CMP3 - Cross Media Publishing

October 23, 2012

http://www.cross-media.net

http://www.bitmotion.de

Bitmotion
René Fritz <r.fritz@bitmotion.de>

Contents

Chapter 1

Introduction

CMP3 is a system for data and content processing and transformation. In other words: it can produce PDF files from your data. But that's not the whole truth because producing PDF is just one example of the possibilities.

So what mean "cross media publishing"? The idea is to use your content produced and stored at one location and feed different channels to publish on several target platforms and media. CMP3 with it's modular design can be configured to retrieve content from any source, modify the data with processors using XSLT or PHP and finally transform the content in any target format.

The target format is not necessarily a print format like PDF, but could also be XML, HTML, Text, and many other.

For more information you may have a look at `http://www.cross-media.net`.

Chapter 2

Installation

While CMP3 is currently a TYPO3 extension, CMP3 itself is (or should be) independent from TYPO3. In principle CMP3 works without TYPO3.

2.1 As TYPO3 extension

Install the extension as usual. Please read the following sections for needed dependencies.

2.2 Standalone

Currently nothing available here. The plan is to move CMP3 to http://www.packagist.org at some point, so we will explain here how to install the composer package.

2.3 Dependencies

2.3.1 Tidy

Tidy is used for HTML processing and needs to be available as cli or as PHP module.

In Debian you can install the PHP module as follow:

```
# sudo apt-get install php5-tidy
```

2.3.2 XSLT

It is very likely that XSLT is needed in your CMP3 project, so this is a dependency.

In Debian you can install the PHP module as follow:

```
# sudo apt-get install php5-xsl
```

This will give you XSLT 1 support.

2.4 Optional dependencies

Following tools are not needed as long as none of the processors are used.

2.4.1 FOP

FOP is a XSL-FO processor which is used to process PDF files from XML data.

There are other (commercial) processors out there which could be easily integrated in CMP3 but that isn't available yet.

2.4.1.1 Installation

In Debian you can install the fop package. Version 1.0 or newer is recommended.

Download fop eg. https://launchpad.net/ubuntu/+source/fop/

install:

```
# sudo dpkg -i fop_1.0.dfsg2-6_all.deb libfop-java_1.0.dfsg2-6_all.deb
# sudo apt-get install -f
# sudo apt-get install libservlet2.4-java
```

You might need an updated version of libxmlgraphics-commons-java too.

2.4.1.2 Font install

With FOP version prior to 1.0 you need to create font metrics with

```
# fop-ttfreader FranklinITCBQ-Demi.ttf FranklinITCBQ-Demi.xml
```

add font in fop.xconf

HINT: Maybe the step with fop-ttfreader is no longer needed with FOP 1.0.

2.4.1.3 Hyphenation

CMP3 itself provides the hyphenation pattern for fop. Please have a look in the example fop.xconf how to use that.

2.4.2 PoDoFo Impose

This is used for imposition of PDF files. If you do not know what imposition is you might want to have a look here: http://en.wikipedia.org/wiki/Imposition.

Unfortunately the current Debian package lacks lua support for podofoimpose which means you have to build the package by yourself and enable lua.

#TODO more details

The tested version 0.9.1.

2.4.3 SAXON (XSLT 2)

Saxon is a XSLT processor which is needed when XSLT 2 is used.

#TODO

Chapter 3

Technical overview

We've learned in our projects and customer requests that it is just NOT possible to build one, three or five applications that would fit for most of our cross media projects. Therefore CMP3 is not an application. It's a tool you can easily use in your application. To make a tool that you can use in multiple projects you have to make it modular. So you have to identify the typical elements of a cross media application.

A cross media application typically has ...

- an element to fetch the data from a data source
- some processing of the data
- and finally a transformation to produce the target format like pdf

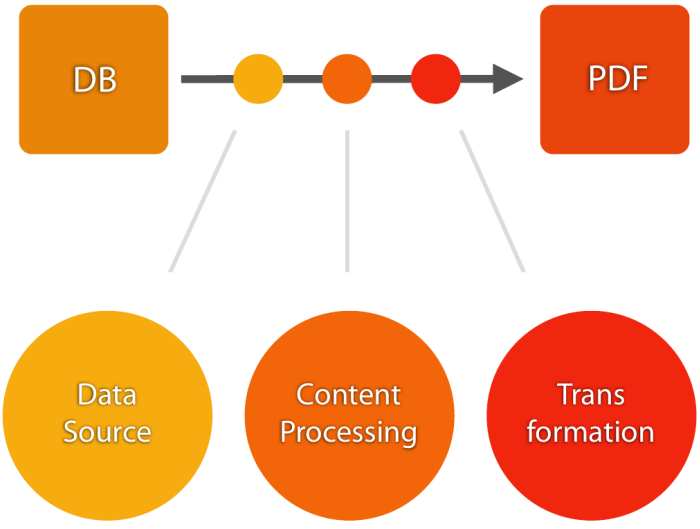

Figure 3.1: Processing parts of CMP3

And that's exactly the way we splitted the functionality in separate parts in CMP3 to make it modular and reusable.

