TUG 2007 — program and information

Tuesday July 17

9 am–5 pm	track 1: *LaTeX workshop,* Sue DeMeritt & Cheryl Ponchin
9 am	track 2: *MetaPost workshop,* Hartmut Henkel
10:30–10:45 am	*break*
10:45 am	track 1 continues
10:45 am	track 2a: *Beamer & TikZ workshop,* William Slough & Andrew Mertz
10:45 am	track 2b: *ConTeXt workshop,* Hans Hagen
12:30–2 pm	*lunch*
2 pm	track 1 continues
2 pm	track 2a continues
2 pm	track 2b: *Lua and LuaTeX,* Roberto Ierusalimschy and Taco Hoekwater
3:30–3:45 pm	*break*
5–7 pm	*registration & reception,* at the Tula Community Center

Wednesday July 18

8–9 am	*registration*	
8:30 am	Karl Berry, TeX Users Group	*Welcome*
8:35 am	Peter Wilson, Herries Press	keynote: *Between then and now — A meandering memoir*
9:30 am	Barbara Beeton, AMS & TUG	*STIX fonts and Unicode*
9:55 am	Jonathan Kew, SIL	*SIL font projects*
10:15 am	*break*	
10:30 am	Dick Koch, Univ. of Oregon	*Multiple TeX distribution support in MacTeX*
11:05 am	Jim Hefferon, St. Michael's College	*CTAN package sourcing*
11:45 am	Jonathan Kew	*XeTeX Live*
12:20 pm	Morten Høegholm, DTU	*LaTeX3 project update*
12:45 pm	*lunch*	
1:45 pm	Klaus Höppner, DANTE e.V. & TUG	*Typesetting tables with LaTeX*
2:25 pm	David Allen, Univ. of Kentucky	*Three-dimensional graphics in LaTeX*
3:05 pm	Morten Høegholm	*The breqn package: revised and revised*
3:45 pm	*break*	
4:00 pm	Ari Stern, Caltech	*Incorporating LaTeX text with LaTeXiT*
4:20 pm	Leonard Rosenthol, Adobe Systems	*Everything you wanted to know about PDF but were afraid to ask*
5 pm	q&a	

Thursday July 19

8:30 am	Robert Burgess, Cornell Univ.	*CrossTeX: A modern bibliography management tool*
9:10 am	Andrew Mertz & William Slough, Eastern Illinois University	*Programming with PerlTeX*
9:50 am	Chris Rowley, Open University	*Vistas for TeX*
10:30 am	*break*	
10:45 am	Paul Topping, Design Science	*MathType 6's TeX input for MS Word and Wikipedia*
11:25 am	William Hammond, SUNY Albany	*Dual presentation with math from one source*
12:05 am	Barry MacKichan, MacKichan Inc.	*Design decisions for a structured front end to LaTeX*
12:45 pm	*lunch*	
1:45 pm	Don DeLand, Integre	*From TeX to XML: The legacy of techexplorer and the future of math on the Web*
2:25 pm	Eitan Gurari, Ohio State Univ.	*LaTeX conversion into normalized forms and speech*
3:05 pm	Paulo Ney de Souza, UC Berkeley	*Long-time preservation strategies for TeX-sourced content*
3:45 pm	*break*	
4 pm	q&a, TUG meeting	
7 pm	*banquet* (at the Aztec Center)	

Friday July 20

8:30 am	Roberto Ierusalimschy, PUC-RIO	*About Lua*
9:30 am	Hans Hagen, Pragma ADE	*Introduction to the LuaTeX project*
9:30 am	Taco Hoekwater, Elvenkind BV	*The Lua TeX interface: Extra tables and callbacks*
10:30 am	*break*	
10:45 am	Hans Hagen	*LuaTeX attributes*
11:25 am	Idris Hamid, Colorado State Univ.	*Arabic script typography*
12:05 pm	Hans Hagen	*Zapfino as torture test*
12:45 pm	*lunch*	
1:45 pm	Nelson Beebe, Univ. of Utah	*Extending TeX and METAFONT with floating-point arithmetic*
2:25 pm	Taco Hoekwater	*MPLib: Turning MetaPost into a reusable component*
3:05 pm	Hans Hagen	*ConTeXt MkIV*
3:45 pm	*break*	
4 pm	Idris Hamid	*Critical editions*
4:40 pm	panel	
≈ 5 pm	*end*	

Conference logistics

This information (and more) is online at `http://tug.org/tug2007/sdsu.html`.

- The conference location is the Cuicacalli Hall building of San Diego State University, San Diego, California; the talks and workshops will take place in the Cuicacalli Seminar Room (first floor). Here is the address: San Diego State University — Cuicacalli Residence Hall
  5150 E Campus Drive / San Diego, CA 92115 / USA.
- The dormitory accomodations are the third floor of an adjoining building, Tacuba Residence Hall.
- Every participant, whether staying in the Tacuba Residence Hall or off-campus, must sign in upon arrival at Cuicacalli Hall front desk where you will receive a meal card (breakfast and lunch for overnighters, lunch only for commuters), a parking permit if needed, and Internet access information ($10/week). The Cuicacalli Hall front desk phone number is 619-594-2622.
- If you are driving, park in nearby Parking Garage #3; get permit at the front desk.
- If arriving by trolley, exit at the SDSU stop, take the elevator to the ground floor, cross over the pedestrian bridge, turn left. The bridge ends at Tepeyac Hall. Walk a bit east, next building is Cuicacalli Hall, enter at main entrance.
- The initial registration and reception is on Tuesday, 17 July, 5 pm to 7 pm at the Tula Community Center (near the other buildings). Snacks and nonalcoholic beverages will be served. Please check in at the registration table, pick up your name tag, conference booklet and other items. If you're not able to attend the reception, you can register the next morning from 8 am–8:30 am outside the conference room.
- A swimming pool and volleyball court are located in the central quad area of the residence halls. Other recreational facilities such as basketball courts, weight room, bowling alley, etc., are available in the Aztec Recreation Center: see `http://arc.sdsu.edu/membership/index.php`. Near the top of the page is a link to a guest pass. Click on it, fill it out and print it. This entitles you to one free workout. Beyond that, workouts are $10 or $12 for 24 hours (the rate is may go up, we're not sure). You need to have ID with you to purchase the daily pass, and most likely to use the guest pass.

Mac OS X meeting

At Tuesday lunch there will be gathering of those interested in TeX on Mac OS X. It may continue in the afternoon after the Beamer+Ti*k*Z workshop, time and interest permitting.

TUG members meeting

After the regular session on Thursday, we will hold a TUG user group meeting for anyone interested. Several TUG board members will be present at the conference: Barbara Beeton, Karl Berry, Jon Breitenbucher, Sue DeMeritt, Steve Grathwohl, Jim Hefferon, Klaus Höppner, and Cheryl Ponchin, as well as TUG's executive director, Robin Laakso. We will welcome Dick Koch to the board, and report on TUG's current status and future outlook.

We invite discussion of any TUG-related business at this time: ideas for outreach to additional communities, additional initiatives to undertake, existing projects to support, or any other user group-related topics.

Banquet & soapbox

The conference banquet will be held in the Aztec Center (on the SDSU campus near the conference building) at 7 pm on Thursday, July 19. We will have a few door prizes at the banquet (as usual). If you haven't signed up for the banquet, it's not too late. Please just let us (email `office@tug.org`).

Thanks to Peter Flynn for suggesting a 66–99 second soapbox at the banquet, where anyone can speak for a minimum of 66 seconds and a maximum of 99 seconds on a TeX-related topic:

- You can report a success, gripe about a problem, lament a failure, share an insight, ask a question, or explain a solution.
- No intros, no questions, no hacking on earlier speakers; just you, the mike, and the audience ...
- ... well, and a moderator with a timer who will cut you off when your 99 seconds are up.
- No slides, overheads, whiteboards, blackboards, flipcharts, chalk, markers, or other props.
- Come prepared or make it up on the spur of the moment—no experience necessary.

Between then and now — A meandering memoir

Peter Wilson

I was asked to talk about something interesting — perhaps how I came to develop the `memoir` class. Following this suggestion the first part is about how I became involved with LaTeX and friends and why the `memoir` class. To me all this is not particularly interesting as it falls into the personal 'been there, done that' category. What I find more interesting is how the written word has been presented. The second part briefly describes this, starting four millenia ago with Cuneiform and, with a few stops along the way, ending at recent times.

STIX fonts and Unicode

Barbara Beeton

The goal of the STIX project is to provide fonts usable with other existing tools to make it possible to communicate mathematics and similar technical material in a natural way on the World Wide Web. This has involved two major efforts: enlarging Unicode to recognize the symbols of the mathematical language, and creating the fonts necessary to convert encoded texts into readable images.

This ten-year effort is finally resulting in fonts that can actually be used for the intended purpose.

Support for multiple TeX distributions in i-Installer and MacTeX

Dick Koch

We discuss a data structure by Gerben Wierda and Jérôme Laurens which makes it easy to use multiple TeX distributions on Mac OS X.

CTAN package sourcing

Jim Hefferon

I have some experimental software to improve the way in which packages are added to the Comprehensive TeX Archive Network (CTAN).

XeTeX Live

Jonathan Kew

With the release of TeX Live 2007, XeTeX has "come of age" and entered the mainstream of the TeX world. The XeTeX engine, which provides built-in Unicode and OpenType support, is now a standard part of a TeX Live installation, and thus is readily available to any user who installs this distribution.

This presentation will show how users can take advantage of XeTeX to easily use additional fonts in TeX or LaTeX documents, with no complex installation or setup procedures. It will also show how non-Latin scripts such as Chinese, Arabic, Devanagari, and many others can be typeset just as easily as English, thanks to full Unicode support throughout the system.

In addition, some of the newest developments in XeTeX (beyond the TeX Live 2007 release) will be discussed and demonstrated. These include support for the Graphite rendering technology for complex scripts; extensions that can simplify Chinese/Japanese character spacing and mixed-script typesetting; and more complete Unicode math support.

Typesetting tables with LaTeX

Klaus Höppner

From a LaTeXoligist's point of view, LaTeX is a perfect tool to typeset nearly everything in a beautiful manner. Without any doubt, LaTeX may typeset tables, but it is easy to produce bad tables with ugly lines and text touching the lines. This talk is intended to introduce how to typeset tables with LaTeX on a beginners' level, mentioning some typographic aspects, showing some packages that help the author in formatting tables and then telling how to typeset tables with page breaks.

Three-dimensional graphics in LaTeX

David Allen

PSTricks and its add-ons allow production of some three dimensional graphics. Two other programs, Asymptote and Sketch, implement graphic languages. This presentation gives a brief introduction to each of these programs and examples of their use.

The breqn package: revised and revived

Morten Høegholm

The breqn package facilitates automatic linebreaking of displayed equations. Originally developed by the late Michael Downes of the American Mathematical Society, it is now being restructured completely as part of a thesis project.

Incorporating LaTeX text into graphics and presentations with LaTeXiT

Ari Stern

It is often a challenge to combine text created in LaTeX with content from other software, such as graphics and presentation software, while maintaining a consistent typographical style. (For instance, text labels in mathematical figures often do not match the surrounding text.) This demo will show how this can be done easily using LaTeXiT, a free utility for Mac OS X included with the MacTeXdistribution. Examples will include 2D and 3D mathematical figures in Adobe Illustrator, as well as presentations using Apple Keynote.

CrossTEX: A modern bibliography management tool

Robert Burgess

CrossTEX is a new bibliography management tool that aims to render bibliographies less error-prone, databases easier to maintain, and documents easier to cite. It is based on an object-oriented data model that minimizes redundant information in bibliographic databases. It enables works to be cited not through syntactic object keys but through semantic information that uniquely identifies the work. It also supports customization and automation to a much greater extent in order to avoid errors, typos, and inconsistencies in bibliographies, while providing users with fine-grained control over the typesetting of references and citations. Other features include support for new object types, such as urls and patents, that have become common in citations, direct generation of HTML documents, easy programmability using a well-known language, and extensive databases of published works. It is backwards-compatible with existing BibTEXdatabases, and, overall, builds on BibTEX's strengths while fundamentally fixing the problems that lead to errors in BibTEX-formatted documents.

Programming with PerlTEX

Andrew Mertz, William Slough

PerlTEX couples two well-known worlds — the Perl programming language and the LaTEX typesetting system. The resulting system provides users with a way to augment LaTEX macros with Perl code, thereby adding programming capabilities to LaTEX that would otherwise be difficult to express. In this paper, we provide a brief tutorial on Perl and illustrate the use of PerlTEX with a variety of examples. Although Perl may perhaps be best known for its string manipulation capabilities, we will demonstrate how PerlTEX indirectly provides support for "programming" graphics through the use of additional packages such as Ti*k*Z.

MathType 6.0's TEX input for MS Word and Wikipedia

Paul Topping

MathType is well-known for its point-and-click user interface for editing math. However, some users feel more comfortable typing math using TEX, so in MathType 6.0 we have added a TEX input mode. This provides the user with the best of both worlds: TEX for initial entry, point-and-click and drag and drop for easy editing and manipulation. Since MathType can save equations in several graphics formats and objects, it provides a direct path from TEX to Microsoft Word, PowerPoint, and virtually any document or application. Since many blogs and wikis accept a variant of TEX math syntax and expose it in their web pages, we are now able to support both authoring and reuse of equations in these environments. In particular, MathType users can now copy equations out of the thousands of Wikipedia pages containing equations for use in educational and research authoring. In addition, MathType users can create equations and paste them directly into new Wikipedia content.

Dual presentation with math from one source using GELLMU

William Hammond

A contemporary author writing an article for "dual presentation" has in mind both the classical printed presentation of an article and the modern web form of an article based on HTML.

There are two main approaches for achieving dual presentation that are relevant to the TEX community.

— Write a LaTEX article, and use a program that translates to HTML.
— Write an article in a suitable XML document type, such as DocBook or TEI, and use standard software for generating LaTEX and HTML.

Both methods present challenges to authors who have been accustomed to using LaTEX.

Since mid-2002 the second-generation form of HTML that supports mathematical content has been supported by the two most widely deployed web browsers, but not many articles seem to have appeared on the web in this form so far. The most likely reason is difficulty of production.

This talk will address the use of "generalized LaTEX" to produce dual content from a single LaTEX-like source. This method combines the reliability of XML document transformation with many of the conveniences available when writing LaTEX markup.

From TEX to XML: The legacy of techexplorer and the future of math on the Web

Don DeLand

In 1997, IBM Research first released techexplorer, a browser plugin for rendering TEX markup directly within browsers. Since Integre took over techexplorer development in 2003 there have been relatively few advances in browser technology, but tremendous developments in collaboration tools and other web-based applications. This talk gives a brief history of the techexplorer project and explains why its development has shifted away from TEX to its current focus on native XML/

MathML authoring. Although delivering math in web browsers continues to be a frustrating process, "Web 2.0" holds substantial promise for a new generation of web-based applications that support mathematics.

LaTeX conversion into normalized forms and speech

Eitan M. Gurari

LaTeX is an authoring language designed for producing documents through native TeX compilers. During the years different applications have been developed to accept LaTeX input through engines programmed from scratch. Those engines are restricted in power to subsets of LaTeX features. The first part the presentation will demonstrate how TeX4ht can be used to translate general LaTeX constructs into restricted dialects of LaTeX recognizable by such engines. The jsMath dialect for LaTeX rendering through JavaScript (`http://www.math.union.edu/ dpvc/jsmath/`) will be employed for a target.

T.V. Raman introduced in 1994 his AsTeR program for automatically rendering technical documents into audio. His pioneering work assumed a LaTeX subset for an input. Current recommendations for speech browsers address XML documents abiding to the SSML and CSS specifications (`http://www.w3c.org`). The second part of the presentation will exhibit TeX4ht translations of LaTeX into speech through XML-based documents.

Long-time preservation strategies for TeX-sourced content

Paulo Ney de Souza

The amount of published material in the world has grown exponentially since Gutenberg's invention, with a rate of doubling every 7 1/2 years right now. Electronic publishing will only increase this rate, posing new challenges for the long-time preservation of records and usability for the future.

TeX has changed us into our own typists and even graphics designers sometimes, but at the same time has provided the best strategy for preservation of scientific content we have. This talk will examine some of these strategies and how MSP — a non-profit scientific publisher — has used it to improve usability of journals over time.

About Lua

Roberto Ierusalimschy

Lua is an embeddable scripting language that aims for simplicity, small size, portability, and performance. Unlike most other scripting languages, Lua has a strong focus on embeddability, favoring a development style where parts of an application are written in a "hard" language (such as C or C++) and parts are written in Lua. Currently Lua is used in a vast range of applications, being regarded as the leading scripting language in the game industry.

In this talk I will give an overview of the language, covering not only the technical aspects of the language but also its origins back in 1993, its evolution, and its current status.

The Lua TeX interface: extra tables and callbacks

Taco Hoekwater

Besides adding Lua as a scripting language to LuaTeX, we are also opening up the internals of TeX.

There are two sides to that: on the front side, we allow Lua code to inspect (and sometimes even change) what is going on inside TeX proper, via the addition of TeX-related tables. On the back side, there is an interface defined that allows Lua code to literally replace the compiled-in behaviour of TeX. It is actually possible to overload certain section of TeX-the-program completely.

This talk will give a wide overview of what is already possible as well as what is not yet possible, but already planned for the future.

LuaTeX Attributes: The new kid on the block

Hans Hagen

When you switch fonts in TeX grouping keeps changes to another font local to the group. But there is more than fonts. Most macro packages provide color support and in ConTeXt we also support some PDF related features like outlines and hidden invisible ink. These features all share a common problem: we need to keep track of their state in the page stream and across pages and splitting content also takes some care. A maybe less obvious example are hyperlinks, which are natively supported by pdfTeX (although ConTeXt does it slightly differently).

