmyk(0.0,1.0,0.0,0.0);
```

```
    color: yellow;
    color: -cchip-cmyk(0.0,0.0,1.0,0.0);
    color: black;
    color: -cchip-cmyk(0.0,0.0,0.0,1.0);

    /*
        Lab D50 example
        provide 3 values, first value in the range 0.0..100.0,
        the 2nd and 3rd value in the range -127.0 .. 128.0
    */
    color: pink;
    color: -cchip-lab(67.74,70.68,-10.31);
    /*
        DeviceGray example
        provide 1 value in the range 0.0..1.0
        Note: 0.0 is black, and 1.0 is white
    */
    color: gray;
    color: -cchip-gray(0.3);
    /*
        ICC based Gray example
        provide 1 value in the range 0.0..1.0
        Note: 0.0 is black, 1.0 is white
        first argument must be a valid path to the respective ICC profile
    */
        color: darkgray;
        cchip-icc-gray('../iccprofiles/gamma_2-2.icc', 0.3);
    /*
        ICC based RGB example
        provide 3 values in the range 0.0..1.0
        first argument must be a valid path to the respective ICC profile
    */
    color: green;
    color: -cchip-icc-rgb('../iccprofiles/sRGBIEC61966-2.icc', 0.0,0.5,0.0);
    /*
        ICC based CMYK example
        provide 4 values in the range 0.0..1.0
        first argument must be a valid path to the respective ICC profile
    */
    color: green;
    color: -cchip-icc-cmyk('../iccprofiles/PSOcoated_v3.icc', 1,0.0,1,0.0);
    /*
        spot color examples with CMYK alternate space
```

```
        provide
        - name of the spot color
        - 4 values in the range 0.0..1.0
        - [optional] tint value for the spot color
        Note: for DeviceGray or ICC based alternate spaces adjust per syntax from
above
    */
    color: orange;
    color: -cchip-cmyk('Spot P 34-8 C', 0.0,0.75,0.98,0.0);
    color: green;
    color: -cchip-cmyk('Forest Green', 0.81,0.0,0.92,0.22);
    color: purple;
    color: -cchip-cmyk('Deep Purple', 0.84,1.0,0.0,0.12);
    color: magenta;
    color: -cchip-cmyk('Magenta', 0.0,0.1,0.0,0.0);
    color: black;
    color: -cchip-cmyk('Die line', 0.0,0.0,0.0,1.0);
}
    /*
        First, define a -cchip-devicen rule to establish a DeviceN colorspace
        Consists of:
        • -cchip-devicen-name: name by which the DeviceN colorspace is referenced
inside CSS
        • -cchip-components: definitions of the colorants of the DeviceN color-
space via spot color definitions
        In order to actually use the color, see '-cchip-devicen' below
    */
    @-cchip-devicen{
        -cchip-devicen-name: "test-colorspace-name";
        -cchip-components:
            -cchip-cmyk('Cyan', 1.0, 0.0, 0.0, 0.0)
            -cchip-cmyk('Magenta', 0.0, 1.0, 0.0, 0.0)
            -cchip-cmyk('Yellow', 0.0 ,0.0 ,1.0 ,0.0)
            -cchip-cmyk('Black', 0.0, 0.0, 0.0, 1,0)
            -cchip-cmyk('Fifth colorant', 0.5, 0.5, 0.5, 0.0)
            -cchip-cmyk('Sixth colorant', 0.0, 0.5, 0.5, 0.2);
        }
.pdfchip_devicen_colorspace {
    /*
        DeviceN color example
        provide
        - name of the DeviceN color space
        - as many values in the range 0.0..1.0 as there are colornats in the Devi-
```

```
ceN color space
    */
    color: -cchip-devicen('test-colorspace-name', 1.0, 0.0, 0.0, 0.0, 0.0, 0.0);
}
```

Click to copy

Setting PDF ExtGState parameters (overprint, etc.)

