MAPS

NUMMER 49 • VOORJAAR 2019

NEDERLANDSTALIGE TEX GEBRUIKERSGROEP

Voorzitter
Hans Hagen
voorzitter@ntg.nl

Secretaris
Taco Hoekwater
secretaris@ntg.nl

Penningmeester
Ferdy Hanssen
penningmeester@ntg.nl

Bestuursleden
Frans Goddijn
Piet van Oostrum

Postadres
Nederlandstalige TEX Gebruikersgroep
Baarsjesweg 268-I
1058 AD Amsterdam

ING bankrekening
IBAN: NL53INGB0001306238
BIC: INGBNL2A

E-mail bestuur
ntg@ntg.nl

E-mail MAPS redactie
maps@ntg.nl

WWW
www.ntg.nl

De **Nederlandstalige TEX Gebruikersgroep (NTG)** is een vereniging die tot doel heeft de kennis en het gebruik van TEX te bevorderen. De NTG fungeert als een forum voor nieuwe ontwikkelingen met betrekking tot computergebaseerde documentopmaak in het algemeen en de ontwikkeling van 'TEX and friends' in het bijzonder. De doelstellingen probeert de NTG te realiseren door onder meer het uitwisselen van informatie, het organiseren van conferenties en symposia met betrekking tot TEX en daarmee verwante programmatuur.
De NTG biedt haar leden ondermeer:

- Tweemaal per jaar een NTG-bijeenkomst.
- Het NTG-tijdschrift MAPS.
- De 'TEX Live'-distributie op DVD/CDROM inclusief de complete CTAN software-archieven.
- Verschillende discussielijsten (mailing lists) over TEX-gerelateerde onderwerpen, zowel voor beginners als gevorderden, algemeen en specialistisch.
- De FTP server `ftp.ntg.nl` waarop vele honderden megabytes aan algemeen te gebruiken 'TEX-producten' staan.
- De WWW server `www.ntg.nl` waarop algemene informatie staat over de NTG, bijeenkomsten, publicaties en links naar andere TEX sites.
- Korting op (buitenlandse) TEX-conferenties en -cursussen en op het lidmaatschap van andere TEX-gebruikersgroepen.

Lid worden kan door overmaking van de verschuldigde contributie naar de NTG-giro (zie links); vermeld IBAN zowel als SWIFT/BIC en selecteer shared cost. Daarnaast dient via `www.ntg.nl` een informatieformulier te worden ingevuld. Zonodig kan ook een papieren formulier bij het secretariaat worden opgevraagd.
De contributie bedraagt € 35. Voor studenten geldt een tarief van € 18. Dit geeft alle lidmaatschapsvoordelen maar *geen stemrecht.* Een bewijs van inschrijving is vereist. Een gecombineerd NTG/TUG-lidmaatschap levert een korting van 10% op beide contributies op. De prijs in euro's wordt bepaald door de dollarkoers aan het begin van het jaar. De ongekorte TUG-contributie is momenteel $105.

Afmelding kan met ingang van het volgende kalenderjaar door opzegging per e-mail aan de penningmeester.

MAPS bijdragen kunt u opsturen naar `maps@ntg.nl`, bij voorkeur in LATEX- of ConTEXt formaat. Bijdragen op alle niveaus van expertise zijn welkom.

Productie. De Maps wordt gezet met behulp van een LATEX class file en een ConTEXt module. Het pdf bestand voor de drukker wordt aangemaakt met behulp van pdftex 1.40.19 en luatex 1.0.9 draaiend onder MacOS X 10.14. De gebruikte fonts zijn Linux Libertine, het niet-proportionele font Inconsolata, schreefloze fonts uit de Latin Modern collectie, en de Euler wiskunde fonts, alle vrij beschikbaar.

TEX is een door professor Donald E. Knuth ontwikkelde 'opmaaktaal' voor het letterzetten van documenten, een documentopmaaksysteem. Met TEX is het mogelijk om kwalitatief hoogstaand drukwerk te vervaardigen. Het is eveneens zeer geschikt voor formules in mathematische teksten.
Er is een aantal op TEX gebaseerde producten, waarmee ook de logische structuur van een document beschreven kan worden, met behoud van de letterzet-mogelijkheden van TEX. Voorbeelden zijn LATEX van Leslie Lamport, $\mathcal{AMS}$-TEX van Michael Spivak, en ConTEXt van Hans Hagen.

Contents

Van de penningmeester

In januari viel bij de meeste leden de factuur voor de contributie van 2019 in de digitale brievenbus. Daarop konden jullie zien dat het contributiebedrag dit jaar is verlaagd. De laatste contributiewijziging dateert van de invoering van de euro: in 2002 is de jaarcontributie op € 40 gezet.

Sinds 2006 schommelt het eigen vermogen van de vereniging tussen de € 19.000 en € 24.000. Dit is veel meer kapitaal dan de vereniging nodig heeft om te functioneren. De vereniging heeft tot 2015 nog behoorlijke rente-inkomsten gehad van dat kapitaal, maar daarna is de spaarrente gekelderd. Een eigen vermogen van € 10.000 is meer dan voldoende voor een vereniging als de onze, daarmee kan de vereniging in theorie 2 tot 3 jaar een jaarlijkse MAPS uitbrengen en de mail- en webserver in de lucht houden, zonder dat daar een cent contributie tegenover staat.

We hebben al een aantal jaar het beleid om het kapitaal te verminderen, maar dat heeft nog niet veel zoden aan de dijk gezet. Er wordt al een tijd € 3.500 per jaar gedoneerd aan fontprojecten. Maar omdat ook de overige kosten zijn teruggelopen, is het eigen vermogen nog niet significant afgenomen. Dit komt voornamelijk doordat er al een aantal jaar slechts een MAPS per jaar wordt uitgegeven. De MAPS is onze grootste kostenpost, naast de eerdergenoemde donaties.

De in 2015 afgesproken donatie van € 3.000 per jaar aan het Gyre fontproject loopt in 2019 af. Als de contributie niet zou wijzigen, zouden we in 2020 de situatie hebben dat er ca. € 6.000 binnenkomt aan contributie en ca. € 4.000 uitgaat aan de MAPS, de mail- en webserver, en de jaarlijkse bijeenkomst: een overschot van € 2.000.

Met de nieuwe contributie van € 35 per jaar betekent dit voor 2020 nog steeds een contributie-inkomst van ca. € 5.000. Om het eigen vermogen te doen slinken, is er dus nog ruimte voor een groot of een aantal kleine projecten waaraan one vereniging een geldelijke bijdrage kan leveren.

Het ledental slinkt ook nog steeds, dus de inkomsten zullen geleidelijk teruglopen. Op de e-maillijst die het bestuur gebruikt voor onderlinge discussie (in feite is die e-maillijst ons vergadermedium) hebben we het hier ook over gehad, voordat we het besluit namen de contributie te verlagen. We vinden het op dit moment verantwoord om de contributieverlaging door te voeren.

Nog een belangrijk punt om af te sluiten: tijdens de ledenvergadering van 20 mei 2009 ben ik als penningmeester in het bestuur gekozen. Dat betekent dat ik in 2021 genoodzaakt ben af te treden. Dit lijkt nog ver weg, maar ik heb de ervaring dat het altijd erg lastig is kandidaten te vinden. Ik zou het erg prettig vinden als er onder onze leden iemand is die het leuk zou vinden om vanaf 2021 het stokje over te nemen. Idealiter zou ik vanaf de ledenvergadering van volgend jaar een kandidaat wegwijs willen maken, zodat het stokje in 2021 moeiteloos kan worden overgedragen.

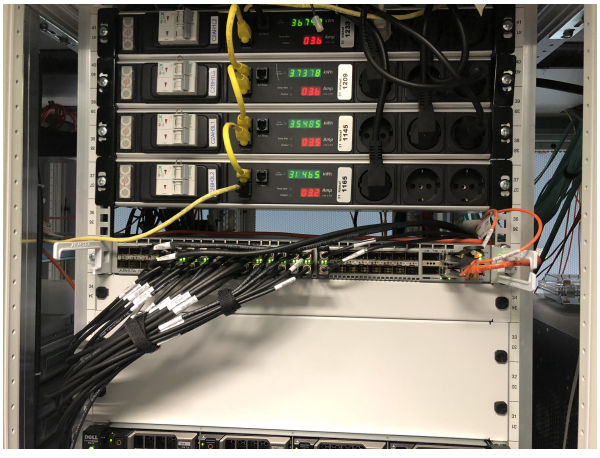
Ferdy Hanssen
9 februari 2019

Verslag 57ste NTG bijeenkomst

Een goede NTG bijeenkomst zet je aan het denken.

Je pikt niet zomaar wat tips op maar in plaats daarvan kan het gebeuren, zoals tijdens de 57ste NTG bijeenkomst, dat je inzicht krijgt in de manier waarop letters en andere typografische tekens worden ontworpen en hoe je, door extreme uitvergroting en het in beeld brengen van de rand-lijnen, ziet hoe raak- en scharnierpunten van wat in wezen kaarsrechte balkjes zijn, afgeronde knieschijven kunnen worden. Bij het dikker of cursief maken van een font komt er meer kijken dan vettere lijnen tekenen of de bovenzijde iets naar rechts duwen. Hans Hagen gaf daarover veel uitleg en lardeerde zijn betoog met tal van voorbeelden, onder andere terugblikkend op het jaren geleden door Adobe zeer ambitieus uitgerolde concept van Multiple Master fonts. Een fantastische gedachte waar helaas bar weinig van tot stand is gekomen. Daarop voortbordurend heeft Hans een methode geïmplementeerd waardoor we opnieuw een aantal fonts zelf kunnen manipuleren, vooropgesteld dat we een idee hebben wat we willen en dat we durven te besluiten welke van de vele variabelen we aan willen passen.

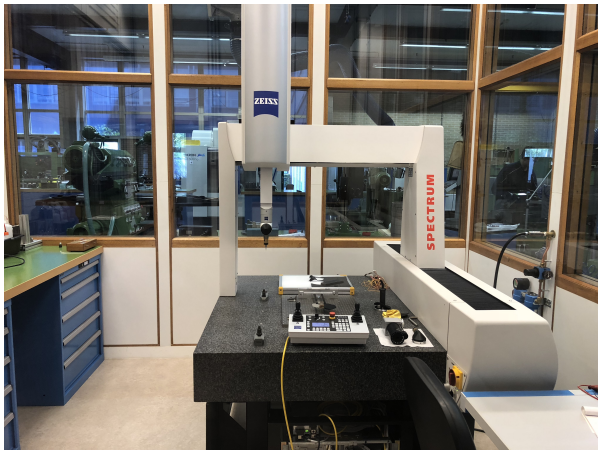
Tijdens de rondleiding van Dennis van Dok en een van zijn collega's van het Nikhef (Nationaal instituut voor subatomaire fysica) hebben we enkele ‚hot spots' mogen bekijken. Een daarvan was de huidige com-



puterruimte, waar vele honderden processoren roodgloeiend staan met aan een zijde ijskoude lucht en aan de andere kant een gangpad waar tropische warmte uit de computers blaast. Men streeft ernaar de totale energieconsumptie beneden de 400 kW te houden maar dat is niet eenvoudig.

Een verdieping lager is ooit de allereeerste internetverbinding van het Europese vasteland aangelegd. Nog steeds heeft het instituut er plezier van, ook al is een naburig ‚datahotel' gebouwd waar twee torens van elf verdiepingen tjokvol computers zijn geplaatst. Het in eigen huis houden van zo'n dataverbinding legt het instituut intussen geen windeieren.

Veel onderzoek wordt er elders in het gebouw gedaan naar zeer kleine deeltjes zoals neutrino's en in de hal staat een actieve kast waar ook de voorbijganger kan observeren hoe vaak er een neutrino door onze ruimte vliegt. De duistere kast heeft een installatie waardoor de binnenkomst en het vertrek van zo'n deeltje zichtbaar wordt als een blauwe flits en daarvan konden we er een flink aantal met eigen ogen vaststellen. Dergelijke deeltjes gaan razendsnel en ze hebben totaal geen hinder van objecten waar wij zelf ontzag voor hebben. Zo vliegt 'n neutrino met speels gemak aan de ene kant de aarde binnen om er met dezelfde snelheid



een ogenblik later aan de andere kant uit te ketsen. De vraagt dringt zich op of een neutrino net zoveel bravoure heeft indien deze door een vat helium bij het absolute nulpunt probeert te gaan. Dat moet eens worden nagevraagd.

Er worden ook bollen geproduceerd die er op het eerste gezicht uitzien als zeer hippe kwasi-industriële discolampen of miniatuurversies van diepzee-duikklokken en deze objecten ter grootte van een forse volleybal worden aan lange snoeren diep in de middellandse zeer afgezonken, waar vervolgens in alle rust, vrij van de ruis die je elders hebt, neutrino's kunnen worden betrapt en bestudeerd.

Een plek waar we ditmaal niet aan toe kwamen is de afdeling waar men scans kan maken van oudheden, zoals een bal uit de tijd van de farao's waarbij heel gericht diep in de materie kan worden geschouwd om na te gaan hoe men zoiets destijds vervaardigde, en daarbij kan het zomaar zijn dat zelfs de vingerafdruk diep in het voorwerp, in een van de lagen waaruit het destijds is opgebouwd, kan worden waargenomen.

Intussen is al het besluit gevallen dat we graag nogmaals bij het Nikhef te gast willen zijn, allereerst natuurlijk om over TEX kwesties en typografie te praten, maar ook om op de hoogte te blijven van hetgeen die slimme gasten daar uitpluizen.

Frans Goddijn

Naschrift van Dennis

Hoi Frans,

Er zijn een paar kleine correcties op details. Je weet hoe mensen daar op blijven letten nietwaar?

Het grote apparaat waar je naar verwijst die deeltjes zichtbaar maakt heet een vonkenkamer. Het is geen neutrinodetector (was dat maar waar!) want die zijn heel moeilijk te vangen. Het soort deeltjes wat deze vangt zijn andere vormen van kosmische straling, veelal afkomstig van botsingen van astrodeeltjes met de atmosfeer. Ze worden zichtbaar gemaakt doordat de deeltjes het gasmengsel in de kast enigszins ioniseert waardoor het gas elektrisch geleidend wordt, en de hoge spanning tussen de platen zorgt dan voor een vonk langs het pad van het deeltje.

Voor het vinden van neutrino's hebben we inderdaad de ‚bollen' gemaakt die als een parelsnoer op de bodem van de Middellandse Zee komen te staan.

Je verwijst naar het stroomverbruik van onze dataruimte; de 400 kWh is onze limiet, want het koelsysteem is er niet op gemaakt meer te kunnen wegkoelen. We zitten hier nog lang niet aan, maar we letten wel op dat we systemen kopen met een goede prestatie-per-watt-verhouding. Uiteindelijk is elektriciteit niet gratis, het is een aanzienlijke portie van ons totale budget. Ik denk dat je wel kunt stellen (zonder dat ik exacte cijfers heb) dat een computer in zijn bedrijfsjaren ongeveer evenveel kost als de initiële aanschaf.

Dennis van Dok, Nikhef

Impressie hackerskamp SHA2017

Abstract
Vorig jaar was er een groot hackerskamp op de Flevopolder. Omdat TEXnicians zich ook onder de naam "hackers" mogen scharen, was er dit jaar zelfs een heus "LaTEX village" aanwezig op het kamp. Een kort verslag van het kamp.

Keywords
hackerskamp, SHA2017

Een hackerskamp

Elke vier jaar wordt er een groots (ca. 3000 bezoekers) internationaal hackerskamp in Nederland georganiseerd. Een beetje verwarrend is het wellicht, dat elke iteratie een andere naam draagt. Ditmaal werd het kamp SHA2017 genoemd, een verwijzing naar de familie van cryptografische hashfuncties "SHA".

Voor onbekenden van het fenomeen "hackerskamp" kan het kamp worden omschreven als een congres voor hackers met lezingen, workshops en allerlei andere hackers- en nerdy-activiteiten. Veel van die lezingen gaan over computerbeveiliging, van academisch gepubliceerde papers of breed in de media uitgelichte hacks, tot individuen die hun eigen ervaringen delen over het beveiligen of kraken van systemen. Een ander groot gedeelte van de presentaties gaat over privacy, sommige objectief en informatief, maar uiteraard zijn er ook veel politiek gemotiveerd. Tot slot horen ook zeker lezingen over hardware hacking, ethiek of kunst bij een hackerscongres thuis.

Alhoewel de lezingen een groot deel van het programma uitmaken, kom je ook mensen tegen die nog niet naar één lezing zijn geweest. Dit komt omdat het kamp eigenlijk een groot sociaal evenement is. Veel hackers nemen hun zelfgebouwde hardware mee, stellen het tentoon en vertellen de voorbijgangers graag over hun creaties. Zo ontstaan levendige gesprekken over wat er nog verbeterd aan zou kunnen worden of over hoe je het zelf zou kunnen maken. Deze discussies leiden zo weer tot nieuwe ideeën of nieuwe creatieve projecten.

Het LaTEX village

Het waren heus niet alleen de hardware hackers die graag een gesprekje aangaan met andere bezoekers, ook software projecten waren alom vertegenwoordigd op het terrein met hetzelfde doel. Zo was ook de LaTEX *community* aanwezig met het LaTEX *village* als uitvalsbasis (in een *village* konden mensen met dezelfde interesse bij elkaar kamperen).

Het LaTEX *village* was georganiseerd door DANTE, de Duitse evenknie van NTG. Naast het bieden van een ontmoetingsplaats, werden er ook beginnerscursussen geven over het gebruik van LaTEX en kon je er diverse boeken over TEX en aanverwanten kopen. Erg leuk was het om mede-enthousiastelingen te ontmoeten en te kunnen keuvelen over onderwerpen als TEX en typografie ("ken je deze package/website al?"), maar ook over het feit dat er dit jaar nu eindelijk genoeg ruimte in het busje was om de koelkast mee naar het kamp kunnen nemen en dat dat toch echt wel bij een hackerskamp paste ("genoeg anderen die hun desktop meeslepen").

Merchandise

Toen mijn ogen vielen op de merchandise die DANTE had meegenomen, was ik direct verkocht. Uiteraard moest ik het T-shirt met de tekst "I'm a LaTEX fetishist!" hebben. Eindelijk kon ik nu mijn vrienden en familie laten zien dat ik inderdaad zo gek ben van dat afschuwelijke LaTEX als ze al vreesden. Vol trots liep ik met mijn nieuwe aanwinst het terrein over, wetende dat ik hier niet vreemd zou worden aangekeken in het midden van duizenden andere hackers met nerdy T-shirts van hun favoriete project.

Aangekomen bij de foodcourt, besloot ik een hapje te gaan eten. Terwijl ik vriendelijk te woord werd gestaan, maakte de foodtruck medewerker een opmerking over mijn T-shirt. Ik vroeg of hij ook LaTEX kende. "Ja, LaTeX", dat kende hij wel. Ah, een beginner, bedacht ik me, dat de X een hoofdletter χ is en je dus anders uitspreekt weet per slot van rekening niet iedereen. Nog nét voordat ik betaalde, realiseerde ik mij dat hij de combinatie "LaTeX" en "fetishist" héél anders interpreteerde. Ik probeerde nog gauw duidelijk te maken dat ik het wel over de "programmeertaal LaTEX" had en dat het "iets met computers" te maken had. Of het helemaal is overgekomen, betwijfel ik.

Tim van de Kamp
ntg at timvandekamp dot nl

Take Notes – Take Two

Notes handling module revised and extended

Abstract
Second, revised and extended, version of a module for processing of notes. Notes are classified according to category/subcategory and can contain information about subject, author, date, source, etc. The typesetting of the notes can be filtered according to several criteria. Many aspects of the formatting are easily configurable.

Introduction

Active email lists, as for example the ConTEXt-list, produce many emails discussing some problem or other topics. Often these discussions form threads chaining successive postings, where each posting incorporates much content of the previous one. Collecting all the emails for later reference produces a lot of redundancy. Moreover, the continuous repetition of previous content tends to make reading a tedious business.
Clearly better accessibility can be attained when the threads worth retaining are condensed to a single or a few successive notes, describing the gist of the discussion. This idea motivated the development of this module. It serves as a means to collect, store, select and reproduce the stored notes. Usage of this module is not restricted to storing email discussions, of course. It can be applied to many topics one deems worth remembering.
The above formed the motivation from which the first version of this module originated. The development of software pieces like this is in my case highly governed by the need of the moment. With new (self imposed) tasks at hand, new possibilities literally scream for implementation. This was the case when I turned my attention to researching the origins of my family. The mass of facts to be taken into account (hundreds in the archives of the 18th century alone) could be kept under control only when put into uniform records. Thus an adapted version came into existence as a spinoff from the original module: it could keep track of all persons of interest playing a role in the source documents. Because two nearly identical programs are a bad thing, especially with respect to bug fixing and maintenance, it was decided to combine them in a new and extended `takenotes` module. Also this offered a chance to straighten out the code and revert from a few bad structural decisions.

Overall structure

This module is dependent on other modules: `hvdm-lua`, `hvdm-ctx`, `hvdm-voc` and `hvdm-xml`. They are loaded from within the main module file, now named `hvdm-tak` (from *tak*enotes, as is easily guessed).
Notes are stored in the main document within the root node `<takenotes>` or may be put into separate files to be loaded on the fly. A `<note>` has the following general structure:

```
<note attribute=".."  attribute="..">
  <subject> ... </subject>
  <author> ... </author>
  <source> ... </source>
  <text> ... </text>
  <remark> ... </remark>
  <!-- etcetera -->
</note>
```

This works fine with one `<note>` per file but not with several. In that case the first one is processed, the others are 'forgotten'. In order to have them processed properly they must be be enclosed inside a `<notes>` node which then functions as their root. The same applies to node collections which can be read from a file or buffer, as explained below.

```
<notes>
  <note> ... </note>
  <note> ... </note>
   <!-- etcetera -->
</notes>
```

Thus notes can be typeset from various sources:

```
<takenotes attributes="">
  <!-- Local notes -->
    <notes>
    <note> ... </note> <note> ... </note> ...
    </notes>
  <!-- Included from file -->
    <notes file="somenote.xml"/>
  <!-- Included from directory -->
    <notes folder="somedirectory"/>
  <!-- Included from mtxrun -->
    <notes arg="someparameter"/>
</takenotes>
```

The first three possibilities speak for themselves, but the last one needs an explanation. ConTEXt typesetting is set in motion from the `mtxrun` script. That script has parameters which can be picked up by the executing program. An example will make this clear. Suppose one has a tailored ConTEXt file `template.tex` that should be used to process individual notes, each in their own file. This can be accomplished by running the template file with inside it `<notes file="somenote.xml"/>`, each time changing `somenote.xml` to a new value. For a few hundred files this is a herculean task, to say the least. How much easier if but one command could do them all.[1] The last option above is meant to accomplish just that. In general ConTEXt is called from the command line with:

```
mtxrun --script context file
```

We add our own parameter, naming it for the sake of this example `someparameter`, and run with:

```
--someparameter="somenote.xml" template.tex
```

The template file is then run on `somenote.xml`. In order to do this for a great many of files, all what is needed is a shell script that repeatedly executes the `mtxrun` with a shell parameter for `file`. At the end of this article two `bash` shell scripts are outlined that together accomplish this task.

Note structure
Minimal template for a note:

```
<note
  category="" subcategory="" tag="" date=""
  key="" language=""  text="" obsolete="">
  <!-- child nodes -->
</note>
```

The data pertaining to a note are split between the attributes and its child nodes. Attributes are used to classify the note and will serve as keys in the filtering of notes during the processing stage. These attributes have the following meaning.

- ⋆ `category` – *Mandatory* Freely chosen designation for the topic or type of document under which the subsequent notes are categorized by the user.
- ⋆ `subcategory` – A category can be refined into subcategories, making finer meshed searches possible. An example from my collection of genealogical data: `category="register"`, `subcategory="marriage"`.
- ⋆ `tag` – Optionally notes can be tagged in the (left) margin with this tag, usually a single short word.
- ⋆ `date` – The date pertaining to the note. Usually the date on which the content was produced originally or the date of the event it describes.
  - A date of 6 or 8 digits will be interpreted as `yyyymmdd` or `yymmdd` respectively; the latter designating a date in the current 21st century. Thus `17530427` will be interpreted as the 27th of April 1753 and `190401` as the 1st of April 2019.
  - Other acceptable date formats are `dd-mm-yyyy`, `d-m-yyyy`, `yyyy-mm-dd`.
  - Four consecutive digits will unsurprisingly be treated as a year.
  - Two years separated by one or more dashes, possibly surrounded by whitespace, constitute a range of years; sorting is on the first year of the pair. An example is "1750 - 1780".
  - Dates given as '`May 12,1981`' are printed as such, but cannot take part in date sorting.
  - A missing date will generate a warning, but it can be appropriate to leave out the date. In that case use `date="nodate"` to silently suppress output of a date.
  - Dates found in error will generate a warning.
- ⋆ `key` – A number of space and/or comma separated keywords that can be used to filter out notes of particular interest.
- ⋆ `language` – Signifies in the standard two-letter code the language in which the content of the note is written. Needed only when that language differs from the language in which the full document is typeset and should be reflected in the output. The internationalization of certain terms in the output uses this value.
- ⋆ `text` – Even if the user has chosen to generally suppress `<text>` nodes (see below), still some notes will be meaningless without it. Setting `<note text="on">` (or `yes`, `true`) forces the `text` nodes out for this specific note. Conversely, the text on the note can be suppressed regardless of the selected setting with `text="off"` (or `no`, `false`).
- ⋆ `obsolete` – Flags obsolesce of the note and is used to suppress its output without the need to remove the note physically. It then can be easily reinstated when need arises. Values `on`, `yes`, `true` and `off`, `no`, `false` makes the note obsolete/active, the latter being the default. For obsolete notes, when their output is selected, a bare minimum of child nodes is displayed.

Child nodes and their presentation
The child nodes carry the information proper. A few standard child nodes are implemented already, chosen as being the most general. Others can be added by the user at will. We shall refer to the child nodes of `<note>` as 'legends'.
When typeset, the legends precede the text content contained in `<text>` nodes. They adapt their heading to the language set for the ConTEXt document or the `language` attribute on the `<note>`. Provided, of course,

the language equivalents are known to the module (as is the case for english, dutch, french and german for the standard legends). It is not difficult to add other translations in other languages (see the explanation of using <vocabulary> later on).
Legends <subject> and <abstract> will be placed but once in the output and then their first occurrence is taken, the others being ignored. Legends <alert> and <source> on the contrary will be gathered from everywhere in the note and grouped together. The other legends are not constrained.
Typesetting is done in a certain order. First comes the information taken from the note's attributes. Then the legends in the order determined by the program. Note however, that the order in which they are put inside the <note> can be chosen freely. The order in the enumeration below roughly reflects the ordering in the output.

- ⋆ <subject> – *Mandatory* The very first legend containing a short description or title of the subject of the note; only its first occurrence is used. The subjects of all notes are optionally combined in a list from where they can be reached by clicking on them. The subject is never suppressed, not even on obsolete notes when these are printed.
- ⋆ <author> – Author(s) of the information and/or the writer(s) of the note, presented together comma separated.
- ⋆ <source ref="" link=""> <url> – Carries the source or location of the information. Generally containing a link to an URL, either local or on the internet. It can be a posting on the internet or a reference into a (public) archive.
  Two attributes are provided for locating the source: `ref` is what will be printed and *must be present*, `link` is the underlying URL. The link attribute (if not absent or an empty string) is checked and an error message presented if it cannot be reached. Otherwise an active link is made so that clicking on it brings the underlying document in view. *Beware: spaces in these links must be explicitly coded as %20 or else the link will not be found!*
  Sometimes it is preferred to embed the <source> references within the text rather then to group it at the <note> level. Then the reference is printed both in the <source> group as well as locally. The first occurrence will carry the interactive link, the latter being bare plaintext. Also, in that case the content of the node is not ignored as is otherwise done; this content appears behind the legend en is useful for example to annotate the source.
  The <url> is synonym to <source> and has the same effect. Differentiation between them allows one to activate/deactivate them separately.
- ⋆ <person omit=""> – This is a legend with child nodes, meant for the description of persons and derived from my genealogical interest.
  The legend has one attribute `omit`. Setting this to `on` (or `yes`, `true`) marks this person eligible to be omitted if chosen so on a takenotes run. It allows one to register all persons figuring in a document and at the same time differentiate between the more important ones and others. A minimum of child nodes is provided, but others can be added by including their node names in the <takenotes> attributes (see below). Be aware of the fact that nodes not marked as such will be ignored inside the <person> node; otherwise there would be too much interference with the formatting.
  - `name` – The name of the person. It is an option to collect these names in a list that then will be found at the end of the <takenotes> run behind the list of subjects.
  - `title`, `surname`, `between`, `name` – Alternative to putting the full name inside <name> is to split it up into [<title>] <name> [<between>] <surname> (`title` and `between` being optional). The presence of a not empty <surname> triggers this alternative.
  - `named` – Especially in the past, people were often not very much concerned with spelling issues. Therefore names in documents will differ in spelling or are clearly misspelled. Also, in certain documents a latinized form of the name is used; for example in church archives recording baptisms one may find 'Adrianae' meaning 'from Adriana'. Another variant is 'Joannes' for 'Johannes', not to speak of abbreviations like 'Jo:es'. To cope with all these variations `named` is provided. It is useful for registering the name exactly as encountered in the document, while at the same time keeping the standard name in `name`. This entry will be typeset within parentheses behind the proper name.
  - `relation` – People are related to each other, such as 'son of', 'spouse of', etc. Adding their relation helps to differentiate between people with the same name but different ancestry.
  - `role` – The way people act in the document, for example as buyer of a house, bride in a marriage, deceased in a burial.
  - Add new ones through the `on` attribute of <takenotes>, because child nodes not being registered as legend will be ignored.[2] Optionally add their translation to the vocabulary (see below).
- ⋆ <user-legends-nodes> – The number of legends preprogrammed is limited and more often than not insufficient for the task at hand. The module has to be informed then in order to be able to place the

additional legends correctly. The method chosen is a very simple one. To add a <location> legend for example, list that node's name in the on attribute: <takenotes on="location">. Thereafter <location> is recognized as a legend and typeset as such.
On the other hand, nodes present but unwanted must be marked as such or their content will appear unstructured in the legend list. To avoid this they must be enumerated in the off attribute: off="location" will suppress the <location> node. The alert and subject have been made exempt of this mechanism, guaranteeing they will not be excluded.
Because the underlying definitions are made globally, there is no need to repeat them within the same ConTEXt run.