In order to make such features easier (and more robust) to implement luaTeX provides attributes. These behave like fonts but are by design indifferent of what they represent. They travel with the nodes (each node can have attributes) and it's up to the macro package to make sure that the intended behaviour takes place. For this Lua code is used in combination with processing node lists, either by using callbacks or by postprocessing boxes.

In this talk I will explain what attributes are, and how they can be of use to macro writers.

Zapfino: Hermann's torture test for TEX

Hans Hagen

The next couple of years TEXies have to explore the new landscape of OpenType fonts. Most of the implementation details will be hidden beyond user interfaces of macro packages. However this does not hide the potential mess that users can invoke when they start enabling or disabling features related to fonts.

Thanks to the Oriental TEX project Taco can spend substantial time on coding luaTEX which in turn means that I have lots of testing and protyping on my plate. We also spend much time on discussing the interfaces and extensions to the program and due to this as well as realistic testing luaTEX develops rapidly. However, in order to fulfill the requirements of the Oriental TEX project we need to be able to typeset high quality Arab. Since I'm more familiar with Latin and since I had the Zapfino Pro handwriting font waiting for me in OpenType format I decided to use that font as benchmark for advanced node processing in luaTEX. It proved to be a worthy contender. In the process we were able to optimize node support in luaTEX and it also triggered reimplementing the OpenType tables (from FontForge format 1 to format 2).

In this presentation I will discuss how we deal with advanced features that are part of OpenType fonts like Zapfino. I will also explain how such features are implemented in Lua and TEX code.

Extending TEX and METAFONT with floating-point arithmetic

Nelson Beebe

This paper examines how TEX and METAFONT handle numbers, and the historical reasons for the design of their arithmetic. It briefly surveys historical and current computer arithmetic, and suggests how TEX and METAFONT could enjoy a more flexible computational system *without* loss of their important and distinguishing feature of platform-independent results for typesetting, and for font design.

This work is based on current progress in standardization of computer arithmetic, on proposals for extending the C and C++ programming languages, and on the development by the current author of a large portable numerical function library that significantly enhances the computational environments of more than a half dozen widely-used programming languages, and could do so as well for many scripting languages.

MPLib: Turning MetaPost into a reusable component

Taco Hoekwater

Currently, MetaPost can only be used as a stand-alone program: it is not possible to combine your application with a library offering MetaPost drawing facilities. Likewise, it is not feasible to write a system service to create graphics, because the burden of restarting the executable each time makes such a system horribly inefficient.

The MPlib project aims to convert MetaPost's functionality into a form that *will* allow these things to happen. The goal is to create a reusable component that is fully re-entrant, has an extra layer for the configuration of input and output handling, and has error handling strategies that can be controlled by the user.

The project is funded by the joint TEX User Groups worldwide.

ConTEXt MkIV: luaTEX hits the road

Hans Hagen

In ConTEXt dealing with different backends has never complicated the system because driver dependent features are isolated rather well and use a driver independent layer. Different frontends are supported by conditional sections in the code which are dealt with at format generation time. When XƎTEX came around, frontend issues became more noticeable and with luaTEX we can safely say that we're dealing with a new kind of TEX (although it is still downward compatible with pdfTEX).

In order to use luaTEX to its full power, i.e. go beyond what pdfTEX provides, one needs to extend of even rewrite substantial parts of macro packages. As we develop luaTEX, Taco and I do lots of tests and we use ConTEXt as a testbed. In the process large parts of this macro package are replaced and the related version is tagged MkIV (its precedecessor has tag MkII). When luaTEX has become stable we will rewrite parts of ConTEXt even more drastically but the current state already gives a good impression of the impact that luaTEX will have on writing macros.

In this talk I will discuss what impact luaTEX has on the current releases of ConTEXt, and what the consequences will be for its future.

LaTeX class — Cheryl Ponchin, Sue DeMeritt

Contents

Beamer and Ti*k*Z Workshop

Andrew Mertz
`aemertz@eiu.edu`

William Slough
`waslough@eiu.edu`

For this workshop, we will introduce LaTeX users to two packages: beamer and Ti*k*Z. Although these packages can be used separately, they are a natural combination.

The beamer package provides flexible and powerful environments which can be used to create PDF-based documents suitable for presentations. The Ti*k*Z package provides flexible and relatively simple ways to specify graphics within LaTeX documents. Unlike mouse-driven graphics programs, Ti*k*Z uses a coordinate system and a language-based approach, somewhat akin to the picture environment of LaTeX.

In this workshop, we provide examples and hands-on exercises which span a wide spectrum of use, including basic slides, use of themes, colors, production of handouts, use of dynamic effects, animations, and various graphic primitives. Participants wishing to complete the hands-on exercises should bring a computer to the workshop.

Outline of Beamer and Ti*k*Z Workshop

1. Introduction and motivation
2. Using the beamer package: fundamental level
    (a) Basic layout and processing of a beamer document
    (b) Anatomy of a frame
    (c) Adding content to a frame
    (d) Verbatim text and fragility
3. Exercises with beamer: round 1
4. Using the Ti*k*Z package: fundamental level
    (a) The tikzpicture environment
    (b) Points
    (c) Paths
    (d) Coordinates
    (e) Drawing primitives: rectangles, circles, arcs, ellipses, curves
5. Exercises with Ti*k*Z: round 1
6. Using the beamer package: advanced level
    (a) Adding color to frames
    (b) Fonts and themes
    (c) Multiple columns
    (d) Incremental frame content
    (e) Use of overlays
    (f) Handouts
    (g) Animations
7. Exercises with beamer: round 2
8. Using the Ti*k*Z package: advanced level
    (a) Nodes
    (b) Loops
    (c) Transformations: rotation, scaling, shifting
    (d) Clipping
    (e) Scope
    (f) Plotting
9. Exercises with Ti*k*Z: round 2
10. Free-form explorations with beamer and Ti*k*Z
11. Conclusions, questions and wrap-up

Between Then and Now — A Meandering Memoir

Peter Wilson
Herries Press
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries dot press (at) earthlink dot net

Abstract

I was asked to talk about something interesting — perhaps how I came to develop the memoir class. Following this suggestion the first part is about how I became involved with LaTeX and friends and why the memoir class. To me all this is not particularly interesting as it falls into the personal 'been there, done that' category. What I find more interesting is how the written word has been presented. The second part briefly describes this, starting four millenia ago with Cuneiform and, with a few stops along the way, ending at recent times.

memoir, *n.* a fiction designed to flatter the subject and impress the reader.

With apologies to Ambrose Bierce

We are the inheritors of an ancient tradition, one that goes back for more than four thousand years. It has taken me a long time to start to appreciate it, and had it not been for LaTeX I never would have realised that it was there.

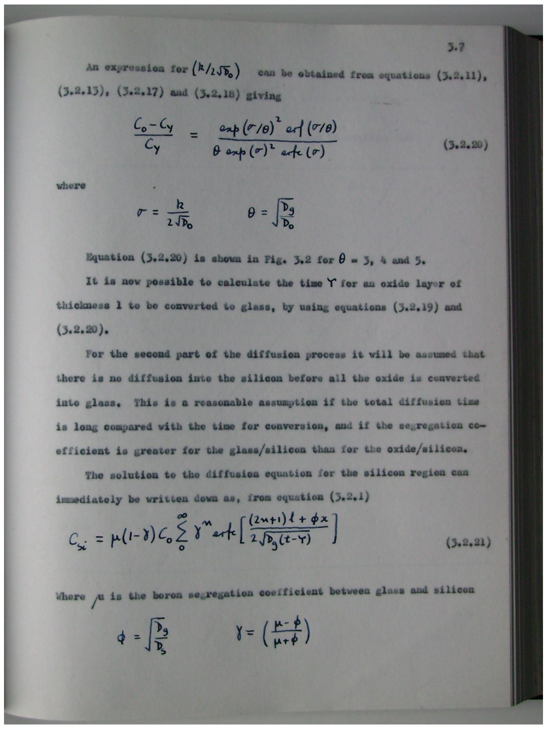
1 Neophyte

In 1973 I had to submit six bound copies of my thesis — one for my supervisor, another for the external examiner, the third for the University library, a fourth for myself, and two spare in case something untoward happened.[1] A very kind secretary typed it for me, one original and five carbon copies. I had to insert all the mathematics by hand (see Figure 1, original size 7½ by 10 inches), and in the last carbon copy that was about all that was legible.

Round about 1980 I came across a computer program called RUNOFF that would do a reasonable job of printing technical reports, provided you didn't mind adding in any mathematics by hand and you could overlook the fact that all we had was a dot matrix printer with too few dots.

Relief came in 1985 when I was introduced to LaTeX; no more hand insertions, justified text, different fonts, a professional look, and no looking back.

I used it for all my internal company reports and paper submission to journals — this was before we could ship documents around electronically so in



3.7

An expression for $(k/2\sqrt{D_o})$ can be obtained from equations (3.2.11), (3.2.13), (3.2.17) and (3.2.18) giving

$$\frac{C_o - C_y}{C_y} = \frac{\exp(\sigma/\theta)^2 \operatorname{erf}(\sigma/\theta)}{\theta \exp(\sigma)^2 \operatorname{erfc}(\sigma)} \qquad (3.2.20)$$

where

$$\sigma = \frac{k}{2\sqrt{D_o}} \qquad \theta = \sqrt{\frac{D_g}{D_o}}$$

Equation (3.2.20) is shown in Fig. 3.2 for $\theta = 3$, 4 and 5.

It is now possible to calculate the time Υ for an oxide layer of thickness l to be converted to glass, by using equations (3.2.19) and (3.2.20).

For the second part of the diffusion process it will be assumed that there is no diffusion into the silicon before all the oxide is converted into glass. This is a reasonable assumption if the total diffusion time is long compared with the time for conversion, and if the segregation coefficient is greater for the glass/silicon than for the oxide/silicon.

The solution to the diffusion equation for the silicon region can immediately be written down as, from equation (3.2.1)

$$C_{si} = \mu(1-\gamma)C_o \sum_0^\infty \gamma^n \operatorname{erfc}\left[\frac{(2n+1)l + \phi x}{2\sqrt{D_g(t-\Upsilon)}}\right] \qquad (3.2.21)$$

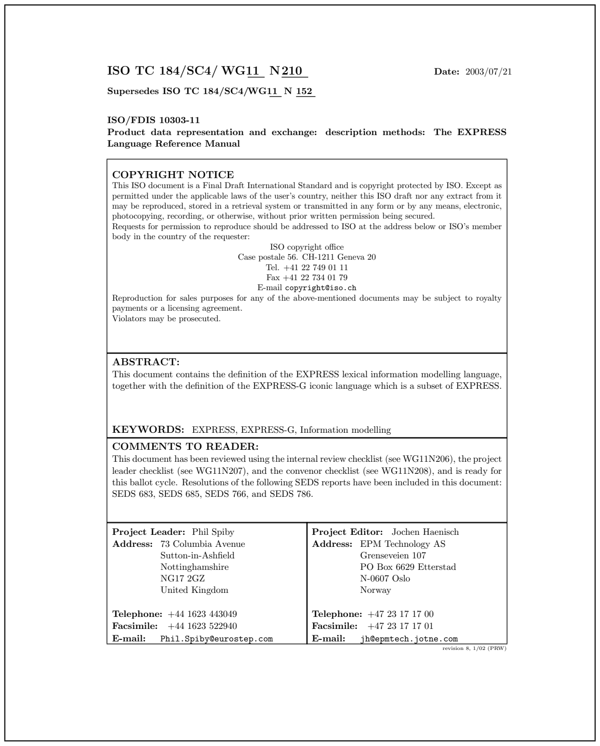
Where μ is the boron segregation coefficient between glass and silicon

$$\phi = \sqrt{\frac{D_g}{D_s}} \qquad \gamma = \left(\frac{\mu-\phi}{\mu+\phi}\right)$$

Herries Collection

Figure 1: Page from PhD thesis, 1973

[1] It did. The binder bound one copy with some pages upside down and others back to front!



ISO TC 184/SC4/ WG11 N210 **Date:** 2003/07/21

Supersedes ISO TC 184/SC4/WG11 N 152

ISO/FDIS 10303-11
Product data representation and exchange: description methods: The EXPRESS Language Reference Manual

COPYRIGHT NOTICE
This ISO document is a Final Draft International Standard and is copyright protected by ISO. Except as permitted under the applicable laws of the user's country, neither this ISO draft nor any extract from it may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, photocopying, recording, or otherwise, without prior written permission being secured.
Requests for permission to reproduce should be addressed to ISO at the address below or ISO's member body in the country of the requester:

ISO copyright office
Case postale 56. CH-1211 Geneva 20
Tel. +41 22 749 01 11
Fax +41 22 734 01 79
E-mail `copyright@iso.ch`

Reproduction for sales purposes for any of the above-mentioned documents may be subject to royalty payments or a licensing agreement.
Violators may be prosecuted.

ABSTRACT:
This document contains the definition of the EXPRESS lexical information modelling language, together with the definition of the EXPRESS-G iconic language which is a subset of EXPRESS.

KEYWORDS: EXPRESS, EXPRESS-G, Information modelling

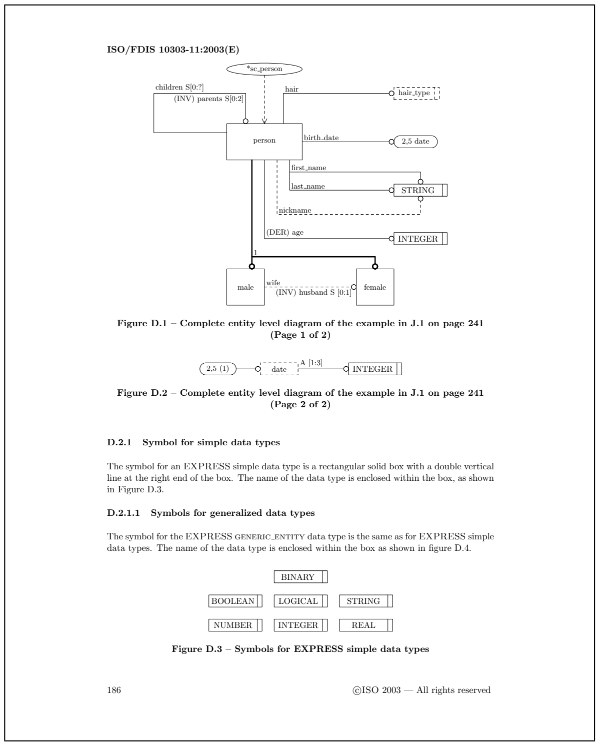
COMMENTS TO READER:
This document has been reviewed using the internal review checklist (see WG11N206), the project leader checklist (see WG11N207), and the convenor checklist (see WG11N208), and is ready for this ballot cycle. Resolutions of the following SEDS reports have been included in this document: SEDS 683, SEDS 685, SEDS 766, and SEDS 786.

Project Leader: Phil Spiby	**Project Editor:** Jochen Haenisch
Address: 73 Columbia Avenue Sutton-in-Ashfield Nottinghamshire NG17 2GZ United Kingdom	**Address:** EPM Technology AS Grenseveien 107 PO Box 6629 Etterstad N-0607 Oslo Norway
Telephone: +44 1623 443049	**Telephone:** +47 23 17 17 00
Facsimile: +44 1623 522940	**Facsimile:** +47 23 17 17 01
E-mail: `Phil.Spiby@eurostep.com`	**E-mail:** `jh@epmtech.jotne.com`

revision 8, 1/02 (PRW)

Herries Collection

Figure 2: Cover sheet for *ISO/FDIS 10303-11:2003*



ISO/FDIS 10303-11:2003(E)

Figure D.1 – Complete entity level diagram of the example in J.1 on page 241 (Page 1 of 2)

Figure D.2 – Complete entity level diagram of the example in J.1 on page 241 (Page 2 of 2)

D.2.1 Symbol for simple data types

The symbol for an EXPRESS simple data type is a rectangular solid box with a double vertical line at the right end of the box. The name of the data type is enclosed within the box, as shown in Figure D.3.

D.2.1.1 Symbols for generalized data types

The symbol for the EXPRESS GENERIC_ENTITY data type is the same as for EXPRESS simple data types. The name of the data type is enclosed within the box as shown in figure D.4.

Figure D.3 – Symbols for EXPRESS simple data types

186 ©ISO 2003 — All rights reserved

Herries Collection

Figure 3: Page 186 from *ISO/FDIS 10303-11:2003*

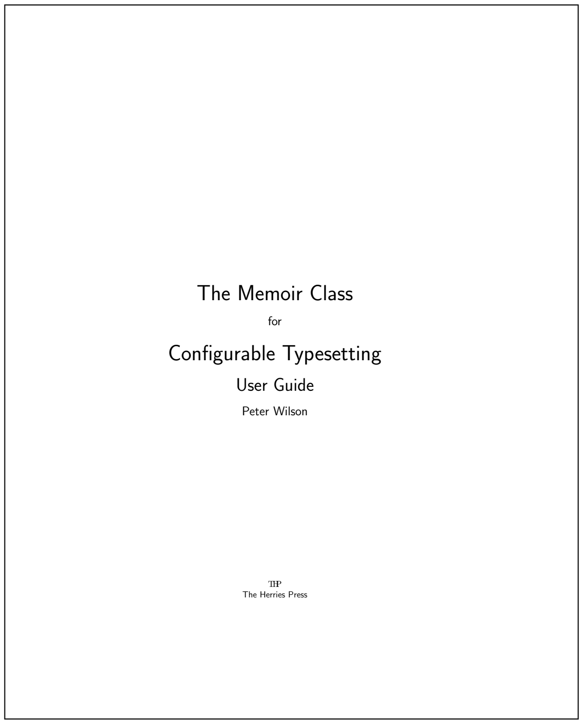
some sense it didn't matter what you used to create them as they would either be copied or retyped.