```
        .pdfchip_extgstate {
                    -cchip-flatness-tolerance : 5.0;     /* >= 0.0; default: 1.0 */
                    -cchip-smoothness-tolerance: 0.0;    /* 0.0 … 1.0;  default:  -
1.0 */
                    -cchip-text-knockout: 1;             /* 0 or 1; default: 0 */
                    -cchip-overprint: 1;                 /* 0 or 1; default: 0 */
                    -cchip-overprint-mode: 0;            /* 0 or 1; default: 0 */
                    -cchip-stroke-adjustment: 1;         /* 0 or 1; default: 0 */
                    -cchip-rendering-intent:  perceptual;
                        /* absolute-colorimetric or relative-colorimetric or per-
ceptual
                           or saturation; default: relative-colorimetric */
                    opacity: 0.5                         /* 0.0..1.0; default: 1.0
*/
                    mix-blend-mode: multiply;
                    background-blend-mode: difference;
                        /*  normal | multiply | screen | overlay | darken | light-
en | color-dodge |
                            color-burn | hard-light | soft-light | difference | ex-
clusion | hue |
                            saturation | color | luminosity; default: normal
*/        }
        Example:
        .background-spot_orange-ICCbasedcmyk {
            -cchip-overprint: 1;
            -cchip-overprint-mode: 0;
            -cchip-rendering-intent: absolute-colorimetric;
            background-color: orange;
            background-color: -cchip-icc-cmyk('./ISO Coated v2 (ECI).icc', 'Or-
ange',0.0,0.8,0.8,0.0, 0.75);
        }
```

Click to copy

Barcode objects (for 1D and 2D codes)

```
======= Syntax example for barcode objects: =======
    <object type="application/barcode">
        <param name="data" value="123456789012">
        <param name="type" value="EAN 13">
        <param name="modulewidth" value="0.33mm">
        <param name="barwidthreduction" value="10%">
        <param name="textplacement" value="none">
    </object>
    - important:
        - height and width may be provided through CSS styling (e.g.:
style="width:30mm; height:30mm;")
        - for professional use it is best though to use modulewidth to define the
size
        - for any inkjet or similar processes, substantial barwidth redution will
normally be necessary!
        - need help or guidance? send an email to support@callassoftware.com
======= Types of barcodes / 2D Codes: =======
<param name="type" value="Code 11">
<param name="type" value="Code 2 of 5 Standard">
<param name="type" value="Code 2 of 5 Interleaved">
<param name="type" value="Code 2 of 5 IATA">
<param name="type" value="Code 2 of 5 Matrix">
<param name="type" value="Code 2 of 5 DataLogic">
<param name="type" value="Code 2 of 5 Industry">
<param name="type" value="Code 39">
<param name="type" value="Code 39 Full ASCII">
<param name="type" value="EAN 8">
<param name="type" value="EAN 8 + 2 Digits">
<param name="type" value="EAN 8 + 5 Digits">
<param name="type" value="EAN 13">
<param name="type" value="EAN 13 + 2 Digits">
<param name="type" value="EAN 13 + 5 Digits">
<param name="type" value="EAN/UCC 128">
<param name="type" value="UPC 12">
<param name="type" value="Codabar 2 Widths">
<param name="type" value="Code 128">
<param name="type" value="DP Leitcode">
<param name="type" value="DP Identcode">
```

```
<param name="type" value="ISBN 13 + 5 Digits">
<param name="type" value="ISMN">
<param name="type" value="Code 93">
<param name="type" value="ISSN">
<param name="type" value="ISSN + 2 Digits">
<param name="type" value="Flattermarken">
<param name="type" value="GS1 DataBar (RSS-14)">
<param name="type" value="GS1 DataBar Limited (RSS)">
<param name="type" value="GS1 DataBar Expanded (RSS)">
<param name="type" value="Telepen Alpha">
<param name="type" value="UCC 128">
<param name="type" value="UPC A">
<param name="type" value="UPC A + 2 Digits">
<param name="type" value="UPC A + 5 Digits">
<param name="type" value="UPC E">
<param name="type" value="UPC E + 2 Digits">
<param name="type" value="UPC E + 5 Digits">
<param name="type" value="USPS PostNet 5 (ZIP)">
<param name="type" value="USPS PostNet 6 (ZIP+cd)">
<param name="type" value="USPS PostNet 9 (ZIP+4)">
<param name="type" value="USPS PostNet 10 (ZIP+4+cd)">
<param name="type" value="USPS PostNet 11 (ZIP+4+2)">
<param name="type" value="USPS PostNet 12 (ZIP+4+2+cd)">
<param name="type" value="Plessey">
<param name="type" value="MSI">
<param name="type" value="SSCC 18">
<param name="type" value="LOGMARS">
<param name="type" value="Pharmacode One-Track">
<param name="type" value="PZN7">
<param name="type" value="Pharmacode Two-Track">
<param name="type" value="Brazilian CEPNet">
<param name="type" value="PDF417">
<param name="type" value="PDF417 Truncated">
<param name="type" value="MaxiCode">
<param name="type" value="QR-Code">
<param name="type" value="Code 128 Subset A">
<param name="type" value="Code 128 Subset B">
<param name="type" value="Code 128 Subset C">
<param name="type" value="Code 93 Full ASCII">
<param name="type" value="Australian Post Custom">
<param name="type" value="Australian Post Custom2">
<param name="type" value="Australian Post Custom3">
<param name="type" value="Australian Post Reply Paid">
```