- <non-legend-nodes> – Comprises all XML-nodes not programmed in this module and not mentioned in either the on or the off attributes. Nothing special is done.
- <group> – Sometimes one may feel the need to meddle with the strict order and selection of nodes at the note level. Then enclose the content with a <group> (and pray all may proceed as intended).
- <alert legend=""> – This is a remark typeset in the alertcolor (see below). Where remarks can be optionally suppressed alerts will be suppressed in obsolete notes only, but otherwise they are always present.
- <remark legend=""> – Meant for placing remarks or other additional information not having a place in the regular text portion of the note. Attribute legend takes the place of the heading 'remark' in the output. An example is legend="todo" for a reminder.
- <abstract> – Speaks for itself, an abstract of the contents of this note. The first one is printed, later occurrences are ignored.
- <text title="" indent=""> – Textual content of the note, in one or more <text>-blocks, all printed in the order of occurrence.
  When title is present, it is typeset centered at the start of the text block. The indent attribute can be used to explicitly switch indenting on or off locally.

Takenotes attributes

The root node <takenotes> effects the extraction and presentation of the collection of notes.

Selection of notes

The number of notes can grow so large that finding something specific becomes tedious, especially in a printed document. Here two strategies can be followed. The first is examining the resultant pdf in a program as Adobe Reader and using its search facilities. The second is to narrow the output to just the notes of interest. In order to accomplish that, a variety of selection criteria can be applied, either separately or in combination.

- The first set of selection criteria is based on the value of attributes on the enclosing <note attributes>. Selections can be made on category, subcategory, a range of dates, key and obsolescence.
- The second type of selection looks into the content of the note. A search pattern is applied to the full content and the note is displayed when that pattern matches somewhere. Both case sensitive and case insensitive searches can be initiated. Because the searching is executed in Lua, not just fixed patterns but also more complicated general patterns are possible.

Most of the custom settings are made globally in order to facilitate their setup in subsequent <takenotes>'s in one ConTEXt run; for example each in its own chapter. The selection criteria are an exception to this.
Attributes steer the selection of notes according to criteria related to the attributes on the note. The selection criteria work independently from each other and together let pass those notes that conform to all of them.

- category, subcategory – When present and not empty the value of these attributes takes part in a selection based upon the category/subcategory attribute on each <note>; the comparison is case insensitive.
- from and until – Attribute from carries the lowest date on the <note> that will be selected. Conversely until selects the last date included in the output. If both are present they constitute an interval. Dates must be formatted according to the rules explained above. Free format dates or notes without a date will be assigned the lowest date the module provides (1-1-4712 BC).[3]
- key – Will be matched to the comma or whitespace separated list of keywords on the <note>. If there is match, the note will be selected. This matching is case sensitive.
- select – The value of this attribute is a Lua search pattern applied to all sub-nodes within the <note>. A matching pattern contributes to typesetting of the note, in combination with the other selection criteria that are activated.
  When there are uppercase letters in the search pattern a case sensitive search is done, otherwise it is case insensitive.[4] If <text> blocks are suppressed they are left outside the search and therefore do not influence the selection.

Selection of features

Typesetting of the notes is determined twofold. First one may choose which legends will appear and which

not. This regards both the legends builtin as well as those provided by the user, as already has been outlined above.
Second are flags that influence the output in many respects. Activation and deactivation are done with the `on=""` and `off=""` attribute respectively. Values within the attribute must be separated by commas, may have spaces between them, but not divided with a newline.
Flags are capitalized, legend names all lowercase. A catalogue of the possibilities follows.

Choosing the legends

- By default on are the legends:
  `abstract`
  `alert` (always)
  `author`
  `person`, `relation`, `role`
  `remark`
  `source`
  `subject` (always)
  `text`
  `url`
- By default on are the flags:
  `Category`
  `Date`
  `Omit`
  `Sort`
  `Sortup`
- By default off are the flags:
  `Filename`
  `Key`
  `Namelist`
  `Subjectlist`
  `Newpage`
  `Nobreak`
  `Number`
  `Obsolete`
  `Tag`
  `Verbose`
- `Namelist` – If `<person>` nodes are present, their `name` will be collected and afterwards put in a two-column list in alphabetic order. When this list is present a link to it will appear at the top of each page.
- `Subjectlist` – The subjects are collected and output in alphabetical order. If the `Number` attribute has been set, the subjects in the list will be tagged with the sequence number of the corresponding note. If `Date` is selected, the note dates are attached.
  When the list of subjects is present a link to it will appear at the top of each page. Furthermore, the lines in the subject list are active and carry to the page in the document where this specific subject resides. All links are local to the current instance of `<takenotes>` as is the case for the list of names.

Tailoring the appearance

- `Newpage` – Start a new page for each note.
- `Nobreak` – Prevents breakup of a note over a page boundary by placing it in a framed box. Be aware of the fact that long notes will play havoc with page boundaries.
- `Filename` – Turns on a note separator with the name of the file in it. It can be combined with `Number`.
- `Number` – Turns on a separator with the sequence number of the current note in it. Can be combined with `Filename`.
- `Obsolete` – Include obsolete notes in the output which are otherwise ignored. If included their content is curtailed to the subject and the text color altered (by default dimmed to gray).
- `Omit` – Its use is discussed with the `<person>` child node.
- `Sort`, `Sortup` – Notes are sorted on their date in ascending order by default. Turn that into descending order by adding `Sortup` to the `off` attribute. Notes with a missing or empty date are grouped together.

Style and color

The content of the notes can be given different font and style from the rest of the document. An example of the use of a different font for the text would be to typeset program code in a monospaced font.
The following values are available for style options: `bf`, `it`, `bi`, `bs`, `sl`, `tt`, `sc`, `rm`, `ss`, `tf`, `tfa`, `tfb`, `tfc`, `tfd`, `tfx`, `tfxx`, `ttx`, `ttxx`, `os`, `hw`, `cg`, `bold`, `italic`, `bolditalic`, `italicbold`, `slant`, `slanted`, `boldslanted`, `slantedbold`, `smallcaps`, `oldstyle`, `mediaeval`, `normal`, `serif`, `regular`, `roman`, `sans`, `sansserif`, `mono`, `type`, `teletype`, `handwritten`, `calligraphic`, `big`, `verybig`, `heavy`, `veryheavy`, `small`, `tiny`, `smallmono`, `tinymono`.
These values translate to corresponding font commands. Several font commands may be combined as for example `legendstyle="bf,os"`, but not all combinations are effective. Be aware of the fact that not all of the above can be considered pure style options. For example the use of `tt` might not change the font family but has a profound influence on the look of the text nonetheless.

- `abstractstyle` – Used to set a different style for the text in the abstract.
- `legendstyle` – Style used for the legends.
- `textfont`, `textstyle` – Font and style for the text blocks, abstracts inherit the same font but may differ in style.

Colors may be left at their default values, switched to the (black) text font by providing the value `none` or given a color at will.

- `alertcolor` – Color used for the alerts.
- `backgroundcolor` – When set the background of the notes will be colored.

- ⋆ `linkcolor` – Color used for active links.
- ⋆ `obsoletecolor` – Color used for the obsolete notes when printed.
- ⋆ `omitcolor` – Color used for the `<person>` node when marked for omission in case they are printed.

Languages
Elements of the output can be internationalized through definition and use of one or more vocabularies. This applies to the language in which the legends are typeset, but depending on the effort one wishes to invest many parts of the text can undergo internationalization. The component effectuating this is the module `hvdm-voc` which is incorporated by the module `hvdm-xml`.
It all starts with the creation and filling of a vocabulary. By default there is a vocabulary `languages` defined (in part) by the code below. A named vocabulary is automatically created the first time one attempts to add something to it.

```
<vocabulary name="myvocab">
<word add="dutch">
  <en>dutch</en>
  <nl>nederlands</nl>
  <de>niederländisch</de>
  <fr>n&eacute;erlandais</fr>
</word>
</vocabulary>
```

In the document the code shown above is loaded from a buffer or a file with:

```
<vocabulary name="myvocab" buffer="aBuffer"/>
<vocabulary name="myvocab" file="aFile"/>
```

Individual translations may be loaded one by one with:

```
<vocabulary name="myvocab" add="greek">
  <en>greek</en>
  <nl>grieks</nl>
  <de>griechisch</de>
  <fr>grec</fr>
</vocabulary>
```

Translations are retrieved by:

```
<vocabulary name="myvocab" get="greek"/>
<vocabulary name="myvocab" Get="greek"/>
<vocabulary name="myvocab" GET="greek"/>
```

With the current language being ‘en’, it results in greek, Greek and GREEK. However, after changing to german the results change accordingly:

```
<vocabulary use="german"/>
```

Now it is griechisch, Griechisch and GRIECHISCH. Similarly the default vocabulary to use can be set:

```
<vocabulary set="myvocab"/>
```

If the document is changing vocabularies often, it might be safer to use the vocabulary’s name on each node explicitly. The more because a word not known in the vocabulary at hand, will appear untranslated. Most of the operations on vocabularies can be called directly from a ConTEXt-document with macro’s `translate`, `Translate`, `TRANSLATE`, etc. See the code of the module for the details.

Example
Two examples of a note. The first is in the Dutch language as designated by the attribute `nl`. Note by comparing the XML and its output, that flagging a note as obsolete suppresses everything except the subject title when printed nonetheless. The second is an example from a set of genealogical data. It also serves the purpose of showing how HTML-elements from module `hvdm-xml` can be used.

onderwerp: Cryptografie cursus
datum: 1-1-2018
categorie: Cryptografie/cursus
auteur: dr. Hans van der Meer
opmerking: This course is in Dutch.
samenvatting: Cursus cryptografie
Algemene inleiding tot de cryptografie en cryptoanalyse met oefeningen voor studenten.

Note the difference in output between an active note and an obsolete one.

Vervallen
onderwerp: Cryptografie cursus

```
<?xml version="1.0" encoding="UTF-8"?>
<takenotes on="Nobreak"
  backgroundcolor="lightgray">
<note date="20180101"
  category="Cryptografie" subcategory="cursus"
  key="cryptografie cryptoanalyse"
  language="nl" obsolete="no">
  <subject>Cryptografie cursus</subject>
  <author>dr. Hans van der Meer</author>
  <abstract>Cursus cryptografie</abstract>
  <text>
    Algemene inleiding tot de cryptografie en
    cryptoanalyse met oefeningen voor studenten.
  </text>
  <remark>This course is in Dutch.</remark>
</note>
</takenotes>
```

subject: Marriage of Arnoldus Vorst and Maria Catharina van Hoven
date: 11-1-1727
category: archive/marriage
source: HGA 0321-01 inv.16 f.189
location: The Hague
▷ *person:* Arnoldus Vorst
relation: father of Maria Vorst
role: groom
age: 24
▷ *person:* Maria Catharina van Hoven (Hove)
relation: mother of Maria Vorst
role: bride
age: 22
remark: Married pro deo at town hall.

```
<?xml version="1.0" encoding="UTF-8"?>
<takenotes on="Nobreak, location, age"
  backgroundcolor="lavenderblush"
  from="1-1-1700" until="31-12-1750">
<color name="lavenderblush"
  green=".941176" blue=".960784"/>
<note
  category="archive" subcategory="marriage"
  date="11-1-1727" key="vorst" language="en">
<subject>
  Marriage of Arnoldus Vorst and
  Maria Catharina van Hoven
</subject>
<location>The Hague</location>
<source ref="HGA 0321-01 inv.16 f.189"/>
<ul sym="4"><li>
<person>
  <name>Arnoldus Vorst</name>
  <named></named>
  <relation>father of Maria Vorst</relation>
  <role>groom</role>
  <age>24</age>
  <profession>carpenter</profession>
</person>
</li><li>
<person>
  <name>Maria Catharina van Hoven</name>
  <named>Hove</named>
  <relation>mother of Maria Vorst</relation>
  <role>bride</role>
  <age>22</age>
</person>
</li></ul>
<remark>Married pro deo at town hall.</remark>
</note>
</takenotes>
```

Note here that the location and the age have been activated with `on="location, age"` and thus placed on each person, while the undeclared profession is absent.

Bash scripts for using templates

To get an impression of the scripts that run a template with the `arg` option on several <note> containing files, an extract follows below. The full scripts can be obtained from the internet. One such place is the *publications page* on `hvandermeer.com` (use the link *ConTeXt module distribution*).

```
#!/bin/bash
#   ¢Hans van der Meer hvandermeer.com
#   Run template tex-file on targets through notesone script.
for filename in "$@"; do
    notesone "$TEMPLATEFILE" "$filename"
done
```

```
#!/bin/bash
#   ¢ Hans van der Meer hvandermeer.com
#   Process one file with ConTeXt through template.
mtxrun --script
  context --$TARGETARGUMENTNAME="$2" "$1" >/dev/null
#   Change the output pdf to the target basename.
mv "$SOURCENAME.pdf" "`dirname "$2"`/$TARGETNAME.pdf"
```

Notes

1. For the fans: Yes, you can hear The Lord of the Rings here.
2. This may seem overly restricted, but it prevents the module of becoming too complicated and is at the same time beneficial for a clear presentation of these data.
3. For those who wonder: the lowest value for the Julian date.
4. In Lua patterns elements such as %A, %S are used to suppress certain categories of characters and thus are distinct from uppercase text. The search mechanism implemented is aware of that distinction.

Hans van der Meer
havdmeer@ziggo.nl

Bits and Pieces from ConTEXt mailing list

A Collection of TEX and METAFONT Notes

Abstract
My Takenotes module for processing notes is used to present a selection from the notes collected mainly from the ConTEXt users group on the internet.

Introduction

Active email lists, as for example the ConTEXt-list, produce many emails discussing problems and other topics. Often these discussions form threads chaining successive postings, each posting incorporating much of the previous ones. Collecting all the emails for later reference therefore tends to produce a lot of redundancy. Moreover, the continuous repetition of previous content renders reading rather tedious.
Clearly easier reading can be achieved when threads worth retaining are condensed to a single note, describing the gist of the discussion. This idea motivated the development of the Takenotes module in the first place, as a means to collect, store, select and reproduce those notes. Here are presented some of the notes I have collected over time and which are deemed useful for wider dissemination. Selection has been quite haphazard, though. Enjoy!

The notes

note-1

subject: Long division macro
date: 5-11-1996
author: David Arsenau
source: comp.text.tex

A very involved, but interesting macro to typeset long divisions from the bare minimum of data.

```
\newcount\gpten % power-of-10 - which digit now
\countdef\rtot2 % running total - remainder
\countdef\LDscratch4 % scratch
\def\longdiv#1#2{%
\vtop{\normalbaselines \offinterlineskip
  \setbox\strutbox\hbox{%
    \vrule height 2.1ex depth .5ex width0ex}%
  \def\showdig{$\underline{\the\LDscratch\strut}$%
    \cr\strut\the\rtot\,\cr\noalign{\kern-.2ex}}%
  \global\rtot=#1\relax
  \count0=\rtot\divide\count0by#2\relax
  \edef\quotient{\the\count0}%\show\quotient
  % make list macro out of digits in quotient:
  \def\temp##1{\ifx##1\temp\else
    \noexpand\dodig ##1\expandafter\temp\fi}%
  \edef\routine{\expandafter\temp\quotient\temp}%
  % process list to give power-of-ten:
  \def\dodig##1{\global\multiply\gpten by10 }%
  \global\gpten=1 \routine
  % display one digit in quotient (zero ignored):
  \def\dodig##1{\global\divide\gpten by10
    \LDscratch =\gpten
    \multiply\LDscratch by##1%
    \multiply\LDscratch by#2%
    \global\advance\rtot-\LDscratch \relax
  \ifnum\LDscratch>0 \showdig \fi
  }%
  \tabskip=0pt
  \halign{\hfil##\cr % \halign entire division
    #2\,$/\mkern2mu\the\rtot\,
      \rlap{/\quotient}$\cr\noalign{\kern-.2ex}%
    \routine\cr % do each digit in quotient
}}}
```

Example:

```
\longdiv{1132}{57}
```

$$\begin{array}{r} 57\,/\,1132\,/\,19 \\ \underline{570} \\ 562 \\ \underline{513} \\ 49 \end{array}$$

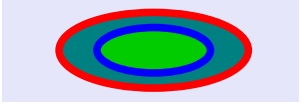
note-2

subject: Drawing ellipse
date: 1-7-2000
author: Christian Stapfer
source: comp.text.tex

Easy and not so easy way of drawing an ellips.

```
a := 5mm; b := 2mm;
% Not so easy:
  n = 31; draw for i = 0 upto n-1:
    (a * cosd(i*360/n), b * sind(i*360/n)) ..
  endfor cycle withcolor red;
% Much easier:
  draw fullcircle scaled (3*a) yscaled (b/a);
```

Used in the example below to draw and fill some ellipses. *Example:*



note-3

subject: Inline text fractions
date: 22-6-2012
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to get fractions on the baseline?
Answer: use the following code:

```
\definefontfeature[fraction][frac=yes]
\definehighlight[textfraction]
    [style={\feature[+][fraction]}]

No so neat: \m{3\vfrac{1}{2}}
much better: 3\textfraction{1/2}
```

Example: No so neat: $3\,^{1}/_{2}$ much better: 3½

note-4

subject: Math fractions
date: 6-4-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Hans Hagen redid the MKIV code for math fractions, illustrated with the following example:

```
\definemathfraction[myfrac][mathstyle=script]
\definemathfraction[myfracx]
  [mathstyle=script,alternative=outer]
\startformula
  {a^{2^{2^2}}\over a}
  \frac{a^{2^{2^2}}}{a}
  \frac{a^{2}}{b}
  \myfrac{a^{2}}{b}
  \myfracx{a^{2}}{b}
\stopformula
```

Example:

$$\frac{a^{2^{2^2}}}{a} \quad \frac{a^{2^{2^2}}}{a}\frac{a^2}{b} \quad \frac{a^2}{b}\frac{a^2}{b}$$

note-5

subject: Blanks instead of indented paragraphs
date: 10-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: I want to have empty lines between paragraphs, not only indenting.
Answer: Use `\setupwhitespace[line]`

note-6

subject: Placing footnotes and endnotes
date: 11-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to place either footnotes or endnotes?
Answer:

```
\placenotes
```

is an abbreviation for

```
\placenotes[footnote]
```

but for endnotes explicitely use

```
\placenotes[endnote]
```

note-7

subject: Arc symbol
date: 14-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to place an arc symbol under or over?
Answer: use `$\underparent{ABC}\overparent{XYZ}$`

Example: $\underparent{ABC}\overparent{XYZ}$

note-8

subject: Coloring footnote numbers
date: 15-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to color the number of a footnote?
Answer: Use for the number in the running text

```
\setupnote[footnote][textcolor=color]
```

and for the number in the footnote block

```
\setupnotation[footnote][headcolor=color]
```

note-9

subject: Show only sections in the TOC
date: 16-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to restrict the Table of Contents to specific elements e.g. sections?
Answer: use `\placelist[section,subsection]` to place sections and subsections only.

note-10

subject: Restrict TOC to sections and subsections
date: 16-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to restrict the Table of Contents to specific elements like sections and subsections?
Answer: use

```
\placelist[section,subsection]
```

to place only sections and subsections.

note-11

subject: Why ConTeXt macros are not always found
date: 16-4-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Question: Why does a global search through the ConTEXt sources sometimes misses a macro?
Answer: The macro `\completecontent` is an example of a macro not found by searching the source code. The

reason is that it is formed by combining parts and a result of the multilingual interface of ConTEXt. In this case one had to look for `e!complete`.

note-12

subject: Suppress caption number in float
date: 16-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: How to suppress the number in a float caption?
Answer: use `\placefigure[nonumber,…]{…}{…}`.

note-13

subject: Inhibiting hyphenation
date: 21-4-2013
author: Hans Hagen
source: ntg-context@ntg.nl

In reaction to a request added the macro `\unhyphenated`. This just sets `\lefthyphenmin` to its maximum value. Another possibility is setting `\normallanguage=0` because that language has no hyphenation patterns.

note-14

subject: Drawing line under header
date: 29-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to draw a line under the header?
Answer: Use `\setupheader[text][after=\hrule]`

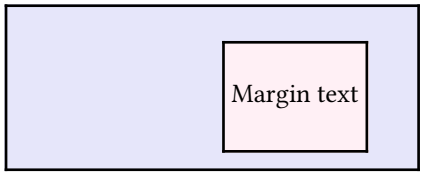
note-15

subject: Placing margin text besides float
date: 29-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to associate text in the margin with a figure in a float?
Answer: see the following code snippet using the second argument to `\startplacefigure`. (The example is a bit contrived, because the code is meant to be used on a page, not inside a frame.)

```
\define\FigureText{\dowithnextbox{%
  \startlinealignment[middle]
    \copy\nextbox\rlap{\hskip\rightmargindistance
    \framed[frame="off,
      width=\rightmarginwidth,height=\nextboxht,
      background=color,backgroundcolor=lavender]
        {\floatuserdataparameter{text}}}
  \stoplinealignment}\hbox}
\setupfloat[figure][command=\FigureText]
\startplacefigure[title=TheTitle][text=TheText]
    \externalfigure[TheFigure]
\stopplacefigure
```

Example:



Note that in the example the frame around the margin text has been drawn in order to illustrate the mechanism. Also the width of that frame has been set to `fit` in order to force the example inside the column.

note-16

subject: Changing the format of chapter numbers
date: 29-4-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to change the format of the chapter number not only in the chapter heading, but in the table of contents too? In this case from "Chapter 1" to "Chapter 1.0".
Answer: see the following code snippet.

```
\define[1]\ChapterConversion
  {\convertnumber{Numbers}{#1}.0}
\defineconversion[ChapterConversion]
  [\ChapterConversion]
\setuphead[chapter][conversion=ChapterConversion]
\completecontent
\chapter{First Chapter}
\section{First Section}
```

note-17

subject: Chapter title in header only
date: 2-5-2013
author: Piotr Kopszak
source: ntg-context@ntg.nl

Question: how to use chapter titles in the headers only?
Answer:

```
\setuphead[chapter][placehead=no]
\setupheadertexts[][chapter][][]
```

note-18

subject: Keep footnotes on page
date: 23-7-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Question: How to keep footnotes on the page where they are called?
Answer:

```
\setupnotes[footnote][split=verystrict,scope=page]
```

Note however that this may result in unused white-space between the text and the footnotes.

note-19

subject: Colors can be scaled and mixed
date: 22-8-2013
author: Hans Hagen, Hans van der Meer

source: ntg-context@ntg.nl

Colors can be evenly mixed as in:

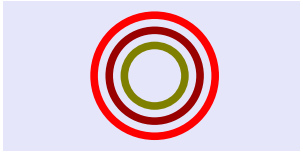
```
\definecolor[redyellow][.2(red,yellow)]
\definecolor[magentablue][.5(magenta,blue)]
```

Resulting in red-yellow and magenta-blue.

Another example of this in MetaPost shows the effect of reducing the strength of the color by multiplying it with a factor between 0 and 1. Be aware then of the fact that smaller multipliers darken the color, approaching black (the middle circle in the example).

```
\startMPcode
a := 4mm; pickup pencircle scaled 1mm;
draw fullcircle scaled (4*a) withcolor red;
draw fullcircle scaled (3*a) withcolor .6red;
draw fullcircle scaled (2*a) withcolor .5[red,green];
\stopMPcode
```

Example:



note-20

subject: First line or word(s) made different
date: 22-8-2013
author: Hans Hagen
source: ntg-context@ntg.nl

The first line of each paragraph can be made different with:

```
\definefirstline[fancy][alternative=line,
    color=red,style=\setfontfeature{smallcaps}]
\setfirstline[fancy] ... \par
```

THIS IS TEXT ILLUSTRATING THE CHANGE IN FONT, style and color of the first line, while the lines following it are not influenced by the changes applied to the first.

The next example uses:

```
\definefirstline[fancy][alternative=word,n=3,
    color=blue,style=\setfontfeature{smallcaps}]
\setfirstline[fancy] ... \par
```

THIS IS TEXT illustrating the change in font, style and color of the first three words, while the words following it are not influenced by the changes applied to the first.

note-21

subject: Dropcapital as initial letter
date: 28-8-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Typesetting a dropcapital with `\setupinitial[]` followed by `\placeinitial\strut`. Parameters on `\setupinitial`.

```
font=FONTatSIZE: font specification
style=STYLE: for example \tex{bfd} for large bold
n=NUMBER: line where footline of dropcapital
m=NUMBER: number of letters to capitalize
distance=DIMENSION: offsets the following text
hoffset=DIMENSION: shift +left -right
voffset=DIMENSION: 0pt=top  from there +down -up
```

```
[style={\tfc\bi},n=1,m=4]
```

This is a starting line for illustrating the default dropcapital. This is a line for illustrating the dropcapital. This is a line for illustrating the dropcapital.

```
[n=2,m=4,voffset=.8\lineheight,distance=2em]
```

This is a starting line for illustrating the default dropcapital. This is a line for illustrating the dropcapital. This is a line for illustrating the dropcapital.

```
[font=Bold at 32pt,n=2,hoffset=-10pt,distance=16pt]
```

This is a starting line for illustrating the dropcapital. This is a line for illustrating the dropcapital. This is a line for illustrating the dropcapital. This is a line for illustrating the dropcapital.

note-22

subject: Restrict number of ConTeXt runs
date: 3-10-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to restrict the number of ConTEXt runs?
Answer: there are two possibilities:

1. Execute the run once with: `mtxrun --once`
2. Put a restriction in the first line of the document containing: `% nofruns=1`

note-23

subject: Drawing text along a path
date: 13-10-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Question: How to draw text along a path in MetaPost?
Answer: See the example.

```
\startMPcode
path p, q, r;
p := halfcircle xyscaled 35mm; draw p;
q := halfcircle xyscaled 44mm; draw q;
r := reverse halfcircle xyscaled 38mm;
draw followtext(r,
  "$*$~some text but not that long~$*$");
\stopMPcode
```

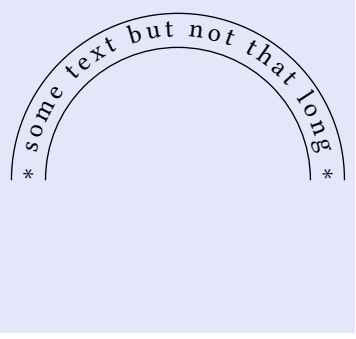
There is an older method to accomplish this with the MetaPost library txt:

```
\useMPlibrary[txt] ...
\startuseMPgraphic{followtokens} ...
```

This has two drawbacks. The first is the use of hard-coded variable `RotPath` and the second is the fact that the text does not exactly follow the given path, but is

offset slightly from it. In contrast `followtext` follows the path exactly, as can be understood by comparing code and figure.

Example:



note-24

subject: Framing part of math formula
date: 21-10-2013
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to frame part of a formula in math?
Answer: see next example.

```
\definemathframed[mcframed] [location=mathematics]
\startformula
  \ln(1+x) = \mcframed[frame=off,background=color,
    backgroundcolor=red,foregroundcolor=white]
    {x-{\frac {x^2}{\frac{x^3}{\frac{x^3}{3}}}}}
      +\frac{x^3}{3}-\cdots
  = \mcframed{x-{\frac {x^2}{\frac{x^3}
      {\frac{x^3}{3}}}}}+\frac{x^3}{3}-\cdots
\stopformula
```

Example:

$$\ln(1+x) = \boxed{x - \frac{x^2}{\frac{x^3}{\frac{x^3}{3}}}} + \frac{x^3}{3} - \cdots = \boxed{x - \frac{x^2}{\frac{x^3}{\frac{x^3}{3}}}} + \frac{x^3}{3} - \cdots$$

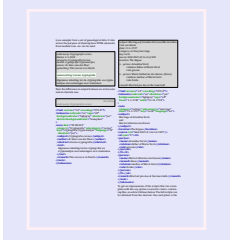
note-25

subject: Insert page by number from pdf
date: 9-12-2013
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Inserting a specific page by number from an existing pdf file can be done with:

```
\externalfigure[filename][page=pagenumber]
```

Example:



note-26

subject: Counters for pagenumber explained
date: 8-5-2014
author: Wolfgang Schuster
source: ntg-context@ntg.nl

ConTEXt uses three different counters for the pages of the document:

1. The realpage counter is used for internal references to pages, this counter should never be reset because it is needed to have unique numbers for each page.
2. The userpage counter which is shown in the header, TOC etc. and you can reset its value at the begin of a new sectionblock etc.
3. The subpage counter can be used when you want to divide a certain sectionblock into smaller parts to have local page numbers for a certain part of your document.