I became involved in the development of the International Standard 10303 *Industrial automation systems and integration — Product data representation and exchange*, commonly known as STEP, both as the editor and as a technical contributor. ISO had strict rules about the layout of the typewritten documents we would be submitting, which they would then retype for their publishing system, merrily adding typos as they went along. We managed to persuade them to take camera-ready copy so they could eliminate the typo introducing stage. We used LaTeX, of course, as it produced high quality output and, further, it was non-proprietry and we were working in a non-proprietry area.

The draft standard grew to about 2000 pages before we were allowed to split it up into parts to be published separately. Some part editors, for whatever reason, started to use wordprocessors instead of LaTeX. In the meantime I had developed a class for ISO standards in general, and ISO 10303 in particular.

Figure 2 shows the cover sheet for the part of the standard defining the EXPRESS and EXPRESS-G information modeling languages. The cover sheet was implemented using the `picture` environment and all that an author had to do was use a few macros for the text — rather like for the `\maketitle` command. Also as part of my work on STEP I developed the MetaPost `expressg` package for drawing BLA (box, line, annotation) diagrams like the ones in Figure 3.

I eventually moved to the National Institute of Standards and Technology (NIST) in Maryland where the secretariat for STEP was based (Kemmerer, 1999). Someone up the management chain decided that the whole thing should be maintained as SGML documents (or portions thereof) in a database. As they were one of the major supporters of using wordprocessors I was surprised that they chose LaTeX as the publishing system and I spent a considerable time writing a LaTeX to SGML translator, and vice-versa. Unfortunately ISO kept changing their formatting requirements, LaTeX authors kept introducing their own macros, the SGML team kept changing their DTD, and the wordprocessor users were going to be involved at some indefinite date in the future. The experience made me really appreciative of Eitan Gurari's `TeX4ht` (Gurari, 2007). I left before any document made it through the system, which I think has died the death it deserved.



The Memoir Class

for

Configurable Typesetting

User Guide

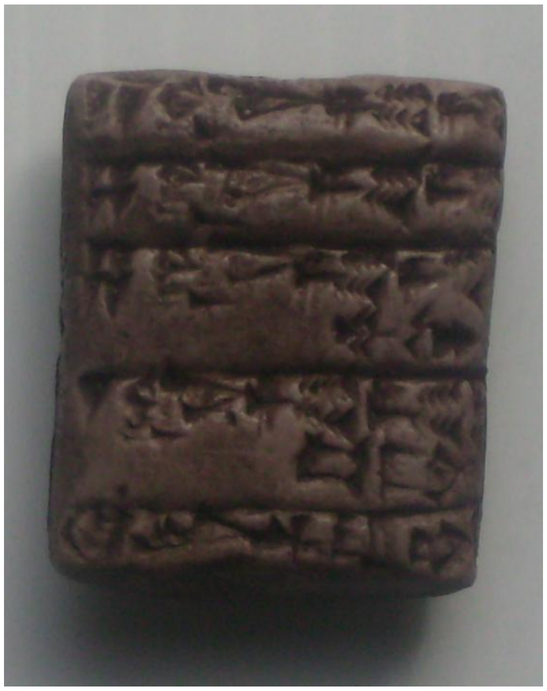
Peter Wilson

TIP

The Herries Press

Herries Collection

Figure 4: Title page of the memoir class user manual



Herries Collection

Figure 5: Sumerian cuneiform tablet, circa 2112–2004 BC

Some of the documents had got up to 1200 pages which caused enormous difficulties to the poor souls who had to use 'the' wordprocessor.

This led me on to the development of my LaTeX memoir class. I didn't want to be bitten by the ISO experience again, so I felt that a class that would let me change the document formatting easily without having to delve into its innards would be very useful. I had written a few packages that helped in formatting bits and pieces and decided to incorporate them into the class. Then there were other packages that I quite often used and integrating those, or their functionality, seemed reasonable, thus ensuring that they would all work well together. Then, like Topsy, it 'just growed'. Now it encompasses the functionality of more than 30 popular packages.

Putting everything together got me started on wondering how a document should be put together. This led to a long trail. One portion was trying to get a better idea about the typographer's craft. And as typographers deal with letter forms that led me to the history of the alphabet and the story of the letter forms that we use now.

2 Early Writing

Writing was invented in ancient Mesopotamia, an area which roughly corresponds to modern day Iraq. The earliest recorded writings are by the Sumerians from around 3300 BC, who used pointed sticks or reeds or to impress marks into wet clay tablets that were subsequently dried. The result is what we call Cuneiform.[2] We are still in the business of recording writing.

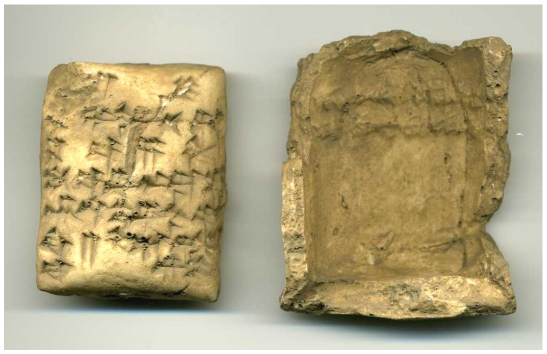
As the city states arose and society became more complex writing was necessary to help the bureaucrats and merchants keep track of things and so that tax collectors and others could go about their business in a fair manner.

Figure 5 shows a replica of a Sumerian cuneiform tablet dating back to between 2112 and 2004 BC, from the Third Dynasty of Ur, about the time of the Biblical Abraham. The original is 1¼ by 1¼ by ⅜ inches. The scribes would write on the front and the back of a tablet, and sometimes on the sides as well.

Cuneiform writing was adopted by the Babylonians even though their language was not like Sumerian, and Figure 6 shows a replica of a Babylonian tablet and its clay envelope, from about 1790 BC. The tablet is 1½ by 1¾ by ½ inches.

The package is a receipt for an amount of grain sufficient for one man for 6 months. The same text is on the outside of the clay envelope as on the tablet;

[2] From the Latin *cuneus* meaning wedge.



Herries Collection

Figure 6: Babylonian cuneiform tablet and envelope, circa 1790 BC

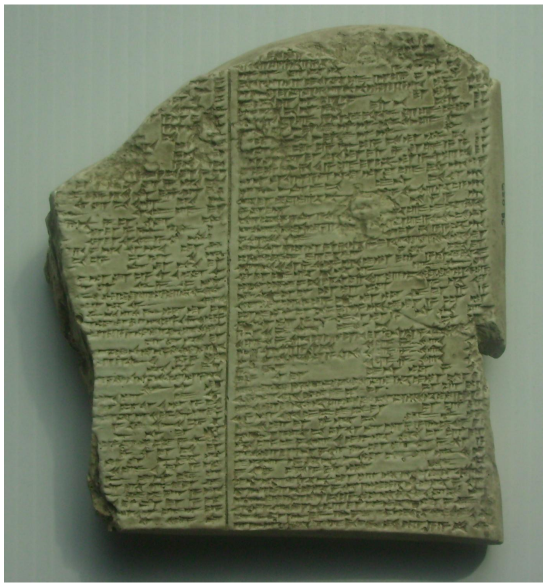
if there was doubt about the external message then the envelope could be broken and the external and internal messages compared.

Writing evolved from that needed for simple record keeping to, for instance, be able to write people's names or to record the majestic deeds of the ruler. The earliest literary tablets containing parts of the *Epic of Gilgamesh*, which is by far the world's oldest epic, date back to about 2100 BC. The Gilgamesh story has been pieced together from thousands of pieces of broken cuneiform tablets (George, 2000). Figure 7 shows a replica of one of the many tablets found by Sir Austen Henry Layard in 1850–53 in the ruins of King Ashurbanipal's library at Ninevah which was destroyed in 612 BC. This particular one contains much of what is called 'Tablet 11' of the Epic which includes the best preserved story of a Deluge[3] or Flood, well pre-dating the Biblical version which was written around the 9th century BC.

The tablet, which is 5 3/4 by 5 3/4 by 1 1/4inches, was first translated in 1872 by George Smith working at the British Museum. Wallis Budge (Budge, 1925) described the event like this:

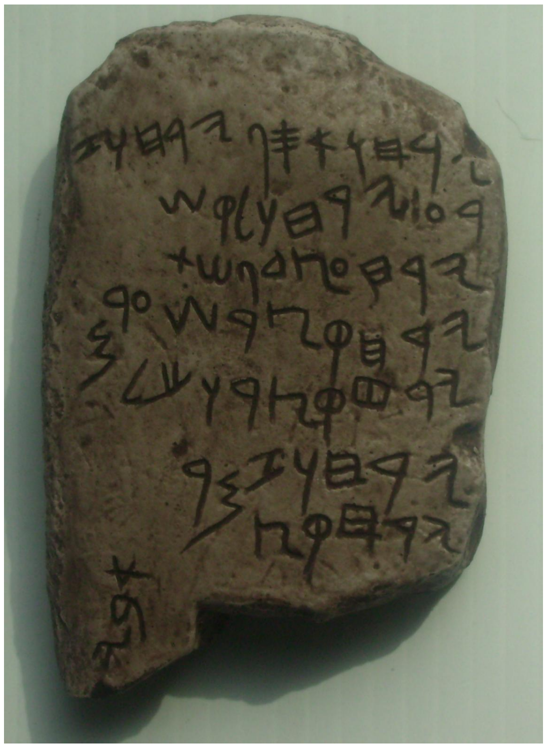
> Smith took the tablet and began to read over the lines which Ready [the conservator who had cleaned the tablet] had brought to light; and when he saw that they contained the portion of the legend he had hoped to find there, he said "I am the first man to read that after two thousand years of oblivion". Setting the tablet on the table, he jumped up and rushed about the room in great excitement, and, to the astonishment of those present, began to undress himself!

[3] There is evidence that the catastrophe occurred around 7500 BC when the Black Sea's water level rose by 400 feet during the course of about a year (Ryan and Pitman, 2000).



Herries Collection

Figure 7: Epic of Gilgamesh (tablet 11), circa 650 BC



Herries Collection

Figure 8: Gezer Calendar, circa 925 BC

Figure 8 shows a replica of a soft piece of limestone rock from around 925 BC. This was found in 1908 by R.A.S. Macalister at Tell el-Jazari (the historic city of Gezer) about 20 miles NW of Jerusalem. The tablet is 3 by 4½ by ⅝ inches. The text is written right to left in what some say is in a Proto-Hebrew script while others (Healey, 1990, p. 30) say it is in the Phoenician[4] script. It is a calendar of agricultural tasks and seasons. The tablet's inscription is:

𐤉𐤓𐤇𐤅𐤀𐤎𐤐𐤉𐤓𐤇𐤅𐤆
𐤓𐤏𐤉𐤓𐤇𐤅𐤋𐤒𐤔
𐤉𐤓𐤇𐤏𐤑𐤀𐤐𐤔𐤕
𐤉𐤓𐤇𐤒𐤑𐤓𐤔𐤏𐤓𐤌
𐤉𐤓𐤇𐤒𐤑𐤓𐤅𐤊𐤋
𐤉𐤓𐤇𐤅𐤆𐤌𐤓
𐤉𐤓𐤇𐤒𐤑
𐤀𐤁𐤉

In the following transliteration I have added inter-word spaces that are not in the original. Also, the first two lines on the tablet contain the first three lines of the calendrical information.

z wḥry ps' wḥry
šql wḥry ʿr
tšp 'ṣʿ ḥry
mrʿš rṣq ḥry
lkw rṣq ḥry
rmz wḥry
ṣq ḥry
yb'

And a translation is:

Two months are [olive] harvest,
Two months are planting [grain],
Two months are late planting;
One month is hoeing up flax,
One month is harvest of barley,
One month is harvest and feasting;
Two months are vine tending,
One month is summer fruit.

It is signed in the bottom lefthand corner with the name 'Abijah'.

3 Manuscripts

Our modern alphabets date back to around 1600 BC, and in particular to the Phoenician script and alphabet. By various routes this spread out from the Middle East, changing as time went on to accommodate different languages (Wilson, 2005).

Throughout the ages scribes have always taken great care in the appearance of their work, especially with religious works.

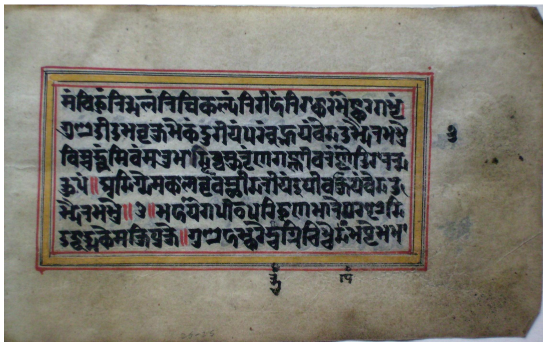

Herries Collection

Figure 9: Leaf from a copy of the *Bhagavad Gita*, Kashmir, circa 1800

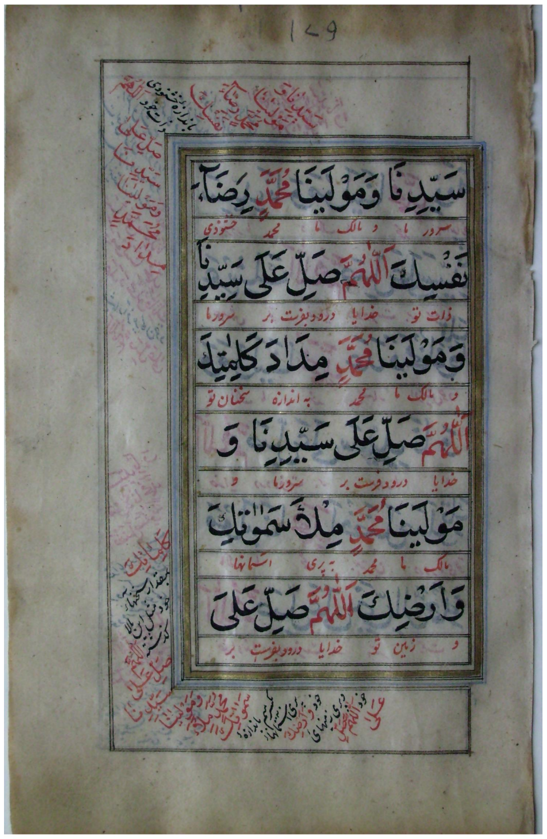
Figure 9 is a leaf from a Kashmiri copy of the *Bhagavad Gita*. The original is 5½ by 3¼ inches overall in a black Devanagari script surrounded by a yellow, red and blue border, on burnished paper. It dates to the late 18th or early 19th century. The *Bhagavad Gita* (The Song of the Lord) is a poem forming part of the Hindu epic, the *Mahabharata* which dates back to the first millenium BC, consisting of an eve of battle dialogue between the warrior prince Arjuna and Lord Krishna (in the person of his charioteer).

Figure 10 shows a leaf from a copy of *Delail al-Khayrat* — the book of *Blessings on the Prophet* composed by Muhammad ibn Sulayman al-Jazuli (d. 1465). The original of the leaf is 4 11/16 by 7⅝ inches. It was written about 1690 by Mohammed Azeem for Nawab Sadullah Khan who was the Prime Minister of the Moghul emperor Shah Jehan — the builder of the Taj Mahal. The Arabic text is black with an interlinear Persian translation in red and a commentary in the margins around the main text. The border is in gold and a light blue.

Figure 11 is a leaf from an Indian copy of the *Koran*. The original is 4¾ by 7 11/16 inches overall. The Arabic script is in black ink surrounded by a main border, 3⅛ by 5⅜ inches, in gold and blue. The interlinear decoration is in gold leaf.

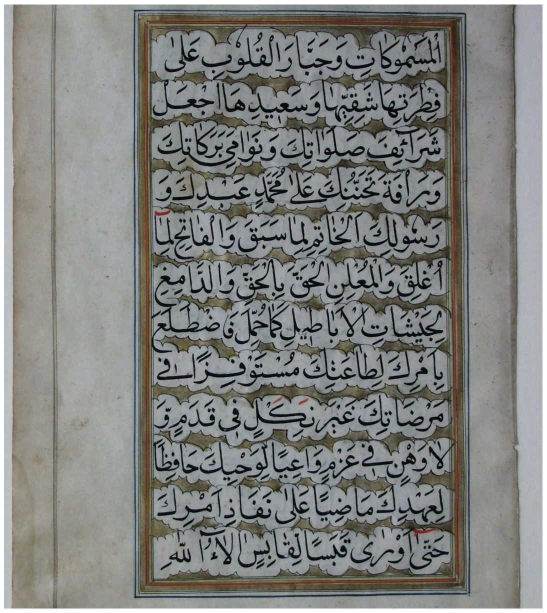
Arabic texts are famous for their calligraphy but there are other cultures as well where calligraphy is an esteemed art. Figure 12 is number 66 from the series of Japanese woodblock prints *Ogura*

[4] To me it looks remarkably like Phoenician.