```
<param name="type" value="Australian Post Routing">
<param name="type" value="Australian Post Redirect">
<param name="type" value="ISBN 13">
<param name="type" value="Royal Mail 4 State (RM4SCC)">
<param name="type" value="Data Matrix">
<param name="type" value="EAN 14 (GTIN 14)">
<param name="type" value="VIN / FIN">
<param name="type" value="Codablock-F">
<param name="type" value="NVE 18">
<param name="type" value="Japanese Postal">
<param name="type" value="Korean Postal Authority">
<param name="type" value="GS1 DataBar Truncated (RSS)">
<param name="type" value="GS1 DataBar Stacked (RSS)">
<param name="type" value="GS1 DataBar Stacked Omnidir (RSS)">
<param name="type" value="GS1 DataBar Expanded Stacked (RSS)">
<param name="type" value="PLANET 12 digit">
<param name="type" value="PLANET 14 digit">
<param name="type" value="Micro PDF417">
<param name="type" value="USPS Intelligent Mail Barcode (IM)">
<param name="type" value="Plessey Bidirectional">
<param name="type" value="Telepen">
<param name="type" value="GS1 128 (EAN/UCC 128)">
<param name="type" value="ITF 14 (GTIN 14)">
<param name="type" value="KIX">
<param name="type" value="Code 32">
<param name="type" value="Aztec Code">
<param name="type" value="DAFT Code">
<param name="type" value="Italian Postal 2 of 5">
<param name="type" value="DPD">
<param name="type" value="Micro QR-Code">
<param name="type" value="HIBC LIC 128">
<param name="type" value="HIBC LIC 39">
<param name="type" value="HIBC PAS 128">
<param name="type" value="HIBC PAS 39">
<param name="type" value="HIBC LIC Data Matrix">
<param name="type" value="HIBC PAS Data Matrix">
<param name="type" value="HIBC LIC QR-Code">
<param name="type" value="HIBC PAS QR-Code">
<param name="type" value="HIBC LIC PDF417">
<param name="type" value="HIBC PAS PDF417">
<param name="type" value="HIBC LIC Micro PDF417">
<param name="type" value="HIBC PAS Micro PDF417">
<param name="type" value="HIBC LIC Codablock-F">
```

```
<param name="type" value="HIBC PAS Codablock-F">
<param name="type" value="QR-Code 2005">
<param name="type" value="PZN8">
<param name="type" value="DotCode">
<param name="type" value="Han Xin Code">
<param name="type" value="USPS Intelligent Mail Package (IMpb)">
<param name="type" value="Swedish Postal Shipment Item ID">
```

Click to copy

Optional parameters for barcode objects

```
======= Optional parameters (default values are enclosed with **asterisks**):
=======
<param name="format" value="A##B###C&">
    String formatting
    Provides control over how strings are formatted. For more details see section
4.6 Format in the Barcode Reference.
<param name="modulewidth" value="0.33mm">
    **-1**, units: mm, in, mils, **pixel**
    Module width
    Provides control over the Module width. For more details see section 4.2 Mod-
ule Width in the Barcode Reference.
<param name="hres" value="600">
    **-1**
    Horizontal resolution
    Providing the horizontal resolution triggers an optimisation of the module
width for best possible consistency of bars and gaps in the barcode and thus the
barcode readability. For more details see section 4.2 Module Width in the Barcode
Reference.
<param name="vres" value="600">
    **-1**
    Vertical resolution
    Providing the vertical resolution triggers an optimisation of the module width
for best possible consistency of bars and gaps in the barcode and thus the barcode
readability. For more details see section 4.2 Module Width in the Barcode Refer-
ence.
<param name="textplacement" value="none">
    above, **below**, none
    Text placement
    Provides control over the positioning of the human readable text relative to
the barcode proper. Applies only to 1D codes.
```