Each of these three counters has a command for setup:

- `\setuprealpagenumber`
- `\setuserpagenumber`
- `\setupsubpagenumber`

The old `\setuppagenumber` command is a synonym for the `\setuserpagenumber` command because this is the counter you have to change for your documents.

note-27

subject: Setting style and conversion for numbers of in-command
date: 22-5-2014
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to set the style en conversion of numbers in references for the `\in` command?
Answer:

```
\defineprocessor[sectionstyle][style=italic,...]
\defineconversionset[sectionconversion]
  [sectionstyle->Romannumerals][]
\setupreferencestructureprefix[section][default]
  [prefixconversionset=sectionconversion]
```

note-28

subject: Customize items in itemize
date: 29-5-2014
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Customizing items in itemize by example:

```
\define[1]\ItemCommand{%
  \hangindent=1.2in\relax
  \simplealignedbox{1.2in}{flushleft}%
    {\rlap{#1}\hfil\quad\hss---}%
  \removeunwantedspaces\ignorespaces}
\setupitemize[command=\ItemCommand]
```

Note how the `\rlap` is used to keep the —'s aligned.

Each item is followed by the argument.

1. First item: — Example text. Example text. Example text.
2. Second item: — Example text. Example text. Example text.

Similarly numbers in the TOC can be customized as in the following code;

```
\define[1]\ChapterListNumber
    {\simplealignedbox{\listparameter{width}}%
        {flushright}{#1}}
\setuplist[chapter][width=2em,distance=1em,
    numbercommand=\ChapterListNumber]
```

note-29

subject: Special quoting of paragraphs
date: 1-6-2014
author: Stéphane Goujet
source: ntg-context@ntg.nl

Question: how to do special oldstyle typesetting with quoted blocks?
Answer: the trick is the use of `\localleftbox`, define a quotation block as follows:

```
\define\qbopen{«}%         symbol on first line
\define\qbrep{\hbox{»~}}% symbol on next lines
\define\qbclose{»»»}%      symbol on last line
\definestartstop[bloccite]
  [before={\qbopen~\bgroup\localleftbox{\qbrep}},
   after={\egroup\qbclose}]
```

Note the use of the `\hbox` in the definition of `\qbrep` without which the space after the quote on the subsequent lines did not appear. Use within `\startbloccite` ... `\stopbloccite`

Example:

« This text is quoted in an oldstyle manner and made long enough to show the effect with more than one line. » »»»

note-30

subject: New macro for mode definition
date: 26-6-2014
author: Hans Hagen
source: ntg-context@ntg.nl

A new macro `\definemode` that is speeding up testing on mode settings. Use instead of the older `\enablemode` and `\disablemode`.

```
\definemode[themode][yes] % enables mode
\definemode[themode][no]  % disables mode
\definemode[themode][keep]% save previously used
```

note-31

subject: Suppress specific sections from the TOC
date: 5-7-2014
author: Hans Hagen
source: ntg-context@ntg.nl

Question: How to suppress specific sections from the Table of Contents?
Answer: use specially defined sections like the `silentsection` in the following example:

```
\definehead[silentsection][section]
\placelist[chapter,section]
\chapter{first}\section{first}
\chapter{first}\silentsection{first}
```

note-32

subject: Make and use plain format in ConTeXt
date: 11-7-2014
author: Hans Hagen
source: ntg-context@ntg.nl

Generate the plain format from the ConTEXt system with:

```
mtxrun --script plain --make
```

then run with:

```
mtxrun --script plain myfile.tex
```

and inspect the options aavailable with:

```
mtxrun --script plain
```

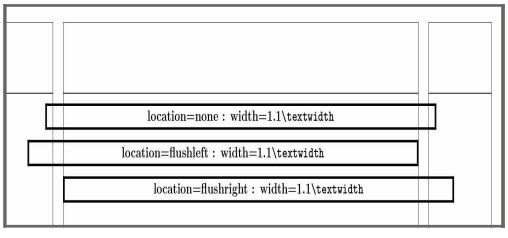
note-33

subject: Extending a figure into the margin
date: 25-7-2014
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to extend a figure into the left or right margin?
Example:

```
% loc = none, flushleft, flushright respectively.
\setupfloat[figure][location=loc]
\startplacefigure
    \framed[width=1.1\textwidth]{location=loc}
\stopplacefigure
```

Note the peculiar counterintuitive effect of left and right: flushleft aligns with right edge of the textarea!



note-34

subject: Buffers, environments and doifmode warning
date: 30-7-2014
author: Hans Hagen
source: ntg-context@ntg.nl

With the arguments of macros like `\doifmode{…}{…}` one should be careful because when one passes arguments their catcodes are frozen. In case of trouble it is better to use `\startmode[…]` etc. as these do not pick up arguments.

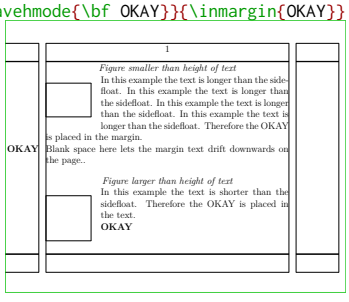
note-35

subject: Placing sidefloats and inmargin
date: 15-10-2014
author: Hans Hagen
source: ntg-context@ntg.nl

Placing floats on the side in the margin can interfere with the use of `\inmargin`. This shows how `\doifelsesidefloat` catches a clash with an extending sidefloat. The second sidefloat is too long and pushes the OKAY to the text-body. If there is blank vertical space directly below the block with the first sidefloat, it will drift downwards towards towards the first text that is typeset. Prevent this by affixing a `\null` (an empty hbox) to force the sidefloat out.

```
% figure smaller than height of text
  \placefigure[left,nonumber]{}
    {\framed[height=15mm,width=2cm]{}}
    ... text ... \par
  \doifelsesidefloat
    {\dontleavehmode{\bf OKAY}:}{\inmargin{OKAY}}
% figure larger than height of text
  \placefigure[left,nonumber]{}
    {\framed[height=20mm,width=2cm]{}}
    ... text ... \par
  \doifelsesidefloat
    {\dontleavehmode{\bf OKAY}}{\inmargin{OKAY}}
```

Example:



note-36

subject: Changing and switching page backgroundcolor
date: 15-10-2014
author: Hans Hagen
source: ntg-context@ntg.nl

Change the backgroundcolor of a full page and even switch it temporarily.

```
\setupbackgrounds[page]
    [background=color,backgroundcolor=cyan]
% = cyan page ... text ... \page
\setupbackgrounds[page]
    [background=,backgroundcolor=]
% = uncolored page ... text ... \page
\setupbackgrounds[page]
    [background=color,backgroundcolor=orange]
% = orange page ... text ... \page
  \pushbackground[page]
    \page\setupbackgrounds[page]
        [background=color,backgroundcolor=green]
    % = green page ... text ... \page
  \popbackground
% = return to previous orange ... text ...\page
```

note-37

subject: Alternate drawing and clearing with fill in MetaPost
date: 18-10-2014
author: Hans van der Meer

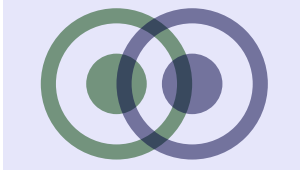
Alternating drawing and clearing with the MetaPost fill command can be done in two ways. Also, this example shows the use of `withtransparency`.

1. use reverse before the path of the `fill`
2. use a new feature, the `eofill` command that alternates between drawing and clearing

Both `fill`'s in the example produce the same picture, albeit in different colors.

```
a := 20mm;
fill fullcircle scaled a --
  reverse fullcircle scaled .8a --
  reverse fullcircle scaled .4a -- cycle
    withcolor green/4 withtransparency (1,.5);
  currentpicture := currentpicture shifted (-.5a,0);
eofill fullcircle scaled a --
  fullcircle scaled  .8a --
  fullcircle scaled  .4a -- cycle
    withcolor blue/4 withtransparency (1,.5);
  currentpicture := currentpicture shifted (.5a,0);
```

Example:



note-38

subject: Using euler font for math
date: 1-1-2015
author: John Kitzmiller
source: ntg-context@ntg.nl

Get *Euler* font substituted as math font. All fonts, except *pagellaovereuler*, can be changed for others or kept as is by using `\fontclass` instead of the name of the font. Note that the `\appendtoks` is needed because Euler exists in `rm` style only. The example below is produced by the following code:

```
\usetypescriptfile[euler]
\definetypeface[myeuler]% to select current font
  [rm][serif][pagella][default]% use [\fontclass]
\definetypeface[myeuler]
  [mm][math][pagellaovereuler][default]
Formula was: $y = ax^2 + bx + c$\crlf becomes:
\bgroup\switchtobodyfont[myeuler]
  \appendtoks \rm \to \everymathematics
  $y = ax^2 + bx + c$
\egroup
```

Example:

Formula was: $y = ax^2 + bx + c$
becomes: $\mathrm{y} = \mathrm{a}x^2 + \mathrm{b}x + \mathrm{c}$

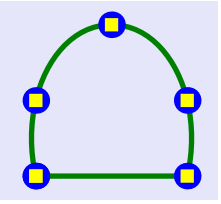
note-39

subject: Drawing in MetaPost directly from Lua code
date: 4-1-2015
author: Hans Hagen
source: ntg-context@ntg.nl

Drawing from a Lua table directly into a page. Note the mandatory substitution of `\letterhash` for # in '#data' and `\letterpercent` for % in '%s' as required in Lua.

```
\startluacode
local data = {{0,0},{0,2},{2,4},{4,2},{4,0}}
for i=1,\letterhash data do data[i] = string.formatters
    ["(\letterpercent s,\letterpercent s)"]
    (unpack(data[i]))
context.metafun.start()
context.metafun("path p; p := (\letterpercent s -- cycle)
    scaled 5mm;",table.concat(data,"--"))
context.metafun("draw p withpen pencircle
   scaled 2pt withcolor green/2;")
context.metafun("drawpoints p withpen pencircle
    scaled 10pt withcolor blue;")
context.metafun("drawpoints p withpen pensquare
    scaled 5pt withcolor yellow;")
context.metafun.stop()
\stopluacode
```

Example:



note-40

subject: How to ignore nodes in xml processing
date: 15-1-2015
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to ignore an XML-node?
Answer: define a nonexisting setup for that node, for example:

```
\startxmlsetups example:setups
    \xmlsetsetup{#1}{a|b|c|d}{example:*}
    \xmlsetsetup{#1}{x}{example:nonexisting}
\stopxmlsetups
```

This will ignore nodes `<x>...</x>`.

note-41

subject: Updating fonts
date: 27-1-2015
author: Hans Hagen
source: ntg-context@ntg.nl

Normally it is automatically detected if a font is updated or when a font is not found. Only when one changes fonts (locations) it is needed to rebuild the database using `--reload` or `--reload --force` for a full reload. ConTEXt itself will do a fast update when needed.
Rebuild with: `mtxrun --generate` or `context --generate`
The main thing you need to keep in mind as user is:

`mtxrun --generate` : when the tree changes
`mtxrun --script font --reload` : when fonts were moved, added or removed. If the cache is wiped all happens automatically anyway.

note-42

subject: Footnotes in separate groups
date: 15-2-2015
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to divide footnotes in separate groups? See next example that makes footnotes in two groups, each separately numbered:

```
\definenote[NoteA]\definenote[NoteB]
Tekst-A\NoteA{footnote A.}
Tekst-B\NoteB{footnote B.}
```

note-43

subject: Coloring the background of text areas
date: 26-2-2015
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to differentiate text areas by backgroundcolor?

```
\definetextbackground[one]
  [frame=off,backgroundcolor=yellow]
\definetextbackground[two]
  [frame=off,backgroundcolor=green]
\setnewconstant\kindofpagetextareas 1% low level
\starttextbackground[one] ... \stoptextbackground
\starttextbackground[two] ...  \stoptextbackground
```

Example: ... text ... ... text ...

note-44

subject: Testing if on left or right page
date: 15-3-2015
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to test for a left or right page?
Answer: use `\doifrightpageelse`.

note-45

subject: How to change the ?? for an unknown reference
date: 27-3-2015
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to change the mark ?? for an unknown reference?
Answer: in the next example defined as red, monospace and in the margin

```
\def\dummyreference{\inmargin[style=\tt,color=red]{??}}
```

note-46

subject: Locally suppress page header or footer
date: 22-5-2015
author: Hans Hagen

source: ntg-context@ntg.nl
The header (or mutatis mutandis footer) will be suppressed by `\page[header]`. This changes the state on the current page, but on itself will not generate a page-break.
With `\page[header,yes]` one breaks the page and the current one will have its header suppressed. On the next page the header is reinstalled.
Note that `\page[header,yes]` and `\page[yes,header]` differ in their effect.

note-47

subject: Coloring elements of footnote
date: 6-9-2015
author: Wolfgang Schuster, Hans van der Meer
source: ntg-context@ntg.nl
Color the footnotemark in the text with:
```
\setupnote[footnote][textcolor=color]
```
Color the footnotemark in the footnote with:
```
\setupnotation[footnote][headcolor=color]
```
Color the text of the footnote with:
```
\setupnotation[footnote][color=color]
```
An alternative for the footnotemark in the footnote that allows more freedom, macro `\high` is needed to keep the mark in a high position:
```
\def\myfootnotemark#1{\color[color]{\high{#1}}}
\setupnotation[footnote][numbercommand=\myfootnotemark]
```

note-48

subject: Formatting roman chapter but arabic section numbers
date: 25-9-2015
author: Pablo Rodriguez
source: ntg-context@ntg.nl
Question: how to format chapter numbers as romannumerals but section numbers as arabic numerals?
Answer:
```
\setuphead[chapter][conversion=Romannumerals]
\definestructureconversionset[sectionnumbers][0,R][n]
\setupheads[sectionconversionset=sectionnumbers]
```

note-49

subject: Restrict table of contents to chapters only
date: 28-9-2015
author: Wolfgang Schuster
source: ntg-context@ntg.nl
Question: how to restrict the table of contents to chapters only?
Answer: use `\placelist[chapter]`

note-50

subject: Typesetting version of LuaTeX en ConTeXt
date: 6-10-2015
author: Hans Hagen
source: ntg-context@ntg.nl
The versions of current ConTeXt and LuaTeX can be typeset with the following macros:
```
\contextversion
\the\luatexversion
\luatexbanner
```
Note the need of `\the` for the `\luatexversion`.

The current ConTEXt is version 2019.01.07 16:10.
The current LuaTEX is version 109.
Banner: This is LuaTeX, Version 1.09.0 (TeX Live 2019/dev)

note-51

subject: Stretching words to length
date: 8-10-2015
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: how to stretch words to a certain length?
Answer: use the following code sample:
```
\stretched[features=none,width=DIMENSION]{word}
```

Example: w o r d w o r d w o r d

note-52

subject: Defining a smaller bullet
date: 29-11-2015
author: Hans Hagen
source: ntg-context@ntg.nl
Qeustion: how to define a smaller bullet vertically centered?
Answer:
```
\definesymbol[smallbullet]
  [\raise.1ex\hbox{\mathematics{\scriptstyle\bullet}}]
\symbol[smallbullet]
```

Example: normal bullet = "•" small bullet = "•"

note-53

subject: Centering content vertically on the page
date: 21-12-2015
author: Wolfgang Schuster
source: ntg-context@ntg.nl
Question: vertically centering content with
```
\null\vfill … \vfill\null
```
does not work as before, how to?
Answer: define a `makeup` for the page as follows:
```
\definemakeup[centered]
    [pagestate=start,headerstate=start]
\startmakeup[centered] … \stopmakeup
```

note-54

subject: Filling and drawing together in MetaPost
date: 25-2-2016
author: Hans Hagen
source: ntg-context@ntg.nl
Besides a separate `draw` (outline) and `fill` (inside) oper-

ation, MetaFun now adds `fillup` doing both at the same time. It leads to more efficient pdf code.

note-55

subject: Different behaviour of framed with align
date: 16-5-2016
author: Wolfgang Schuster
source: ntg-context@ntg.nl

On 18-4-2013 the question arose if certain values of the align parameter in `\framed` will lead to an `\hbox` or to a `\vbox` for its content. It turns out that horizontal mode is forced by a strut preceding the content of the framed. The code `\framed[strut=no,autostrut=no]` produces vertical mode at the start of the framed in case one sets a value for the width of the framed. (Trying out all variations in align and autostrut with and without width set in an example program is left as an exercise to the reader :-)

note-56

subject: Hashmark and ampersand in alignments and preambles
date: 16-5-2016
author: Hans van der Meer
source: LuaTeX Reference

The characters # and & must be typed of old in TEX as \# and \&. For & this has changed in LuaTEX, no need for the backslash anymore; the hashmark is produced by `\letterhash = #`.

However, both characters also have a special meaning inside alignment templates `\halign` and `\valign`. For use in these templates LuaTEX now offers the aliases `#=\alignmark` and `&=\aligntab`.

note-57

subject: Extra indentation with setupdelimitedtext
date: 16-5-2016
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Question: Why is the indentation using `\startblockquote` twice what it should be with the following code?

```
\setupdelimitedtext[blockquote][
  before={\startnarrower\noindentation},
  after={\par\stopnarrower}]
```

Answer: The blockquote environment is indented by default, you can disable it with

```
\setupdelimitedtext[blockquote][leftmargin=0pt]
```

Example of both cases:

Without `leftmargin=0pt`: This is a text meant to stretch over at least two lines in the typeset text.

With `leftmargin=0pt`: This is a text meant to stretch over at least two lines in the typeset text.

note-58

subject: List of fonts installed
date: 18-5-2016
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Fonts installed on the system are listed with:

```
\usemodule[fonts-system]
\showinstalledfonts
```

note-59

subject: Floats and text on alternating pages
date: 9-9-2016
author: Hans Hagen
source: ntg-context@ntg.nl

Placing floats on the even pages and text on the odd ones in doubledsided documents can be accomplished by the following code.

```
\setuppagenumbering[alternative=doublesided]
\newtoks\SavedFloats
\appendtoks\the\SavedFloats
    \global\SavedFloats\emptytoks\page
  \to \everybeforeoutput
\page[right]
-- some text --
\appendtoks \placefigure[here]{}{}\to \SavedFloats
-- some text --
```

Each float is appended to tokenregister `\SavedFloats` which is automatically flushed before a page of text is submiited to the output.

note-60

subject: Name of current font
date: 9-10-2016
author: Hans van der Meer

Retrieve current font names with `\truefontname{style}`

The example uses:

```
fontclass = \fontclass
truefontname{Regular} = \truefontname{Regular}
etcetera.
```

fontclass	maps
Regular	file:LinLibertine_R.otf
Serif	file:LinLibertine_R.otf
Italic	file:LinLibertine_RI.otf
Bold	file:LinLibertine_RB.otf
BoldItalic	file:LinLibertine_RBI.otf
Sans	file:lmsans10-regular
SansBold	file:lmsans10-bold
Mono	file:Inconsolatazi4-Regular.otf

note-61

subject: Using TeX register values in Lua code
date: 9-10-2016
author: Hans Hagen
source: ntg-context@ntg.nl

In the example the value of a dimen register is written to a file.

```
\startluacode
  io.savedata("tempnotes.txt", "textwidth = "
    .. number.topoints(tex.dimen.textwidth)
    .. " = " .. tostring(math.floor(
      tex.dimen.textwidth/(65536*2.8452))) .. "mm")
\stopluacode
\typefile{tempnotes.txt}
```

Example:

```
textwidth = 223pt = 78mm
```

note-62

subject: Variations in alignment: maxaligned
date: 9-10-2016
author: Hans van der Meer

Besides `\leftaligned`, `\midaligned` `\rightaligned` there is a new one `\maxaligned`. Below first `\midaligned{a b c d}` and then `\maxaligned{a b c d}`, showing their difference. With `\maxaligned` the items are spread out most.

Example:

a b midaligned c d
a b maxaligned c d

note-63

subject: Formatting of caption label
date: 10-10-2016
author: Wolfgang Schuster
source: ntg-context@ntg.nl

Set the separator (= *prefix*) between the chapter/section-number and the figure/table-number (= *floatnumber*). Note that the level of the chapter or (sub)section must be given, as is done below for a section numbering.

```
\setupcaptions
  [prefixsegments=section,prefixconnector=..]
```

And add something (the *numberstopper*) after the caption label with

```
\setupfloatcaption[suffix=..]
```

The example used `$\diamond$` for the prefixconnector and `$\bullet$` for the suffix:

1 The Chapter Title

1.1 This is a section in the document

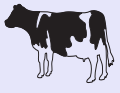


Figure 1◇1 ● Used: "◇" and "●"

note-64

subject: Footnotemarker before and after text
date: 13-10-2016
author: Hans Hagen
source: ntg-context@ntg.nl

A footnote is usually placed after the text to which it refers. It is however possible to place it at the front using the option `\setupnotes[footnote][anchor=next]`. In effect this interchanges the whitespace before and after the footnotemarker. The code for the example:

```
\startlocalfootnotes
follows text\footnote{footnote 1} ---
\setupnotes[footnote][anchor=next]
precedes \footnote{footnote 2} text
\placelocalfootnotes
\stoplocalfootnotes
```

Example:

follows text[1] — precedes [2]text

1. footnote 1
2. footnote 2

note-65

subject: Environment with typesetbuffer and getbuffer
date: 16-10-2016
author: Wolfgang Schuster
source: ntg-context@ntg.nl

This note illustrates the difference between `\typesetbuffer` and `\getbuffer`.

The command `\typesetbuffer` puts its content in an external file which is then processed as a separate document. Therefore it cannot and does not use the settings from the main document. These settings should be put inside the buffer processed by `\typesetbuffer`. This contrasts with the command `\getbuffer` where typesetting is fully embedded in the current environment.

note-66

subject: Inclusion of hbox in a MetaPost figure
date: 22-2-2017
author: Hans Hagen
source: ntg-context@ntg.nl

In MetaPost included in the text it is possible to use the contents of an `\hbox` from the TEX-side. Example:

```
\newbox\MyMpBox
\setbox\MyMpBox\hbox{foo}
\startMPcode
  draw textext("\copy\MyMpBox");
  draw textext("\copy\MyMpBox") rotated 45 shifted(1cm,0);
\stopMPcode
```

Example:

foo foo

note-67

subject: Figure on empty page
date: 24-2-2017
author: Hans Hagen
source: ntg-context@ntg.nl

Question: how to put a figure on a completely empty

page with no header and footer?
Answer: use

```
\startplacefigure[location={page,high,header,footer}]
```

note-68

subject: Matching baselines with different font sizes
date: 14-3-2017
author: Hans Hagen
source: ntg-context@ntg.nl

The problem is to typeset two chunks of text with different font size next to each other while keeping their baselines matched. It can be solved by typesetting the texts in frames put on the same baseline. Note the fancy font definition that is used to pass the font to the `foregroundstyle` parameter.

```
\definefont[whatever][Sans at 12pt]
\setupframed[frame=off,foregroundstyle=whatever]
\dontleavehmode
\inframed{TITLE}\relax
\dontleavehmode
\inframed{\definedfont[Sans at 6pt] SUB-TITLE}
```

Example: TITLE SUB-TITLE

note-69

subject: Passing parameters to MetaPost with Lua code
date: 19-3-2017
author: Hans Hagen
source: ntg-context@ntg.nl

Parameters can be directly passed to MetaPost in a Lua table. See the text 'foo' inside two concentric circles from the next example:

```
\startluacode
   document.mydata = {text="foo",size=25}
\stopluacode
\startMPcode
  draw fullcircle scaled
    lua("mp.print(document.mydata.size)");
  draw fullcircle scaled
    lua("mp.print(1.25*document.mydata.size)");
  draw textext
    (lua("mp.quoted(document.mydata.text)"));
\stopMPcode
```

Example:

note-70

subject: Name and use of current fontstyle
date: 7-4-2017
author: Hans Hagen
source: ntg-context@ntg.nl

Macro `\fontstyle` delivers the current fontstyle.
This can be used, for example, in

```
\doifelse{\fontstyle}{rm}{..}{..}
```

Also useable in Lua code with (as done in the example):

```
\startluacode
if tokens.getters.macro("fontstyle") == "rm" then
   context("Fontstyle rm here indeed.")
else
   context("Fontstyle not rm here.")
end
\stopluacode
end
```

Example: Fontstyle rm here indeed.

note-71

subject: How to expand nested doif's
date: 21-5-2017
author: Wolfgang Schuster
source: ntg-context@ntg.nl

This note explains the use of the expanded variants of macros `\doif`. These are `\expdoif` etcetera. They are needed when the conditions are nested. In the following code:

```
\doifelse{a}{\doifelse{x}{x}{a}{b}}{yes}{no}
```

the inner `\doifelse` evaluates to "a" but the outer one is not seeing this, so the result of its execution is "no". Using the expanded version for the inner test instead will present the evaluated result to the outer test and thus

```
\doifelse{a}{\expdoifelse{x}{x}{a}{b}}{yes}{no}
```

yields the expected result "yes".

note-72

subject: Am I in front- text- or backmatter?
date: 14-7-2017
author: Hans Hagen
source: ntg-context@ntg.nl

Question: How do I determine if I am in frontmatter, bodymatter or textmatter?
Answer: use the following tests:

```
\doifelsemode{*frontpart}{yes}{no}
\doifelsemode{*backpart} {yes}{no}
```

note-73

subject: Placing an ornament in the corner of a frame
date: 23-8-2017
author: Aditya Mahajan
source: ntg-context@ntg.nl

The framecorner can be embellished with an ornament, a text in this example. The ornament is just a `\framedtext` and customizable as such.

```
\defineornament[FrameTitle][alternative=a]% or b
   [frame=off,width=fit,...]
\FrameTitle{Info}{\startframedtext
    [width=fit,toffset=\lineheight]% or try it out
 This is the info.\stopframedtext}
```

In typesetting the example it appeared necessary to determine the value of `toffset` by trial and error, because the expected value of `\lineheight` didn't work out as ex-

pected. *Example:*

Info

This is the info.

note-74

subject: Difference in typing with lines=yes/no
date: 30-8-2017
author: Aditya Mahajan
source: ntg-context@ntg.nl

This example shows the (subtle) difference between the values yes and no for the `lines`-parameter in typing verbatim text.

```
\starttyping[lines=no]
Here is some text with very very long line that goes on
\stoptyping
\starttyping[lines=yes]
Here is some text with very very long line that goes on
\stoptyping
```

Example:

```
Here is some text with very very long
line that goes on [lines=no]

Here is some text with very very
long line that goes on [lines=yes]
```

note-75

subject: Breaking text with a sidefloat
date: 5-9-2017
author: Hans Hagen
source: ntg-context@ntg.nl

Question: How to break off the filling of text in the middle of sidefloat?

Answer: use `\flushsidefloats`

```
% Text before sidefloat
\startplacefigure[location={left,none}]
  \externalfigure[][]
\stopplacefigure
% Text neighbouring sidefloat
\flushsidefloats
% Text after sidefloat
```

Example:

note-76

subject: Paragraphs narrower after first line
date: 20-12-2018
author: Wolfgang Schuster
source: ntg-context@ntg.nl

The problem was to typeset paragraphs with the first line at the left margin and all other lines, possibly including those in further paragraphs, offset to the right. Be aware that its application can be a bit tricky as this author experienced.

1. Every paragraph first line shifted left:

```
\definedelimitedtext[narrowtext]
  [leftmargin=2em,indenting={yes,-2em},
    left={\dontleavehmode}]
```

2. First paragraph first line only shifted left:

```
\noindentation  % otherwise unwanted indenting
\definehspace[outdent][-2em]
\definedelimitedtext[narrowtext][leftmargin=2em,
  rightmargin=0pt,location=paragraph,
  left={\dontleavehmode\hspace[outdent]}]
```

Every paragraph first line shifted left

This is a text meant to make a long paragraph such that it stretches over more than one line in order to demonstrate the behaviour of definedelimitedtext.
This is a text meant to make a long paragraph such that it stretches over more than one line in order to demonstrate definedelimitedtext.

First paragraph only first line shifted left

This is a text meant to make a long paragraph such that it stretches over more than one line in order to demonstrate the behaviour of definedelimitedtext.
This is a text meant to make a long paragraph such that it stretches over more than one line in order to demonstrate the behaviour of definedelimitedtext.

note-77

subject: Parameter passing to Lua
date: 4-1-2019
authors: Hans Hagen, Hans van der Meer
source: Hans Hagen private communication

Question: Why does my call to Lua crashes when passing data with embedded strings?