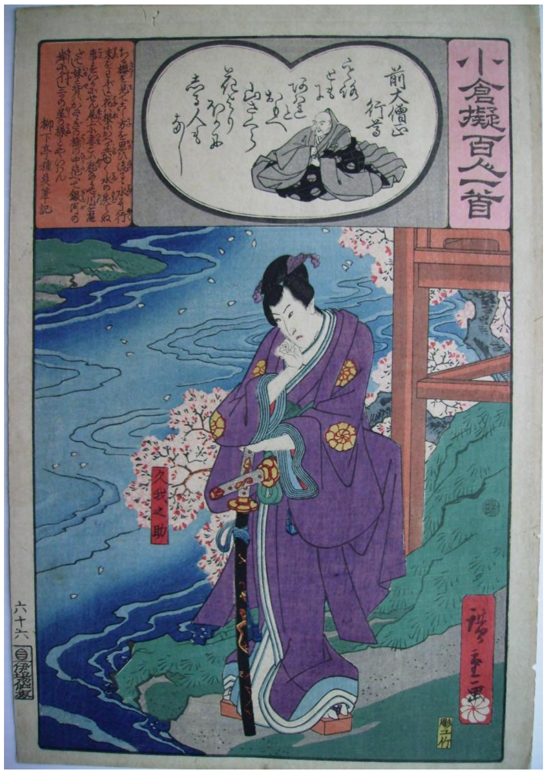
Herries Collection

Figure 10: Leaf from a copy of *Delail al-Khayrat* Arabic/Persian, circa 1690



Herries Collection

Figure 11: Leaf from a *Koran*, India



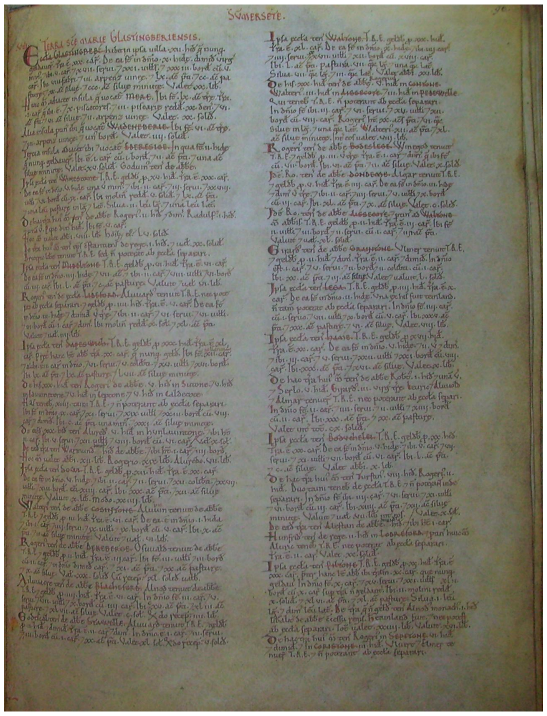
Herries Collection

Figure 12: *Ogura Imitation of 100 Poets* no. 66, by Hiroshige, circa1846

Imitation of 100 Poets illustrating a famous anthology of 100 poems by 100 poets that was assembled by the poet Fujiwara no Teiko in 1235. The woodblock print publisher Iba-ya Sensburō commissioned three artists — Kuniyoshi, Hiroshige and Kunisada — to produce the prints in the series which were published between 1845 and 1847. This one by Hiroshige illustrates a poem by Daisōjō Gyōson (1055–1136). The poem reads:

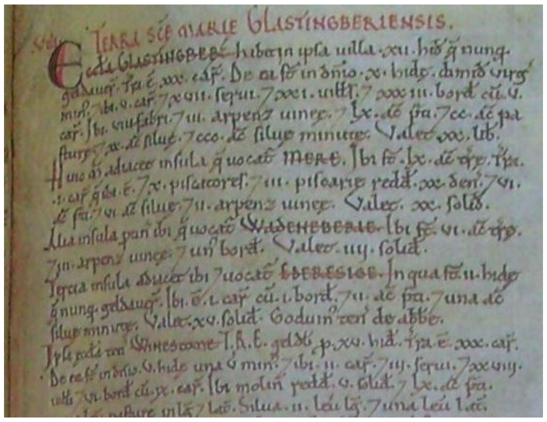
Morotomi ni	Let us, each for each
Aware to omoe	Pitying, hold tender thought,
Yamazakura	Mountain cherry flower!
Hana yori hoka ni	Other than thee, lonely flower,
Shiru hito mo nashi	There is none I know as friend.

The main illustration shows a contemplative Kuganosoke (the hero of the play *Imoseyama*) outside a pavilion on the bank of a river. The title of the series is at the top right in large kanji characters and at the top left is a description of the main illustration in smaller kanji. The lozenge contains a portrait of the poet and the poem itself in a highly calligraphic style. The original is in the standard *oban* size of approximately 9½ by 14 inches. To



Herries Collection

Figure 13: Page from the Domesday Book, England 1086



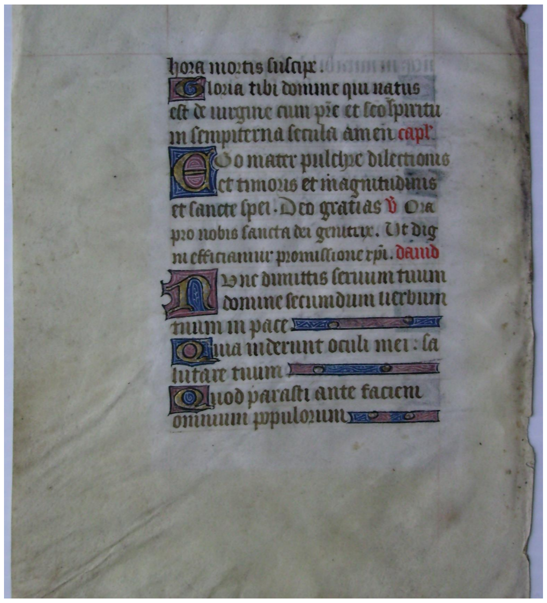
TERRA SCĒ MARIE GLASTINGBERIENSIS.
Eccla GLASTINGBERĒ habet in ipsa uilla .xii. hid q nung.
geldauer. Tra ē .xxx. car. De ea sunt in dnio .x. hide. dimid uirg
...
ire uel Isfulam. 7 in arpent uinee. Valet. xx. solid.
Alia insula pertin ibi q uocat WADENEBERIE. Ibi sunt .vi. ac tre.
7 in arpent uinee. 7 un bord. Valet .iiij. solid.
Tercia insula adiacet ibi 7 uocat ERERESIGE. In qua sunt .ii. hide
q nung. geldauer. Ibi ē .i. car cu .i. bord. 7 ii. ac pa. 7 una ac
silue minute. Valet .xv. solid. Godum ten de abbe.
Ipsa eccla ten WINESCOME. T.R.E. geldb p .xx. hid. Tra ē .xxx. car.

Herries Collection

Figure 14: Domesday Book (enlarged), England 1086

print it there would have been one carved woodblock for each colour in the picture, with the picture being gradually built up one colour at a time. Registration between the individual blocks and with the paper is critical. Even seemingly simple pictures could require ten or more blocks.

Coming closer to home, European books were mainly written in Latin. Literacy was essentially confined to the Church, the Papal See and monasteries in particular, and to clerks in noble courts.



hora mortis suscipe.
Gloria tibi domine qui natus
est de uirgine cum patre et sancto spiritu
in sempiterna secula amen. capl'
Ego mater pulchre dilectionis
et timoris et magnitudinis
et sancte spei. Deo gratias. V. Ora
pro nobis sancta dei genitrix. Ut dig
ni efficiamur promissione xpi. dauid
Nunc dimittis seruum tuum
domine secundum uerbum
tuum in pace.
Quia uiderunt oculi mei: sa
lutare tuum.
Quod parasti ante faciem
omnium populorum.

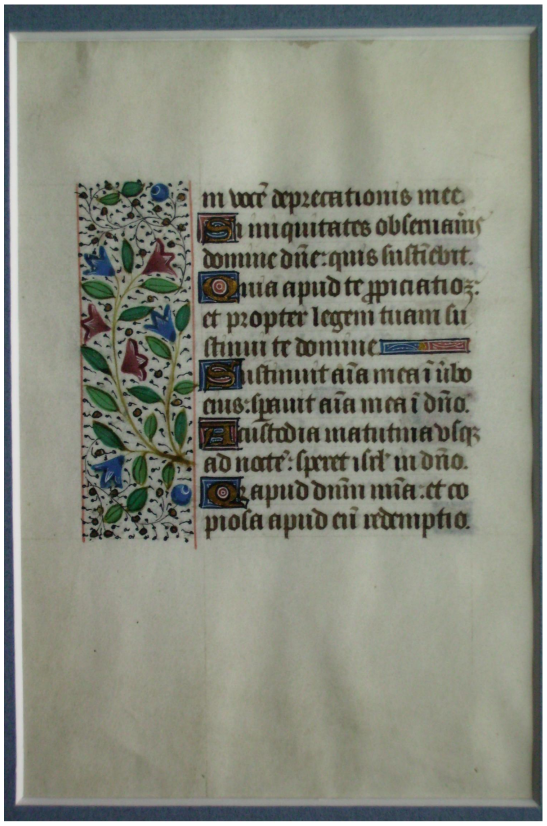
Herries Collection

Figure 15: Book of Hours, France, circa 1445

Most works that have survived were religious in nature but rulers required administrative records of all kinds. One of the most famous is the *Domesday Book* that William the Conqueror (circa 1028–1087) ordered to be compiled in 1086. It is a survey of the newly conquered England, from Yorkshire to the South Coast, arranged by county, and listing all the landowners and the worth and taxes paid on their properties (Hinde, 1985). Figure 13 shows one page from the book that starts with information about Glastonbury in the County of Somerset. The text is in Latin, in two columns of 44 lines each, written in a Carolingian minuscule script. An enlarged view of the top of the left column is shown in Figure 14. Some headings are in red, but the text is not without errors.

Many beautiful manuscripts were written by scribes in monasteries, some for use by the Church and others for rich patrons. Many of the latter are elaborately decorated and illuminated.

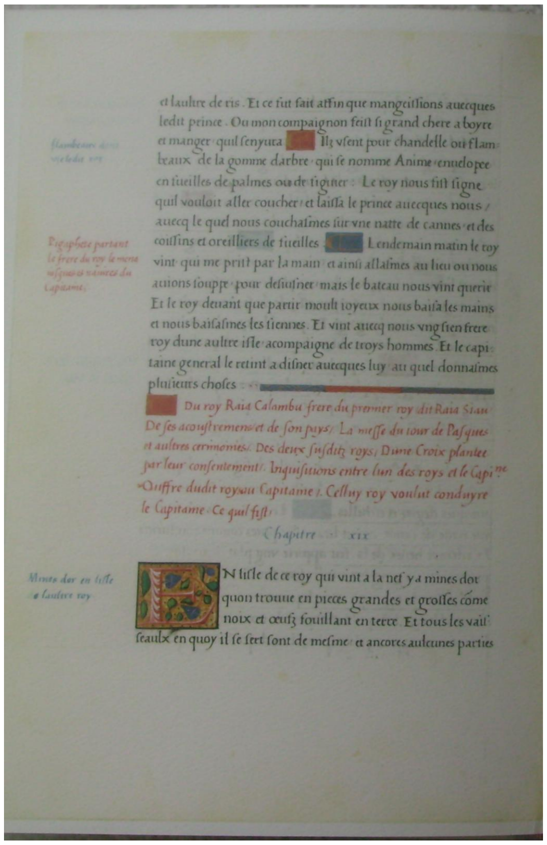
Figure 15 is a leaf (verso) from a Benedictine Book of Hours produced in France around 1445. The original vellum leaf is 5¾ by 8 inches. The Latin text, 3 by 4¼ inches, is in the Gothic Textura Quadrata bookhand in a light brown ink. The versal initials are in liquid gold on grounds of red and blue with white tracery. The paragraph endings use the same style.



in voce deprecationis mee.
Si iniquitates obſeruaueris
domine dñe: quis ſuſtinebit.
Quia apud te ꝓpiciatio ẽ:
et propter legem tuam ſu
ſtinui te domine.
Suſtinuit aĩa mea ĩ ũbo
eius: ſpauit aĩa mea ĩ dño.
A cuſtodia matutina uſqꝫ
ad noctẽ: ſperet iſrl' in dño.
Quia apud dñm mĩa: et co
pioſa apud eũ redemptio.

Herries Collection

Figure 16: Book of Hours, France, circa 1450



flambeaux dont vſe ledit roy

Pigaphete partant le frere du roy le mena iuſques es nauires du Capitaine;

et laultre de ris. Et ce fut fait affin que mangeiſſions auecques
ledit prince. Ou mon compaignon feiſt ſi grand chere a boyre
et manger quil ſenyura Ilz vſent pour chandelle ou flam
beaux de la gomme darbre qui ſe nomme Anime enuelopee
en fueilles de palmes ou de figuier: Le roy nous fiſt ſigne
quil vouloit aller coucher et laiſſa le prince auecques nous/
auecq le quel nous couchaſmes ſur vne natte de cannes et des
couſſins et oreilliers de fueilles Lendemain matin le roy
vint qui me priſt par la main et ainſi allaſmes au lieu ou nous
auions ſouppe pour deſiuſner mais le bateau nous vint querir
Et le roy deuant que partir moult ioyeux nous baiſa les mains
et nous baiſaſmes les ſiennes. Et vint auecq nous vng ſien frere
roy dune aultre iſle acompaigne de troys hommes. Et le capi
taine general le retint a diſner auecques luy au quel donnaſmes
pluſieurs choſes:

Du roy Raia Calambu frere du premier roy dit Raia Siau
De ſes acouſtremens et de ſon pays/. La meſſe du iour de Paſques
et aultres cerimonies/. Des deux ſuſditz roys/. Dune Croix plantee
par leur conſentement/. Inquiſitions entre lun des roys et le Capi^ne^
Offre dudit roy au Capitaine/. Celluy roy voulut conduyre
le Capitaine. Ce quil fiſt.

Chapitre xix

Mines dor en liſle a laultre roy

EN liſle de ce roy qui vint a la nef y a mines dor
quon trouue en pieces grandes et groſſes cõme
noix et œufz fouillant en terre. Et tous les vaiſ
ſeaulx en quoy il ſe ſert ſont de meſme et ancores aulcunes parties

Herries Collection

Figure 17: Magellan's Voyage Around the World, 1524

A more decorative example is shown in Figure 16 which is a leaf from a Book of Hours produced in France, perhaps at Rheims, around 1450 or maybe a little later. The original vellum leaf is 3¾ by 5⅝ inches. The Latin text, 2½ by 2⅞ inches, is in the Gothic Textura Quadrata bookhand in a dark brown ink. The versals are in liquid gold with additional decoration in red and blue. The floriated decoration uses green as well as the other colours.

In a different vein, and a different script, Figure 17 is a page from Antonio Pigafetta's account of Magellan's circumnavigation (1519–1522), beautifully written in a humanist bookhand. There are four surviving manuscripts, one in the Venetian dialect of Italian, and three in French. Pigafetta probably completed his work in 1524 and it would then have been copied out by professional scribes. The manuscript now at the Bernicke Library at Yale University consists of 103 vellum leaves, measuring 7½ by 11¼ inches, with 27 lines to a page (Pigafetta, 1969). The page in the illustration shows the end of chapter XVIII, a summary (in red) of the next chapter, and the title and first four lines of chapter XIX. The marginal notes, in red and blue, are a summary of the corresponding paragraphs in the main text.

4 Printed books

In the West, printing using moveable type was invented by Johannes Gutenberg around 1440–1450, although the earliest printed book known is a 9th century Chinese woodblock printing of the *Diamond Sutra*. Gutenberg had to experiment to determine the formula for a suitable ink and also to discover a good metal alloy for the type itself. He came up with lead to which he added antimony for strength and hardness and tin for toughness.[5]

In order to be successful in the market, Gutenberg had to produce books that equaled those produced by the scribes, except that they did not necessarily have to be decorated so lavishly. The scribes, though, used many ligatures and other techniques to try and have non-ragged text blocks. To compete with them Gutenberg's font for his 42-line Bible,

[5] This is still the basis for type today; Monotype casting machines use lead with 15–24% antimony and 6–12% tin.

published around 1455, consisted of some 290 characters though all the text is in Latin which requires a basic character set of only forty letters — twenty lowercase letters and twenty caps — and some punctuation marks (Thorpe, 1999).

The 42-line Bible is set in two columns of 42 lines each. It is believed that about 135 copies were printed on paper and 40 on vellum. The page size was 12 by 16 1/2 inches and it is estimated that more than five thousand calfskins were required for the vellum copies.

The *Nuremberg Chronicle* was published in 1493 in Nuremberg and was the first book to combine text with illustrations that illuminated the words (instead of using randomly selected woodblock engravings that happened to be at hand). As was usual then the book did not have a title page: Latin scholars call it the *Liber Chronicarum* and in German it is called *Die Schedelsche Weltronik* after its author Hartmann Schedel. The book was printed and published by Anton Keberger with a print run of about 1500 Latin copies and 900 German ones. Around 400 Latin and 300 German copies have survived.

There are 1809 woodcut illustrations printed from 645 originals, so many were used multiple times, usually portraits. For example a single woodcut was used to represent Alcuin, Cato, Dante, Paris and Plutarch on different pages. The woodcuts were created by Michael Wolgemut and Hans Pleydenwurff, with perhaps one or two by Albrect Dürer who was apprenticed to Wolgemut at the time.