```
<param name="textdistance" value="0.5mm">
    **-1**, units: mm, in, mils, **pixel**
    Text distance
    Provides control over the distance of the human readable text from the barcode
proper. Applies only to 1D codes.
<param name="bearerbars" value="topbottom">
    **none**, top, bottom, topbottom
    Bearer bars
    Provides control over the presence and position of bearer bars. For more de-
tails see section 3.3 Barcode Glossary, Bearer Bars, 6.1.43 ITF-14 and 6.1.66 UPC
SCS (Shipping Container Symbols)in the Barcode Reference.
<param name="bearerwidth" value="0.5mm">
    **-1**, units: mm, in, mils, **pixel**
    Bearer width
    Provides control over the width of bearer bars. For more details see section 3.
3 Barcode Glossary, Bearer Bars, 6.1.43 ITF-14 and 6.1.66 UPC SCS (Shipping Con-
tainer Symbols)in the Barcode Reference.
<param name="notchheight" value="0.5mm">
    **-1**, units: mm, in, mils, **pixel**
    Notch height
    Provides control over the notch height. For certain types of barcodes like e.g.
 "EAN 13", some of the bars are typically longer than the rest of the bars. This
parameter provides control over by how much they will be longer.
<param name="barwidthreduction" value="1%">
    **0**, units: %, mm, in, mils, **pixel**
    Bar width reduction (BWR)
    Provides control over the bar width reduction. For more details see section 4.
3 Bar Width Reduction (Pixel Shaving) in the Barcode Reference.
<param name="quietzoneleft" value="0.5">
    **0**
    Quiet zone left
    Provides control over the quiet zone on the left. For more details see section
4.4 Quiet Zone in the Barcode Reference.
<param name="quietzoneright" value="0.5">
    **0**
    Quiet zone right
    Provides control over the quiet zone on the right. For more details see sec-
tion 4.4 Quiet Zone in the Barcode Reference.
<param name="quietzonetop" value="0.5">
    **0**
    Quiet zone top
    Provides control over the quiet zone at the top. For more details see section
4.4 Quiet Zone in the Barcode Reference.
```

```
<param name="quietzonebottom" value="0.5">
    **0**
    Quiet zone bottom
    Provides control over the quiet zone at the bottom. For more details see sec-
tion 4.4 Quiet Zone in the Barcode Reference.
<param name="quietzoneunit" value="X">
    **X**, mm, in, mils, pixel. X: multiples of module width
    Quiet zone unit
    Provides control over the unit used for controlling the quiet zone. For more
details see section 4.4 Quiet Zone in the Barcode Reference.
```

Click to copy

Minimal boilerplate HTML file

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title> ->->-> useful title <-<-<- </title>
<!-- referenced CSS or script files (adjust as needed):
<link rel="stylesheet" href="./folder/file.css"  type="text/css" />
    <script src="./folder/file.js"></script>
-->
<script>
function cchipPrintLoop() {
cchip.printPages( 1 ); // limit output to 1 page
console.log ("One and only page created (output is limited to 1 page)");
}
</script>
    <style>
    body, * {
    background: transparent;
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    -webkit-box-sizing: border-box;
    }
        @page {
            size: 216mm 303mm; /* A4 page with room for 3mm bleed on each side) */
            -cchip-trimbox: 3mm 3mm 210mm 297mm;
            -cchip-bleedbox: 0mm 0mm 216mm 303mm;
```

```
            -cchip-cropbox: 0mm 0mm 216mm 303mm;
        }
    </style>

</head>
<body>
<div>
<p>Put some content here, or use JavaScript to generate some, or include some
SVG</p>
<svg width="100mm" height="100mm" viewBox="0 0 100 100">
<g transform="translate(20,-5) scale(0.7)" >
<rect x="0" y="0" height="100" width="100" style="stroke:none; fill: -cchip-cmyk(0.
2,0.0,0.0,0.0)"></rect>
<polygon
points="65,10 5,96 97,39 10,39 80,97"
stroke-linejoin="round"
style= "
fill:-cchip-cmyk('My spot color',0.4,0.0,0.8,0.0);
fill-rule:nonzero;
-cchip-overprint: 1;
stroke:-cchip-cmyk(0.1,0.9,0.0,0.0);
stroke-width:2;"/>
 </g>
</svg>
</div>
</body>
</html>
```