Answer: A parameter from ConTEXt is often passed as a string, which on the Lua end is received as such. Thus one can see the following interfacing:

```
\def\mycall#1{\directlua{tex.print(me.mycall("#1"))}
```

Which when called from ConTEXt can look like:

```
\mycall{My sample text}
```

what in the following programcode on the Lua side is received in parameter `fromtex` from where it can be manipulated; for example to concatenate it with another string:

```
me.mycall = function(fromtex)
  local result = "This is received " .. fromtex
  -- Here fromtex literally is "My sample text"
```

But there is a viper hiding in this approach, which will manifest itself when the input has embedded strings; for example in:

```
\mycall{My "sample" text}
```

Lua now received `"My "sample" text"`: no longer a simple string but an expression that must be evaluated. Eventually this can lead to an (at first inconspicuous) error.

The remedy is as simple as effective: use the other more robust Lua string representation `[[...]]`. And thus the safest way to pass a string with unknown content is:

```
\def\mycall#1{\directlua{tex.print(me.mycall([[#1]]))}
```

Hans van der Meer
hvandermeer.com

LuaTEX 1.10, a stable release

Na wat eerste experimenten is het LuaTEX project in 2006 van start gegaan. In eerste instantie waren Taco Hoekwater, Hartmut Henkel en Hans Hagen betrokken, later kwam daar Luigi Scarso bij terwijl Hartmut wat uit het zicht verdween. Vanaf het begin zijn de ConTEXt gebruikers betrokken bij het project, niet in de laatste plaats omdat ze gewillige testers waren. Via `contextgarden.net` waar Mojca Miklavec de regie heeft, onder andere over de compile farm, kunnen gebruikers distributies downloaden met de meest recente features. Het is dus eigenlijk een in Nederland geïnitieerd project van de ConTEXt ontwikkelaars en gebruikers.

Een paar jaar terug is versie 1.0 van LuaTEX gelanceerd: in principe was de interface stabiel. Binnenkort lanceren we versie 1.10, met wat kleine uitbreidingen en natuurlijk verbeteringen en fixes. Dit is een verdere stap naar stabilisering. Omdat ook andere macro pakketten gebruik gingen maken van LuaTEX werd duidelijk dat verder experimenteren geen echte optie meer was, meer daarover later.

Samengevat biedt LuaTEX het volgende:

- Een up-to-date TEX engine, met een volledig UTF-8 code pad. De input is UTF, de logging ook, en karakters en shapes (glyphs) maken gebruik van de in Unicode toegestane ranges.
- Een open font interface die het mogelijk maakt zowel traditionele 8 bit fonts (meestal zijn dat Type1 fonts) te gebruiken maar ook OpenType fonts. En dat alles onder volledige controle van TEX en Lua.
- Een extensie interface die het via zogenaamde Lua callbacks mogelijk maakt de input te manipuleren, zelf extra commandos te definiëren (met behulp van scanners), en de tussenresultaten, in de regel zogenaamde node lists, te onderscheppen en manipuleren.
- Dat alles wordt gedaan in Lua, een relatief eenvoudige maar krachtige scripting taal, met een stabiele reputatie en geen afhankelijkheden van grote, steeds wijzigende en uitdijende bibliotheken.
- Een bewezen stabiel PDF backend dat efficiënt ondersteuning biedt aan uit PDF overgenomen uitbreidingen als font expansie maar ook kan omgaan met de uit Omega overgenomen mogelijkheden om bijvoorbeeld van rechts naar links te typesetten.
- Images zijn gepromoveerd naar een standaard functionaliteit. Ook is er een interface naar het image subsysteem.
- Er zijn in de loop der tijd wat extra primitieven bijgekomen (vooral handig voor ontwikkelaars) terwijl wat pdfTEX extensies zijn verdwenen omdat we Lua hebben als vervanger.
- De codebase is destijds door Taco overgezet van Pascal naar C en is in de loop der tijd uitgebreid en opgeschoond. De engine is efficiënt en goed te onderhouden. De performance is ok en als men Lua gebruikt kan men de LuaJIT variant kiezen die een snellere virtuele machine heeft.

Kortom, na meer dan een decennium is er een TEX engine die zijn werk goed kan doen en die eigenlijk vanaf het begin (in ieder geval door ConTEXt gebruikers) gewoon in productie kon worden gebruikt.

Stabiliteit is belangrijk binnen de TEX wereld en daarom is versie 1.10 min of meer het eindpunt van een ontwikkeling. Natuurlijk komen er updates, maar net zoals pdfTEX al jaren stabiel is in functionaliteit, zo zal ook LuaTEX dat moeten zijn.

Echter, omdat we nog wel wat ideeën hebben, gaan we wel door met experimenteren. Dit is binnen de ConTeXt community relatief eenvoudig. Er is bijvoorbeeld een bewezen effectieve infrastructuur die dat faciliteert. Daarnaast hebben we niet te maken met gebruikers (denk daarbij aan instellingen en uitgevers) die helemaal geen verandering willen en zelfs structurele aanpassingen in een macro pakket niet toejuichen. In ConTeXt is dat zelden een probleem omdat we een consistente interface hebben en veel is geïntegreerd. De transitie van MkII naar MkIV is redelijk soepel verlopen: de belangrijkste wijzigingen voor gebruikers hadden te maken met encodings en fonts.

Een follow-up van LuaTeX, denk aan LuaTeX versie 2+, zal dan ook wederom binnen het ConTeXt domein plaatsvinden. Op die manier heeft niemand anders er last van. Een beetje zoals in de eerste jaren van LuaTeX, toen alleen ConTeXt er gebruik van maakte en we drastisch konden experimenteren. Ik herinner me nog de tijden dat de binaries en ConTeXt code dagelijks heen en weer gingen tussen Taco en mij, waarbij we ideeën konden uittesten. Ik heb daar prettige herinneringen aan.

We kunnen sommige aspecten van een follow up terugsluizen naar LuaTeX. Standaard zal ConTeXt deze engine gebruiken en we zijn dus gebaat bij consistentie (in fixes) en functionaliteit, maar omdat we geen stabiliteit willen doorbreken is er dus gewoon een extra engine (die waarschijnlijk alleen door ConTeXt wordt gebruikt). De werktitel voor dit project is MetaTeX, een naam die al langer binnen het project wordt gebruikt, en de extra engine zal de naam `luametatex` hebben zodat er geen verwarring ontstaat met de standaard `luatex` engine. De ConTeXt gebruikers zullen vooralsnog gewoon LuaTeX gebruiken, maar kunnen dan als alternatief experimenteren met de follow up.

Gedurende de LuaTeX ontwikkeling hebben we steeds bijgehouden wat we deden en er frequent over gerapporteerd, op papier en tijdens bijeenkomsten. Er zijn inmiddels vier documenten in de ConTeXt distributie die de stadia en ontwikkelingen daarbinnen samenvatten. Een vijfde document dat de follow up bijhoudt ontstaat op het moment van dit schrijven.

Hans Hagen

Basic image formats

In TEX a graphic is not really known as graphic. The core task of the engine is to turn input into typeset paragraphs. By the time that happens the input has become a linked list of so called nodes: glyphs, kerns, glue, rules, boxes and a couple of more items. But, when doing the job, TEX is only interested in dimensions.

In traditional TEX an image inclusion happens via the extension primitive `\special`, so you can think of something:

```
\vbox to 10cm {%
  \hbox to 4cm {%
    \special
      {image foo.png width 4cm height 10cm}%
    \hss
  }%
}
```

When typesetting TEX sees a box and uses its dimensions. It doesn't care what is inside. The special itself is just a so called *whatsit* that is not interpreted. When the page is eventually shipped out, the dvi-to-whatever driver interprets the special's content and embeds the image.

It will be clear that this will only work correctly when the image dimensions are communicated. That can happen in real dimensions, but using scale factors is also a variant. In the latter case one has to somehow determine the original dimensions in order to calculate the scale factor. When you embed eps images, which is the usual case in for instance dvips, you can use TEX macros to figure out the (high res) bounding box, but for bitmaps that often meant that some external program had to do the analysis.

It sounds complex but in practice this was all quite doable. I say 'was' because nowadays most TEX users use an engine like pdfTEX that doesn't need an external program for generating the final output format. As a consequence it has built-in support for analyzing and including images. There are additional primitives that analyze the image and additional ones that inject them.

```
\pdfximage
  {foo.png}%
\pdfrefximage
  \pdflastximage
  width 4cm
  height 10cm
\relax
```

A difference with traditional TEX is that one doesn't need to wrap them into a box. This is easier on the user (not that it matters much as often a macro package hides this) but complicates the engine because suddenly it has to check a so called extension whatsit node (representing the image) for dimensions.

Therefore in LuaTEX this model has been replaced by one where an image internally is a special kind of rule, which in turn means that the code for checking the whatsit could go away as rules are already taken into account. The same is true for reusable boxes (xforms in pdf speak).

```
\useimageresource
  {foo.png}%
\saveimageresource
  \lastsavedimageresourceindex
  width 4cm
  height 10cm
\relax
```

While dvips supported eps images, pdfTEX and LuaTEX natively support png, jpg en pdf inclusion. The easiest to support is jpg because the PDF format supports so called jpg compression in its full form. The engine only has to pass the image blob plus a bit of extra information. Analyzing the file for resolution, dimensions and colorspace is relative easy: consult some tables that have this info and store it. No special libraries are needed for this type of graphic.

A bit more work is needed for pdf images. A pdf file is a collection of (possibly compressed) objects. These objects can themselves refer to other objects so basically we have a tree of objects. This means that when we embed a page from a pdf file, we start with embedding the (content stream of the) page object and then embed all the objects it refers to, which is a recursive process because those objects themselves can refer to objects. In the process we keep track of which objects are copied so that when we include another page we don't copy duplicates.

A dedicated library is used for opening the pdf file and looking for objects that tell us the dimensions and fetching objects that we need to embed. In pdfTEX

the poppler library is used, but in LuaTEX we have switched to pplib which is specially made for this engine (by Paweł Jackowski) as a consequence of some interchange that we had at the 2018 BachoTEX meeting. This change of library gives us a greater independence and a much smaller code base. After all, we only need access to pdf files and its objects.

One can naively think that png inclusion is as easy as jpg inclusion because pdf supports png compression. Well, this is indeed true, but it only supports so called png filter based compression. The image blob in a png file describes pixels in rows and columns where each row has a filter byte explaining how that row is to be interpreted. Pixel information can be derived from preceding pixels, pixels above it, or a combination. Also some averaging can come into play. This way repetitive information can (for instance) become a sequence of zeros even when actual changes in pixel values took place. And such a sequence can be compressed very well which is why the whole blob is compressed with zlib.

In pdf zlib compression can be applied to object streams so that bit is covered. In addition a stream can be png compressed, which means that it can have filter bytes that need to be interpreted. But the png file format can do more: the image blob is actually split in chunks that need to be reassembled. The image information can be interlaced which means that the whole comes in 7 separate chunks that get overlaid in increasing accuracy. Then there can be an image mask part of the blob and that mask needs to be separated in pdf (think of transparency). Pixels can refer to a palette (grayscale or color) and pixels can be encoded in 1, 2, 4, 8 or 16 bits where color images can have 3 bytes. When multiple pixels are packed into one byte they need to be expanded.

This all means that embedding a png file can demand a conversion and when you have to do that each run, it has a performance hit. Normally, in a print driven workflow, one will have straightforward png images: 1 byte or 3 bytes which no mask and not interlaced. These can be transferred directly to the pdf file. In all other cases it probably makes sense to convert the images beforehand (to simple png or just pdf).

So, to summarize the above: a modern TEX engine supports image inclusion natively but for png images you might need to convert them beforehand if runtime matters and one has to run many times.

Hans Hagen

Is TEX really slow?

The question

Sometimes you read complaints about the performance of TEX, for instance that a LuaTEX job runs slower than a pdfTEX job. But what is actually a run? Consider the following example (in ConTEXt speak):

```
\starttext
    Hello \TeX\ {\bf world}!
\stoptext
```

In the next few pages I will try to explain what happens when you process some text and why even such a simple TEX job takes about half a second to process on my laptop.

Starting up

When we start up the TEX engine (one of pdfTEX, XETEX or LuaTEX), the first thing the program does is figuring out in what environment it is running. How is the TEX resource tree organized and what format file is to be loaded (we assume a production run here)? The format file, which is just a saved memory dump, relates to a macro package and after it has been loaded additional loading can happen, triggered by the macro package. A ConTEXt format file for LuaTEX is 11.6 MB (11.0 MB for LuajitTEX), 8.3 MB for pdfTEX and 4.8 MB for XETEX. Just for the record: creating a MkIV format for LuaTEX takes 4.8 seconds (a bit less for LuajitTEX), making a MkII format for pdfTEX needs 10.8 seconds and for XETEX uses about 6.9 seconds, just to indicate the spread. If you're a LATEX user, can you predict the relative values for that package?

In LATEX, before you start the text in your document, you load one or more packages, while in ConTEXt everything is already loaded. So, due to the much smaller format file normally LATEX wins here in terms of speed, unless you start adding lots of functionality via packages. Before the actual typesetting begins for sure a couple of fonts have to be loaded and the math subsystem has to be set up. All this takes time but with ssd disks and plenty of memory on a modern machine with proper caching of files, it happens quite fast. Next time you reboot your machine, just compare the first TEX run with a successive one and you will see how fast disks and proper file caching help. When you have a short document, the startup time matters, but when you have a document of 200 pages, it can often be neglected. Nevertheless it helps accepting performance penalties when you consider what happens.

The time spent on initializing the file system depends on the size of the TEX resource tree that you use. If you decided to install all there is, you pay a price because the file databases are large. On the other hand, if you consider leaving out plain TEX or ConTEXt, you're not saving much. For instance, if you use an eight bit engine and therefore eight bit fonts, and when you then also want to typeset some cjk text, you end up with huge (for instance) ttf fonts being turned into a collection of small fonts. We're talking about hundreds of extra files here. And in a TEX tree there can be many such font collections. Another example is the TEXGyre collection. When you use a wide engine (Unicode aware) you can do with a dozen fonts (otf), but when you use an eight bit engine, you end up with hundreds of files (tfm, vf, afm, pfb, map files, etc.), each for a specific encoding. So, if speed matters: try to be lean and mean. When I started with using LuaTEX in ConTEXt, we already were using so called minimals with pdfTEX: an as small as possible TEX tree. By using Lua instead of the built-in libraries we could actually speed up the startup even more.

Loading the format itself is basically just copying bytes to well defined memory locations. The only factor that can delay this is when we use formats that are portable across operating systems and hardware architectures. In that case we get a hit from byte swapping: little endian becomes big endian. The interesting fact here is that most users are on platforms that suffer from this swapping, but the good news is that nowadays distributions don't use portable formats. When we found out that this delayed format creation and loading, in LuaTEX we therefore permanently disabled this feature. We also compress the format (zip level 3) which speeds up loading too. One reason why a pdfTEX format is relatively large is that it is not compressed while it has a lot of hyphenation patterns embedded too. The reason why a ConTEXt format file for LuaTEX is much larger than one for XETEX (which also has patterns) is that in MkIV (which is ConTEXt for LuaTEX) we have quite a lot of Lua code and also quite some metadata about for instance characters, something that XETEX gets from elsewhere.

So, when the bytes in the format file have become tokens in TEX memory space, the job can start. Loading a bunch of extra macros is pretty fast, and often not measurable, but there are exceptions. If you load for instance `tikz` a truckload of files gets looked up and loaded, and that takes time. Loading additional files is not just copying bytes, but involves converting them to tokens, storing them in memory, maybe doing some calculations, etc. It sounds bad, but you just get what you ask for and the rewards are probably worth it. The only difference between engines in this stage is that XETEX and LuaTEX are using utf which involves a bit more work in parsing the input.

Nowadays fonts are seldom preloaded so the ones to be used have to be read from disk and converted into a suitable format for TEX. And this is again where differences between engines start showing up. Loading a tfm file is pretty fast as it's just some byte juggling and there are not that many bytes involved: widths, a limited set of heights and depths, possibly a handful of ligature definitions and some kerning pairs. A wide engine has to interpret a rather large OpenType font resource and filter the information that it needs: dimensions but also OpenType features (although this depends on how these are processed). Some of these properties are passed from Lua to TEX. However, when you use more than basic Latin, loading these much larger files pays off because we don't need to deal with specific encoding related instances.

Somewhat related is loading of hyphenation patterns. In LuaTEX this is delayed so you only load what is needed but otherwise these are part of the format file and the more languages a package supports, the more get loaded. Because nowadays patterns are defined in utf this means that making a format file (which doesn't happen often) takes more time in pdfTEX than in the other engines, because this encoding is to be somehow mapped onto a (bunch of) eight bit encoding(s). This brings us to the users input.

Often TEX is used for English documents but what if your document is encoded in utf and has substantial amounts of Greek, Cyrillic or Chinese? In that case you end up with lots of two and three byte codes. In pdfTEX this can be handled by making the first byte an active character (a command) that then looks ahead and interprets the following bytes. This is often not enough because each code range might demand its own font, so switching fonts is needed too. In XETEX and LuaTEX that comes for free. So, conclusions about LuaTEX being slower than pdfTEX depend on the language and script you use! For instance, in pdfTEX typesetting Arabic is macro magic, while in LuaTEX Lua and in XETEX the engine do the work.

We already mentioned math. Here it depends on the way math is supported in a macro package. Do we use many families and eight bit fonts, or do we use OpenType math, or do we use a mixture? Do we set up most and store it in the format, or do we delay this till runtime? Do we want to mix different math fonts in one document? Do we use regular and bold (heavy) math? Do we support bidirectional math?

Anyway, by the time we're past `\starttext` and can start with typesetting, we already let the engine do a lot. When you're staring at your console, also realize that this all happens in an interpreted language, where macros get expanded, token lists get created and destroyed and all calculations happen by interpretation. Often TEX looks ahead and has to push back tokens into its input. And, grouping means that we have a save stack so that after a group ends adapted registers have to be restored. Although it is not strictly true, you can consider TEX to run on a virtual machine with a large instruction set. And as the meaning or macros can change any time, there is not that much just-in-time optimization possible. If you say `\tracingall` at the top of you document you get an idea what we're talking about.

Typesetting text

When TEX sees the `Hello \TeX\ {\bf world}!` the `H` tells the engine to start a new paragraph. That itself can trigger a lot because a macro package can hook all kind of actions into `\everypar`. But, when that is done, TEX starts consuming the characters that make up the paragraph and it expands macros on its way till `\par` or an empty line is seen. At that moment characters, kerns, penalties, glue, rules ... all became nodes that got appended to the current node list. When done with that TEX will launch the par builder.

There is a conceptual difference between pdfTEX and LuaTEX with respect to the paragraph builder. I can't speak of XETEX as I don't know how that works internally but it can't be far from either of these two. In pdfTEX constructing the so called node list and figuring out line breaks is interleaved with hyphenation, font kerning and ligature building. Traditional TEX is very optimized to do only what is needed here.

In LuaTEX these stages are split: we collect, then we hyphenate, build ligatures, kern glyphs, and then break the result into lines. Each is optional and can be overloaded by callbacks (which is why there are split stages). This is how for instance OpenType font features can be applied and special demands of scripts can be met: by intercepting the node list at certain points in the process. Performance wise, pdfTEX is the winner here. But, only when we're talking basic Latin. Anything more complex, and especially a mix of languages and scripts pays a price. Even then XETEX and LuaTEX can be slower simply because more advanced font features are applied. Quality carries a burden (although today's

documents in most cases don't look much better than before).

The paragraph building itself is more or less the same in the engines: first TEX tries without hyphenation. Keep in mind that in LuaTEX the list always is in a hyphenated form, i.e. has discretionary nodes added. If that fails, a hyphenated pass is applied and when TEX is not satisfied, an emergency pass can happen that will stretch or shrink spacing to the extent that makes all happy.

The pdfTEX engine introduced protrusion (hanging glyphs in the margin) and expansion (stretching glyphs so that excessive spacing become less prominent). It uses additional font instances that get created on the fly. In LuaTEX we support the same but don't do it the same way because there we keep information in the glyph nodes. This is more efficient and also gives nicer code. No matter what method is chosen, enabling these mechanisms hit performance. And it looks like some TEXies really think that all will look better so they enable this feature by default, so they always suffer.

As an intermezzo: enabling something by default can always come at a price. A good example is `synctex`, which will add a 5 to 10% overhead to a run. The same is true for inefficient styles. You can load lots of fonts that you never use and it will add runtime.

After the par builder has done its work it hands over the result to the box builders or page builder. There decisions will be made and additional action can be triggered. For instance, the page builder can decide to launch the output routine which is a hook that can do lots of things, depending on the macro package. One of these can be constructing the page body, adding headers and footers and shipping out the page to the output medium, for instance a pdf file.

Shipping out a page is not that spectacular. The page is just a nested linked list that gets serialized to a stream of pdf codes. But, in the process information is collected about what characters from what fonts are used. If used, hyperlinks can be injected. Location specific information can be flushed to an auxiliary file. The more features you enable the more runtime is involved. There is probably not that much difference between the engines here.

Because in LuaTEX nearly all steps can be replaced or extended by callbacks (Lua functions) a macro package can add its own overhead. And here is where comments about performance become somewhat lame. The more you hook in, the more overhead is added. And the worse the code is, the larger the penalty. One cannot blame an engine for that. Because the LuaTEX engine itself is quite efficient, users (or macro packages) can add seconds or even minutes to a run. It is often easier to blame LuaTEX than your own programming skills and/or demands.

This effect is not limited to callbacks. Macros and rendering feature related mechanisms can be inefficient too. A macro package writer can pay attention to that, but a user can have a dramatic impact by adding bad macros, redundant font switches, useless calculations etc. It can really add up! Of course this also relates to how often something is used. For instance, an image inclusion subsystem can involve lots of (possibly inefficient) code, but because the number of images is normally small it has no significant impact on the run. The final inclusion of the resource will definitely have more impact.

In the small example text above, not much is happening: only a font switch. But again, a simple `\bf` can be either a straightforward switch to a font (basically changing the current font id) or it can involve some more: housekeeping, loading a font, triggering related mechanisms to also adapt to bold, etc. In most cases one can assume that a macro package writer has done a decent job on it.

In this small sentence we see the `\TeX` macro. The original definition by Don Knuth is not that complex but still involves moving glyphs around. In ConTEXt the definition is such that the rendering adapts itself to the current font as good as possible. You really don't want to see the full expansion (10.000 such expansions take half a second so in practice it goes unnoticed.)

Wrapping up

Once we've arrived at the `\stoptext` the engine needs to wrap up the result. At that moment, it is known which fonts are used and what characters from these fonts are referred to. The shapes of the used glyphs need to be embedded. Normally this process is quite efficient, but just as one can see some hiccup before `\starttext`, an extra hiccup can happen after `\stoptext` (or whatever your macro package uses).

After the run, a decision has to be made about successive runs to get the table of contents right, fix cross references, sort registers, massage bibliographies and more. In ConTEXt dealing with this is part of the regular run but one can also delegate this to an external program. Anyhow it adds to the runtime (or time between runs). Macro packages differ in the way they deal with this.

Conclusion

In ConTEXt performance is measured in pages per second. An average document does between 20 and 30 pages per second. Of course when you need multiple runs the effective average drops. But, because input can also be for instance xml, performance is then also influenced by interpreting this format. And, when your pdf needs to be tagged, again some overhead can be added (typically you can delay that overhead till the

final run). When you have all kinds of MetaPost code, some runtime gets added (but not much) even when that happens realtime during typesetting. When you use color or backgrounds, in text or tables or in the page body, it comes at a price. But, even then, runtime is still acceptable: processing the 300 page LuaTEX manual on my laptop currently takes some 13 seconds and although processors don't become faster I bet that on a more modern machine it goes below 10 seconds (maybe even lower than I expect) but I can't test that right now.

So let's summarize the above. When you feel that your TEX job runs slow, try to answer these questions:

- ☐ What engine do I use, an eight bit or a Unicode aware one?
- ☐ What kind of fonts do I use, eight bit (Type1) or OpenType?
- ☐ How much do I ask from the font system?
- ☐ Do I really need expansion and/or protrusion?
- ☐ How much math magic do I need?
- ☐ Do I really need to load all these extra packages (modules)?
- ☐ Is my styling okay?
- ☐ Are my own macros, when used in abundance, top notch?
- ☐ Do I really need to enable all these features now?

Quite likely pdfTEX, XƎTEX and LuaTEX will all stay around, so you can choose whatever suits you best. However, the choice might also depend on to what extent the macro package supports all engines. For ConTEXt there is not much choice. All recent development relates to LuaTEX, so you're stuck with that. The good news is that on average a LuaTEX run is faster than one with pdfTEX or XƎTEX. It also offers way more, which is why most users made the switch.

Typesetting the MetaFun manual could easily take many minutes in pdfTEX, but in LuaTEX it takes less than 20 seconds. We sometimes process collections of xml files, for instance math schoolbooks of hundreds of pages. Thousands of small files are combined runtime based on student profiles and the colorful result has more (small) images than pages, typeset alongside the text. Realtime content selection, xml coding error correction, all happens each run. This takes less than a minute for the few runs needed to construct the document compared to close to an hour in the pdfTEX setup (if it's possible at all). Comparing performance is more than clocking a run.

When pondering processing speed, it might help to think of what actually has to happen when your document gets processed. Then you might also consider how fast a 300 page equivalent webpage (with the same amount of tables, images, math, etc) would render and with what memory footprint (apart from assembling that page). Or, how would a word processor do a change in page 5 that affects all following pages. Or how does a desktop publishing system deal with your work in progress in terms of preview, memory management, generating output, etc. Probably TEX and friends, which are rather robust and reliable, won't come out that bad. So we end with asking ourselves:

- ☐ Is the alternative, using another program, really faster?

An the equally valid question:

- ☐ Can an alternative tool support me in a way that I like?

Hans Hagen

Dagboek van een Informaticus

Mijn eerste ervaring—of aanvaring?—met TEX moet zijn geweest ergens rond 1995 zijn geweest, als jonge twintiger, student informatica aan de universiteit van Amsterdam. U moet zich de situatie trachten voor te stellen: ik woonde nog altijd thuis bij mijn ouders, in een dorpje ver van Amsterdam. De keuze voor die studie was ingegeven door een fascinatie met computers die begon toen in mijn lagere-schooljaren een Commodore 64 het huis binnenkwam en die nooit meer voorbij is gegaan. De Commodore werd verdrongen door een opeenvolgende reeks van *IBM compatible* PC's die vader via het PC-Privé programma had aangeschaft. Eerst uitgerust met MS-DOS en later Microsoft Windows 3.1.

Het contrast met de computers op de universiteit was groot. Als tweedejaars student kregen we een UNIX account op het SunOS systeem van de faculteit, wat ook meteen het portaal naar de rest van de wereld was. Ik kreeg mijn eerste e-mailaccount en bijdehantere medestudenten lieten me zien welke FTP sites de beste spullen hadden.

De grafische Sun werkstations hadden een heel ander uiterlijk, gaven een heel ander *gevoel* dan de PC van thuis. Het oogde allemaal wat professioneler, voelde wat stabieler. Mettertijd leek het thuis-PC'tje steeds meer te krimpen, steeds onbehaaglijker te voelen, alsof het te heet werd gewassen. Naarmate ik behendiger werd op de UNIX commandoregel, kreeg mijn interactie met de DOS-prompt steeds meer het karakter van een gesprek met de dorpsgek: korte zinnen, geen moeilijke woorden gebruiken.

In één opzicht was de PC thuis echter superieur: deze was uitgerust met Ami Pro, een grafisch *desktop-publishing*-programma wat zijn gelijke niet kende op de Sun machines. Het programma kon moeiteloos overweg met een keur aan TrueType lettertypes en maakte het een fluitje van een cent om stijlen te definiëren voor koppen, broodtekst, opsommingen, etc. Alle typische elementen waren aanwezig: uitlijning, werken in kolommen, witruimtes, bladspiegel, inhoudsopgave, nummering, referenties, invoegen van afbeeldingen; in die tijd had het naar mijn smaak weinig tekortkomingen. Uiteraard is dit product van de markt gedrukt door het onontkoombare Microsoft Word, maar dit terzijde.

Ik herinner me dat ik na wat vruchteloos rondzoeken op het Internet van vóór het wereldwijde web bij de systeembeheerder op de faculteit binnenstapte en vroeg wat men eigenlijk gebruikte voor het maken van drukwerk. Het antwoord was nogal kortaf: dat was TEX en/of LATEX en daarmee kon ik het doen. Hierop volgde dus mijn eerste aanvaring met dat systeem en dat ging ongeveer als volgt.

Ik logde in op één van de werkstations in de practicumlokalen, opende een Xterm en tikte:

```
tex
```

waarop het programma mij onverwijld antwoordde:

```
This is TeX, Version 3.14159
**
```

met de cursor achter de beide sterretjes. Er werd dus iets van mij verwacht. Ik had ook wel iets verwacht en zeker meer dan alleen dit. Ik weet niet meer wat ik hierna ingetikt heb, maar het leidde nergens toe. De geheimen van TEX bleven voor mij vooralsnog verborgen.

Ik vroeg een oudere medestudent om hulp, en die legde mij eerst uit dat ik ten eerste geen TEX maar LATEX moest gebruiken (maar dat commando gaf precies hetzelfde resultaat toen ik het probeerde) en ten tweede dat ik eerst een bestand moest schrijven met een zekere structuur en een groot aantal mysterieuze commando's en ten derde dat ik eigenlijk het boek van Lamport zou moeten aanschaffen.