3.1 Processing chains

As figure 3.1 suggests the whole process is organized in a chain. Those processing chains are the main concept of CMP3 and makes it very flexible.

Let's have a closer look at the parts of the processing chain.

3.1.1 Data Source

A data source have a fetcher which does the database queries for example. But a data source does also a normalization which means the data is transformed into an XML format[1]. That format doesn't only provide the data itself but also some meta information of the data types which could be integer, string or a date for example. Additionally the CMP3XML provides meta information about the format of the data which could be line, multiline, rich text, header and things like that.

3.1.2 Content processing

The next step in the processing chain is content processing. In most applications it is needed to do some content processing before the final transformation. In CMP3 you can use as many content processors as you want. Each processor returns the modified CMP3XML which is passed to the next processor and so on.

3.1.3 Transformation

When the processing chain is finished a transformation is done which produces the final output format like pdf.

3.1.4 Chain configuration

As you can see the whole process is organized in a chain. And what's so special in CMP3 is that those chains are not hardcoded in PHP but they can be defined using TypoScript[2]:

```
source.single_product = \Cmp3\Source\Typo3Xml

  fetcher = \Cmp3\Source\Typo3Query fetcher {

    selection {
```

[1] We will see in later examples that it is possible not to use CMP3XML but any other format.

[2] For those who don't use CMP3 in TYPO3 context, TypoScript is the configuration 'language' of TYPO3. We use TypoScript to define a processing chain even if we don't use TYPO3.

```
        10 = product
      }

      queries.product {
        table = tx_kmpproduct_product
        enableFields = default
        constraints {
          uid = {JobData:item}
        }
      }
    }
  }


  processing.single_product {

    10 = \Cmp3\ContentProcessing\XPath_Typo3RteRender
    10.xpath = //field[@format="typo3_rte"]

    20 = \Cmp3\ContentProcessing\XPath_Typo3TableRender
    20.xpath = //field[@format="typo3_table"]
    20.headerPos = left

    30 = \Cmp3\ContentProcessing\XPath_QRCode
    30 {
      xpath = //field[@name="print_url"]/value
      backgroundColor = #FFFFFF
      foreColor = #000000
      padding = 0
      moduleSize = 4
      ecclevel = M
    }
  }


  transformation.xml2pdf_fop = \Cmp3\Transformation\Fop
  transformation.xml2pdf_fop {
    stylesheet = EXT:myproject/leaflet_a4.xsl
    engine.fop.config = EXT:myproject/fop.xconf
  }
```

You can find configuration for a data source, processing and transformation. As you see there are names used like *single_product* or *xml2pdf_fop*. Giving them names means you can add more processing parts with other names.

3.1.5 Job

Finally we have to connect those parts in a so called „job configuration“. As you can see we use the names of configured data source, processing and transformation. This makes it easier to combine those elements to new chains. In a job we can also define multiple parts, and composer. Let's say you want to produce a document

which consists of two parts with different processing chains. You can easily add a part 20 here which produces a second PDF file for example.

A so called composer could be used to merge those files and produces one PDF file. In this example we use a composer which modifies the generated PDF to shrink it's file size. As you can see using TypoScript to define the processing chains makes the system very flexible.

```
job.MyProject.title = Leaflet A4
job.MyProject {
  parts {
    10.source = single_product
    10.preProcessing = single_product
    10.transformation = xml2pdf_fop
  }
  compose {
    10 = \Cmp3\Composer\PdfShrink
    10.quality = ebook
  }
}
```

For example to render a leaflet with a different layout only the transformation has to be replaced. This modular approach allows also to replace the transformation by another one that produces a different output format like InDesign. (icml, idml, tagged text). And of course you can define totally different jobs with other data sources and different processing to produce a great variety of target formats and layouts.

3.2 Running a job

So how do I use CMP3 in my own applications?

```
$objQueue = new \Cmp3\Job\Queue;
$objJob = $objQueue->CreateJob($strJob, $objConfig);
$objQueue->RunJob($objJob);
$objResult = $objQueue->GetResult($objJob);
```

This is the PHP code which is needed to call the CMP3 System and render a job. And while we are using TypoScript to define the job, it is not needed to change the PHP code for other transformations or layouts. If you need a processing which can't be done with the current system, just write a small Content Processor for your needs, add it to the processing chain in TypoScript and you are done.