Click to copy

Minimal boilerplate SVG file

```
<?xml version="1.0" encoding="utf-8"?>
<svg version="1.1"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
     width="216mm" height="303mm">
<style type="text/css">
@page {
            size: 216mm 303mm; /* A4 page with room for 3mm bleed on each side) */
            -cchip-trimbox: 3mm 3mm 210mm 297mm;
            -cchip-bleedbox: 0mm 0mm 216mm 303mm;
            -cchip-cropbox: 0mm 0mm 216mm 303mm;
```

```
}
</style>
<defs></defs>
<rect x="1.5mm" y="1.5mm"
width="213mm" height="300mm"
stroke="-cchip-cmyk(1.0,0.2,0.0,0.0)"
fill="-cchip-cmyk(0.0,0.1,1.0,0.0)"
style= "-cchip-overprint: 1; -cchip-overprint-mode: 1;"
>
</rect>
<foreignObject x="25mm" y="20mm" width="80mm" height="80mm">
<object xmlns="http://www.w3.org/1999/xhtml"
type="application/barcode"
style="margin: 0; padding: 0;
background-color:-cchip-cmyk(0.0,0.0,0.0,0.0);
color: -cchip-cmyk(1.0,0.0,1.0,0.4);">
<param name="type" value="Data Matrix"> </param>
<param name="data" value="Created from SVG file."> </param>
<param name="modulewidth" value="1mm"> </param>
<param name="quietzoneleft" value="1"> </param>
<param name="quietzoneright" value="1"></param>
<param name="quietzonetop" value="1"></param>
<param name="quietzonebottom" value="1"></param>
<param name="quietzoneunit" value="X"></param>
</object>
</foreignObject>
</svg>
```

Click to copy

pdfChip-file-attachments.zip

13. pdfChip Switch app

13.1 Starting with the pdfChip app

The callas pdfChip app is a regular Enfocus Switch app. This means you have to install it from the Enfocus Appstore, just as you would install any other Switch app.

The app is free, but in order to use callas pdfChip with it, you'll have to make sure pdfChip is licensed with either a trial or full license. The app doesn't install callas pdfChip, rather it works with the callas pdfChip installed on your system, so you have to make sure the application is properly installed.

Adding the pdfChip app to a flow

Once the app is installed in Enfocus Switch, it will show up in the "Apps" group in the "Flow elements" pane. In order to use it in a flow, drag it onto the flow canvas.

The pdfChip app always requires at least one incoming connection, and requires exactly one outgoing connection. To know what you should send through the incoming connection, refer to the "Ways to use the app in a Switch flow" article. To get more information about specific properties of the app, refer to the "pdfChip app properties" article.

The pdfChip app always outputs the PDF file created from the supplied template. For each input file, the output always is a single PDF file upon succesful conversion.

Handling errors

If something goes wrong during conversion (which could be caused by the supplied JSON or HTML template, or could be pdfChip specific such as missing licenses), the input to the app is *failed*. This is Enfocus Switch speak for "the incoming job is moved to the problem jobs folder". The regular error handling practices of Enfocus Switch can be used from that point on.

Make sure to also consult the Switch log where you might find additional information to troubleshoot the issue.

13.2 Ways to use the app in a Switch flow

The creation of PDF documents with callas pdfChip needs two things in general: a template and payload (the data that will be loaded into the template). Accordingly, the way the app can be setup corresponds to where this information comes from. There are two main ways this can be setup and these are described in this article.

2.1.The template flows through the Switch flow

In this case, the Switch flow generates or picks up an HTML template and sends it through the flow. At some point, this template meets the pdfChip app in the flow, which converts the template to a PDF file that then travels on through the workflow.

In the pdfChip app, use the “Type of input data” property to enable this scenario. Set the property to:

- **Plain HTML/SVG file**, or,
- **Folder with index.html**

No payload

If you’re simply trying to convert HTML or SVG to PDF and your template doesn’t require additional input data to determine how the template looks like, you’re done. The pdfChip app simply converts your (static) template to PDF which ends up in the output folder.