Dit maakte mij erg boos. Hoe kon je arme studenten nu dwingen om zulke dure boeken te kopen voor zoiets essentieels! Maar een informaticastudent laat zich niet zo gemakkelijk uit het veld slaan. Voor een veteraan van vakken zoals complexiteitstheorie is geen uitdaging te groot.

Aangezien het wereldwijde web nog even op zich liet wachten moest ik het hebben van FTP en Usenet, maar uiteindelijk ontdekte ik de schatkamer van CTAN. Alles wat ik ooit zou willen weten was daar te vinden. Na wat gesnuffel ontdekte ik de broncode van het TEXbook, de officiële handleiding van het systeem, en ik besloot dat dit het boek moest zijn wat ik zou gaan lezen om het systeem te leren kennen. Ik leg hier een bekentenis af, namelijk dat ik als arme student de ingebouwde beveiliging uit de broncode heb verwijderd om het boek in zijn volledigheid (behalve de illustraties) te kunnen afdrukken op de nagelnieuwe laserprinter van de faculteit. Ik heb het later (toen ik wat salaris had) wel met Knuth goedgemaakt door verschillende boeken van zijn hand te kopen.

Voor wie het TEXbook niet kent: behalve dat het een uitputtende handleiding is van TEX zelf, is het ook een leerzaam en leesbaar werk. Vooral de eerste paar hoofdstukken vormen een heldere introductie van de achterliggende bedoeling van het programma. Knuth geeft bij sommige paragrafen met verkeersborden aan dat er een gevaarlijke bocht volgt in de lijn der gedachten, en hij raadt aan om bij eerste lezing deze paragrafen over te slaan. Ik kan deze raad van harte onderschrijven, zeker omdat ik slachtoffer werd van mijn eigenwijsheid en zelfoverschatting toen ik toch alles wilde weten. Het boek wordt bij eerste lezing veel korter en verzandt niet in details.

Wat ik ook leerde van dit boek was dat ik er toch verstandiger aan zou doen om me verder toe te leggen op LaTEX, want dat *format* had meer hulpstukken ingebouwd voor het schrijven van kant en klare documenten. Uiteindelijk heb ik de bijbehorende boeken ook maar gekocht (en er geen spijt van gekregen).

Als informaticus sprak LaTEX me enorm aan. Wat kon er nou leuker zijn dan een document schrijven als een computerprogramma? De resultaten waren oogverblindend mooi, hier kon de tekstverwerker thuis niet aan tippen. Wat me er uiteraard toe bracht om thuis ook met TEX aan de slag te willen. Aangezien de thuiscomputer geen internet had, zat er niets anders op dan de MikTeX installatie te downloaden op de universiteit en deze op paar dozijn 3.5"floppy's te transporteren naar huis. (Later zou ik hetzelfde doen met mijn eerste Linux distributie, maar dan met een paar honderd floppy's. De thuis-PC werd dual-boot en ik zou voor altijd afscheid nemen van Dos en Windows.) Dit luidde een nieuwe fase in van experimenteren met LaTEX.

Beheersing van dit obscure systeem gaf me een machtig gevoel. De beheersing en controle over elk aspect van de drukkunst (maar dan in digitale vorm) wakkerde bij mij het gevoel aan in de voetsporen van Gütenberg te zijn getreden. De kunst van goede typografie, zo heb ik later geleerd, is dat het eigenlijk helemaal niet mag opvallen. Ik was toen nog niet zo wijs, maar ik begon al wel een zintuig te ontwikkelen voor slechte typografie. Dit zintuig is zowel een zegen als een vloek, want wie het eenmaal heeft ziet het overal waar hij kijkt; zolang mensen met Word en Powerpoint blijven werken zal daar geen verandering in komen.

Een van de toepassingen had te maken met het vak calculus. Hoewel gezegend met een redelijke wiskundeknobbel heb ik een aantal vaardigheden zoals integraalrekenen op school danig verwaarloosd, en daar in mijn studie flink spijt van gekregen. Ik moest serieus aan de bak voor dat vak, en na een paar mislukte pogingen besefte ik dat het maken van het huiswerk en de daarmee gepaard gaande oefening onontbeerlijk was. Dat het huiswerk echter stomvervelend was maakte het allemaal niet leuker. Wat het natuurlijk wel leuker maakte was om al het huiswerk uit te werken in LaTEX. De docent was er in ieder geval van onder de indruk.

Het echte werk begon bij mijn schaatsclub. De secretaris van de club was een echte duizendpoot, die naast al het geregel ook het mededelingenblad verzorgde. Dit was een maandelijks boekwerkje vol met wedstrijduitslagen, overzichten en dergelijke, met veel zorg en liefde bijeengeknipt en geplakt en gekopieerd in honderdvoud op de kopieermachine op zijn werk. Het schrijfwerk leek afkomstig van een ouderwetse schrijfmachine. Hier lag een taak voor iemand met typgrafische en redactionele vaardigheden, die bovendien ingewijd was in een geheimzinnig systeem voor het produceren van ongeëvenaarde kwaliteit drukwerk. Ik bood aan het samenstellen van het clubblad voor mijn rekening te nemen en nam me voor dit naar een hoger niveau te tillen.

Dennis van Dok

Belangrijke onderdelen voor een programmaboekje

Abstract
Al vele jaren maak ik programmaboekjes voor het Nieuwegeins Kamerkoor, en steeds met LaTeX, resp. LuaTeX. Dit artikel beschrijft enkele macro's die ik heb gebruikt voor het maken van het boekje. Het op de pagina – doorgaans A5 – uitlijnen van een liedtekst en de vertaling daarvan is meestal handwerk. Het package *verse* bleek niet geschikt, dit artikeltje bevat ondermeer een alternatief.

Keywords
gedicht met vertaling, kolommen, bladvullende kop, programmaboekje

Het begin

Het maken van een programmaboekje voor een concert van ons kamerkoor is altijd veel werk:

☐ maken van een attractieve voorpagina (meestal op basis van het affiche dat al eerder voor het concert is gemaakt)

☐ verzamelen van teksten van het bestuur, over de dirigent en solisten, over de componist(en)

☐ de liedteksten met bijbehorende vertaling

☐ illustraties verzamelen

☐ advertenties aanvragen en/of aanpassen.

Een collegazanger zorgt meestal voor de vertalingen, die worden in MS-Word aangeleverd.

Ik concentreer me hier op de weergave van de liedteksten. In de boekjes is het gebruikelijk de originele tekst van een stuk – de *gezongen* teksten – links op de pagina te plaatsen en de corresponderende vertalingen aan de rechterzijde. A5 is daar een beetje krap voor en het *verse*-package doet veel indentering en biedt geen optie (voorzover ik weet) voor een vertalingskolom. Bovendien is het een gedoe om (bijna) iedere dichtregel te voorzien van \\. Hetzelfde geldt voor `\usepackage[lyric]{songs}`.

Siep Kroonenberg deed mij indertijd een aantal suggesties m.b.t. de opzet van een class-bestand, en ik voegde de tweekolomsweergave van de liedteksten toe: `\beginvers`, `\vertaaldvers` en `\eindevers`.

Waarom is dit bruikbaar: je kunt zelf de indeling bepalen van de teksten, je hoeft geen \\ om een regel af te breken, een lege regel hoef je niet te forceren met `\lr` of `\\[\baselineskip]`, gewoon een of meer regels leeg laten.

Te lange liedregels zijn natuurlijk vaak een probleem, zoek een goed punt om zo'n lange regel af te breken, en het tweede deel van die regel kun je zo nodig met spaties laten inspringen. Te lange regels worden overigens automatisch afgebroken, maar dat wordt niet mooi. Het blijft handwerk.

Er zijn natuurlijk liedteksten die niet vertaald hoeven, daarvoor is het paar `\beginversgeenvertaling` en `\eindevers` beschikbaar, voor lange regels is dan natuurlijk de hele `\textwidth` beschikbaar.

Voorbeelden van de toepassing (zie eerste afbeelding):

```
\beginvers
So at last, our fable
            has reached its final page.
Or, is it the beginning
            of another timeless age?

We roam, we stay, we cry,
and laugh with pain.
Searching for the answers, but all in vain.
...
\vertaaldvers
Zo zijn we aan het slot
               van ons verhaal gekomen.
Of is dit het begin van nieuwe tijden?


We zwerven, we blijven, we huilen,
en lachen ondanks ons verdriet.
We zijn op zoek naar antwoorden,
             maar vinden er geen.
...
\eindevers
```

Droom van de rode kamer

Epilogue **Het verhaal op de steen**

Een ronddolende monnik stuit op de magische steen en schrijft er het verhaal van de Jia-clan op. Is het wellicht de Baoyu zelf die de karakters geduldig in de steen grift? Fictie en werkelijkheid lopen hier in elkaar over.

So at last, our fable
has reached its final page.
Or, is it the beginning
of another timeless age?

Zo zijn we aan het slot
van ons verhaal gekomen.
Of is dit het begin van nieuwe tijden?

We roam, we stay, we cry,
and laugh with pain.
Searching for the answers, but all in vain.

We zwerven, we blijven, we huilen,
en lachen ondanks ons verdriet.
We zijn op zoek naar antwoorden,
maar vinden er geen.

For this is merely a story
told by a nameless stone,
penned with joyful cheers,
carved by hot and bitter tears.

Want dit is slechts een verhaal
verteld door een naamloze steen,
blijmoedig opgeschreven,
gekerfd met hete en bittere tranen.

Everyone call the author fool;
none his honest message hears.

Iedereen noemt de schrijver een dwaas;
niemand die
zijn oprechte boodschap hoort.

The wind,
the blossoms,
the moonlight,
the snow...
and this Dream of Red chamber,
that my grief for the lost golden days
may disclose.

De wind,
de bloesem,
het maanlicht,
de sneeuw...
en deze Droom van Rode Kamer,
onthullen mijn verdriet
om de verloren bloeitijd.

Ah...

Ah...

test van het gebruik van twee kolommen:

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.
Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Nieuwegeins Kamerkoor
Ernst van der Storm

Als morgen het volgende leven aanbreekt **Rúguǒ míngtiān jiùshì xià yīshēng**

Met iedere ademhaling gaat de tijd aan ons voorbij.
Stoppen onze dromen ook wanneer we het einde bereiken?
Iedere ochtend en avond wordt ons lichaam zwakker.
Op de dag des oordeels, heeft je hart dan vrede gevonden?

Onze omgeving is het toneel voor vreemde dingen,
al zoekend en tastend willen we weten
hoe we een gebroken parelketting weer heel moeten maken
en hoe we weggevallen noten moeten zingen.

Als morgen het einde van je leven is,
wat zou je vandaag dan doen?
Als morgen het einde van je leven is,
wat zou je vandaag dan doen?

Ik bewaak mijn leven met warmte,
en laat een spoor in de levensrivier achter.
Ik koester bewust mijn gezondheid,
en ontsteek het licht van mijn leven.
% let op: commentaar in het vers komt wél op de pagina!

vertaaldgedicht.tex gegenereerd op 15 juni 2018 om 20:50 u

1 2

De macro's uit de classfile:

```
% commentaar in de vers-omgeving geeft verrassingen,
% vermijd in de verzen de "%" en macroaanroepen
\newcommand{\beginvers}{\begin{minipage}%
[t]{.48\textwidth}%
\begin{alltt}\sloppy\normalfont}
% onvertaalde tekst links

\newcommand{\vertaaldvers}{\end{alltt}%
\end{minipage}\hfill\beginvers}
% vertaling rechts

\newcommand{\eindevers}{\end{alltt}%
\end{minipage}} % afsluiting

\newcommand{\beginversgeenvertaling}%
{\begin{minipage}[t]{1\textwidth}%
\begin{alltt}\normalfont}
```

Paginabrede kop

De kop boven het weergegeven artikel heb ik gemaakt met

```
\newcommand\gezelschap[1]%
  {\resizebox
    {\linewidth}{!}
    {\bfseries \#1\par}} % sk autosize
```

(suggestie Siep) de in #1 aangeleverde string wordt aan de tekstbreedte aangepast door automatisch de omhullende box en daarmee de fontgrootte te bepalen.

De kolommen-omgeving

Siep maakte op basis van multicols een eenvoudige afgeleide daarvan. De tekst loopt mooi door over een paginagrens naar de volgende pagina twee kolommen.

Tot slot

Als afgeleide van het voorstel-class-bestand van Siep Kroonenberg heb ik een miniversie *boekje9A5min.cls* gemaakt waar allerlei kleine macro's *niet* in zitten, onder andere die voor de figuren/advertenties, en die voor de programmapagina, omdat die iedere keer verandert.

Bovendien moet het klein blijven.

Ernst van der Storm
Nieuwegein

MuPDF Tools

Abstract
The application MuPDF (http://mupdf.com) is a very fast, portable, open-source PDF previewer and development toolkit actively supported by Artifex, the creators of GhostScript (http://artifex.com).
But MuPDF is not *just* a very fast, portable, open-source PDF previewer and toolkit. It also comes with a handy collection of command-line tools that are easily overlooked.
The command line tools allow you to annotate, edit, and convert documents to other formats such as HTML, SVG, PDF, and PNG. You can also write scripts to manipulate documents using Javascript.
This small paper gives a quick overview of the possibilities.

Introduction

In recent versions of the MuPDF distribution, most of the tools have been combined into a single front-end program called `mutool`. This combines the functionality of the about half a dozen programs from earlier releases. If you use an older version of MuPDF, there will be little programs like `muclean`, but in the new combined version that functionality is now available as `mutool clean`. A similar command-line adjustment is needed for the other old command-line tool names.

In the following, I am using MuPDF 1.14.0. While the functionality of the separate tools remains roughly the same, not all versions of MuPDF have the exact same options. If you want to know the options that 'your' version of a command supports, just key in the name without arguments. Just `mutool` will provide a list of all known tools, and *e.g.* `mutool clean` will show the list of options specific to the `clean` tool.

mutool clean – rewrite PDF file

If you are familiar with the general structure of PDF documents and you often work with PDF documents handed to you from other sources, this is probably the most valuable of all the tools.

Its main purpose is to 'clean up' a PDF. It can perform garbage collection on unused objects and clean up the page streams.

`mutool clean` can also convert a PDF into (near) ASCII by decompressing all the internal structures. The output is still a valid PDF, but since it has very little to no binary data any more, it can easily be inspected and possibly edited in a regular text editor. Beware though: this tends to make the file larger.

Alternatively, `mutool clean` can also convert a PDF into 'linearized' format for distribution on the web.

mutool convert – convert document

Like the name suggests, this tool can convert a PDF file into a variety of different formats. Noteworthy supported formats in version 1.14 are: PNG, PNM, PCL, PostScript, PDF, SVG, HTML and plain text. Each of these has a number of sub-options to control the output format.

This is like the `convert` command from ImageMagick, except you do not need to have Ghostscript installed, it is generally faster, and uses less memory. On the down side, there are fewer output formats and options supported.

mutool create – create PDF document

With `mutool create`, you can create a PDF from text snippets that specify a page content stream. Each of these snippets becomes a page in the output PDF. It parses some special comments inside of those snippets to define images and fonts and page size, so for example a snippet could look like this:

```
%%MediaBox 0 0 300 300
%%Image Im0 /Users/taco/Downloads/22843.png
% Draw an image.
q
200 0 0 200 50 50 cm
/Im0 Do
Q
```

and the result would be a one-page PDF with that image centered in the page.

Because you will have to write the page content stream, this is not a tool for beginners in PDF. But it is a lot easier to create a PDF this way than to write the PDF completely from scratch, because the required PDF objects and object references are generated by `mutool create`. Nevertheless, just using TEX is easier (albeit not as fast).

mutool draw – convert document

This is like `mutool convert`, except that it has more difficult to use options and uses a different syntax for those options. It is better to first see whether `mutool convert` can do what you want and only if it cannot, then look at `mutool draw`.

mutool trace – trace device calls
Produces a debug dump of the PDF document as an XML file. This can be useful to track what the MuPDF library is actually doing, but too much information is lost from the PDF to do much else (at least, that is my experience so far).

mutool extract – extract font and image resources
Extracts images and embedded font resources from a PDF document, dumping them as separate files in the current directory.

mutool info – show information about PDF resources
Dumps detailed information about various PDF document internals to the standard output.

mutool merge – merge pages from multiple PDF sources into a new PDF
Combines one or more PDF documents (or pages from them) into a new combined PDF document. The fact that it can combine 'one' PDF means that this is an easy way to extract pages from a PDF. In fact, this is what I use to generate the separate article files for the on-line version of the Maps.

mutool pages – show information about PDF pages
In particular, `mutool pages` shows the various bounding boxes for all of the pages in XML format on the standard output.

mutool portfolio – manipulate PDF portfolios
PDF portfolios are a way of putting multiple independent PDFs into a single container PDF. With `mutool portfolio` you can create or modify the contents of such a portfolio.

mutool poster – split large page into many tiles
Splits each of the pages inside a PDF into tiles that then become the separate pages of the output PDF. It does not alter the page streams, `mutool poster` just creates adjusted bounding boxes for those separate pages so that they offer a different viewport to the original content stream.

This is a useful trick if you want to print a large PDF on a desktop printer, but it does not offer any extra features like registration marks.

mutool sign – manipulate PDF digital signatures
For now (this is a very new tool) this does nothing except verify an existing signature in a signed PDF document. And mine does not even do that yet, because it seems to be a build option that it not turned on in the MacOS version…

mutool run – run javascript
The MuPDF library has a quite elaborate interface to javascript that can be used with `mutool run` to execute scripts. These scripts have direct access to the command-line and the MuPDF internals both for interpreting and for creating PDF documents, so very powerful things can be done.

The full interface is documented on the Artifex web pages.

mutool show – show internal PDF objects
`mutool show` is a tool for displaying the internal objects in a PDF file on the standard output. This can be very useful in combination with 'grep' for example, or if you do not want to load a multi-megabyte PDF in a text editor.

Taco Hoekwater

Een kleine wegwijzer naar TEX documentatie

Abstract
Het kan lastig zijn om de informatie te vinden die je nodig hebt, en dat geldt ook voor TEX- en LaTEX-gebruikers. Maar ik hoop hier te laten zien dat je meestal niet lang hoeft te zoeken. TeX Live en MiKTeX installeren vrijwel volledige documentatie[1]. Online zijn er eveneens zeer complete en goed doorzoekbare overzichten.

Geïnstalleerde documentatie

De handleiding voor een specifiek pakket kan worden opgevraagd door op de command-line

```
texdoc pakket-naam
```

in te typen. Dit werkt zowel bij MiKTeX als bij TeX Live[2].

TeX Live installeert ook een bestand `doc.html` in de root van de installatie. Dit bestand bevat links naar alle pakket- en andere documentatie die is inbegrepen in TeX Live (tenzij er bij installatie is gekozen om de documentatie niet te installeren).

MiKTeX ontbeert een dergelijk overzicht, maar de CTAN catalogus, zie hieronder, is een goed alternatief.

De CTAN catalogus

CTAN (`http://mirror.ctan.org/`) is het centrale archief voor TEX-gerelateerde software. Ga naar `https://www.ctan.org/` om software en documentatie te zoeken. Met de 'Extended search' faciliteit kun je op diverse kenmerken zoeken, onder andere 'package descriptions', 'Topics' en 'File names'. Als je één of andere kant-en-klare oplossing nodig hebt dan is dit een goede plek om te zoeken.

Elke package heeft zijn eigen pagina. Deze pagina bevat een aantal nuttige items, waaronder:

□ links naar documentatie. Als je nog niet precies weet of het pakket iets voor je is kun je hiermee zonder verdere omhaal een eerste indruk krijgen.

□ een vermelding van de beschikbaarheid onder MiKTeX en TeX Live, of anders een download link

Soms zijn er suggesties voor alternatieven. Het kan nooit kwaad om even verder te kijken, vooral als het pakket al erg oud is.

Helaas is er niet langer meer een pagina met *alle* packages, elk met een korte beschrijving.

LaTEX inleiding en naslag

Speciaal aanbevolen:

□ *The Not so Short Introduction to LaTEX2e* is een uitgebreide inleiding die regelmatig wordt bijgewerkt. Er is een Nederlandse vertaling, maar die is zwaar verouderd. Pakket-naam: `lshort-english`.

□ *LaTEX2e: An unofficial reference manual* is recentelijk fors uitgebreid met meer geavanceerde onderwerpen. Deze handleiding maakt ook deel uit van de documentatie voor Texmaker en TeXstudio. Pakket-naam: `latex2e-help-texinfo`

□ Hét naslagwerk voor symbolen is *The Comprehensive LaTEX Symbol List*, pakket-naam: `comprehensive`.

Lettertypes

Een fantastisch overzicht van beschikbare lettertypes vind je op de `http://www.tug.dk/FontCatalogue/` website. Lettertypes zijn op verschillende manieren ingedeeld. Op de overzichtspagina's krijgt elk lettertype een regeltje voorbeeld-tekst. Elk lettertype heeft ook een pagina met uitgebreidere voorbeelden, en met de benodigde preamble regels. Er staat ook vermeld of het lettertype onderdeel is van TeX Live. Zie figuur 2.

En verder. . .

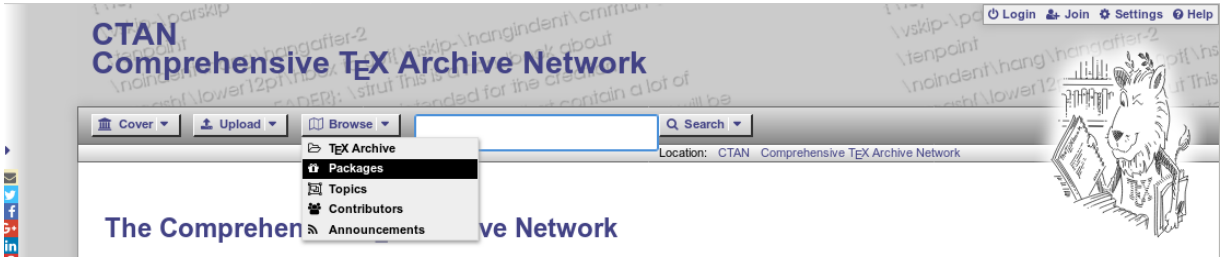
Dit is maar een beknopte selectie van TEX-gerelateerde informatie. Een veel uitgebreidere opsomming is te vinden op de TUG-pagina *TeX Resources on the Web* (`www.tug.org/interest.html`). Hierin staat ook een lijst van commercieel uitgegeven boeken.

En natuurlijk zijn er zoekmachines voor als je er toch niet uitkomt.

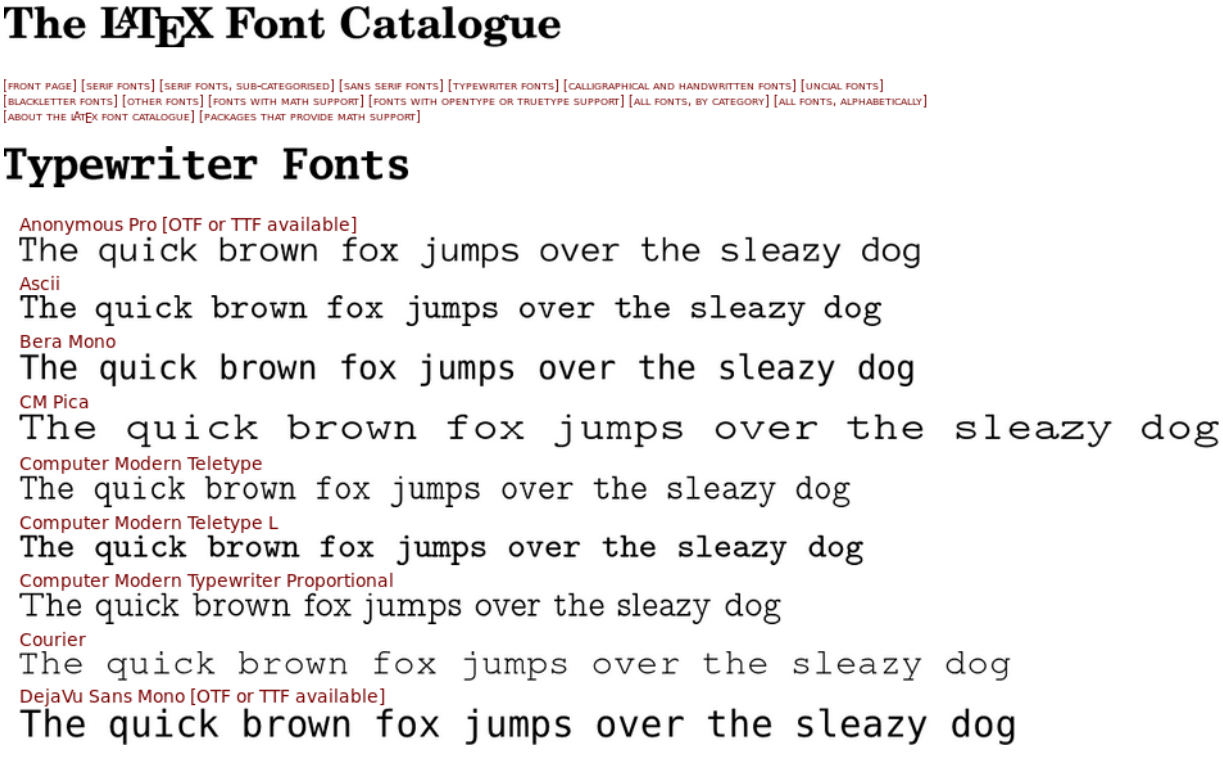
Notes

1. Bij TeX Live is dit een installatie-optie.
2. Onder MiKTeX doet het `mthelp` commando hetzelfde.

Siep Kroonenberg



Figuur 1. De CTAN catalogus: `https://www.ctan.org/`



Figuur 2. De LaTeX font catalogus: `http://www.tug.dk/FontCatalogue/`

Veel pagina's scannen, één pdf

Abstract
Voor een koor of een orkest is het soms nodig uit een muziekboek een stuk te kopiëren, met het resultaat een aantal gescande afbeeldingen – meestal van het type jpeg of pdf. Hieronder beschrijf ik een methode om met een paar LaTeX-macro's de kantlijnen van alle pagina's recht en symmetrisch te krijgen, de scans op de gehele pagina weer te geven en het resultaat als één pdf bruikbaar te maken voor afdruk. Bijvoorbeeld de trapeziumvorm van een scan corrigeren, dat gaat alleen met meer specifieke software zoals DigiKam.
Gebruik van een editor met kolom-editmogelijkheden kan handig zijn.

Inleiding

De hier beschreven methode is bedoeld om met behulp van LaTeX meerdere gescande pagina's tot één A4-document samen te stellen.

Iedere scan apart bewerken met een programma als Digikam of Gimp is een alternatief, zeker als het nodig is om perspectivisch te vervormen, de pixeldichtheid aan te passen, de jpeg-kwaliteit te variëren, of vlekken in een scan wegpoetsen.

De benadering waarvoor ik hier gekozen heb is over algemeen voldoende, werkt louter met LaTeX, meer heb je niet nodig.

We gaan uit van een aantal scans van bladmuziek

Die scans worden, als het goed is, zo netjes mogelijk gemaakt, maar vaak zitten er zwarte repen of vlekken aan de rand van zo'n scan omdat het origineel niet in het midden en/of niet precies recht heeft gelegen of je wilt dat het resultaat vergroot, pagina-vullend wordt afgebeeld.

De hier gebruikte werkwijze is niet zo bijzonder maar wel effectief: lelijke randen kunnen worden weggehaald, de scan kan uitgespreid worden over de hele pagina en zo nodig (om)gedraaid worden. Ook een voordeel is dat je meerdere pagina's in hetzelfde document tegelijk kunt aanpassen en dat na compileren het eindresultaat van al die wijzigingen meteen zichtbaar is in het einddocument.

De opzet van het systeem, het werkbestand

De basis van het systeem is het LaTeX-werkdocument, waarvan hieronder een voorbeeld wordt weergegeven.

Let op de zeven à acht parameters, met name `\Raster`, `\Witrand` en `\Bron` – waarmee de omschakeling met `%` van de belangrijkste features beheerd wordt – en verder: `\Naam`, `\Title`, `\Subtitle`, `\Author`, de info voor titels op een eventuele voorpagina.

Bij het werkdocument hoort, behalve `scancorr.cls`, een hulpbestand in Postscript dat wordt gebruikt om een raster af te beelden over alle pagina's zodat het gemakkelijker is de afstanden *te schatten* waarmee de randen van scans moeten worden getrimd.

```
%!lualatex
\documentclass{scancorr}
%\pageav % optioneel de output op A5 zetten
\begin{document}
    \Bron{.}                    % bronmap bijv. {/c/EPSON_scan}
    \Naam{Ernst van der Storm}
   \Title{Ah! mio cor}
\Subtitle{Giorgio Federico Haendel}
  \Author{Aria}
    \Info                       % meta info for pdf
  \Raster                       % hulpraster
 \Witrand                       % witte rand

%\TitlePage                     % frontpagina
%\EmptyPage                     % lege pagina

% #1: bron waaruit pagina gehaald wordt
% #2 #3 #4 en #5 zijn trimwaardes in mm
% #6: draaihoek in graden (+=antiklok)
% #7: paginanummer in bron
%             #1           #2   #3   #4   #5    #6   #7
%            bron          lft  rgt  top  bot   ang  pag
\jpgpdftrim{example0.pdf} {000}{000}{000}{000}{0.00}{001}
\jpgpdftrim{example0.pdf} {000}{000}{000}{000}{0.00}{002}
\jpgpdftrim{example0.pdf} {000}{000}{000}{000}{0.00}{003}
\jpgpdftrim{example0.pdf} {000}{000}{000}{000}{0.00}{004}
\jpgpdftrim{example0.pdf} {000}{000}{000}{000}{0.00}{005}
\end{document}
```

Figuur 1. Broncode van een werkbestand

Ook kan men later met een witte rand van 1 cm breed eventuele *ongerechtigheden* van het scannen *aan de rand van de pagina's* maskeren.