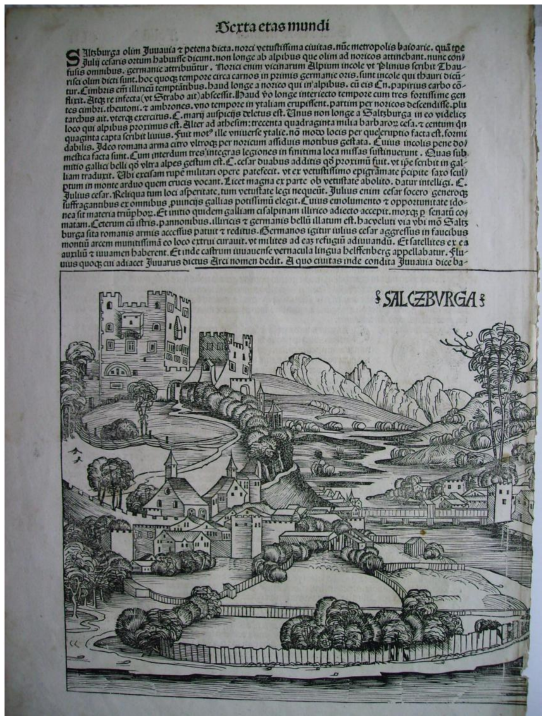
The pages are large, 12 by 17 1/2 inches. Views of cities were printed as a double spread. Spaces were left in the text for the woodcuts; in the more luxury volumes the woodcuts were hand coloured.

The Chronicle divides the history of the world into seven ages:

1. Creation to the Deluge
2. ends with the birth of Abraham
3. ends with the reign of King David
4. ends with the Babylonian captivity
5. ends with the Incarnations of Jesus
6. from the birth of Christ to the end of the world
7. the age of the Anti-Christ
8. the Last Judgement

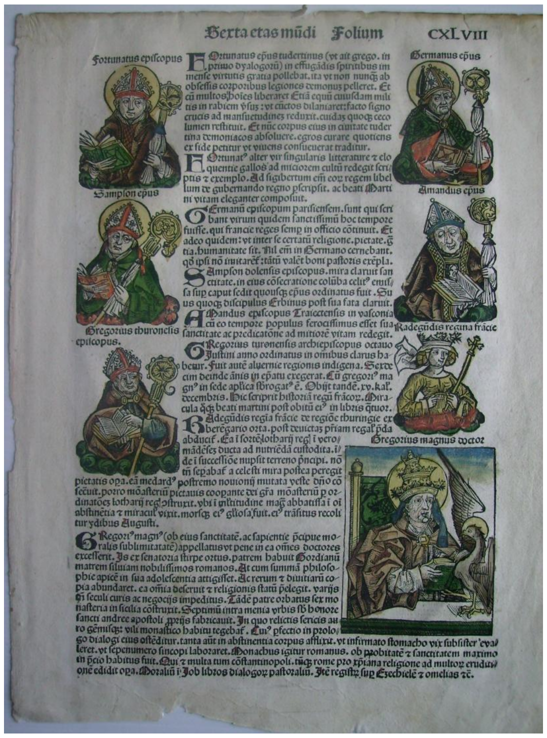
Beloit College has an extensive web site (`http://www.beloit.edu/~nurember`) devoted to their copy of the *Nuremberg Chronicle* which has coloured illustrations.

Figure 18 is Folio CLII (verso) from the *Nuremberg Chronicle*. At the bottom is half of a double spread picture of Salzburg (the other half is on the recto of Folio CLIII).



Herries Collection

Figure 18: *Nuremberg Chronicle* (1493): Folio CLIIv



Herries Collection

Figure 19: *Nuremberg Chronicle* (1493): Folio CXLVIIIr

Figure 19 is Folio CXLVIII (recto) from the *Nuremberg Chronicle*. The hand coloured pictures are of various ecclesiastical personages and at the lower right a queen (Radegudis regina fracie) and a doctor (Gregorius magnus doctor). The original for this picture is 12 by 15 1/2 inches (over the years 2 inches have disappeared from the lower margin).

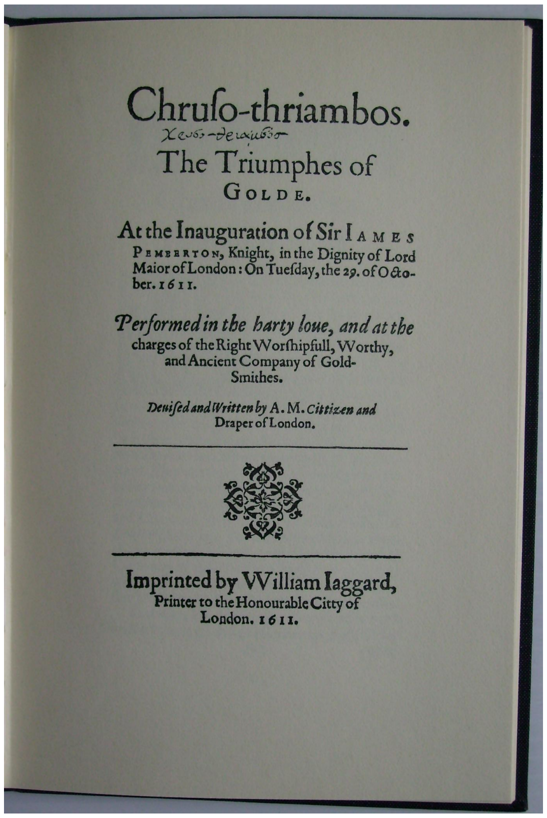
Books had, of course, been made and sold long before Gutenberg. In London, for example, the publishing trade was regulated by the Guild of Stationers which was incorporated in 1403. At that time stationers were either booksellers who sold manuscripts that they had copied; or illuminators who illustrated and decorated manuscripts; or bookbinders who bound manuscripts. Stationers would also sell the materials that they used. Unless you were a member of the Guild you could do none of these things.

Following Gutenberg, printing rapidly spread out over much of Europe. In England, for example, Caxton set up his shop in 1476, Theoderic Rood was printing in Oxford between 1478 and 1485, and John Sieberch in Cambridge in 1520. The Stationers Guild received a royal charter in 1557 and was responsible for regulating the printing industry over all the country, which meant that they had a monopoly on book production — once a member asserted ownership of a text (or 'copy') no other member could publish it. This is the origin of the term 'copyright'.

In Germany books were usually printed in a gothic type but the rest of Europe moved to types based on the humanist tradition that had been maintained in Italy.

Figure 20 is the title page of a reprint of *Chruso-thriambos* or *The Triumphs of Golde* by Anthony Mundy, published in 1611. The original is 6 by 9 inches. The pageant *Chruso-thriambos* was written and produced at the request and charge of the Worshipful Company of Goldsmiths in honour of Sir James Pemberton, a goldsmith, the newly elected Lord Mayor of London. Page 8 (numbered 26 in the book containing the reprint) from the body is shown in Figure 21.

Ambroise Paré (1510–1590) served as the official royal surgeon for kings Henry II, Francis II, Charles IX and Henry III of France, and did much to advance medical procedures, particularly surgery. A page from the first English translation of his major work, by Thomas Johnson and printed in 1634 by Th. Cotes and R. Young, is shown in Figure 22. The original is 8 by 12 1/2 inches and is set using an Oldstyle type, possibly Garamond. Paré's major contributions included the abandonment of boiling



Chruſo-thriambos.
Χρυσο-θριαμβος
The Triumphes of
GOLDE.

At the Inauguration of Sir IAMES PEMBERTON, Knight, in the Dignity of Lord Maior of London: On Tueſday, the 29. of October. 1611.

Performed in the harty loue, and at the charges of the Right Worſhipfull, Worthy, and Ancient Company of Gold-Smithes.

Deuiſed and Written by A. M. *Cittizen and* Draper of London.

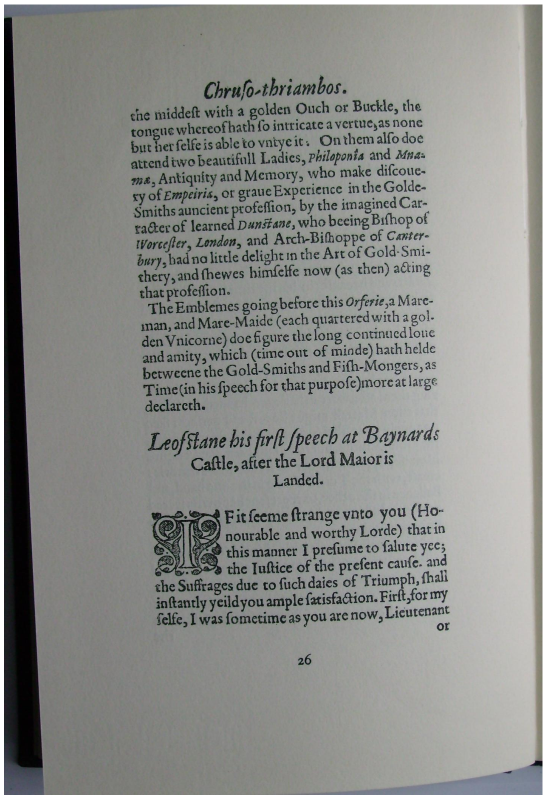
Imprinted by William Iaggard, Printer to the Honourable Citty of London. 1611.

Herries Collection

Figure 20: *Chruso-thriambos:* Title page, 1611

oil for the treatment of gunshot wounds in favour of egg yolk, oil of roses and turpentine which worked far better. He also introduced the use of ligatures instead of cauterisation during amputations, and was especially adept at devising ingenious and efficient artificial limbs and new surgical instruments. All in all he seems to have been afflicted with a great deal of commonsense.

A book, *Eygentliche Beschreibung Aller ände auff Erden* about 16th century trades, was published in Frankfurt in 1568 which included several woodcuts from drawings by Jost Amman. Figure 23 is one of these showing a printing shop. The two men in the background are setting type, taking the characters from the type cases in front of them. The men in the foreground are operating the printing press. The one on the left is removing a sheet of paper that has just been printed and the one on the right is using two circular pads to ink the type for the next sheet. A fresh sheet of paper will replace the one being removed. The flap at the left, with the cutouts, will be folded down to hold the paper in place, then the assembly folded over to lie on top of the type. The final assembly is slid into the press,



Chruſo-thriambos.

the middeſt with a golden Ouch or Buckle, the tongue whereof hath ſo intricate a vertue, as none but her ſelfe is able to vntye it. On them alſo doe attend two beautifull Ladies, *Philoponia* and *Mnama*, Antiquity and Memory, who make diſcouery of *Empeiria*, or graue Experience in the Golde-Smiths auncient profeſſion, by the imagined Caracter of learned *Dunſtane*, who beeing Biſhop of *Worceſter*, *London*, and Arch-Biſhoppe of *Canterbury*, had no little delight in the Art of Gold-Smithery, and ſhewes himſelfe now (as then) acting that profeſſion.

The Emblemes going before this *Orferie*, a Mare-man, and Mare-Maide (each quartered with a golden Vnicorne) doe figure the long continued loue and amity, which (time out of minde) hath helde betweene the Gold-Smiths and Fiſh-Mongers, as Time (in his ſpeech for that purpoſe) more at large declareth.

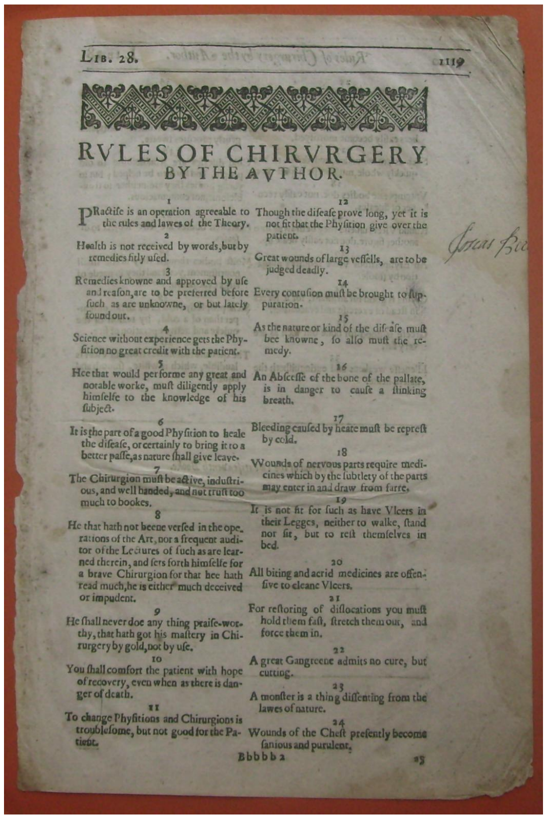
Leofſtane his firſt ſpeech at Baynards Caſtle, after the Lord Maior is Landed.

IF it ſeeme ſtrange vnto you (Honourable and worthy Lorde) that in this manner I preſume to ſalute yee; the Iuſtice of the preſent cauſe. and the Suffrages due to ſuch daies of Triumph, ſhall inſtantly yeild you ample ſatisfaction. Firſt, for my ſelfe, I was ſometime as you are now, Lieutenant or

26

Herries Collection

Figure 21: *Chruso-thriambos:* page 8, 1611



LIB. 28. 1119

RVLES OF CHIRVRGERY
BY THE AVTHOR.

1
PRactiſe is an operation agreeable to the rules and lawes of the Theory.

2
Health is not received by words, but by remedies fitly uſed.

3
Remedies knowne and approved by uſe and reaſon, are to be preferred before ſuch as are unknowne, or but lately found out.

4
Science without experience gets the Phyſition no great credit with the patient.

5
Hee that would performe any great and notable worke, muſt diligently apply himſelfe to the knowledge of his ſubject.

6
It is the part of a good Phyſition to heale the diſeaſe, or certainly to bring it to a better paſſe, as nature ſhall give leave.

7
The Chirurgion muſt be active, induſtrious, and well handed, and not truſt too much to bookes.

8
He that hath not beene verſed in the operations of the Art, nor a frequent auditor of the Lectures of ſuch as are learned therein, and ſets forth himſelfe for a brave Chirurgion for that hee hath read much, he is either much deceived or impudent.

9
He ſhall never doe any thing praiſe-worthy, that hath got his maſtery in Chirurgery by gold, not by uſe.

10
You ſhall comfort the patient with hope of recovery, even when as there is danger of death.

11
To change Phyſitions and Chirurgions is troubleſome, but not good for the Patient.

12
Though the diſeaſe prove long, yet it is not fit that the Phyſition give over the patient.

13
Great wounds of large veſſells, are to be judged deadly.

14
Every contuſion muſt be brought to ſuppuration.

15
As the nature or kind of the diſeaſe muſt bee knowne, ſo alſo muſt the remedy.

16
An Abſceſſe of the bone of the pallate, is in danger to cauſe a ſtinking breath.

17
Bleeding cauſed by heate muſt be repreſt by cold.

18
Wounds of nervous parts require medicines which by the ſubtlety of the parts may enter in and draw from farre.

19
It is not fit for ſuch as have Vlcers in their Legges, neither to walke, ſtand nor ſit, but to reſt themſelves in bed.

20
All biting and acrid medicines are offenſive to cleane Vlcers.

21
For reſtoring of diſlocations you muſt hold them faſt, ſtretch them out, and force them in.

22
A great Gangreene admits no cure, but cutting.

23
A monſter is a thing diſſenting from the lawes of nature.

24
Wounds of the Cheſt preſently become ſanious and purulent.

Bbbbb 2 25

Herries Collection

Figure 22: Page from the first English translation of Ambroise Paré's works, 1634

the lever pulled to press the paper onto the type, the assembly slid out from the press and the printed page removed.

Figure 24 is another of the woodcuts, this time showing a book bindery. In the background there is a sewing frame with a book and the man is sewing the sheets together. In the foreground there is a book in a lying press at the left and at the right the man is trimming the edges of the pages in a sewn book, which is in another lying press, before the covers will be put on. In those days books were often sold without covers so that clients could select the kind they wanted.

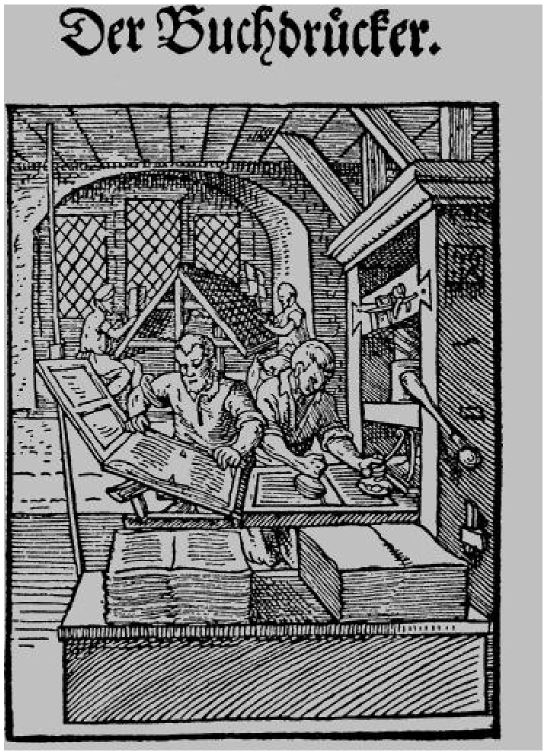
Little changed in the manufacture of books until the middle of the 19th century when some of the processes began to be mechanized (Chappell and Bringhurst, 1999). Figure 25 is is a reconstructed 18th century print shop in Williamsburg, Virgina, 2007. James Mosley, who for 42 years was the Librarian at the St Bride Printing Library in London, said that it was 'the most perfect and accurate working reconstruction of an 18th-century office' that he had ever seen (Mosley, 2003). The paper holder is at the left, the type in the center and the press itself, a so-called *English Common Press*, at the right. The man is preparing to ink the type.

Also at Williamsburg is a reconstructed 18th century bindery, shown in Figure 26. Two sewing frames are in the foreground and a large standing press is in the semi-background.