Dynamic template

If your template requires additional input data, you can specify the path of a JSON file which will be made available to your template during processing in pdfChip. To use this feature, you must set the “Import data from JSON” property to “Yes”. This enables two additional properties.

- **Path to JSON file**: the full path to the JSON file containing the data you want to make available to your template. This usually is a file somewhere on the file system, but with a small Switch script expression could also be configured as the path to a dataset connected to the incoming HTML template.
- **Name**: determines the name of the variable in which the content of your JSON file will be imported. This determines how you're going to be able to use it inside of your template.

2.2.The data flows through the Switch flow

In this case, the Switch flows generates or picks up a JSON file and sends it through the flow. At some point, this JSON file meets the pdfChip app in the flow. The template is now fixed (the same for all files flowing through this app) and the incoming file is the payload. pdfChip uses both pieces of information to generate a PDF file that then travels on through the workflow.

In the pdfChip app, use the "Type of input data" property to enable this scenario. Set the property to:

- **JSON file**

You now also have to specify two additional properties:

- **Name**: determines the name of the variable in which the content of your JSON file will be imported. This determines how you're going to be able to use it inside of your template.
- **Path to HTML file**: the full path to the HTML file you want to use. Typically, this is a file somewhere on the file system. As this property supports variables and script expressions, a different template could be used for different jobs.

13.3 pdfChip app properties

This article describes all pdfChip app properties. It assumes that you are familiar with pdfChip and how it works, and it assumes you're equally familiar with Enfocus Switch, and how properties of tools, apps or configurators work.

1. Path to pdfChip

A required property that is used to set the path to the callas pdfChip command-line application. If you have different versions of pdfChip installed, you should select the one you want to use in Enfocus Switch.

2. Type of input data

Determines which type of job the pdfChip app will expect (refer to the article pdfChip app properties as well). This property allows three different settings.

2.1. Plain HTML/SVG file

Incoming jobs must be a single HTML or SVG file. The name of the input job doesn't matter. This file is used as the template for callas pdfChip.

2.2. Folder with index.html

Incoming jobs must be a folder. This folder *must* contain a file called "index.html". This HTML file is used as the callas pdfChip template. The folder may contain as many other files as necessary to support the HTML file. Subfolders with fonts, CSS files, scripts or anything else is allowed.

2.3. JSON file

Incoming jobs must be a single JSON file. Selecting this enables two additional properties:

- **Name**: determines the name of the variable in which the content of your JSON file will be imported. This determines how you're going to be able to use it inside of your template.
- **Path to HTML file**: the full path to the HTML file you want to use. Typically, this is a file somewhere on the file system. As this property supports variables and script expressions, a different template could be used for different jobs.

3. Underlay and Overlay file

Both properties require you to specify a single HTML file. If specified, these are used as parameters of the callas pdfChip "underlay" and "overlay" command line parameters. Refer to the callas pdfChip documentation to see how these allow advanced pagination techniques.

4. Import data from JSON

To make templates dynamic, calls pdfChip allows importing data from a JSON file and then makes it available to the template while it is being converted to PDF. Set this property to "Yes" if you want to use this functionality, or "No" if you don't need it. Using this functionality enables two additional properties you need to set:

- **Name**: determines the name of the variable in which the content of your JSON file will be imported. This determines how you're going to be able to use it inside of your template.
- **Path to JSON file**: the full path to the JSON file containing the data you want to make available to your template. This usually is a file somewhere on the file system, but with a small Switch script expression could also be configured as the path to a dataset connected to the incoming HTML template.

5. Maximum number of pages

Sometimes you know ahead of time how many pages your resulting PDF should have. Given that pdfChip uses an HTML formatting engine this can lead to problems if your page content comes close to the edge. Use this property to specify what the maximum number of pages is that the generated PDF file should have.

6. Remote content

In most cases, templates use only local data. Sometimes it may be necessary to allow a template to reach out and use remote data (pulled in from the Internet for example). As this is a potential security risk, you have to specifically allow that by setting this property to "Yes". Only do this if this is required in your flow.

7. Use License Server

If instead of licensing pdfChip locally, you use a callas License Server, set this property to "Yes" and use the "License Server" property to specify the URL (and port if necessary) of the License Server(s) you have running.

8. Additional CLI parameters

Use this property to specify any additional command line parameters that would not yet be supported by this pdfChip app through the use of properties.