



Typst

I denna artikel kommer jag att använda mig av typsättningssystemet Typst, vilket är både ett språk och en kompilator som kan producera PDF och SVG (Man arbetar även på att stödja HTML som utformat)

Notera

En fiffig detalj med Typst-kompilatorn är att den innehåller en WebAssembly-host för plugins, dessa kan man använda sig av direkt ifrån sina .typ-filer.

Jag har publicerat en liten Zig-modul för att skapa sådana plugins med hjälp av Zig ⚡

Tanke

Härom dagen funderade jag på att det vore praktiskt om det gick att använda sig av Markdown som källa till ett (hyffsat generellt) dokument skrivet i Typst, denna tanke visade sig inte vara speciellt unik då jag relativt omgående hittade Typst-paketet cmarker vilket beskrivs som **“Transpile CommonMark Markdown to Typst, from within Typst!”**

Precis vad jag var ute efter!

Då var det bara att börja fila på document.typ för att realisera min tanke.

document.typ

Det första vi behöver göra är att importera cmarker från det globala namnutrymmet för Typst-paket.

(Vilket heter *preview* av **anledning**)

```
#import "@preview/cmarker:0.1.5"
```

Efter att vi importerat cmarker så behöver vi deklarerar en dictionary innehållandes lite konfiguration baserad på sys.inputs vilken populeras med hjälp av --input KEY=value flaggan till typst compile.

```
#let cfg = (  
  file: sys.inputs.at("FILE",  
    default: "document.md"),  
  rule: eval(sys.inputs.at("RULE",  
    default: "pagebreak")),  
  page: eval("(" + sys.inputs.at("PAGE",  
    default: "paper: \"a4\", margin: 0.5cm") + ")),  
  text: eval("(" + sys.inputs.at("TEXT",  
    default: "font: \"Inter\", size: 14pt") + ")),  
  code: eval("(" + sys.inputs.at("CODE",  
    default: "font: \"Office Code Pro D\", weight:  
    \"medium\", size: 1em") + ")),  
  line: eval("(" + sys.inputs.at("LINE",  
    default: "stroke: luma(220)") + ")),  
  table: eval("(" + sys.inputs.at("TABLE",  
    default: "stroke: luma(220), inset: 0.4em") + ")),  
  raw: (  
    box: eval("(" + sys.inputs.at("RAW_BOX",  
      default: "" + ")),  
    block: eval("(" + sys.inputs.at("RAW_BLOCK",  
      default: "inset: 1em") + ")),  
  ),  
)
```

Notera

Funktionen sys.inputs.at returnerar en str som jag konkatenerar mellan strängarna "(" och ")"

(då jag vill hantera dessa som dictionary-literaler, utan att behöva inkludera dessa i argumentet till --input)

Sen används eval för att konvertera dessa strängar till värden av typen dictionary.

Nu använder jag mig av ..spread-operatorn för att tillhandahålla argumenten till några olika set-regler

```
#set page(..cfg.page)  
#set text(..cfg.text)  
#set line(..cfg.line)  
#set table(..cfg.table)
```

För att formatera både kodblock och kod i brödtext på samma sätt så binder jag variabeln code med **#let**

```
#let code=text.with(..cfg.code)
```

Sen använder jag denna variabel i två show-regler som är filtrerade på om det är råtext i ett block eller inte.

```
#show raw.where(block: false): it =>  
  box.with(..cfg.raw.box)(code(it))  
#show raw.where(block: true): it =>  
  block.with(..cfg.raw.block)(code(it))
```

Nu är vi redo att använda oss av **#cmarker.render**.

Vi läser in en Markdown-fil med hjälp av read och vi sätter det kontext som ska vara tillgängligt vid parsningen.

- cfg: För att vi i <!--raw-typst --> kommentarer ska ha tillgång till konfigurationen.
- rule: Detta fält används för vad som ska hända med --- (Min konfiguration har standardvärdet pagebreak)
- code: För att vi i <!--raw-typst --> ska ha möjlighet att ändra hur kod formateras.
- image: Vi behöver korrigera hur cmarker hanterar sökvägar till bilder.

```
#cmarker.render(  
  read(cfg.file),  
  scope: (  
    cfg: cfg,  
    rule: cfg.rule,  
    code: code,  
    image: (path, alt: none) => image(path, alt: alt),  
  )  
)
```

Klart! 🎉

document.md

Vi behöver naturligtvis även skriva lite Markdown.

Notera

I kommentarer med formatet <!--raw-typst --> kan man skriva Typst för att styra hur man vill att dokumentet ska formateras. Men man behöver inte använda dessa om man enbart vill skriva Markdown.

```
<!--raw-typst  
#set page(  
  paper: "a4",  
  flipped: true,  
  columns: 2,  
)  
  
#set text(  
  size: 20pt,  
)  
  
#set quote(  
  attribution: [Peter Hellberg],  
)  
  
#show link: underline  
  
#show quote.where(block: true): box.with(  
  stroke: (left: 4pt + rgb("FF6600")),  
  width: 16em,  
)  
-->
```

Vi kan skriva Markdown och använda den i Typst

Med **__ett__** ~flertal~ `trevliga` [funktioner](https://athega.se/)

Undertitel

Lite brödtext[^footnote] som formateras som förväntat.

[^footnote]: En fotnot

> Ett klyftigt citat.

- En
- Lista

1. En
2. Ordnad
2. Lista

```
<!--raw-typst  
#colbreak()  
-->
```

Tabell

Språk	**Hemsida**
Go	<https://go.dev>
Zig	<https://ziglang.org>

Rita

Vi kan även använda oss av Typst för att rita med vektorer:

```
<!--raw-typst  
#circle(radius: 2em, fill: gradient.radial(white,  
rgb("FF6600")).sharp(3))  
-->
```

Eller ladda en SVG från fil och rita ut den:

![Ewok] (ewok.svg)

document.svg

För att generera följande SVG använder jag typst compile document.typ document.svg vilket resulterar i:

Vi kan skriva Markdown och använda den i Typst

Med **ett flertal** trevliga funktioner

Undertitel

Lite brödtext¹ som formateras som förväntat.

> Ett klyftigt citat.

— Peter Hellberg

- En
- Lista

1. En
2. Ordnad
3. Lista

Tabell

Språk	Hemsida
Go	https://go.dev
Zig	https://ziglang.org

Rita

Vi kan även använda oss av Typst för att rita med vektorer:



Eller ladda en SVG från fil och rita ut den:



¹En fotnot

Notera

Om du istället vill generera document.pdf så behöver du bara typst compile document.typ

/ Peter

När denna artikel skrevs var den aktuella versionen av typst v0.13.1