In het weergegeven werkbestand staan het raster en de witte rand beide aan, men kan ze deactiveren door het %-teken ervoor te zetten. In het voorbeeld is na `%\TitlePage` een optie voor een (nagenoeg) lege pagina `\EmptyPage` toegevoegd en je kunt dat commando naar behoefte gebruiken, bijvoorbeeld om ervoor te zorgen dat de oneven pagina's rechts komen.

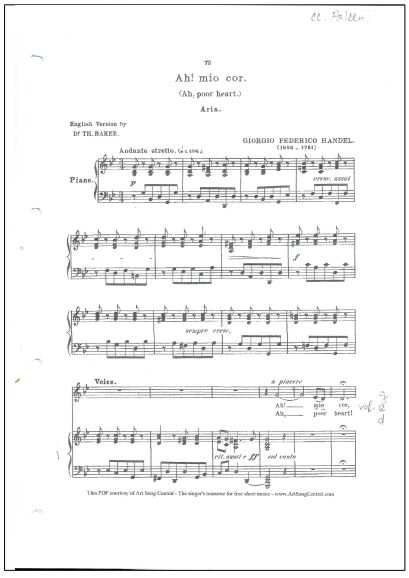
De voorpagina wordt gegenereerd uit de teksten die zijn opgegeven in `\Title`, `\Subtitle` en `\Author`.

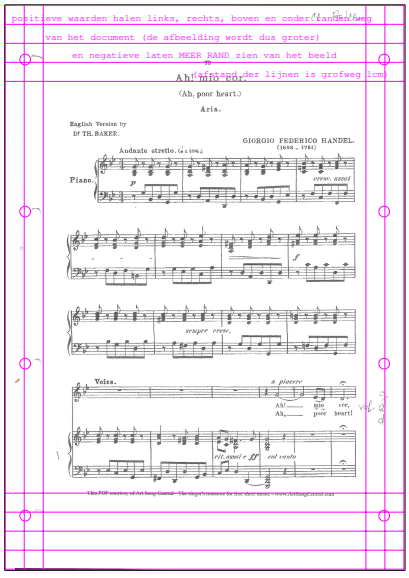
De voorbereiding

Ik beschrijf hier de basisstappen van de werkwijze met pdf's. Werken met andere afbeeldingsbestanden gaat net zo, alleen is het geven van een paginanummer bij jpg's of png's in parameter#7 niet nodig, maar soms misschien handig als geheugensteun bij een groot aantal pagina's. Het is ook mogelijk van willekeurige pagina's uit een ander pdf-bestand een 'nieuw origineel' te maken, dat werkt in principe op dezelfde manier.

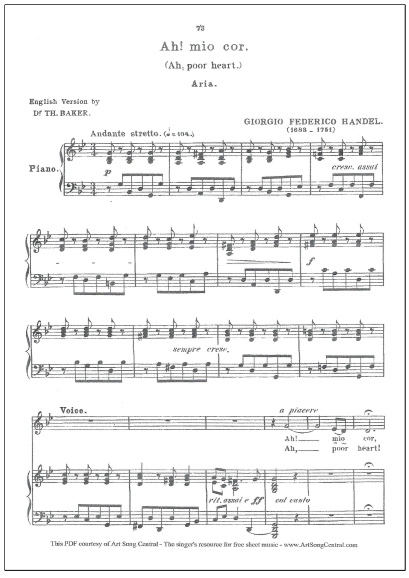
De voorbereiding voor gebruik van de software is als volgt:

- sla de tekst van `scancorr.cls` en van het hulpbestand `hulprastercm.eps` op in de map waar je werkbestanden staan (later kunnen die, als alles werkt, beter naar de gebruikelijke plaatsen verhuisd worden)
- converteer het hulp-eps-bestand bij voorkeur meteen naar `hulprastercm.pdf` bijv. met `epstopdf`
- verzamel alle gescande paginabestanden op één plaats: in de broncode staat het pad er naartoe in de definitie van `\Bron`
- verander indien nodig in het werkbestand de definitie van `\Bron{.}`, de map dus waar de scans in staan


origineel



met raster


na correctie

De *steeds opnieuw genereren*-fase

De productie is een steeds opnieuw compileren van het onderhanden bestand, want de getallen die voor het 'trimmen' ingevuld moeten worden kunnen alleen goed worden na het opnieuw beoordelen van het resultaat, feitelijk dus met *trial and error*.

N.B. het raster is vooral bedoeld als richtlijn voor de marges van het resultaat, je kunt er niet echt aan afmeten *hoeveel* je moet corrigeren

Het best is te beginnen met het hulpraster `\Raster` actief en de witte rand van 1 cm geblokkeerd: `%\Witrand`.

Ter illustratie is het werkbestand ingevuld met een voorbeeldbestand – de scans – example0.pdf.

- bepaal hoeveel pagina's het geheel moet krijgen (evenveel als er zijn gescand, maar een originele pdf kan – zoals hier – ook meerdere bronpagina's bevatten)
- voer de benodigde regels in, voor iedere pagina van het te genereren stuk één regel, zoals:

```
\jpgpdftrim{example0.pdf} {024}{017}{020}{029}{0.00}{001}
\jpgpdftrim{example0.pdf} {025}{019}{020}{028}{-0.3}{002}
\jpgpdftrim{example0.pdf} {026}{020}{020}{029}{0.50}{003}
\jpgpdftrim{example0.pdf} {027}{019}{019}{028}{0.12}{004}
\jpgpdftrim{example0.pdf} {024}{019}{019}{028}{0.00}{005}
```

  Daarin zijn de argumenten:

  1. de naam van de onderhavige bronpagina
  2. linker trimwaarde
  3. rechter trimwaarde
  4. top-trimwaarde
  5. onder-trimwaarde
  6. hoek (in graden)
  7. paginanummer (nodig bij pdf's).

- de vier getallen van de trimwaardes moeten zo goed als zeker aangepast worden want de eerste vier getallen zijn allemaal nul (of zijn blijven staan van 'de vorige keer')
- negatieve waardes zorgen voor *extra* witmarge, bijvoorbeeld als de originele pagina *te dicht langs de rand* is gescand
- hieraan de nodige tijd besteden, steeds opnieuw compileren…

Pagina's kantelen of omdraaien

Parameter vijf, die de pagina kantelt, is meestal 0.00 (niet kantelen) en hoeft, als het om kleine correcties gaat, doorgaans pas in een later stadium aangepast te worden; bijvoorbeeld 1.0° is al veel. Een positieve hoek is kantelen naar links, een negatief getal kantelt naar rechts.

Het is hier ook mogelijk om een pagina *op z'n kop* te zetten door 180 in te vullen, of eventueel een kwartslag, 90° of 270°. Dat moet in de voorkomende gevallen wél eerst ingesteld worden want het omdraaien heeft gevolgen voor de randen die getrimd worden. De volgorde van de trimparameters blijft steeds links, rechts, boven, onder.

Extra voorpagina genereren en de afwerking

Om de paginanummering van het gescande origineel correct te houden – even pagina's links, oneven rechts – kun je met `\TitlePage` eenvoudig een voorpagina toevoegen. Nodig is dan wel om de bij de titelpagina behorende informatie in te voeren: `\Title`, `\Subtitle` en `\Author`. Met `\EmptyPage` kan zo nodig een lege pagina worden toegevoegd.

Aan het eind moet het raster weggehaald worden door die regel met `%\Raster` uit te zetten. Zo nodig kun je de witte rand activeren met `\Witrand` om ongerechtigheden aan de paginarand te maskeren.

Uiteraard kunnen er tússen het gescande materiaal ook vlekken of aantekeningen zitten die je niet in het eindresultaat wil zien. Om dat goed te krijgen moet je de betreffende scan vooraf met een ander programma opschonen, dat is met deze software niet mogelijk. Liedteksten die slecht leesbaar zijn verbeteren, idem.

Als gescand is met een voldoend hoge resolutie levert het vergroten door *includegraphics* – ook al wordt een blad gekanteld – toch een goed resultaat op.

Voor een (kopie van) een partituur is het logisch om de paginagrootte op A4 te houden, maar als de bronbestand afkomstig is van bijvoorbeeld een klein boekje dat oorspronkelijk als pagina-afmeting 20x21 cm2 heeft, kan het handig zijn het resultaat kleiner te printen. Daarvoor is in het werkbestand optie uit het package *memoir* in de class file `\pageav` te activeren; daarmee komt alle output op A5.

Broncode van de *scancorr* class-file

```
\NeedsTeXFormat{LaTeX2e}[1999/12/01]
\ProvidesClass{scancorr}
    [2018/12/06 v1.03 manually trim scans]
\LoadClass[a4paper,fleqn]{memoir}

\RequirePackage{graphicx}
\RequirePackage{xcolor}
\RequirePackage[a4paper,margin=0mm]{geometry}
\RequirePackage{polyglossia}
\setdefaultlanguage{dutch}
\RequirePackage{fontspec}
\RequirePackage{alltt}
\def\Wi{\paperwidth}
\def\Hi{\paperheight}
\usepackage[absolute,overlay]{textpos}\TPGrid{\Wi}{\Hi}
\pagestyle{empty}
```

```
\parindent0pt

\newcommand{\Rasterpagina}{\par%
\begin{textblock*}{\Wi}(0pt,0pt)%
\begin{minipage}[c][\Hi][b]{\Wi}%
\includegraphics[width=1\Wi,totalheight=1\Hi]{hulprastercm}%
\end{minipage}%
\end{textblock*}}
\newcommand{\raster}{\ifx\RA\undefined\else\Rasterpagina\fi}

% \begin{textblock*}{hsize}[ho,ve](hpos,vpos) %\rule{breedte}{hoogte}
\newcommand{\Randpagina}{\par%
  \begin{textblock*}{\Wi}(0pt,0pt)\color{white}\rule{\Wi}{.033\Hi}\end{textblock*}
  \begin{textblock*}{\Wi}(0pt,0pt)\color{white}\rule{.047\Wi}{\Hi}\end{textblock*}
  \begin{textblock*}{\Wi}(.951\Wi,0pt)\color{white}\rule{.049\Wi}{\Hi}\end{textblock*}
  \begin{textblock*}{\Wi}(0pt,.9657\Hi)\color{white}\rule{\Wi}{.0343\Hi}\end{textblock*}
}
\newcommand{\wirand}{\ifx\WR\undefined\else\Randpagina\fi}

%                bb= llx  lly  urx ury
\newlength{\wisselLRTB}
\newcommand{\jpgpdftrim}[7]{
\enlargethispage*{100pt}%
\wisselLRTB=#6pt % 180.0pt komt hierin of 0.1pt enz.
\message{==trim=#2mm #3mm #4mm #5mm,angle=#6,page=#7}
\ifdim\wisselLRTB>260pt% kantel 3/4%
 \includegraphics[trim=#4mm #2mm #5mm #3mm,angle=#6,page=#7,width=\paperwidth,totalheight=\paperheight,clip]{\bronMAP/#1}
\else
 \ifdim\wisselLRTB>175pt% kantel op zijn kop
  \includegraphics[trim=#3mm #4mm #2mm #5mm,angle=#6,page=#7,width=\paperwidth,totalheight=\paperheight,clip]{\bronMAP/#1}
 \else
  \ifdim\wisselLRTB>85pt% kantel een kwart
  \includegraphics[trim=#5mm #3mm #4mm #2mm,angle=#6,page=#7,width=\paperwidth,totalheight=\paperheight,clip]{\bronMAP/#1}
  \else%
  \includegraphics[trim=#2mm #5mm #3mm #4mm,angle=#6,page=#7,width=\paperwidth,totalheight=\paperheight,clip]{\bronMAP/#1}
  \fi
 \fi
\fi
\raster\wirand\newpage}

% titelpagina genereren:
\def\TitlePage{\maketitle\newpage}
% lege pagina toevoegen:
\def\EmptyPage{%
  \mbox{}
  \vskip30mm%
  \centerline{\jobname}%
  \centerline{\tiny Deze pagina is met opzet leeg gelaten}%
  \newpage}

\def\Bron#1{\def\bronMAP{#1}}
\def\Naam#1{\def\eigennaam{#1}}
\def\Title#1{\def\Titel{#1}\title{#1}}
\def\Subtitle#1{\def\stitle{#1}}\def\stitle{\relax}
\def\Author#1{\def\Auteur{#1}\author{#1}}
\def\Raster{\def\RA{}}
\def\Witrand{\def\WR{}}
\def\Info{
  \pdfextension info{
    /Title   (\jobname)
    /Creator (LaTeX)
    /Producer(\eigennaam)
    /Author  (\Auteur)
    /Subject (\Titel, \stitle)
  }
}

\pretitle{\vskip30mm\begin{center}\fontsize{43}{43}\selectfont}
\posttitle{\vskip10mm\fontsize{21}{21}\selectfont\mbox{}\\\stitle\end{center}}
\preauthor{\vskip30mm\begin{center}\fontsize{25}{25}\selectfont}
\postauthor{\end{center}}
\date{}
\endinput
```

Broncode hulpbestandje hulprastercm.eps

```
%%!PS-Adobe-3.0 %% hulprastercm.eps
%%BoundingBox: 0 0 596 842
1 0 1 setrgbcolor %% lichtpaars
/hg 842 def %% hoogte=lengte verticale lijn
/br 595 def %% lengte horizontale lijn
/m { 28.35 } def %% maat van het vakje, komt overeen met 1 cm
/vl { hg dup 0 exch rlineto neg 0 exch rmoveto } bind def %% verticaal
/hl { br dup 0 rlineto neg 0 rmoveto } bind def           %% lijn horiz.
/shr { m 0 rmoveto } bind def     %% stap naar rechts
/shl { m neg 0 rmoveto } bind def %% stap naar links
/svu { 0 m rmoveto } bind def /svd { 0 m neg rmoveto } bind def
/cirkel { 0 0 8 0 360 arc stroke } def
/raster  { gsave
%% eerst verticale: 2 links en 2 rechts
0 1 moveto 0 1 2 { vl shr } for
br 1 moveto 0 1 2 { vl shl } for
%% dan horiz. 4 onder en 4 boven
0 1 moveto 0 1 4 { hl svu } for
0 hg moveto 0 1 4 { hl svd } for
0.002 setlinewidth stroke %% dunne lijnen
grestore } bind def
raster %% op 0 1
gsave m  1.05 mul  2.85 m mul translate cirkel grestore
gsave m  1.05 mul 10.85 m mul translate cirkel grestore
gsave m  1.05 mul 18.85 m mul translate cirkel grestore
gsave m  1.05 mul 26.85 m mul translate cirkel grestore
gsave m 19.95 mul  2.85 m mul translate cirkel grestore
gsave m 19.95 mul 10.85 m mul translate cirkel grestore
gsave m 19.95 mul 18.85 m mul translate cirkel grestore
gsave m 19.95 mul 26.85 m mul translate cirkel grestore
10 819 moveto %% infotekst plaatsen:
/Courier findfont 25 scalefont setfont
%%(afstand der lijnen is grofweg 1 cm) show
%%10 790 moveto %% infotekst plaatsen:
(positieve waarden halen links,) show
10 791 moveto %% infotekst plaatsen:
(rechts, onder en boven randen weg van) show
10 763 moveto %% infotekst plaatsen:
(het document (dus beeld wordt groter)) show
gsave
573 777 moveto %% infotekst plaatsen:
270 rotate
(& negatieve - laten MEER rand zien van het beeld) show
grestore
11 749 moveto %% infotekst plaatsen:
270 rotate
/Courier findfont 15 scalefont setfont
(de buitenste strook wordt rondom met wit afgedekt met \\def\\WR{ja}) show
%%90 rotate
%%10 734  moveto %% infotekst plaatsen:
%%(van het origineel) show
showpage %%EOF
```

Ernst van der Storm
evdstorm@xs4all.nl

Writing my thesis with TEX

A report from the battlefields

Sharing your TEX-setup feels a little like airing your dirty laundry in public. The projects I know are often a mess of haphazardly cobbled up solutions and last-minute changes, duct-taped together with a bunch of macros copy-pasted from random pages on the internet.

Nevertheless (or possibly precisely because of this), sharing these setups and project structures can help other — especially more novice — users to streamline their workflow and lower the threshold to customize their own system. At least, write-ups like these have helped me.[1]

I will follow with a general overview of the setup that I used to write my thesis last year, with an audience in mind that would include myself before I started this thesis. It did get me that sought-after diploma, so you might try your risk with something similar.

First warning: always remember that content goes above style. Don't let the pleasure derived from your TEX-journeys obscure the thing that matters: that which you are trying to convey to the reader. Not even the best design or most comfortable workflow can mend a broken text — or a lack of text, for that matter.

Second warning: I'm not a very capable TEX-user by any means. If anything in the following seems old-fashioned, clumsy or plain wrong: you're probably right. Blame it on a routine built up from anonymous internet advice coming from several different decades and from questionable sources. (You could also take it as the indication of the TEX-proficiency level of an average user — which suggests that better documentation of currently available options is more useful than adding even more complex features to them.)

Background

The thesis I wrote was the conclusion of my bachelor in mathematics, which I followed at the University of Amsterdam (UvA). In the second year of the program, the course 'Computeralgebra / LATEX' is taught as an introduction to typesetting mathematics with LATEX, combined with some programming in Mathematica. Students are then expected to be sufficiently comfortable in LATEX to use it for research projects, an occasional homework assignment, and their thesis (this expectation is met to a certain degree). Also, presentations are expected to be prepared with `beamer`.

For the thesis, we are supplied with a class file with some basic scaffolding: a title page with institute names, supervisors and the logo of the university, and a place to put our abstract.[2] The rest (including design choices) is left to the students.

My thesis was in the area of mathematical logic, specifically set theory. This requires some extra funny symbols, but my setup should work for any scientific thesis or larger project. For an idea of the size: in the end, my thesis was 42 pages long, included 6 chapters and some 10.700 words (according to `texcount`).

Editor

Let's start with the main interface — the editor. I'm using Sublime Text 3 on a MacBook Pro. In the course mentioned earlier, Mac users were advised to use TeXShop since it came bundled with the MacTeX-distribution. I used it for a while, but never liked its design and default settings (it uses a proportional width font for code, and no syntax coloring). It's possible to tweak this to a certain extent, but I already used Sublime Text for other programming tasks, so it was more natural to switch to that.

The Sublime Text-plugin `LaTeXTools`[3] offers various necessary functionalities. Most importantly, it adds a configurable build pipeline, so you can compile while staying within Sublime Text. It also installs syntax highlighting, autofill-options for a.o. citations and references, and command completion. With this enabled, editing becomes very intuitive and swift.

A PDF-previewer with auto-reload (so that the preview reloads every time the PDF is compiled) makes the 'write-compile-read'-cycle even smoother. Unfortunately, Preview.app (the default PDF-viewer on Mac) doesn't offer this functionality. I use the open source alternative Skim instead.[4]

The choice of your editor is usually a very personal one, and discussions about it usually get very heated. I do not claim that Sublime Text is the most suitable editor for everyone — far from it. But in general, picking a good editor and getting comfortable with it and setting it up

```
/project
├── img
│   ├── figure.tikz
│   ├── logo.png
│   └── ...
├── tex
│   ├── 1-introduction.tex
│   ├── 1.1-state-of-the-art.tex
│   ├── 1.2-research-goal.tex
│   ├── 2-experiment-design.tex
│   ├── 2.1-lab-settings.tex
│   └── ...
├── .gitignore
├── project.bib
├── project.tex
├── project.sty
└── style.sty
```

Figure 1. An outline of the directory structure. For the function of `.gitignore`, see section 'Version control'.

properly (learning the most useful shortcuts, installing the right packages, configuring proper file exclusion patterns so your workspace doesn't get cluttered) makes the whole process a lot smoother.

Project organization

At a certain point, keeping all your text in one file becomes unworkable. I break everything up in smaller files quite aggressively — usually, every section gets its own file. Then there is another file per chapter, which collects all its sections using `\input`. These are all organized in a directory called `tex`. Finally, a main file is kept in the root directory. Using `\includes`, it collects all the chapters in `/tex`. This file can then be compiled into a PDF.

Since I want to reuse design and package decisions in later projects, I separated them into two `.sty`-files, that are imported in the main `.tex`-file by `\usepackage`. One of them (`style.sty`) contains general customizations; the other (`project.sty`) contains project-specific choices. See fig. 1 for how this all comes together.

Version control

Although it seems this is a rare opinion among other students (who often use the free version of online LaTeX collaboration tools such as Overleaf — which miss any history tools), to me version control is indispensable. I use Git to save the complete history of my project. Using 'commits', I can add another state of my files in the history tree. This allows me to edit my texts quite ruthlessly, since I know that if I ever change my mind about a part or two, I can always revert it to a previous state, or pick out specific sentences (in practice, I rarely ever actually need this — knowing that the option is there is enough, apparently).

Git offers 'tags', with which you can annotate commits. I use it to mark the commits from which I compiled particular versions — such as the ones I discussed with my supervisor. It's also possible to use 'branches' — particular diversions from the main file history, which I used a couple of times to try out certain new ideas or formats, without messing up the main history. To give an idea: I used 73 commits, 8 tags and 2 branches (besides the 'master' branch).

If you use Git, make sure to properly configure your `.gitignore`-file, which has file and folder exclusion patterns that Git uses (so you don't save all intermediate `.aux` and `.log` files in your history). There is a TeX-specific list on Github that is a good starting point.[5]

Speaking of Github: it is *the* place to synchronize your Git-repository with. If you are working on a project by yourself, it also functions as an external backup (although be sure that you have an extra copy or two saved on some external disks). And if you are collaborating on a project, Github can function as a central server to host your repository. All the participants then have a local copy of the repository, in which they can edit. Afterwards, they send their edits (new commits) back to Github for others to see.

Bibliography

Biblatex seemed the most modern of all options for managing citations, so that was my choice. It comes closest to working 'out of the box'.

One of the most important parts in my workflow is managing the references. You could edit the `.bib`-file directly, but that is quite error-prone and time-consuming. A solution which worked very well for me, is to use the open source Zotero.[6] It is a capable reference manager, and together with the plugin 'Better Bibtex'[7] you can easily exportcitations into a `.bib`-file and keep it updated automatically. If you use the browser plugin 'Zotero Connector' on top of that, you get a workflow that is so seamless it feels slightly magical (see fig. 2):

1. find a reference online
2. import it to Zotero with one click on the Zotero Connector (this automatically saves the PDF of the article, and fills in metadata — usually quite accurately)
3. Zotero automatically updates the `.bib`-file
4. Sublime Text recognizes the new reference, and it is immediately available in the autofill suggestions

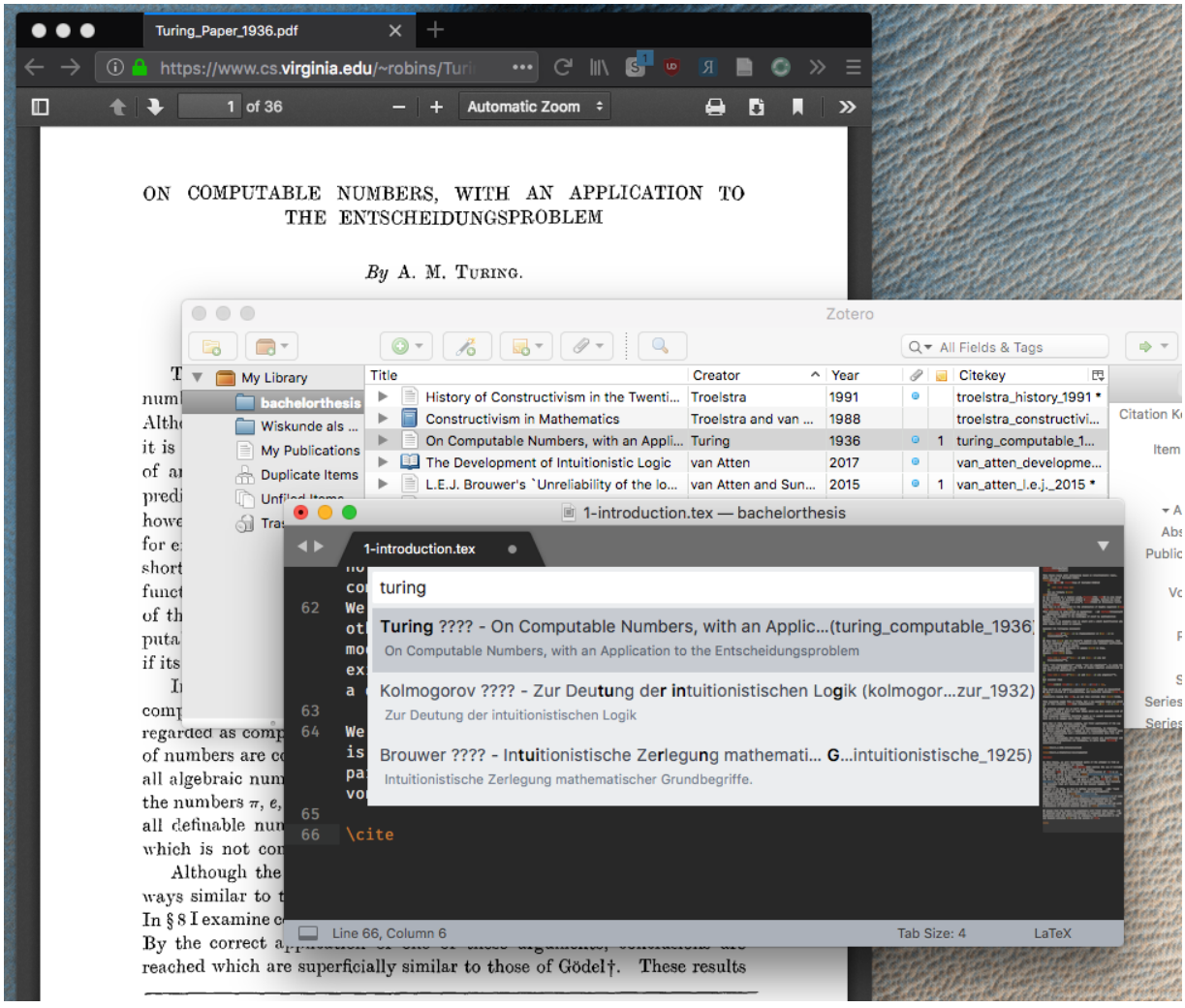


Figure 2. The Zotero workflow: open article in browser; add to Zotero with Zotero Connector (the 'page'-icons in the top right of the browser); Sublime suggest references as soon as you type `\cite`. (Note that there is a bug where Sublime doesn't properly extract the year of the references from the `.bib`-file.)

Design

After staring at mathematics articles in the default LaTeX-layout for so long, I got a bit bored with the default design and tried my hand at customizing it. The main part of this was choosing different fonts. My eye fell on the nice Spectral as a body font, and Fira Sans for titles and captions.[8] To easily use these fonts and customize settings, I switched to the XeTeX-engine. It is then as simple as

```
\setmainfont{Spectral Light}
```

to switch to the Light variant of Spectral as the body font (provided you have the fonts installed on your computer).

For mathematics, the `amsthm`-theorem environments are indispensable. To give them a more unique look, I gave them a line on the left side. This also separates them more clearly from the running text, leading to an interesting vertical typographical dynamic. I used the package `mdframed` to achieve this. This package is usually used to draw boxes around theorems, but you can also use it to draw a line on just one side. Once you have set this package up with the correct line widths and spacings, it is as simple as replacing `\newtheorem` with `\newmdtheoremenv`.