In the days of the American Colonies, printing was not encouraged. Sir William Berkeley, who was the governer of Viginia for 1642 to 1652 and again from 1660 to 1677, spoke for many officials when he said,

> But, I thank God, there are no free schools nor printing, and I hope we shall not have these for hundreds of years; for learning has brought disobedience, and heresy, and sects into the world, and printing has divulged them, and libels against the best government. God keep us from both.

However, using type bought from England, such as those of William Caslon (1692–1766), printing became a thriving business. Figure 27 is Caslon's first specimen sheet, originally printed in 1734. The



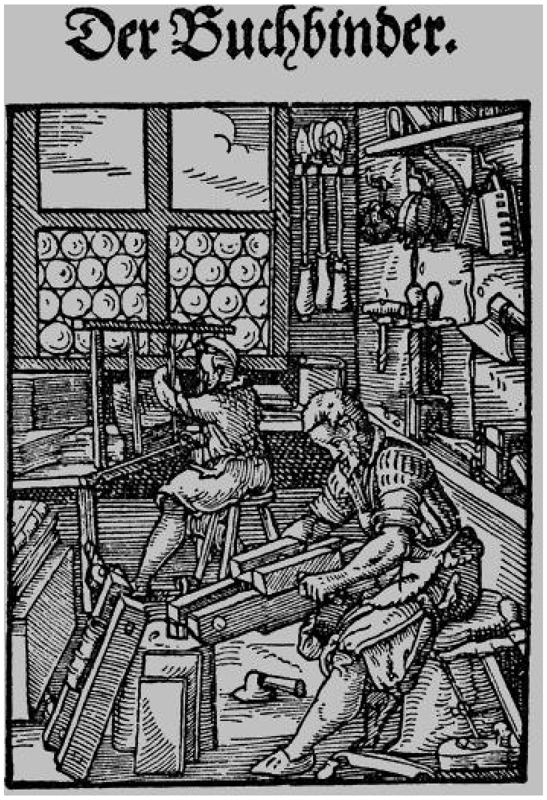
Herries Collection

Figure 23: 16th century printing shop

original is 15½ by 20 ½ inches. As well as the expected roman, italic, and blackletter, the specimens include fonts for the Saxon, Gothick, Coptic, Armenian, Syriac, Samaritan, Arabic, Hebrew (both with and without points), and Greek alphabets. The roman ranges in size from Cannon to Pearl although examples of 6- and 8-line Pica are also shown; the exotics mostly come in a single size although there are three sizes of Greek. There are also several printers ornaments.

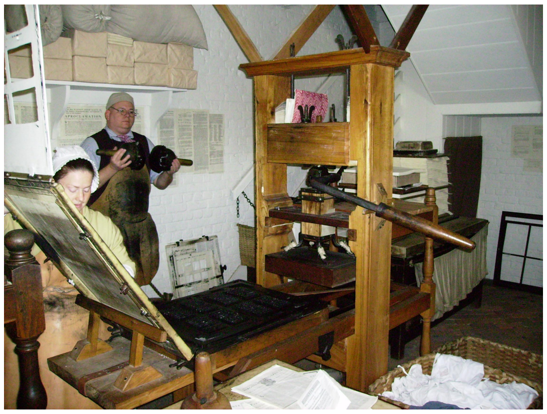
Nowadays, the size of a font is expressed in points but originally names were used. The more common sizes are given in Table 1.

Caslon's type was used in Philadelphia by John Dunlap for the first printing of *The Declaration of Inpependence* in 1776. A more prosaic example of the kind of work done by Colonial printers is Figure 28 showing the title page of *Every Man his own Doctor: or, The Poor Planter's Physician* as printed in Williamsburg by William Parks in 1736. This edition is hand set with Caslon Oldstyle Type. The original is 5 by 7½ inches. The binding of such publications was very easy as the sheets were simply sewn together along the lines of Japanese stab bindings, but not so attractively.



Herries Collection

Figure 24: 16th century book bindery

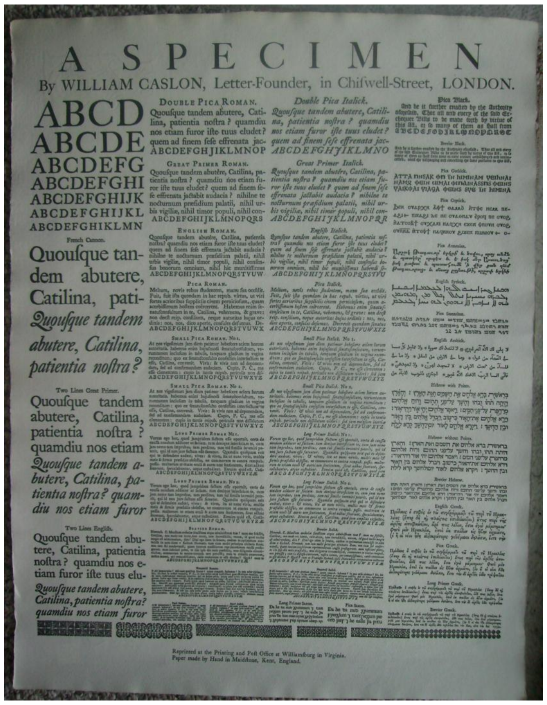


Herries Collection

Figure 25: Reconstructed 18th century print shop (Williamsburg 2007)

Herries Collection

Figure 26: Reconstructed 18th century bindery (Williamsburg 2007)


A SPECIMEN

By WILLIAM CASLON, Letter-Founder, in Chiſwell-Street, LONDON.

ABCD
ABCDE
ABCDEFG
ABCDEFGHI
ABCDEFGHIJK
ABCDEFGHIJKL
ABCDEFGHIKLMN

French Cannon.

Quouſque tandem abutere, Catilina, pati-
Quouſque tandem abutere, Catilina, patientia noſtra?

Two Lines Great Primer.

Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam
Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam furor

Two Lines Engliſh.

Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam furor iſte tuus elu-
Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam furor

DOUBLE PICA ROMAN.

Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam furor iſte tuus eludet? quem ad finem ſeſe effrenata jac-
ABCDEFGHJIKLMNOP

Double Pica Italick.

Quouſque tandem abutere, Catilina, patientia noſtra? quamdiu nos etiam furor iſte tuus eludet? quem ad finem ſeſe effrenata jac-
ABCDEFGHJIKLMNO

GREAT PRIMER ROMAN.

Quouſque tandem abutêre, Catilina, patientia noſtra? quamdiu nos etiam furor iſte tuus eludet? quem ad finem ſeſe effrenata jactabit audacia? nihilne te nocturnum præſidium palatii, nihil urbis vigiliæ, nihil timor populi, nihil con-
ABCDEFGHIJKLMNOPQRS

Great Primer Italick.

Quouſque tandem abutêre, Catilina, patientia noſtra? quamdiu nos etiam furor iſte tuus eludet? quem ad finem ſeſe effrenata jactabit audacia? nihilne te nocturnum præſidium palatii, nihil urbis vigiliæ, nihil timor populi, nihil con-
ABCDEFGHIJKLMNOPQR

ENGLISH ROMAN.

English Italick.

PICA ROMAN.

Pica Italick.

SMALL PICA ROMAN. No 1.

Small Pica Italick. No 1.

SMALL PICA ROMAN. No 2.

Small Pica Italick. No 2.

LONG PRIMER ROMAN No 1.

Long Primer Italick. No 1.

LONG PRIMER ROMAN No 2.

Long Primer Italick. No 2.

BREVIER ROMAN.

Brevier Italick.

Pica Black.

Brevier Black.

Pica Gothick.

Pica Coptick.

Pica Armenian.

English Syriack.

Pica Samaritan.

English Arabick.

Hebrew with Points.

Hebrew without Points.

Brevier Hebrew.

English Greek.

Pica Greek.

Long Primer Greek.

Brevier Greek.

Long Primer Saxon.

Pica Saxon.

Reprinted at the Printing and Poſt Office at Williamſburg in Virginia.
Paper made by Hand in Maidſtone, Kent, England.

Herries Collection

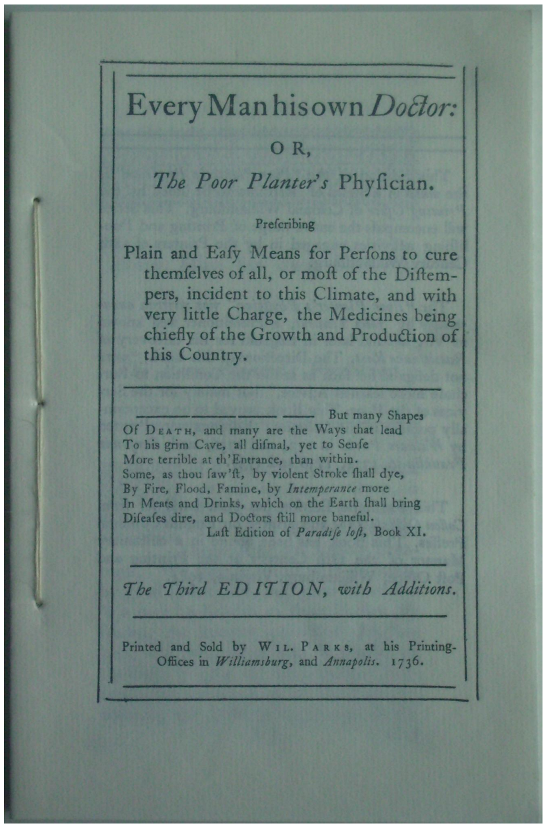
Figure 27: Specimen sheet of Caslon types, 1734

Table 1: Traditional font size designations

Points	Name
4	Diamond
5	Pearl
6	Nonpareil
7	Minion
8	Brevier
9	Bourgois
10	Long Primer
11	Small Pica
12	Pica
14	English
18	Great Primer
24	Double (or Two Line) Pica
28	Double (or Two Line) English
36	Double (or Two Line) Great Primer
48	French Canon (or Four Line Pica)
60	Five Line Pica
72	Six line Pica
96	Eight Line Pica

The book was very popular; two editions were printed by William Parks, and Benjamin Franklin printed three editions between 1734 and 1737. The reprinted version notes that ‘The Directions in the Book “were not designed for such as are in the Condition to Purchase more learned Advice” but mainly for the Services of the Poor’. The Directions mainly seemed aimed at making the patient so uncomfortable that it was better to be well than ill. The recommended treatments for almost everything except physical injuries seemed to involve the letting of copious amounts of blood accompanied by potions aimed at purging anything the patient may have eaten or drunk over the previous couple of days.

Like Caslon, John Baskerville (1706–1775) came from the Birmingham area in England. He printed his first book, Virgil’s *Georgics*, in 1757. Not only did he design his type but he also improved on the printing press of the day and experimented with the formula for ink to produce one that was blacker and more uniform, and also dried quicker which improved the overall efficiency of the printing process. A page from his 1761 edition of *The Plays and Poems of William Congreve* is shown in Figure 29; the original is 5 3/4 by 8 7/8 inches. He invented, and used, a new kind of paper called *wove* rather than the normal *laid* paper. His type had greater contrast between the thick and thin strokes than Caslon’s and was more open. His work was not much appreciated in his native England as it was felt to be too brilliant, or bright, thus hurting the eyes. However he



Every Man his own *Doctor:*

O R,

The Poor Planter's Phyſician.

Preſcribing

Plain and Eaſy Means for Perſons to cure themſelves of all, or moſt of the Diſtempers, incident to this Climate, and with very little Charge, the Medicines being chiefly of the Growth and Production of this Country.

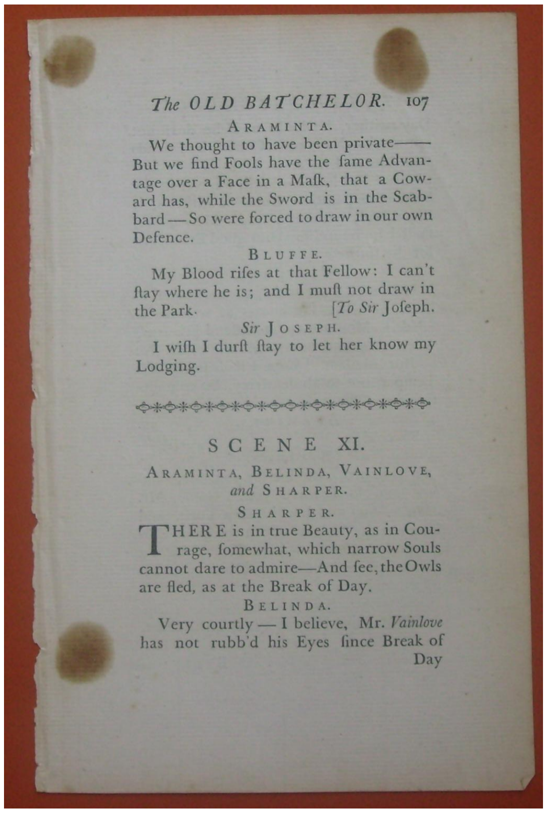
——— ——— ——— —— But many Shapes
Of Death, and many are the Ways that lead
To his grim Cave, all diſmal, yet to Senſe
More terrible at th'Entrance, than within.
Some, as thou ſaw'ſt, by violent Stroke ſhall dye,
By Fire, Flood, Famine, by *Intemperance* more
In Meats and Drinks, which on the Earth ſhall bring
Diſeaſes dire, and Doctors ſtill more baneful.
Laſt Edition of *Paradiſe loſt*, Book XI.

The Third EDITION, *with Additions.*

Printed and Sold by Wil. Parks, at his Printing-Offices in *Williamsburg*, and *Annapolis*. 1736.

Herries Collection

Figure 28: Title page of *Every Man his own Doctor*, Williamsburg, Va, 1736



The OLD BATCHELOR. 107

Araminta.

We thought to have been private——But we find Fools have the ſame Advantage over a Face in a Maſk, that a Coward has, while the Sword is in the Scabbard—So were forced to draw in our own Defence.

Bluffe.

My Blood riſes at that Fellow: I can't ſtay where he is; and I muſt not draw in the Park. [*To Sir* Joſeph.

Sir Joseph.

I wiſh I durſt ſtay to let her know my Lodging.

SCENE XI.

Araminta, Belinda, Vainlove, *and* Sharper.

Sharper.

THERE is in true Beauty, as in Courage, ſomewhat, which narrow Souls cannot dare to admire—And ſee, the Owls are fled, as at the Break of Day.

Belinda.

Very courtly—I believe, Mr. *Vainlove* has not rubb'd his Eyes ſince Break of Day

Herries Collection

Figure 29: Page from Baskerville's edition of *The Plays and Poems of William Congreve*, 1761

had a major influence on continental type designers such as Fournier, Didot and Bodoni.

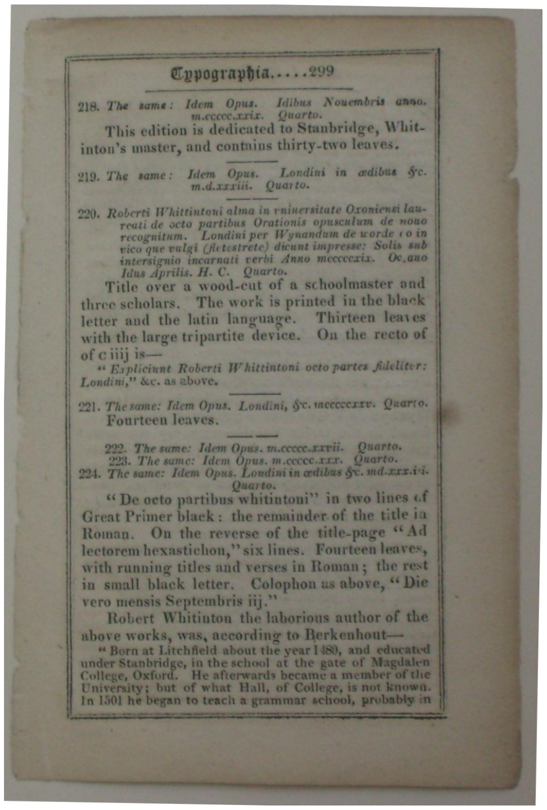
John Johnson (1777–1848) produced an exhaustive survey of typography and printing in his two volume, 1300 page *Typographia, or the Printers' Instructor* published in 1824. The work was produced in four sizes, the largest being royal octavo (6 1/8 by 9 7/8 inches) and the smallest, as shown in Figure 29, being thirty-twomo (3 1/4 by 4 7/8 inches) (Wulling, 1967). The latter is not easy to read because of the small size of the print, from 8pt down to 4pt, but it must have been infinitely more difficult to typeset and proof read the half a million words in the two volumes. The title pages alone, one of which is shown in Figure 31, were built up using over a thousand flowers and rules.

Many nineteenth century printers seem to have felt the need to show off their collection of fonts, often choosing a book's title page as the ideal place for this. Johnson's title page is an amazing piece of printing, but most certainly is not at all representative of the general style. Figure 32 is the title page from *Affectionate Advice to Apprentices*, written in 1827 by the Rector of St. Swithin's at London Stone, for the then Lord Mayor of London. It was distributed widely to many of the young people learning their crafts within the City.[6] This copy was reprinted in 1903. The original size is 4 3/4 by 7 inches. The Victorian lifestyle comes through very clearly: work, obey, learn, and pray. There is no mention of having fun but plenty of advice about avoiding sinful pleasures like going to the theatre to see a play. There is one telling remark, though.

> Our Creator, in great mercy to working people, has commanded every seventh day to be kept to the end of the world as a day of holy rest. If God had not appointed this rest, masters would never in the first instance have thought of giving it to their workpeople.