3.3 Possibilities

What does this all mean? To show you *how* flexible CMP3 is, here is a more complex example.

We want to produce a document in PDF format which consist of four parts. Each part has different data sources and different content formats. We have different

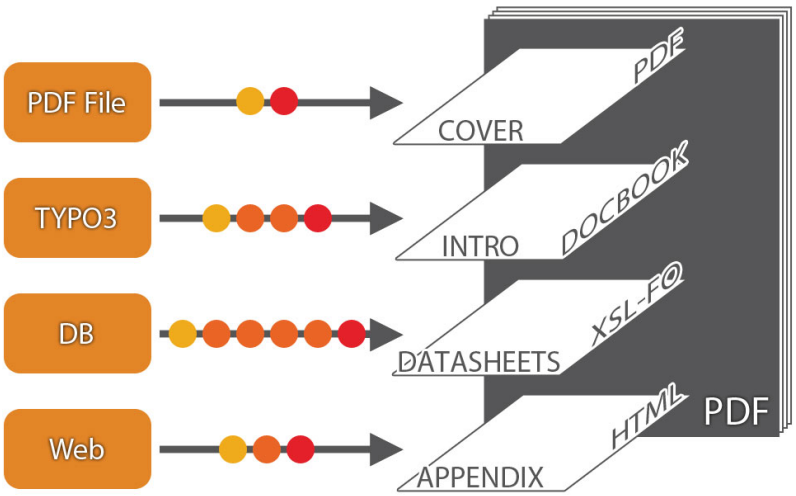

Figure 3.2: A little more complex CMP3 project

data sources like a PDF file, TYPO3, a database and some web content. For each of those sources we use different processing chains to produce PDF. Finally a composer merges the four parts into one PDF file.

3.3.1 Target formats

With CMP3 one can use XSLT or PHP to create a target format, which means (sometimes with the help of external tools) any target format is imaginable. Here are some examples.

- PDF
- XML
- DocBook
- Tex
- InDesign: Tagged Text, ICML, IDML
- HTML
- OpenDocument (ODF)
- RTF
- Text

3.3.2 Processing chains

Due to it's flexible concept a great variety of precessing chains are imaginable. To give you an impression what can be done we give here some examples:

- Data Source > CMP3XML > XSLT > XSL-FO > [FO Processor] > PDF
- Data Source > CMP3XML > XSLT > DocBook > XSL-FO > [FO Processor] > PDF
- Data Source > CMP3XML > XSLT > DocBook > RTF
- Data Source > CMP3XML > XSLT > DocBook > Word
- Data Source > CMP3XML > XSLT > DocBook > ePub

As you see all these chains use CMP3XML as intermediate format but that is not a requirement. You can build chains to process any format. Here are some examples:

- Data Source > HTML > MPDF > PDF
- Data Source > HTML > XSLT > XSL-FO> [FO Processor] > PDF
- Data Source > HTML > XSLT > XML
- Data Source > Text > PHP > Other text format
- Data Source > CMP3XML > XSL-FO > [FO Processor] > PDF
- Data Source > CMP3XML> XSLT > DocBook > ICML
- Data Source > CMP3XML > XSLT > ICML
- Data Source > CMP3XML > XML

You can even build chains that makes absolutely no sense. That means CMP3 is not responsible for a failing chain, it's all your fault. In other words, if anything goes wrong in a chain, the processing stops with an error. While it makes sense to show a web page even if one small box fails to show it's content, generating a print PDF with wrong or missing data is a big mistake and might costs thousands of Pesos.

3.4 Intermediate formats

In principle no intermediate format is necessary for any cross media application, but we want a solution were components can be reused so we don't have to start at zero for every application.

Having a limited set of intermediate formats makes it possible to provide a library of transformations that can be reused in multiple projects.

That was the main idea behind CMP3XML.

3.4.1 CMP3XML

CMP3XML is used as the main intermediate format in CMP3. The format includes all data enriched with meta data which gives information about the data type and format of the data. This helps in the later transformation to generate target document formats as PDF, HTML, XML, DocBook and others.

The cmp3 document is in most cases just the source for transformations and is not used directly in other applications. The transformations are done using XSLT. There are predefined transformations to generate DocBook for example.

3.4.2 DocBook

While CMP3XML - as the intermediate format - will be used in most cases to generate DocBook, DocBook itself can be seen as intermediate format too.

DocBook is a semantic markup language for technical documentation. It was originally intended for writing technical documents related to computer hardware and software but it can be used for any other sort of documentation. As a semantic language, DocBook enables its users to create document content in a presentation-neutral form that captures the logical structure of the content; that content can then be published in a variety of formats. In relation to DocBook semantic means that the documents elements are marked (markup) with tags to give the elements a meaning like header, paragraph or numbered list.