Since the margins of a document printed on A4 are quite wide, I decided to pull the numbering of theorems and titles into the side margins. This can be achieved with a quite clever hack with 'negative phantoms'. Usu-

4 The McCarty realizability model for IZF

After defining IZF in Section 2.2, we had an important open question: is IZF truly intuitionistic? (see Question 2.7) In this chapter, we will prove that it indeed is: the Law of Excluded Middle is *not* a consequence of the axioms of IZF. In order to do this, we construct a model that satisfies all the axioms, but not the Law of Excluded Middle. In particular, it satisfies Church's Thesis

4.1 The realizability structure

We start by describing our realizability structure, which we define in stages and resembles the usual set-theoretic von Neumann universe V (see Figure 4.1 for a visual interpretation):

DEF 4.1 The *universe of realizability sets*, denoted with $V(Kl)$ (where 'Kl' stands for 'Kleene'), is defined by ordinal induction as follows:[1]

$$\begin{aligned} V(Kl)_0 &= \varnothing, \\ V(Kl)_{\alpha+1} &= \mathcal{P}(\omega \times V(Kl)_\alpha), && \text{(for any ordinal } \alpha) \\ V(Kl)_\lambda &= \bigcup_{\alpha<\lambda} V(Kl)_\alpha, && \text{(for a limit ordinal } \lambda) \\ V(Kl) &= \bigcup_{\alpha\in On} V(Kl)_\alpha. \end{aligned}$$

If we look at the definition, we see that a realizability set at stage $V(Kl)_{\alpha+1}$ consists of pairs $\langle n, x\rangle$, where n is a natural number and x is an element of $V(Kl)_\alpha$. McCarty based the idea of a universe with this structure on remarks made by Poincaré (1963) on the constructive notion of

Figure 3. An example of the design: note the custom fonts, the hanging section and definition numbers, and the line besides the definition (some content faded out for clarity).

ally, `\hphantom{text}` is used to insert a space exactly the width of 'text' if it would have been rendered at that place. This can be used to fine-tune the position of mathematical formulae, which helps in aligning them in more complex cases (where `align`-environments are not enough). This trick can be extended to use a 'negative phantom', which places a negative space at that spot. A common definition (don't ask me where I got it — it was floating somewhere on the internet) is the following:

```
\newcommand{\nhphantom}[1]%
  {\sbox0{#1}\hspace{-\the\wd0}}
```

If you then include this in the definition of your `theoremstyle`, you can pull the numbering of theorem-environments as far into the margins as you want — that is, precisely the length of the numbering itself.

A similar procedure works for chapter and section titles, with the aid of the package `titlesec`. I also used the extra options offered by this package to change the default spacing of titles.

For an example of how this all comes together, see fig. 3.

General remarks

Some more general wisdom that appeared to me during this process (which might be obvious to some, but very helpful to others[9])

- Use `\label`'s and `cleveref` for automatic referencing of elements like theorems, chapters and figures. It is as easy placing a label at e.g. a new chapter:

  ```
  \chapter{The best chapter}
  \label{ch:best-chapter}
  ```

  and then elsewhere in the text use this as:

  ```
  Earlier in \cref{ch:best-chapter},
    we discussed [...]
  ```

  Then, `cleveref` (the package which supplies the `\cref`-command) will automatically expand this to

Earlier in chapter 1, we discussed [...]

If you also include the `hyperref`-package, it also changes 'chapter 1' into a link in the PDF that brings you to the right page.

Note that I add the `ch:`-prefix to all chapter-labels. I do something similar for theorems (`th:`), sections (`sc:`), and so on. It helps to remember what you actually referenced, when you find all your `\cref`'s scattered throughout the code.

- Make commands for every special symbol. I was working with set theory and logic, and had to use a lot of special notation. Because I made commands for every new symbol (e.g. `\newcommand{\proves}{\vdash}` for a specific proof-symbol), I could apply the symbols consistently in my text. And if I changed my mind about which symbol to use for which purpose, I only had to update one definition to change all occurrences.
- Add plenty of comments, at least in your `.sty`-files. Some definitions and commands can be quite obscure, and comments help to later remember why you added the code in the first place. For the same reason, when I find a solution for some problem online, I often include the link to the forum post or webpage where I found it.
- When writing, I often leave **todo** notes in the text to remind myself which parts I have to fill in later on. I use the aptly named package `todonotes` for this.
- If possible, consider a dual-screen monitor setup. This allows you to have a window with your editor on one screen, and a preview of the compiled PDF in the other.
- For any new text: write a draft on paper before you dive into your editor. Even though LaTeX-syntax can seem relatively easy once you are used to it, it still slows down your creative process while writing. If I am not careful, I will waste time tweaking small typographic details, instead of using that time to write new text.

Notes

1. Specifically, I picked up advice from a couple of interesting discussions on workflows on the TEX Stackexchange (besides the huge number of solutions to more specific problems that I found there — it's a fantastic resource). For examples, check out the questions tagged 'workflow': `tex.stackexchange.com/questions/tagged/workflow`.
2. The class is available at `github.com/UvA-FNWI/uvamath`.
3. `github.com/SublimeText/LaTeXTools`
4. `skim-app.sourceforge.io`
5. Check the `gitignore`-repository at `github.com/github/gitignore`. You can also use this as an exclude list for your editor.
6. `zotero.org`
7. `github.com/retorquere/zotero-better-bibtex`
8. Both fonts are open source and available on Github (`github.com/productiontype/spectral` and `github.com/mozilla/Fira`)
9. relevant xkcd: `xkcd.com/1053`

Rens Baardman

Following up on LuaTEX

Introduction

From the ConTEXt mailing list one can get an impression of how ConTEXt is used. It's mostly for typesetting books, manuals and sometimes special products and the input is either TEX or xml. I have no clue about articles, but in that domain MS Word, LATEX and maybe even Google Docs compete each other. As LATEX is encouraged (or maybe even enforced) on students for writing reports and such there is also not some body of default styles that are part of the package and probably hardly any user expects pre-cooked solutions. Anyway, I'm always surprised by the diversity of problems that users try to solve. From posts you can also deduce that using Lua becomes more popular as a means to deal with data from various sources. The only measure that we have to what extent the functionality provided is used is the quality of answers on mailing lists and support platforms and they are often of high quality.

It is interesting that outsiders sometimes think that ConTEXt development is driven by commercial usage but this mostly untrue. We started serious development in the 90's because we wanted the materials that we produced to look sort of nice and TEX could be used for that. Over a few decades the package evolved, but to a large extent that was driven by users as well as by exploration of specific user demands. The fact that we could use some features ourselves was nice, but the investment in time is in no way compensated by the revenues. It's the joy of playing with TEX, MetaPost and Lua as well as the interaction with the ConTEXt community that drives development.

Sometimes suggestions are made that one should use a 'proper' computer language instead of TEX lingua, but personally I would not use such an approach as it is rather painful. There is nothing wrong with coding a document in some specific document structure or content related language instead of clumsy function calls. Take alone the concept of grouping properties. It is therefore also not correct to conclude that ConTEXt development eventually will end up with a pure (e.g.) Lua interface to solve typesetting related problems. Hardly anyone asked us to provide solutions using TEX: if we use ConTEXt it's because we know how to use it efficiently. Using TEX coding is a convenient side effect no one really cares about as long as something works.

It must be noted that I, and probably also other users, see LuaTEX as a convenient Lua engine and especially in combination with the load of libraries that comes with ConTEXt, it can do a lot. An extreme example is that I run some domotica applications with LuaTEX and Lua scripts. At the last ConTEXt meeting Taco demonstrated how he uses LuaTEX and friends for his miniature railway tracks. Other presentations demonstrated massive number crunching and visualization of data collected over time.

It often puzzles me why we never run into customers who see(k) the potential of this TEX, MetaPost, and Lua mix. It is pretty hard to beat this combination in performance, quality, convenience and time spent to reach a solution, but it is probably too strange a mix. This lack of market also means that there is no real long term agenda needed and therefore development only relates to exploration of (often very specialized) user demand on the one hand and robust continuity on the other.

What next

Now that LuaTEX has reached a more or less stable state, the question is: what will happen next. Development has been bound rather tightly to ConTEXt development, but now that other macro packages also (can) use LuaTEX, we're sort of stuck and need to freeze functionality in order to guarantee long term stability within the TEX ecosystem. However, as the introduction suggests, we're in no way bound to this, simply because no one pays for development. We can move on as we like. Apart from changes in the way fonts and encodings are dealt with (more modern techniques) ConTEXt is pretty stable in most areas. The user interface is downward compatible so for users progress only has benefits.

There is a frontend that stays rather close to original TEX and MetaPost, there is a backend that produces efficient pdf and provides a high level of control and there are plenty of hooks to make the machinery do as you wish, for instance using Lua. Therefore, while with version 1.0 we got out of the beta stage (already

for years LuaTEX could be used in production but the interfaces evolved over time), with version 1.1 we're more or less frozen in functionality for the benefit of general usage.

But that doesn't stop ConTEXt development. We still want to improve some of the more tricky parts of typesetting (the TEX part), we like to add more graphic capabilities (the MetaPost part) and we will provide more interfaces (the Lua part). In doing so, a follow up on LuaTEX will happen, but in such a way that the version 1.x range will not be impacted much. Of course we might occasionally feed back improvements from the follow up to the ancestor. This further development is again closely related to ConTEXt, and just as in the beginning of LuaTEX development, we will not bother with usage elsewhere: we need the freedom to experiment and the ConTEXt ecosystem and user attitudes provide a healthy environment for that.

An impression

During LuaTEX development some ideas have been postponed but it was around the 2018 ConTEXt meeting that I started thinking about a follow up and actually did some explorative coding in the ConTEXt code base. At that meeting this follow up was mentioned and since there were no objections we can say that at that time the next stage was entered.

In the last few months of 2018 and January 2019 the first versions of what we will tag as LuaMetaTEX (which identifies itself as LuaTEX 2.0 internally) became production ready. At that time the ConTEXt code could also emulate the new behavior on regular LuaTEX, and when tested by some other developers no real problems surfaced. Because the beginning of the year is also the moment that TEXLive code freeze happens the public ConTEXt code is not affected. Also, real experiments can only start when we also release the next generation engine, which will happen later this year.

Therefore, the roadmap is kind of like this. After the code freeze, some of the ConTEXt internals will change and although I expect users reporting issues, there is not much that can't be solved fast. And, ConTEXt users have always been willing to update. Then, later in 2019 the alternative engine will surface and at some point we will switch to this one as default. Binaries can be generated using the compile farm at the ConTEXt garden. Of course it will be possible for users to use stock LuaTEX, but just as when we froze MkII, we might bind some functionality to a specific engine. We will still mostly use MkIV code, but the follow up will be tagged lmtx, which stands for Lua, MetaPost, TEX and xml.

A few details

So what is this new engine and why does ConTEXt need it? And can't we do the same with LuaTEX already? I leave answering this to the reader and focus on what we are doing instead. The current LuaTEX program is constructed from a TEX kernel that itself is derived from pdfTEX and snippets of Aleph. It has the MetaPost library embedded and uses Lua as extension language. The Lua instance is accompanied by some libraries. The ConTEXt distribution is a bunch of TEX, MetaPost and Lua files as well as some resources (mostly fonts) that are a subset of what TEXLive comes with. The process of typesetting is managed by some scripts (`mtxrun` and `context`) and on MS Windows a binary stub is used to launch them. In principle, if we assume MkIV being used, the only binary needed is LuaTEX, plus those stubs.

As it had the complete MetaPost library on board, the dependencies in LuaTEX of a few years back were kind of complex, mostly due to the optional png output. But we don't need that at all. So, when we removed that option, the binary shrunk from some 10MB to about 7MB. These bytes contain the frontend and backend code and some libraries (in some cases the Lua and kpse libraries are external to the main binary). But because ConTEXt doesn't need all these libraries and because only a small part of the backend code is used, the question surfaced "What if we go lean and mean?"

So, as a first step I copied the experimental branch and started pruning. It took a couple of iterations to figure out how to do that best while still being able to produce documents, but step by step code could be removed with success. I started with stripping down the backend. For instance, given the nature of ConTEXt code, it was quite doable to generate the page stream ourselves and at some point only font embedding, image inclusion and object management was left. Actually, pdf inclusion could already be brought under ConTEXt control so only bitmaps had to be dealt with. Replacing the bitmap inclusion by Lua code made it possible to drop the png libraries (jpeg inclusion didn't use libraries). For the record: this actually opens up some possibilities for future image inclusion.

Next came font embedding, which involves subsetting. Because we support variable fonts there was already quite some work done to enable kicking out the code that deals with it possible. As starting from the existing code did not make much sense the standard was taken as reference and eventually embedding worked okay. The only drawback here is that we need a bit more font caching but that is hidden from the user. On the positive side we find ourselves with a potentially more powerful virtual font system.

With respect to the backend, that left object management and because most work was already done in ConTEXt itself, kicking out the whole backend was not that much work. In the end all backend code was indeed removed and the PDF file was generated completely by ConTEXt itself. Interesting is that the generated PDF code looks a bit nicer on the one hand and it also turned out to be a bit more efficient. One reason for this is that the backend kind of knows what the frontend does. There was still a performance hit when comparing for instance compiling the LuaTEX manual but at this time there is no such penalty anymore, simply because other ConTEXt components take advantage of this (and maybe also because in the meantime some code in the engine has been optimized).

With the backend code gone the binary already shrunk significantly. Next to go was the (not used in ConTEXt) `slunicode` library. The Lua image library went away too as did the PDF backend interface (I kept `pplib`). Then removing the font loader was on the agenda and it could be dropped easily because it wasn't used at all. Apart from creating an again smaller binary, it also removed dependencies and compilation became easier and faster. When you look at how TEX deals with fonts, it is clear that not that much is needed. For instance OpenType fonts are processed in Lua and that code uses resources not present (and needed) by TEX, and with the backend gone even less is needed. That meant that I could limit the amount of data passed to and stored in TEX for traditional font handling: TEX only needs dimensions, some math properties, and (only) in so called base mode, kerns and ligatures. This makes the memory footprint smaller and might make loading huge fonts a bit faster. When that code was simplified, it was a small step to removing the traditional TFM loading code, and because virtual fonts are only dealt with in the (no longer hard coded) backend that code could also go. Support for traditional eight bit fonts based on TFM files (normally we turn such fonts into wide fonts using information from AFM and PFB files) is still present but a Lua font loader is used instead.

By the time all this was done I decided to remove the KPSE library because in ConTEXt we use a Lua variant. A future version can bring KPSE back as loadable library (in principle one could use the `ffi` interface for this). By removing all kind of dependencies the startup code was also adapted. The engine can now function as a stub too, which means that a ConTEXt distribution becomes simpler as well. At some point we will default to MkIV only and then only one (or two if you add the LuaJIT companion) binary is needed.

I will not go into too much of the details this time (and there are many). As with the development of LuaTEX, I keep track of choices and experiments in a document and some chapters can make it into articles. In between the mentioned stripping down I decided to also clean up the directional model. In spite of what one might think, there is not that much code dealing with this. If we stick to bidirectional typesetting, the frontend doesn't really care what it sees. It was the backend that took care of reversed text streams and now we provide that backend ourselves. So, when wondering what to do with vertical typesetting, I realized that this was actually not supported in a useable fashion at all so why not let that go? And so it went away and directional support was simplified: we only have two directions now. It is worth mentioning that the concept of a body and page direction already didn't make much sense (it is the macro package that deals with this) so these are gone. Also gone are (related) page dimensions: no hard coded backend means that there is no need for them. The keyword driven direction directives with the three letter acronyms have been removed so we only have numeric ones now. It is trivial to define commands that provide the old syntax.

So how about vertical typesetting? As an experiment I've added some properties to boxes: orientation and offsets; more about that in another article. This is something that is dealt with in the backend so it was a nice experiment to see how easy that could be done. This (plus the new directional approach) is something that might be ported back to stock LuaTEX, but let's first see how it will be used in ConTEXt so that it can stabilize.

The current state

By the time we achieved all this (plus a bit of cleanup) the binary had shrunk to well below 3MB which is nice because it is also used as Lua interpreter. There are hardly any dependencies left. It was a bit cumbersome to explore how much of the code base could be stripped and how the build related scripts and setups could be simplified. One can wonder how simple we can get, given the pretty complex TEX build structure, but this is the motto: "If on some mainstream platform Lua can be compiled without hassles, then compiling LuaMetaTEX should be easy too."

This brings us to some of the main objectives of this project. First of all, the idea is to converge to a relatively small LuaMetaTEX, independent of libraries outside our control, one that is easy to compile. It provides an opened up more or less traditional core TEX but delegates much of the modern font handling to Lua. The backend can be a Lua job too. Generic font code we already provide in ConTEXt, and maybe at some point some of the code might show up as generic; we'll see. Apart from providing hooks (callbacks) the engine stays close to original TEX in concept. After all, that core also delegates the backend to extensions or external programs. We kind of go back to the basics.

I use the LuaTEX manual as well as the ConTEXt test suite for testing but for sure user input is needed. Among the issues to take care of is for instance the ability for users to use `tikz`, which dumps low level PDF code and creates objects. So, for that I needed to provide a compatibility layer. However, that is probably the only package from outside ConTEXt that is used, so I might as well make that bit of the interface a runtime option at some point. After all, there is no need to pollute the regular code. In a similar fashion I provide an `img` library mockup but that one might be dropped: users should use the high level interfaces instead. These are typical issues discussed at meetings.

For quite some time this variant will be a moving target that aims at ConTEXt and the reason for this is in the beginning of this article: we do this for the benefit of users but also for fun. We also want a vehicle that permits us to freely experiment (for instance with more advanced virtual fonts) without worrying about usage elsewhere. And eventually it will be, like its ancestor LuaTEX, a stable general purpose alternative, but one a bit closer to where it all started: original TEX.

In order to make experimenting easier, there will be an updated installer for the ConTEXt garden. That one will use HTTP instead of Rsync and use a MkIV only set of resources. All binaries except for the engine and two runners are dropped and no MkII specific and generic files are present. Apart from power users nobody might notice this move.

Hans Hagen

LaTeX on the Road

An adventure with LaTeX while travelling light

Abstract
This article describes the adventures that I had while working on a small TeX project without my beloved laptop at hand. With only an iPad to do the work and without a local TeX system installed on it, there were several challenges. I document them here so that others can enjoy the struggles I had and can benefit from the solutions when they encounter similar situations.

Keywords
LaTeX, travelling, iPad, Overleaf, git, distributed version management, Github

The context

In July 2018, my wife Cary and I were travelling in South America, to visit friends in Brazil and Bolivia, and additionally to have some vacation. We wanted to travel light, so I had decided not to take my MacBook with me, saving a little bit more than 2 kgs. of weight. We both had our iPhones and iPads (mine is an iPad mini), and we hoped that would do. They were mainly to be used for reading email, interactions on social media, searching for city and transport information, and the like.

I did not expect to do any TeX work, maybe some light programming, for which I had a Python system (Pythonista)[1] on my iPad.

While we were travelling in Brazil, on our way to Bolivia, I got an email from a user of the `multirow` package about a possible bug. It came with a solution which was a very simple substitution, and back home on the laptop, it would have been a few minutes to make the change, check it in into the version control system, do some test, generate a new version of the documentation, and upload the new version to CTAN.

Figuur 1. On our way to Brazil

Because this person had already made a local change, and the problem was not urgent anyway, my first reaction was: I will correct it when I am back home, which, by the way, would be some 2 months later. However when we arrived in Bolivia, where we were staying a couple of weeks, the temptation to solve the problem right there became too large.

But what would have taken at most 10 minutes at home, became a major effort without having a computer with a TeX system. In the end it took me more than two days of struggling, but with victory in the end.

If I would have distributed the package just as a collection of `.sty` files (there are three included), with a separate documentation, the task would have been simple. I could have downloaded the package from CTAN, changed the `.sty` files with a text editor in my iPad, and uploaded them back to CTAN. It might have caused some frowning from the CTAN maintainers if the version number in the documentation would have been different from the one in the `.sty` files, but that would have been temporary anyway.

However, the package was distributed as a `.dtx` file, with a corresponding `.ins` file, and a separate PDF file containing the documentation which is generated from the `.dtx` file. The `.sty` files are also generated from the `.dtx` file with the aid of the `.ins` file. This is the standard setup for most CTAN packages. But this requires the `.dtx` and `.ins` files to be processed by LaTeX (or TeX in case of the `.ins` file). And I did not have a LaTeX distribution on my iPad.

What were the options?

There were in practice two options to solve the problem:

- Install a LaTeX system on my iPad.
- Use an online (cloud-based) LaTeX system.

Figuur 2. Our trip

LaTeX apps on the iPad I found two LaTeX apps on the iOS App Store: Texpad and TeX Writer. Both are offline apps, i.e. you don't need an internet connection to compile your LaTeX documents. But, on the other hand, in order to limit the size of the application, they don't have every package from CTAN installed. You can install additional packages, but as iOS is quite a closed operating system, you are dependent on the developers to supply these packages. Of course you have always the option to add the required files to your project directory, but there might be some cases (for example if you need additional fonts) that this is not sufficient.

Also it isn't clear from the documentation of these packages if they can process something like `.dtx` and `.ins` files to extract the `.sty` files and the documentation for the package, which was essential in my case. I got the impression that they were mainly meant for the 'normal' user to write articles and reports.

They are also not particularly cheap. At this moment Texpad costs €21.99 and TeX Writer €16.99. If I remember correctly they were a little bit cheaper at the time I was travelling. In itself that is not a very steep price, but I did not expect to use it very often, and just for this single case I thought it was too much. And they don't have a tryout version to see if it really fits you, so if you buy one of these, and you don't like it, you effectively lost your money. And then there is this nagging choice: which of the two is best? All in all, I decided not to go that way.

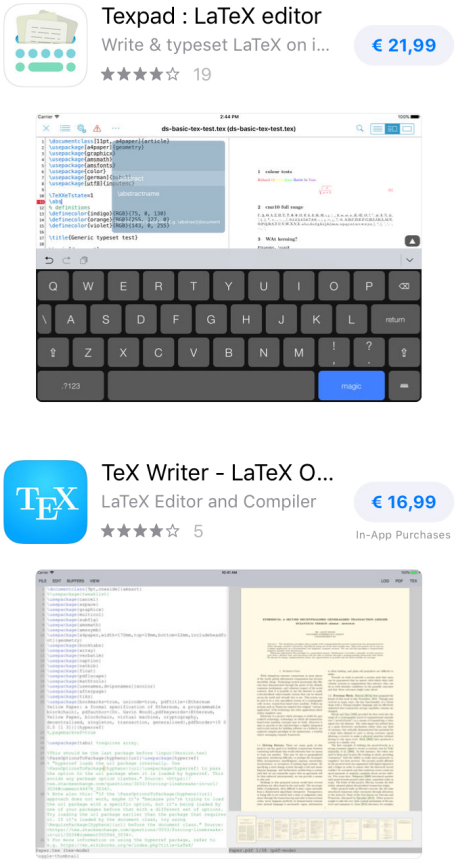
For the cloud-based systems, I had heard about Overleaf (formerly called WriteLatex) and ShareLaTeX, so I decided to investigate these. It appeared that at that time, these two systems were in a processed of being merged. The result was Overleaf version 2 which had the ShareLaTeX interface, but was still in beta phase. For the simple task that I had, a free account would be sufficient, so I started to try that. However, the merging process introduced some teething troubles. In fact it made editing the files from the iPad browser almost impossible. It wasn't clear if this was a specific problem on the iPad, or that the browser interface in general was not yet mature enough. In effect it wasn't usable at all, because its behaviour was very erratic.

I had also tried to use the Overleaf version 1 interface, but I also could not get that working. I have no idea whether these problems were iPad specific, but anyway I could not use it. By the way, the Overleaf editor is now functioning also on the iPad. However, some functionality is not available without an external keyboard, because they are invoked with control keys. For example the search function is invoked by Control-F on Windows and Linux, and by Command-F on MacOS. On an iPad you can't give these with the virtual keyboard. With an external keyboard it is possible. The current Overleaf editor is reasonable. It has some TeX-specific functionality. For example, if you type `\begin{enumerate}` the editor adds `\item` and `\end{enumerate}` and positions the cursor after the `\item` (see figure 3).

```
712 \begin{enumerate}
713     \item |
714 \end{enumerate}
```

Figuur 3. Editor supplies useful parts

Cloud-based LaTeX systems Despite the problems that the editor gave at that time, it seemed to me that this was the best way to go forward. Figure 5 shows the screen from the current version of Overleaf on my MacBook. The default screen has an edit window with the LaTeX source text and a preview window with the resulting PDF. The preview is not live, you have to hit the Recompile button to update it. There is also a file list on the left and it has the possibility to hide or show each of these parts and to adjust the sizes of each part. Especially on the smaller iPad screen it is advisable to



Figuur 4. Texpad and TeX Writer in the iOS App Store

have only the source code part while editing. But even then, the virtual iPad keyboard takes so much space that hardly any source code is visible. See figure 6. Also in this case, the file list at the left would make the edit window even smaller, but the file list can be hidden, as shown in the image.

It helps to put the iPad in portrait mode, as shown in figure 7. But then the keyboard is rather small. For a setup like this to be workable, it would be better to use an external keyboard. There are several keyboards on the market that can be used. They are generally connected through Bluetooth. They are light-weight and don't take much space, so ideal for travelling light. See figure 8. I did not have one at that moment, however.

Setting up the project

Setting up the project is easy. You can create a new project in the Overleaf in the Web interface. You can upload each file individually, or a zip-file with everything included. Overleaf will unpack the zip-file in your project.

Immediately, it became apparent that there was a problem with my project. Overleaf wants you to designate one of your files as the main TEX file, which for me would have been `multirow.dtx`, but it doesn't accept this. It wants to have a `.tex` file. It does not recognise the `.dtx` file as a valid LATEX file. Neither does it want to edit the `.dtx` file, but as the editor was unusable, this was of a minor concern. I would have to edit the files locally on my iPad anyway.

So I had to give it a `.tex` file extension to make it (and myself) happy. I tried two ways

- Copy `multirow.dtx` to `multirow.tex`
- Make a file `multirow.tex` that just contains `\include{multirow.dtx}`

I had expected that each of these would compile the `.dtx` file when the Compile button would be pressed. However, it didn't. It took some time to find out why. My `multirow.dtx` contains a line

```
\DocInput{\jobname.dtx}
```

which is quite usual in `.dtx` files. After some searching I found out that `\jobname` wasn't `multirow` as was to be expected, but `output`. It appears that Overleaf runs the job in a kind of *sandbox* where the jobname of the main file is `output`. You can see that in figure 12.

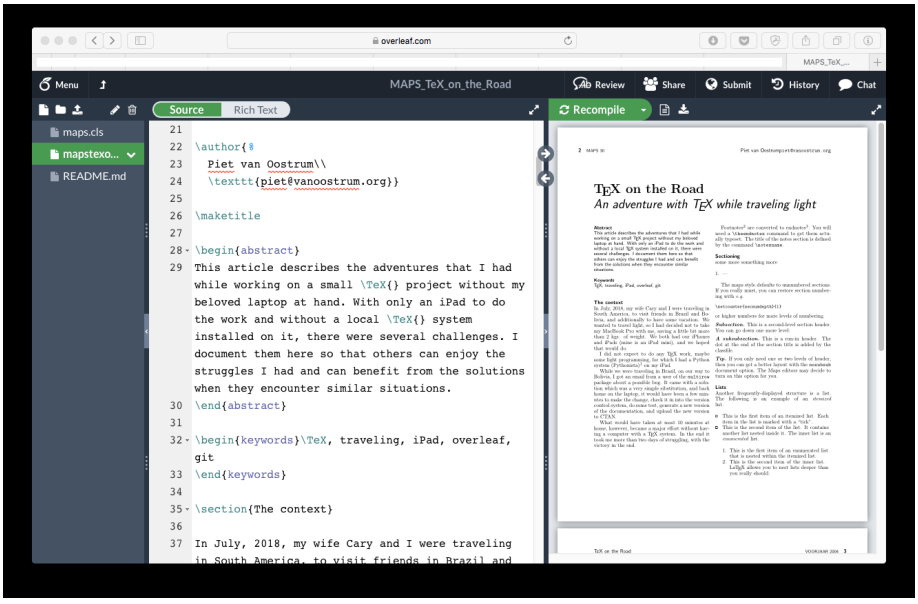
After some googling I found that Overleaf uses `Latexmk`[2] to process the job. It provides a standard, but invisible, `latexmkrc` file that controls the compilation process. However, you can also supply your `latexmkrc` file. This file is described in section 'Latexmk' on page 64.

So the challenge was now to upload a correct `latexmkrc` file, and to update the `multirow.dtx` file. This could be done by uploading these files after each modification, but this might be an error-prone process, and you don't have a record of what has been done. Enter *version management*.

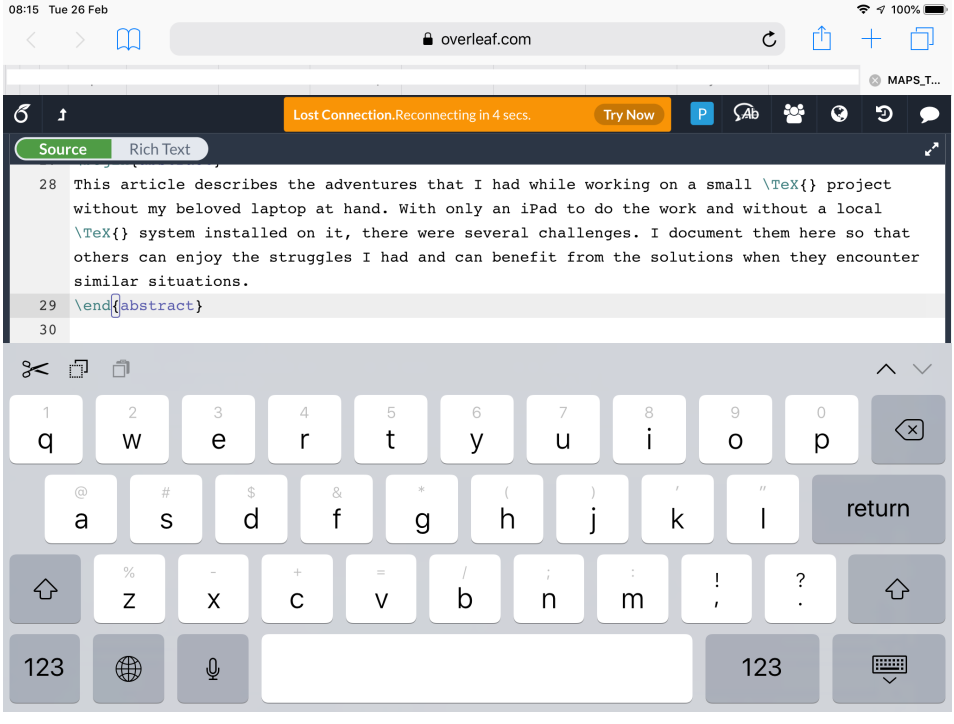
Distributed version management

In any project where you have to make changes more or less regularly, it is important to keep track of what you have done. Also, in general it is useful to have access to previous versions of your project, for example if you want to go back to a previous situation. Some people do this by making copies of their files at regular moments. Sometimes they put the date and the time in the file names, to keep a kind of history. But this soon becomes unwieldy. This is the problem that version management system (also called version control systems) offer a solution for. Each serious developer, whether it is of software or of texts, should consider using a version management system.