William Morris, one of the founders of the Arts and Crafts movement, disliked the erosion of craftsmanship by machines, and in 1891 he established

[6] The Worshipful Company of Goldsmiths, chartered in 1327, still presents it to those seeking to become Freemen of the Company; other Livery Companies may do so as well.



𝔗𝔶𝔭𝔬𝔤𝔯𝔞𝔭𝔥𝔦𝔞.....299

218. *The same: Idem Opus. Idibus Nouembris anno. m.ccccc.xxix. Quarto.*

This edition is dedicated to Stanbridge, Whitinton's master, and contains thirty-two leaves.

219. *The same: Idem Opus. Londini in ædibus &c. m.d.xxxiii. Quarto.*

220. *Roberti Whittintoni alma in vniuersitate Oxoniensi laureati de octo partibus Orationis opusculum de nouo recognitum. Londini per Wynandum de worde ‹o in vico que vulgi (fletestrete) dicunt impresse: Solis sub intersignio incarnati verbi Anno mccccxix. Oc.auo Idus Aprilis. H. C. Quarto.*

Title over a wood-cut of a schoolmaster and three scholars. The work is printed in the black letter and the latin language. Thirteen leaves with the large tripartite device. On the recto of of c iiij is—

"*Expliciunt Roberti Whittintoni octo partes fideliter: Londini,*" &c. as above.

221. *The same: Idem Opus. Londini, &c. mcccccxxv. Quarto.*
Fourteen leaves.

222. *The same: Idem Opus. m.ccccc.xxvii. Quarto.*
223. *The same: Idem Opus. m.ccccc.xxx. Quarto.*
224. *The same: Idem Opus. Londini in ædibus &c. md.xxx.vi. Quarto.*

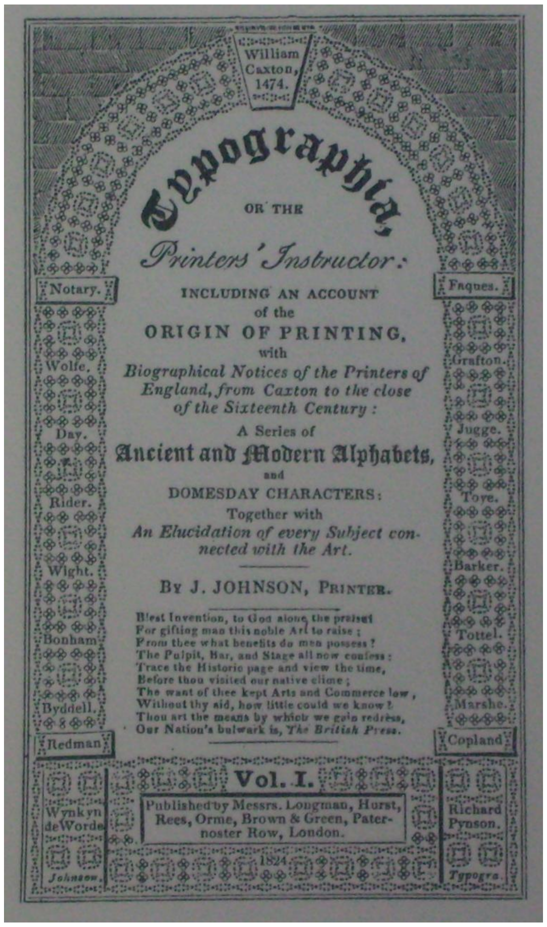
"De octo partibus whitintoni" in two lines of Great Primer black: the remainder of the title in Roman. On the reverse of the title-page "Ad lectorem hexastichon," six lines. Fourteen leaves, with running titles and verses in Roman; the rest in small black letter. Colophon as above, "Die vero mensis Septembris iij."

Robert Whitinton the laborious author of the above works, was, according to Berkenhout—

"Born at Litchfield about the year 1480, and educated under Stanbridge, in the school at the gate of Magdalen College, Oxford. He afterwards became a member of the University; but of what Hall, of College, is not known. In 1501 he began to teach a grammar school, probably in

Herries Collection

Figure 30: Page from John Johnson's *Typographia*, 1824


William Caxton, 1474.
Notary.
Wolfe.
Day.
Rider.
Wight.
Bonham
Byddell.
Redman
Faques.
Grafton.
Jugge.
Toye.
Barker.
Tottel.
Marshe.
Copland
Wynkyn deWorde
Johnson.
Richard Pynson.
Typogra.

𝔗𝔶𝔭𝔬𝔤𝔯𝔞𝔭𝔥𝔦𝔞,

OR THE

Printers' Instructor:

INCLUDING AN ACCOUNT
of the
ORIGIN OF PRINTING,
with
Biographical Notices of the Printers of England, from Caxton to the close of the Sixteenth Century:

A Series of
𝔄𝔫𝔠𝔦𝔢𝔫𝔱 𝔞𝔫𝔡 𝔐𝔬𝔡𝔢𝔯𝔫 𝔄𝔩𝔭𝔥𝔞𝔟𝔢𝔱𝔰,
and
DOMESDAY CHARACTERS:
Together with
An Elucidation of every Subject connected with the Art.

BY J. JOHNSON, PRINTER.

Blest Invention, to God alone the praise!
For gifting man this noble Art to raise;
From thee what benefits do men possess?
The Pulpit, Bar, and Stage all now confess:
Trace the Historic page and view the time,
Before thou visited our native clime;
The want of thee kept Arts and Commerce low,
Without thy aid, how little could we know?
Thou art the means by which we gain redress,
Our Nation's bulwark is, *The British Press.*

Vol. I.

Published by Messrs. Longman, Hurst, Rees, Orme, Brown & Green, Paternoster Row, London.

1824.

Herries Collection

Figure 31: Title page of John Johnson's *Typographia*, 1824

the Kelmscott Press to produce hand made books of the highest quality.

Among others, he produced what is known as the Kelmscott Chaucer, his best known book, consisting of Chaucer's *Canterbury Tales* and all his other works — a total of 31 altogether — which include *The Romaunt of the Rose*, *Troilus and Cressida* and *A Treatise on the Astrolobe*. Although Morris designed the type (Chaucer) and the borders and the decorative initials, 87 woodcuts by Edward Burne-Jones were used as well. The book was published as a limited edition in 1896. There were 425 copies on paper, forty-eight of which were bound in pigskin by Thomas Cobden-Sanderson of the Doves Bindery (later the Doves Press). There were also thirteen copies on vellum. As the pages are 11 3/8 by 16 5/8 inches it is not a book for light reading.

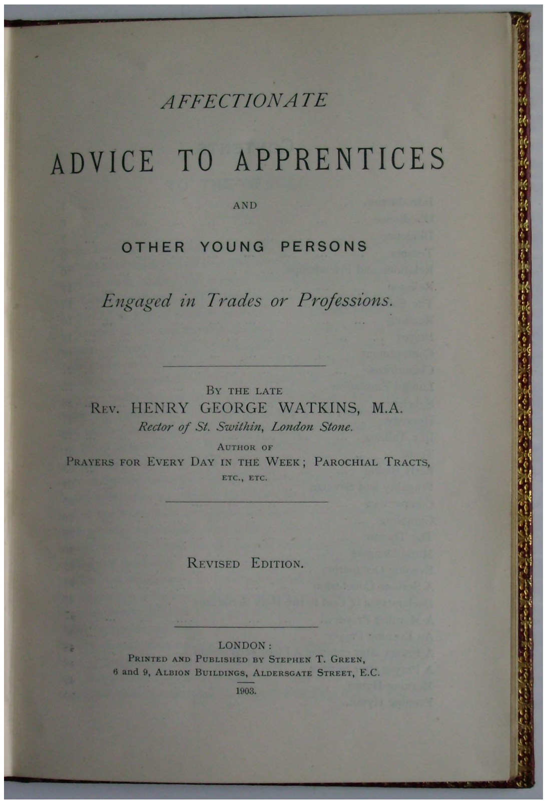
Figure 33 is the opening page of the Prologue to Chaucer's *Canterbury Tales* from a facsimile of the Kelmscott Chaucer. The facsimile is 'slightly reduced in size' where the pages are only 8 5/8 by 12 7/8 inches and weighs 6 1/2 lbs (3 kg).

Morris believed that the factors in bookmaking were all interdependent, that is, the type, paper, ink, imposition and impression all had to be considered together. He also declared that a double spread must always be considered as a whole unit, as demonstrated in Figure 34. Although it has been said (Chappell and Bringhurst, 1999, p. 226) that his style has 'an abundance of thickets and undergrowth', he started people considering a book as a work of art not as simply words on pages, and was instrumental in initiating the move away from the excesses of the Victorian printers.

5 Almost today

The traditions that started to be established in the 16th century are still seen today. Although books are not so lavishly decorated as some from the early days of printing, in general they have calmed down from the freneticism that occurred during the 19th century.

Manuscripts tended to emphasise the capital letter at the start of a paragraph (see Figures 15


AFFECTIONATE

ADVICE TO APPRENTICES

AND

OTHER YOUNG PERSONS

Engaged in Trades or Professions.

By the late
Rev. HENRY GEORGE WATKINS, M.A.
Rector of St. Swithin, London Stone.

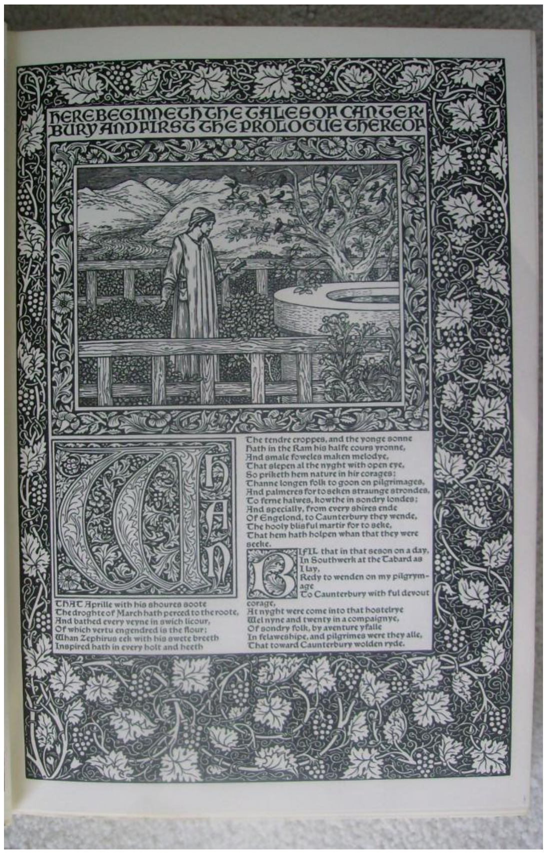
Author of
Prayers for Every Day in the Week; Parochial Tracts,
etc., etc.

Revised Edition.

LONDON:
Printed and Published by Stephen T. Green,
6 and 9, Albion Buildings, Aldersgate Street, E.C.
1903.

Herries Collection

Figure 32: *Affectionate Advice to Apprentices*, 1827


HEREBEGINNETH THE TALES OF CANTERBURY AND FIRST THE PROLOGUE THEREOF

WHAN THAT Aprille with his shoures soote
The droghte of March hath perced to the roote,
And bathed every veyne in swich licour,
Of which vertu engendred is the flour;
Whan Zephirus eek with his swete breeth
Inspired hath in every holt and heeth
The tendre croppes, and the yonge sonne
Hath in the Ram his halfe cours yronne,
And smale foweles maken melodye,
That slepen al the nyght with open eye,
So priketh hem nature in hir corages;
Thanne longen folk to goon on pilgrimages,
And palmeres for to seken straunge strondes,
To ferne halwes, kowthe in sondry londes;
And specially, from every shires ende
Of Engelond, to Caunterbury they wende,
The hooly blisful martir for to seke,
That hem hath holpen whan that they were seeke.

BIFIL that in that seson on a day,
In Southwerk at the Tabard as I lay,
Redy to wenden on my pilgrymage
To Caunterbury with ful devout corage,
At nyght were come into that hostelrye
Wel nyne and twenty in a compaignye,
Of sondry folk, by aventure yfalle
In felaweshipe, and pilgrimes were they alle,
That toward Caunterbury wolden ryde.

Herries Collection

Figure 33: Opening page of the Kelmscott Chaucer's *Prologue*, 1896

and 16), and especially at the start of a major piece of the text as in Figure 17. Versals are still used, as shown in Figure 35 which is the opening page of *The Centaur Types* (Rogers, 1949), but much more rarely than in medieval times. Bruce Rogers (1870–1957) is said to be the 'most accomplished book designer that America has yet produced' (Lawson, 1990, p. 62). He was also the designer of the Centaur type which 'has been one of the widely praised roman types of out time' (ibid, p.72). Rogers described how he came to design Centaur in *The Centaur Types*, which, of course, is set in Centaur and it also includes exact size reproductions of the engraver's patterns. The original size is 6$^{1}/_{4}$ by 9$^{1}/_{2}$ inches.

The *Nuremberg Chronicle*, as in Figure 19, put woodcuts into cutouts in the text. The same idea can be seen in Figure 36 which shows page 3 from *Hammer and Hand* by Raymond Lister with drawings by Richard Bawden (Lister, 1969). The book is a long essay on the ironwork of Cambridge, principally the colleges' wrought iron gates. It was the Cambridge University Printer's Christmas book for

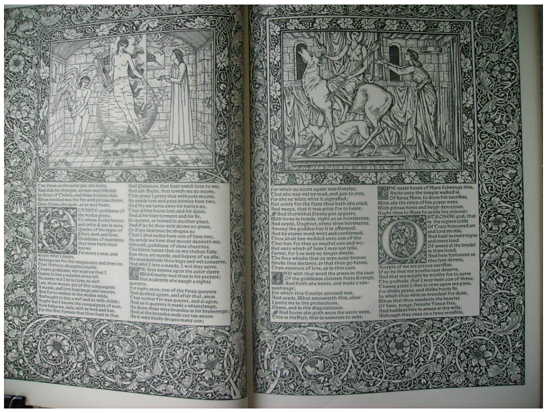
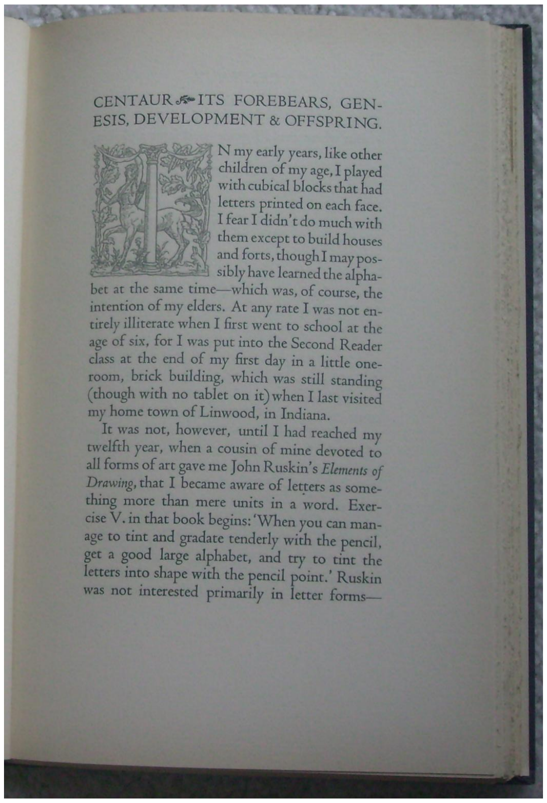

Herries Collection

Figure 34: Double spread from the Kelmscott Chaucer, 1896



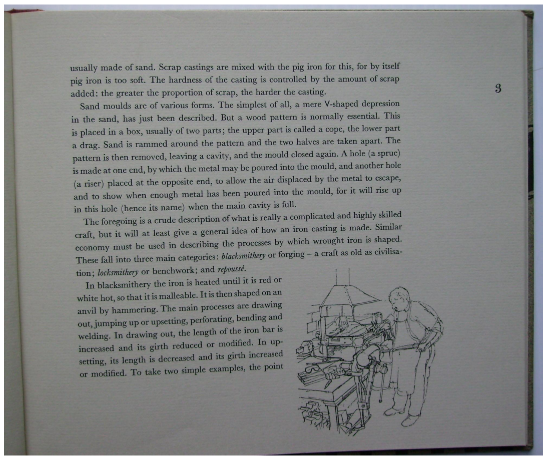
CENTAUR ☙ ITS FOREBEARS, GENESIS, DEVELOPMENT & OFFSPRING.

IN my early years, like other children of my age, I played with cubical blocks that had letters printed on each face. I fear I didn't do much with them except to build houses and forts, though I may possibly have learned the alphabet at the same time—which was, of course, the intention of my elders. At any rate I was not entirely illiterate when I first went to school at the age of six, for I was put into the Second Reader class at the end of my first day in a little one-room, brick building, which was still standing (though with no tablet on it) when I last visited my home town of Linwood, in Indiana.

It was not, however, until I had reached my twelfth year, when a cousin of mine devoted to all forms of art gave me John Ruskin's *Elements of Drawing*, that I became aware of letters as something more than mere units in a word. Exercise V. in that book begins: 'When you can manage to tint and gradate tenderly with the pencil, get a good large alphabet, and try to tint the letters into shape with the pencil point.' Ruskin was not interested primarily in letter forms—

Herries Collection

Figure 35: *The Centaur Types* (1949)



3

usually made of sand. Scrap castings are mixed with the pig iron for this, for by itself pig iron is too soft. The hardness of the casting is controlled by the amount of scrap added: the greater the proportion of scrap, the harder the casting.