That given it is clear that DocBook is meant to be used for structured documents in contrast to CMP3XML which tries to provide as much semantic data as possible. But CMP3XML is not designed for structured documents. Therefore using CMP3XML to generate DocBook is not an requirement. If your data source could be transformed to DocBook more easily without CMP3XML, just do it.

That makes clear that DocBook might not be the right format for other document types like catalogs, mailings and so on. For specialized documents a direct transformation could be used instead of the intermediate DocBook format. On the other hand such a solution means to build a transformation from ground up and much more bigger task than customizing the DocBook transformation.

For DocBook rendering (transformation) multiple solutions exists to create HTML, PDF and other formats. Because of the sematic markup which is targeted to structured documents the generated output is a structured document in all output formats. Having the DocBook format a wide variety of output formats can be generated like PDF, TEX, HTML, RTF and others.

3.4.3 DITA

DITA is also a good candidate for an intermediate format but nothing was done in this direction.

Chapter 4

Application How To

Every CMP3 project is very specific so there is no predefined application to convert data x to output y. You can see CMP3 as a toolbox to create your own application.

Therefore any CMP3 application is encapsulated inside it's own folder or - in case of TYPO3 - in it's own extension. All files (except the CMP3 PHP files) that are needed for processing should be placed inside of that application folder - even if the same files are available inside of the CMP3 folder. This includes xsl, configuration, fonts, templates and so on.

While the API will not change easily the provided xsl files are meant as toolbox or starting point. They will be improved by time and therefore will be changed. Copying all needed files into your application will not break it when CMP3 is updated.

4.1 Examples

Currently no further documentation is available how to build your own CMP3 application, but we have examples.

4.1.1 Tests

Have a look in the *tests* folder for unit tests. While many of them are not interesting for application developers other tests whole jobs with processing and output.

Look into the *fixture* folder for source data and configurations and - after running the tests - look into the *output* folder for the results.

4.1.2 Demo

There's a demo application as TYPO3 extension called cmp3_demo. This is a backend module providing several examples you can try. Have a look in the source code of the module, the *fixture* folder for source data and configurations and the *output* folder for the results.

Chapter 5

Tips and Tricks

5.1 PDF with XSL-FO and FOP

5.1.1 Hyphenation

```
<fo:page-sequence
    master-reference="text-plain"
    id="rc_ucd">
    <fo:flow flow-name="xsl-region-body">
        <fo:block>
        <xsl:attribute name="language">
            <xsl:value-of select="//cmp3:record/@language" />
        </xsl:attribute>
```

5.1.2 Bleedbox (Margin)

TypoScript setup:

```
// cli parameter
engine.fop.parameter = -param bleed '3mm'
```

Add to you stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:fox="http://xmlgraphics.apache.org/fop/extensions"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:cmp3="http://www.bitmotion.de/cmp3/cmp3document"
    version="1.0">

    <xsl:param name="bleed"/>

        <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
```

```
<fo:layout-master-set>

    <fo:simple-page-master
        page-height="841.889763778pt"
        page-width="595.275590551pt"
        master-name="regular-odd">
        <xsl:attribute name="fox:bleed">
          <xsl:value-of select="$bleed" />
        </xsl:attribute>
```

5.1.3 PDF-X

TypoScript setup:

```
// cli parameter
engine.fop.parameter = -pdfprofile PDF/X-3:2003
```

5.1.4 Output device profile (icc)

fop.xconf:

```
<renderers>
    <renderer mime="application/pdf">
        <output-profile>./CoatedFOGRA27.icc</output-profile>
```

5.1.5 Meta data

To add meta data to your generated PDF add something like this to your stylesheet:

```
...
</fo:layout-master-set>


<fo:declarations>
    <x:xmpmeta xmlns:x="adobe:ns:meta/">
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
            <rdf:Description
                rdf:about=""
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <!-- Dublin Core properties go here -->
                <dc:title>
                    <xsl:call-template name="title" />
                </dc:title>
                <dc:creator>
                    <xsl:call-template name="web_address" />
                </dc:creator>
                <dc:description>
```

```
                </dc:description>
            </rdf:Description>
            <rdf:Description
                rdf:about=""
                xmlns:xmp="http://ns.adobe.com/xap/1.0/">
                <!-- XMP properties go here -->
                <xmp:CreatorTool>Bitmotion CMP3 - www.cross-media.net</xmp:CreatorTool
            </rdf:Description>
        </rdf:RDF>
    </x:xmpmeta>
</fo:declarations>
```

Chapter 6

Links

http://wiki.scribus.net/canvas/PDF,_PostScript_and_Imposition_tools