For those readers that are unfamiliar with version management, here follows a brief description. You have

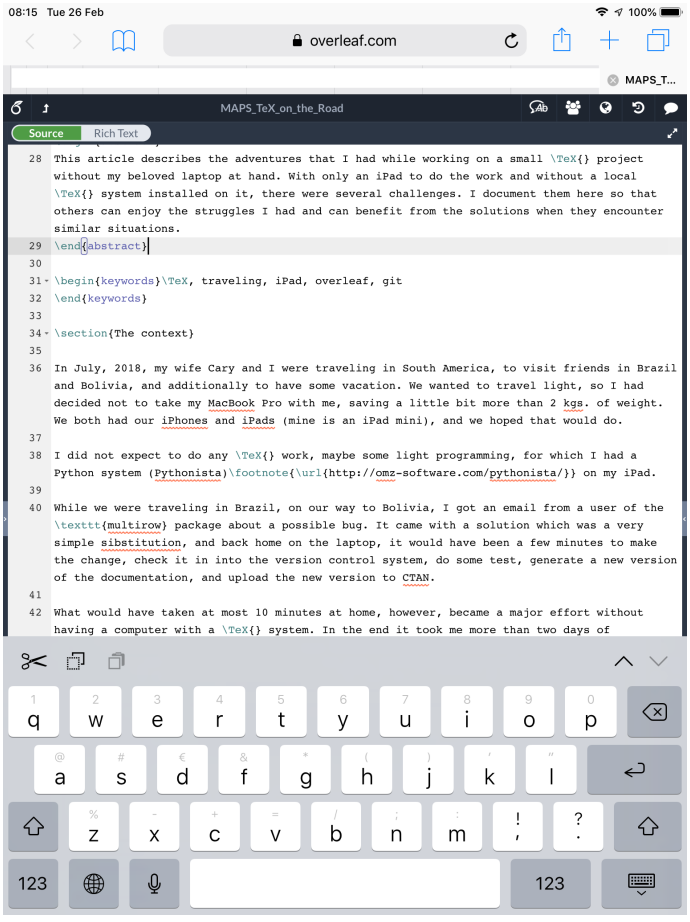


Figuur 5. An early version of this article in Overleaf, with some of the maps.cls documentation still in place



Figuur 6. Overleaf screen with virtual keyboard on an iPad

a *working copy* or *working directory*, which is the collection of files that you work upon in your project. This is just like when you do not use version management. Additionally you have a *repository*, which is a kind of database containing the history of your project. It will contain the state of your *working copy* at certain moments in the past, together with information about who made the changes, and a description of what has changed.



Figuur 7. Overleaf screen with virtual keyboard on an iPad in portrait mode

If, at a certain moment, you have a state of your project that you want to keep, you *commit*, which means a copy is made to the *repository*, together with a description that you enter. The opposite operation (i.e. making a copy from your repository to your working directory) is called *checkout*. You usually have a separate repository for each project. The repository can be on your local computer, or on a server. In the latter case it is possible that different people working on the same project use the same repository. They would then each have their own *working copy*. As they are working independently, these could be different. A version management system usually has provisions to resolve conflicting working copies.

Although these systems can store any type of file, they work best with plain text files. As our TEX sources are plain text, they are ideal candidates for using a version management system.

There are several version management systems available. One older, well-known system is *subversion* (SVN[3]). It usually has the repositories at a central server, but you can also have the repository on your local computer, if you are working alone. As SVN has only one repository per project it is called a centralised version management system.

Centralised version management systems have some big disadvantages for cooperation in teams:

- If you work together the repository must be on a central server, which means you cannot use it when you are offline.
- If you want to keep your changes registered often in the repository, then this can be confusing for the other team members. On the other hand, if you want to keep the repository relatively clean, that is, only commit major updates, then you lose the possibility to keep your own history detailed.

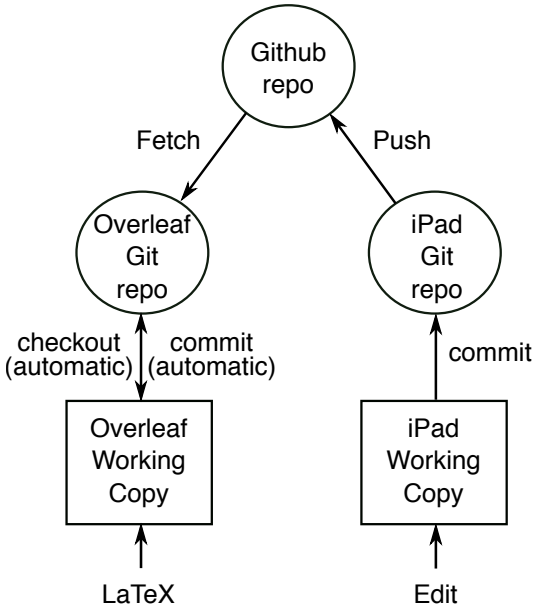
One solution would be to have both a central repository for the team, and your local repository for your own work, but then synchronising these repositories could become tedious. However, this is where *distributed version management systems* have their strength.

Figuur 8. iPad mini with external keyboard

In a *distributed version management system* you can have both a local repository on your computer and a central repository on a server. Or even more than one of each. And these can be easily synchronised. The usual way to work in a team is to have a central repository for the team, and a local repository on each team member's computer. Each team member keeps a history in the local repository. This can be done often, and also offline. When the changes are good enough to be put in the central repository, the team member *pushes* the local changes to the central repository, often after making one set of changes that do not reflect all the details of the work done locally. Other team member can the *fetch* these changes from the central repository when they want to be up-to-date. It is then probable that this is not consistent with some changes that they made themselves. The two sets of changes must then be *merged*. This is the simple explanation. Much more complicated workflows are also possible.

Both centralised and distributed version management systems support the concept of branches. A branch is a separate line of development in your project. For example you have a project that you release from time to time. The development of this release version would for example take place on the main branch in your repository. Now after a release you want to start working on some very new experimental features for a future release. If you would just continue your development, then when a bug in your project is detected, your project is in an unstable state. So you cannot just apply a bug-fix to the current state of your project, but you would have to go back to the state just after the release. As the repository has kept the history of your project, this is easy, but you want also to keep the current state, so that you can go back there after making the bug-fix. Here branches come to the rescue. After your release you create a new branch for your experimental work, and continue working there. When you want to make the bug-fix, you switch back to the main branch. The repository will remember your experimental branch, and after releasing the bug-fix you can switch back to the experimental branch. If you wish you can then also *merge* the fix in your experimental branch. Later when your experiment is successful and you want to release it, you can merge it back to the main branch. You can have as many branches as you want. For example if your bug-fix is expected to be complicated, you can first try it out on a separate branch.

A very popular site for central repositories is Github.[4] This site is based on the distributed version management



Figuur 9. Git workflow

system `Git`. Git is probably the most popular version management system in use today. For my own projects I use Git exclusively nowadays, often only locally, but sometimes in combination with Github.

Use with Overleaf To come back to the project I am currently describing, it appeared that Overleaf also had Git possibilities. Although these were in beta phase at that moment, it could be used for my project. Nowadays you need a paid account on Overleaf to use the Git facilities, but because I had started using them during the beta testing, I have access to them in my free account.

Git can be used in two ways on Overleaf.

- Your Overleaf project can function as a Git repository
- You Overleaf project can be synchronised with a Github repository

I decided to take the Github route, mainly because I have experience with Github and I could not get the direct Git repository on Overleaf working from the iPad. At this moment it is working, but its functionality is very minimalist compared to Github.

In order to use Git on the iPad you need a Git app. I found Git2Go,[5] which is said to be the first app to use Git on iOS. It worked well for my needs, but later I tried two others that I found: Working Copy[6] and TIG.[7] In the appendix I give a comparison of these apps.

My workflow can be found in figure 9. My editing took place in the lower right corner, on the working copy (managed by Git2Go). I could have used the editor that Git2Go provides, but it is not very sophisticated. It does not have syntax highlighting for LaTeX files, and it gives no editing support beyond the standard iPad keyboard. I also had a much better text editing program called Textastic[8]. It has syntax highlighting for LaTeX, good search facilities and an extended keyboard (see fig. 10) that makes it easier to enter non-alphanumeric symbols. Also it has a special provision for easy cursor movement. Git2Go, and the other Git apps mentioned above, function as a kind of file system, which means that Textastic can directly edit their files without copying between the two apps. So the only extra operation to edit in Textastic rather than in Git2Go itself, is switching between the apps. This extra effort I deemed worthwhile for the added comfort of using a good text editor.

After editing the file(s), I switch to Git2Go, commit the change, and immediately push it to the Github repository. Then I switch to Overleaf in the browser, fetch the changes from Github to Overleaf in the Overleaf synchronisation menu, and process the files, hopefully producing a new PDF file. Many times it did not yet work correctly, so I had to go back to Textastic and start a new cycle. The problem wasn't so much in the LaTeX code, as the changes there were very simple. The main problem was getting the `latexmkrc` file correct. The difficulty was that Overleaf did not have good documentation about the context in which the Latexmk program was running. Also, running it on their server did not give as much feedback as running on your own computer. Several times I had to write extra information to a text file, and then download that to the iPad to see what happened. For example, I had to make directory listings, and write them to a text file, just to see what files where generated and what their names were. And the process was a bit tedious because I had to synchronise the files as described above before each try. But after some 50 tries, everything worked perfectly. I will spare you all the attempts that I made, but in the next section I will give you the resulting `latexmkrc` file, and explain what it does.

Latexmk

Latexmk is a program (a Perl script) to process a LaTeX file with all the necessary `bibtex`, `makeindex` and similar calls. It will run LaTeX and these other programs as many times as is necessary to get a completely processed and stable output. For the run-of-the-mill LaTeX file, Latexmk has enough knowledge to know what to do. However, when there are additional requirements, like a non-standard index, glossaries etc. you must give Latexmk a recipe of how to process the various stages. The recipe is given in the `latexmkrc` file, which in fact is also a Perl script. Latexmk has an enormous amount of possibilities, and its manual[9] contains 48 pages. So it took some time to get everything right.

Figuur 10. Textastic extended keyboard

Overleaf provides a standard `latexmkrc` file for its jobs, but as we have seen above, this is not adequate for processing the `.ins` and `.dtx` files. To make Overleaf happy, we must provide a main `.tex` file, but with our `latexmkrc` file we don't use it, so its content is unimportant.

In figure 11 the resulting `latexmkrc` for this process is given, annotated with line numbers. In the remainder of this section I explain what it does.

line 1. This sets the timezone to your local time. This is so that messages with date and time will get your local time, and not the time of Overleaf's servers, which would be useless in most cases. As I was in Bolivia at the time, the timezone was 'America/La Paz'. Now at home it would be 'Europe/Amsterdam'.

line 3-6. In a `.dtx` file the file extension `.glo`, which is normally used for glossaries, is used for the list of changes. And the sorted version, to be created by `makeindex`, will be `.gls`. These lines give a recipe how to create the `.gls` file from the `.glo` file using `makeindex`.

line 8. For processing the normal index in a `.dtx` file `makeindex` needs the additional argument `-s gind.ist`.

line 10. This defines which extra file extensions we need in the process. Besides the already mentioned `.glo` and `.gls`, there is also `.glg` which is the log output of the `makeindex` command from line 5. And the `.txt` extension is used for debugging.

line 12. Here comes the trick to let Overleaf do our work. Normally it will run `pdflatex` on the main TEX file, which in our case is `multirow.tex`. But you can define the `$pdflatex` variable to let it use another command. In our case we let it run the internal function `mylatex` that follows. In this function we do all the preparatory work before we run the actual `pdflatex`.

line 14. Pick up the arguments from the call to `mylatex` in the variable `@args`. This is standard Perl prose.

Latexmkrc file:

```
1  $ENV{'TZ'} = 'America/La Paz';
2
3  add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
4  sub makeglo2gls {
5      system("makeindex -s gglo.ist -o \"$_[0].gls\" \"$_[0].glo\"");
6  }
7
8  $makeindex = 'makeindex -s gind.ist -o %D %S';
9
10 push @generated_exts, 'glo', 'gls', 'glg', 'sty', 'txt';
11
12 $pdflatex = 'internal mylatex';
13 sub mylatex {
14     my @args = @_;
15     (my $base = $$Psource) =~ s/\.[^.]+$//;
16     system("tex $base.ins");
17     # backslashes are interpreted by (1) perl string (2) shell (3) sed regexp
18     # therefore we need 8 backslashes to match a single one
19     system("sed -e s/\\\\\\\\\\jobname/$base/g $base.dtx > $base.tex");
20     return system("pdflatex @args");
21 }
```

Figuur 11. The final latexmkrc file. The line numbers are not part of the file.

line 15. Latexmk puts the name of the main TEX file in `$$Psource` (see page 45 in the Latexmk manual). This line is actually a shorthand for two statements:

```
my $base = $$Psource;
$base =~ s/\.[^.]+$//;
```

The first line copies `$$Psource` to a local variable `$base`. The second line strips of the part after (and including) the last dot. So `multirow.tex` will be transformed to just `multirow`. The reason I use `$$Psource` rather than just using `multirow` is that now the `latexmkrc` file is also usable for other `.dtx` files.

line 16. First we run `tex` on our `.ins` file, which would be `multirow.ins` in our case. This generates the required `.sty` files. This is to ensure that we use the new versions of our `.sty` files, rather than an outdated version in Overleaf's TEX system.

line 19. From our `.dtx` file we generate a copy to our `.tex` file where the text `\jobname` is replaced by the actual base name of our file (in our case `multirow`). This is necessary, as Overleaf supplies a `\jobname` of `output`. So in this case we generate `multirow.tex` from `multirow.dtx`. But this file will input `multirow.dtx` during its processing.
We do the replacement by calling the Unix program `sed`. The `\jobname` is inside a regular expression in `sed`, therefore the backslash must be doubled. But then, this command is processed by the Unix shell, which also interprets backslashes. Therefore we must double all the backslashes again. And then this command is inside a Perl string where backslashes are also interpreted. So we must double them again, and we end up with 8 backslashes to represent a single one.

line 20. Finally we run the real `pdflatex` command with the original arguments. Note that we process the new `multirow.tex` file, because that is what Overleaf expects to do. Also, because this is run in a `sandbox` (i.e. on a copy of the original files in a separate directory), this does not affect our original file.

Finally, we also give an example of the `latexmkrc` file with debugging statements included in figure 12 and the corresponding output in figure 13. You see the values of `$$Psource` and `$base`, the arguments to the `pdflatex` call, and the directory listing at the end of the process. Please note that in the directory listing there is a file `multirow.log`; this is the result of the call `tex multirow.ins`. Note also the generated `.sty` files. The files resulting from the `pdflatex` call on `multirow.tex/dtx` are all called `output.*`. So `makeindex` must also act on these files. In figure 11, line 5, this is accomplished because the file name is given as argument to the function `makeglo2gls`. In line 8 it is accomplished because the patterns `%S` and `%D` are replaced by the *source* and `destination` of the command, respectively. I.e. `output.idx` and `output.ind`.

Latexmkrc with debugging:

```
$ENV{'TZ'} = 'America/La Paz';

add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
sub makeglo2gls {
    system("makeindex -s gglo.ist -o \"$_[0].gls\" \"$_[0].glo\"");
}
$makeindex = 'makeindex -s gind.ist -o %D %S';

push @generated_exts, 'glo', 'gls', 'glg', 'sty', 'txt';

$pdflatex = 'internal mylatex';
sub mylatex {
    my @args = @_;
    Run_subst("echo \"%%B=%B %%R=%R %%S=%S %%T=%T\" > debugout.txt"); ## DEBUG ##
    system("echo '\@args' = \"@args\"              >> debugout.txt"); ## DEBUG ##
    system("echo '\$\$Psource' = \"$$Psource\"     >> debugout.txt"); ## DEBUG ##
    (my $base = $$Psource) =~ s/\.[^.]+$//;
    system("echo '\$base' = \"$base\"              >> debugout.txt"); ## DEBUG ##
    system("tex $base.ins");
    # backslashes are interpreted by (1) perl string (2) shell (3) sed regexp
    # therefore we need 8 backslashes to match a single one
    system("sed -e s/\\\\\\\\\jobname/$base/g $base.dtx > $base.tex");
    $status = system("pdflatex @args");
    system("ls -l                                  >> debugout.txt"); ## DEBUG ##
    return $status;
}
```

Figuur 12. `latexmkrc` file with debug statements

Conclusion

Although working at home on my MacBook is much more comfortable, it is possible to do some serious LaTeX work on your iPad while you are travelling. It takes some effort to find the proper way to do it, however. And I hope this article helps you to get started if you need this work flow.

Appendix – iOS Git apps compared

In this section I compare the three Git apps on iOS that I tried. I did all the production work in Git2Go, but after it was finished I also tried Working Copy and TIG.

Git2Go has a limitation that it only cannot work with Git repositories on all servers. It works with a limited number of services, namely Github, Bitbucket[10] and Gitlab[11]. Other remote repositories can be used if they offer access by the SSH protocol. SSH is one of the two main protocols used to connect to Git servers. The other is HTTPS. Overleaf only offers HTTPS, which Git2Go does not support.
To create a repository on your iPad you must *clone* (i.e. copy) an existing repository on one of the supported servers. You cannot create a local-only repository on the iPad. Once you have the repository on your iPad, you can edit the files in the repository, commit the changes, create new branches. It can fetch from and push to the remote repository, but these are not separate operations. It always does a fetch (which may be empty), followed by a push. It can also merge different branches. It is a limited set of operations compared to the full Git functionality, but it is sufficient for a normal workflow as described above. Also cooperating with other people would be possible, if it doesn't require the more esoteric Git functionality. Git2Go's editor has syntax highlighting for a limited number of programming languages.
Git2Go is free, as long as you only access *public* repositories (i.e. repositories that everybody can see). To access private repositories you would have to buy an upgrade.
For the push operation you will have to login, and Git2Go will remember your username and password, until you explicitly logout.
And occasionally it crashes.
Last minute Note: I tried to re-install Git2Go on my iPhone, and got the message that it was no longer available on the App Store. Also a search in the App Store did not come up with Git2Go. I have no idea if this is a permanent situation, or that it might be in a process of updating.

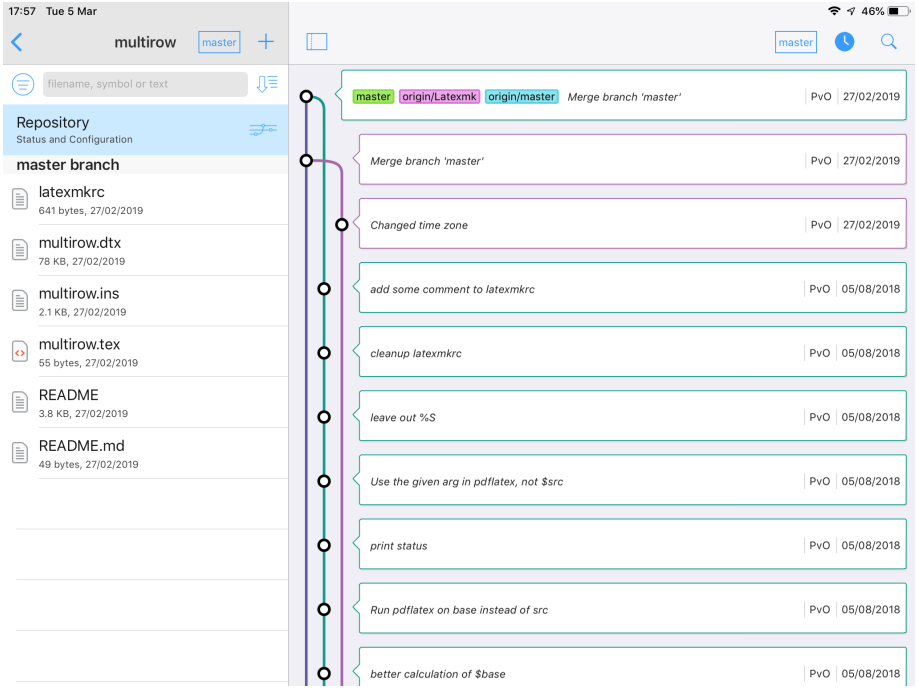
Debug Output:

```
%B=output %R=output %S=multirow.tex %T=multirow.tex
@args = -synctex=1 -interaction=batchmode -recorder -output-directory=/compile --jobname=output multirow.tex
$$Psource = multirow.tex
$base = multirow
total 1176
-rw-r--r-- 1 tex tex   3871 Mar  4 14:12 README
-rw-r--r-- 1 tex tex     49 Mar  4 14:12 README.md
-rw-r--r-- 1 tex tex   1417 Mar  4 14:12 bigdelim.sty
-rw-r--r-- 1 tex tex   1234 Mar  4 14:12 bigstrut.sty
-rw-r--r-- 1 tex tex    203 Mar  4 14:12 debugout.txt
-rw-r--r-- 1 tex tex   1054 Mar  4 14:12 latexmkrc
-rw-r--r-- 1 tex tex  80398 Mar  4 14:12 multirow.dtx
-rw-r--r-- 1 tex tex   2182 Mar  4 14:12 multirow.ins
-rw-r--r-- 1 tex tex   3719 Mar  4 14:12 multirow.log
-rw-r--r-- 1 tex tex   5022 Mar  4 14:12 multirow.sty
-rw-r--r-- 1 tex tex  80398 Mar  4 14:12 multirow.tex
-rw-r--r-- 1 tex tex   3487 Mar  4 14:12 output.aux
-rw-r--r-- 1 tex tex      0 Mar  4 14:12 output.chktex
-rw-r--r-- 1 tex tex  25207 Mar  4 13:10 output.fdb_latexmk
-rw-r--r-- 1 tex tex  20593 Mar  4 14:12 output.fls
-rw-r--r-- 1 tex tex   3281 Mar  4 14:12 output.glo
-rw-r--r-- 1 tex tex   3578 Mar  4 08:13 output.gls
-rw-r--r-- 1 tex tex   3270 Mar  4 14:12 output.idx
-rw-r--r-- 1 tex tex    891 Mar  4 08:13 output.ilg
-rw-r--r-- 1 tex tex   2655 Mar  4 08:13 output.ind
-rw-r--r-- 1 tex tex  34086 Mar  4 14:12 output.log
-rw-r--r-- 1 tex tex 610336 Mar  4 14:12 output.pdf
-rw-r--r-- 1 tex tex 262970 Mar  4 14:12 output.synctex.gz
-rw-r--r-- 1 tex tex   1467 Mar  4 14:12 output.toc
```

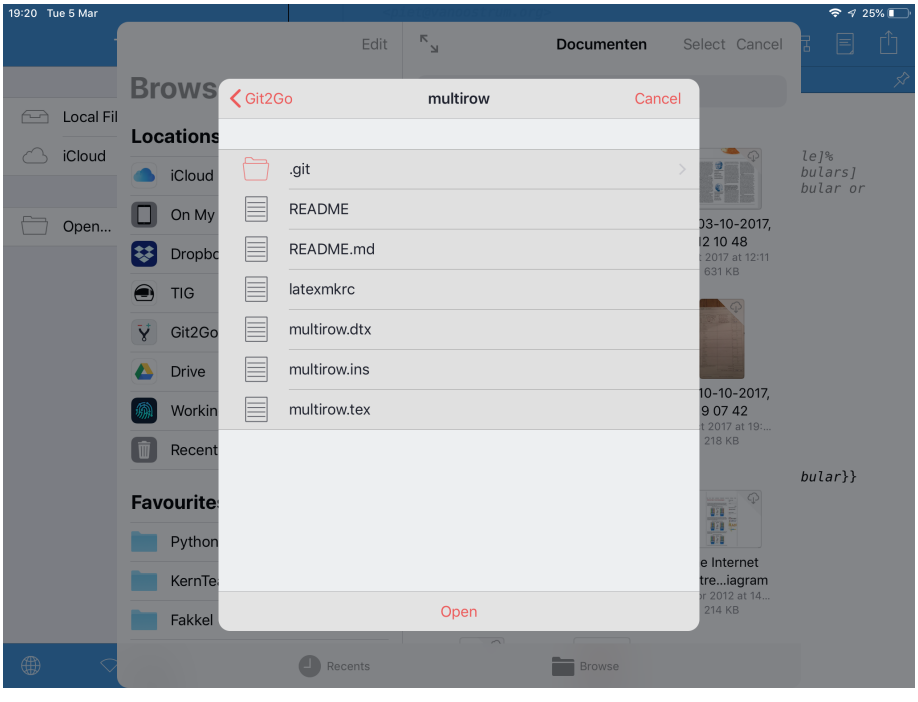
Figuur 13. `latexmkrc` debug output

Working Copy is the nicest of the three apps. It has a very elaborate set of functions. It can connect to all kinds of servers, including Overleaf. However, to use the `push` functionality you have to pay. The price is quite steep (€17.99 at the time of writing), but you can get a free 10 day trial. I used this for writing this article to see how it worked.
Working Copy can clone from existing repositories, including through SSH and HTTPS, and also create local repositories. It can also create a local repository from a `.zip` file. Once you have a repository it can connect your repository to more than one remote repository, which sometimes can be quite handy. For example in the current example, the repository on the iPad could have been connected both to the Overleaf repository and to the Github repository. Of course you will have to be careful not to mess up your workflow.
If your iPad is connected to a Mac or PC with iTunes, you can drag and drop a repository on your computer through iTunes, and it will be copied to the iPad.
Working Copy's editor has syntax highlighting for more than 50 different languages. It can show nice graphical representations of your branches and your commit history (see fig. 14). Besides the `merge` functionality it also has the `rebase` functionality, which is an alternative for merge. For cooperating in large projects this functionality is sometimes necessary.
There is more than fits in this limited space, but Working Copy is far out the best of the three apps. It is expensive, but when you do a lot of work with Git on your iPad, it is worth the price. Working Copy operates in a small market, so the price is understandable. If you really want to do serious work with Git on your iPad, it is recommendable to buy this app.

TIG is the third app I tried. It takes more or less a middle ground between Git2Go and Working Copy. Like Working Copy it can connect to all kinds of repositories, including Overleaf, and it has *push* functionality. And it is free. It can clone existing repositories, and create local ones. It can also connect repositories to more than one remote repository.
Its editor has syntax highlighting support for 166 languages.



Figuur 14. Graphical commit history in Working Copy



Figuur 15. Opening a Git2Go file or repository in Textastic

However, although the functionality is great for a free app, I found its user interface sometimes confusing. And to fetch/push to your remote repositories you have to enter your username and password every time. I did not find a way in which it could remember these. This is very annoying. And it crashed quite often.

As I have mentioned above, all these apps have the facility that you can open their files in an external editor. Figure 15 shows how to open files from Git2Go in Textastic. This is done inside Textastic with the "Open…" button, then selecting "Git2Go". It is then possible to choose "Open" down in the pop-up, which will open the whole directory in Textastic, or select one filename, which will open that file.

Summary:

- If you only need access to repositories hosted by Github, Bitbucket or Gitlab, or repositories that can be accessed by the SSH protocol, and your requirements are modest, you can choose Git2Go (if still available).
- If you need access to repositories that do not fall in the previous categories (such as Overleaf), and you can live with a not so optimal user interface, and your requirements are modest, you can choose TIG. It may be a good choice when you want to connect to a repository that Git2Go does not support, and when you find Working Copy too expensive.
- If you want the top Git app on your iPad (or iPhone) and are willing to pay the price, I would recommend Working Copy. If you want to do serious work with Git, this is the choice and it would be worth the price.

There are nowadays some other Git apps available, but it seems that they are roughly comparable to one of the above. Some of them only support just Github, Bitbucket or Gitlab. I have not found any free app that comes with the functionality of Git2Go or TIG. And paid apps may be slightly cheaper than Working Copy, but they also have less functionality.

Footnotes

1. http://omz-software.com/pythonista/
2. https://mg.readthedocs.io/latexmk.html
3. http://subversion.apache.org
4. https://www.github.com
5. https://git2go.com
6. https://workingcopyapp.com
7. https://itunes.apple.com/us/app/tig-git-client/id1161732225
8. https://www.textasticapp.com
9. http://mirrors.ctan.org/support/latexmk/latexmk.pdf
10. https://bitbucket.org
11. https://gitlab.com

Piet van Oostrum
piet@vanoostrum.org
http://piet.vanoostrum.org