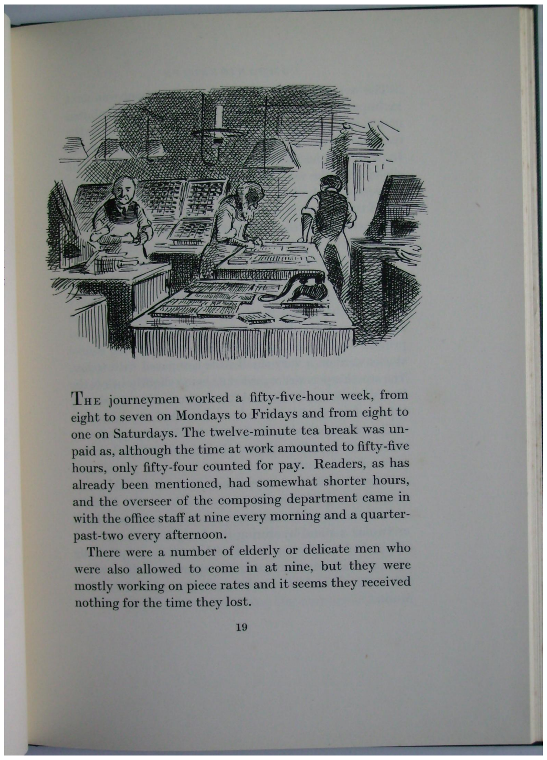
Sand moulds are of various forms. The simplest of all, a mere V-shaped depression in the sand, has just been described. But a wood pattern is normally essential. This is placed in a box, usually of two parts; the upper part is called a cope, the lower part a drag. Sand is rammed around the pattern and the two halves are taken apart. The pattern is then removed, leaving a cavity, and the mould closed again. A hole (a sprue) is made at one end, by which the metal may be poured into the mould, and another hole (a riser) placed at the opposite end, to allow the air displaced by the metal to escape, and to show when enough metal has been poured into the mould, for it will rise up in this hole (hence its name) when the main cavity is full.

The foregoing is a crude description of what is really a complicated and highly skilled craft, but it will at least give a general idea of how an iron casting is made. Similar economy must be used in describing the processes by which wrought iron is shaped. These fall into three main categories: *blacksmithery* or forging – a craft as old as civilisation; *locksmithery* or benchwork; and *repoussé*.

In blacksmithery the iron is heated until it is red or white hot, so that it is malleable. It is then shaped on an anvil by hammering. The main processes are drawing out, jumping up or upsetting, perforating, bending and welding. In drawing out, the length of the iron bar is increased and its girth reduced or modified. In upsetting, its length is decreased and its girth increased or modified. To take two simple examples, the point

Herries Collection

Figure 36: *Hammer and Hand* (1969)



The journeymen worked a fifty-five-hour week, from eight to seven on Mondays to Fridays and from eight to one on Saturdays. The twelve-minute tea break was unpaid as, although the time at work amounted to fifty-five hours, only fifty-four counted for pay. Readers, as has already been mentioned, had somewhat shorter hours, and the overseer of the composing department came in with the office staff at nine every morning and a quarter-past-two every afternoon.

There were a number of elderly or delicate men who were also allowed to come in at nine, but they were mostly working on piece rates and it seems they received nothing for the time they lost.

19

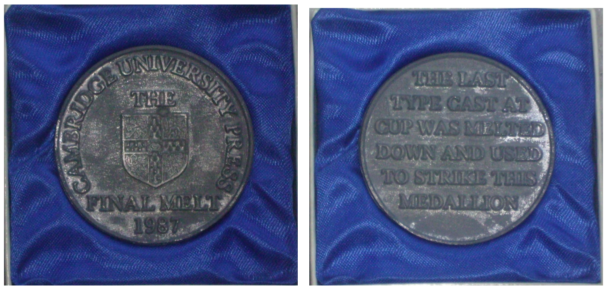
Herries Collection

Figure 37: *A Stickful of Nonpareil:* page 19 (1956)

1969. The original page size is 9¾ by 8⅜, and unusually it is printed on beige paper.

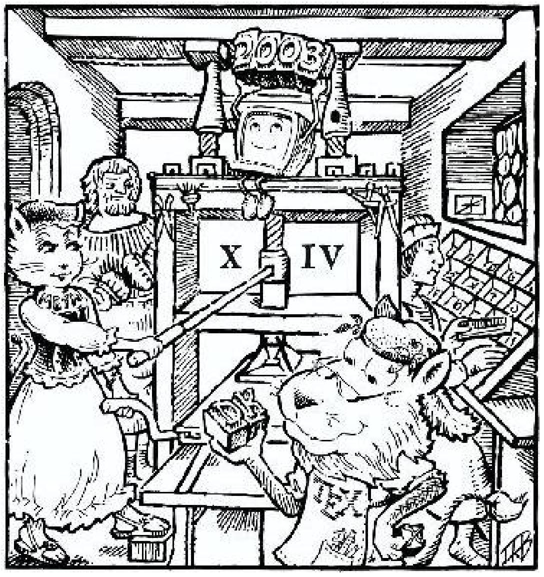
Another element in the design of the *Nuremberg Chronicle* is putting full width illustrations at the top of a page or, as in Figure 18, at the bottom. Figure 37 shows page 19 from *A Stickful of Nonpareil* by George Scurfield and illustrated by Edward Ardizzone (Scurfield, 1956). It was the Cambridge University Printer's Christmas book for 1956. The original is 6½ by 9 inches. 'Nonpareil' is an old printers name for a particular size (6pt) of type, and the book consists of recollections of working at the Cambridge University Press around the end of the nineteenth century. The illustration shows a part of the composing room which is not all that different from the composing area in Jost Amman's 16th century view (Figure 23).

There are, of course, the inevitable changes, both in fashion and, more significantly, in technology. For example, the Cambridge University Press used metal types when it was founded in 1584 and since then all was set by hand until a Monotype composing machine was introduced in 1913 (Black, 1988). Computer-aided phototypesetting and lithographic printing were introduced in the early 1970s.



Herries Collection

Figure 38: Type metal medallion, 1987



Duane Bibby (EuroTEX 2003)

Figure 39: The TEX print shop, 2003

Finally, after four centuries, the last vestiges of the traditional techniques vanished in 1987 when the types that remained in use were finally melted down and cast into commemorative medallions, shown in Figure 38.

On the other hand, Duane Bibby's drawing for the EuroTEX 2003 Conference (Figure 39) shows that the spirit of the tradition lives on.

References

Black, M. H. *Cambridge University Press 1584–1984*. Cambridge University Press, 1988.

Budge, E. A. Wallis. *The Rise and Progress of Assyriology*. London, 1925.

Chappell, Warren and R. Bringhurst. *A Short History of the Printed Word*. Hartley & Marks, 1999.

George, Andrew. *The Epic of Gilgamesh*. Penguin Classics, 2000.

Gurari, Eitan. "TeX4ht: LaTeX and TeX for Hypertext". 2007. Available from `http://www.cse.ohio-state.edu/~gurari/TeX4ht`.

Healey, John F. *The Early Alphabet*. Reading the Past. University of California Press, 1990.

Hinde, Thomas, editor. *The Domesday Book: England's Heritage, Then and Now*. Guild Publishing London, 1985.

Kemmerer, Sharon J., editor. *STEP: The Grand Experience*. Number 939 in Special Publications. National Institute of Standards and Technology, 1999.

Lawson, Alexander. *Anatomy of a Typeface*. David R. Godine, 1990.

Lister, Raymond. *Hammer and Hand: An essay on the Ironwork of Cambridge*. Cambridge University Printer, 1969. Drawings by Richard Bawden.

Mosley, James. "The American Printing History Association 2003 Individual Award: Acceptance Remarks". 2003. (`http://www.printinghistory.org/htm/misc/awards/2003-james-mosley.htm`).

Pigafetta, Antonio. *Magellan's Voyage: A Narrative Account of the First Circumnavigation*. Yale University Press, 1969. In two volumes. Translated and edited by R. A. Skelton from the manuscript in the Beinecke Rare Book and Manuscript Library of Yale University.

Rogers, Bruce. *The Centaur Types*. October House, 1949.

Ryan, William and W. Pitman. *Noah's Flood: The New Scientific Discoveries About the Event that Changed History*. Touchstone, 2000.

Scurfield, George. *A Stickful of Nonpareil*. Cambridge University Printer, 1956. Illustrated by Edward Ardizzone.

Thorpe, James. *The Gutenberg Bible: Landmark in Learning*. Huntington Library, 1999.

Wilson, Peter. "The Alphabet Tree". *TUGboat* **26**(3), 199–214, 2005.

Wulling, Emerson G. *J. Johnson, Typ*. Sumac Press, La Crosse, Wisconsin, 1967.

The STIX Project — From Unicode to Fonts

Barbara Beeton
American Mathematical Society
201 Charles Street
Providence, RI 02904-2294
USA
bnb at ams.org

Abstract

The goal of the STIX project is to provide fonts usable with other existing tools to make it possible to communicate mathematics and similar technical material in a natural way on the World Wide Web. This has involved two major efforts: enlarging Unicode to recognize the symbols of the mathematical language, and creating the fonts necessary to convert encoded texts into readable images.

This ten-year effort is finally resulting in fonts that can actually be used for the intended purpose.

1 Introduction: What is Unicode?

According to the Unicode manual, the original goal of the effort was "to unify the many hundreds of conflicting ways to encode characters, replacing them with a single, universal standard."

Unicode is thus an encoding system capable of representing all the world's languages in a way that will enable any person to interact with a computer in his own language. Nearly all modern computer operating systems are based on Unicode.

The three principal components of Unicode are the character, the block and the plane. A character is the smallest unit, carrying a semantic value. A character may represent a letter, a digit, or some other symbol or function.

A block consists of 256 characters — the number of characters that can be addressed by eight binary digits, addressed as 00–FF. A plane is composed of 256 blocks, for a total of 65 536 characters; there are 16 planes.

The first plane, Plane 0, is referred to as the Basic Multilingual Plane (BMP); if a piece of software claims to support Unicode, it should be able to access every character in the BMP.

Characters are assigned to blocks with the most heavily used given the lowest addresses; assignments are made in half-block (128-byte) chunks.

The first half of block 00 is the basic character set known as ASCII (the formal name is "C0 Controls and Basic Latin"). This contains the upper- and lowercase Latin alphabet, ten digits, various punctuation marks, and a number of control functions; it is the set of characters found on most computer keyboards. The second half of block 00 is known "Latin 1", and includes many accented letters found in western European languages, as well as additional punctuation marks and control characters.

The next few blocks contain:

- the Greek and Cyrillic alphabets;
- a collection of diacritics to be used to compose accented letters not accommodated by Latin 1;
- Hebrew;
- Arabic;
- the scripts for many of the languages of India and southeast Asia.

Each script is allotted a half or full block as needed.

Blocks from 10 to 1F accommodate more language scripts, including extensions for Latin and Greek. Except for very basic symbols such as plus (+) or asterisk (*), non-language characters aren't included until blocks beginning at 20.

2 Who is responsible for Unicode, and how do things get added?

Unicode was developed and is maintained by the Unicode Technical Committee (UTC) an arm of the Unicode Consortium. Members of the consortium include most computer hardware manufacturers and software vendors. To align Unicode with ISO 10646, the standard on which hardware and software are actually based, the UTC works closely with the standardization subcommittee for coded character sets of the International Organization for Standardization.

The UTC members are individuals with various areas of expertise. Most have a strong background in computer software. Many are skilled as well in languages and linguistic-related areas. However, there are very few practicing physical scientists.

If something isn't in Unicode, there is a standard proposal form. This asks for a number of items:

- the repertoire of characters being requested, including character names;
- the context in which the proposed characters are used;
- references to authoritative published sources where the characters have been used;
- relationships the proposed characters bear to characters already encoded;
- contact information for the supplier of a computerized font to be used in printing the standard;
- names and addresses of contacts within national or user organizations.

The on-line description of the proposal review process warns that

- international standardization requires a significant effort on the part of the submitter;
- it frequently takes years to move from an initial proposal to final standardization;
- submitters should be prepared to become involved in the process.

In the case of the STIX proposal, all these warnings were true, in spades.

3 Initial conditions

In 1997, when the STIX project began, Unicode was at version 2.0. It contained several blocks of interest for mathematics:

- combining diacritics (first half of block 03 for text; the last three 16-cell columns of block 20 for diacritics used with symbols)
- Greek (last half of block 03)
- arrows (last half of block 21)
- mathematical operators (block 22)
- miscellaneous technical (first half of block 23)
- geometric shapes (last half of block 25)

None of these blocks was entirely full at that time.

4 Character $\neq$ glyph

Unicode encodes *characters*. Each character has a designated, well defined meaning. It appears in the Unicode charts as a representative glyph, or image. However, since the purpose of Unicode is to convey meaning, the shape of the glyph may vary. To take a trivial example, in text, an "A" has the same code whether it is upright Roman, italic (A), bold (**A**), or sans serif (A). Similarly, an accented "é" can be represented either by one code or by a combination of the letter "e" and the combining diacritic "´".

This is not true for math notation, however. The same letter in different styles (italic, script, Fraktur, bold, ...) means different things. This is illustrated by the Hamiltonian equation from physics:

$$\mathcal{H} = \int d\tau(\varepsilon E^2 + \mu H^2)$$

In 1997, at the beginning of the STIX project, there was no way to unambiguously identify the script $\mathcal{H}$. Based only on the encoding, it was indistinguishable from the H on the right side of the equation:

$$H = \int d\tau(\varepsilon E^2 + \mu H^2)$$

Something more was needed; early proposals by UTC members recommended *markup* (e.g., font changes), such as provided by XML or MathML. However, it was realized that a physicist might wish to search for this entity in a corpus or database, and searching would be much more reliable if it could be done using an unambiguous code.

The UTC solution was to incorporate a substantial set of *mathematical alphanumerics*, about 1,000 characters. These variations on the Latin and Greek alphabets fill four complete blocks (U+1D40–U+1D7F) in Plane 1. Placement outside the BMP was meant to discourage casual users from using these special alphabets for things such as wedding invitations, where stylistic markup is more appropriate.

Another facet of the character/glyph dichotomy is the use in math notation of different-sized operators in text vs. display environments — the size used in text is generally smaller; compare $\sum_{i=0}^{\infty} x_i$ and

$$\sum_{i=0}^{\infty} x_i\,.$$

The sum symbol is just a single character in Unicode. Delimiters (parentheses, brackets, etc.) are also considered to be single characters, but they must be provided in many sizes, including segments suitable for piecing together to span multiple lines.

Unicode takes the position that such substitutions are the responsibility of the application.

5 Requesting additions to Unicode

In addition to the approximately 1,000 mathematical alphanumerics already mentioned, the STIX collection identified roughly 1,000 non-alphanumeric symbols that couldn't be found in Unicode version 2. These were assigned provisional identifiers in the Unicode Private Use Area (PUA) in order to keep track of them. IDs were assigned in order of accession, rather than by shape, usage, or other rational system.

Because of the large number of characters being requested, the UTC invited a representative of STIX to present the proposal in person at a regular UTC meeting, to answer questions directly, rather than carrying on an extensive paper and e-mail interchange. The fact that the proposal was backed by five professional societies and a technical publisher, based on actual experience in their publications, the usual requirement for extensive examples was probably lessened. This did not mean that there was no requirement to justify every symbol; it did, however, allow symbols to be considered in groups rather than individually — if one member of a coherent symbol group (e.g., arrows with a triple stem pointing in several directions) was accepted, the rest of the group was accepted as well.

As noted earlier, Unicode assigns characters in blocks, preferably of groups with some inherent relationships. The UTC experts, acting on usage information provided with the proposed characters, classified them into groups that corresponded to the existing symbol blocks: operators, arrows, geometrics, and so forth. Then began the process of shoehorning them into the code space. First, the gaps in existing blocks were filled with appropriate items. Next, the number of characters in each category was tallied, and new blocks of appropriate sizes assigned. The bulk of the math additions first appeared (on line) in Unicode version 3.2, with the first paper publication in version 4.0.

As of Unicode version 5.0, these new blocks have been added:

- miscellaneous mathematical symbols A (U+2700–U+27EF)
- supplemental arrows A (U+27F0–U+27FF)
- supplemental arrows B (U+2900–U+297F)
- miscellaneous mathematical symbols B (U+2980–U+29FF)
- supplemental mathematical operators (U+2A00–U+2AFF)
- miscellaneous symbols and arrows (U+2B00–U+2BFF)

Not all of these blocks are filled yet, but space has been left where experience has shown growth is likely to occur.

One other key feature was adopted: a *variation selector* — a one-character code (U+FE00 for math symbols) identifying the preceding character as having the same meaning, but an alternate shape which cannot be composed from a base character plus a combining diacritic. An example is the relation $\gneqq$ (U+2269) vs. $\gvertneqq$ (U+2269,U+FE00). The use of the variation selector is very tightly controlled; all characters using it must be accepted explicitly by the UTC. Other shape variations must be indicated by markup and recognized by the application software.

Some important decisions were made during the course of this exercise that should make future submissions progress more smoothly.

First and foremost, it was accepted that mathematics is a language, and that symbols used in this context are as essential as the letter “e” is to English. Another “given” is that math notation is open-ended — mathematicians and other scientists will continue to invent and adopt new symbols, so the job isn’t done, and may never be.

Just within the past few months, a mathematician from Morocco has submitted documentation of mathematical notation in Arabic — it is a mirror image of what we see in European language contexts. This generated a flurry of activity in the UTC to adopt a rational collection of right-to-left symbols to complement the basically left-to-right symbols already present. The new material will appear in Unicode version 5.1.

6 What wasn’t accepted, and why not?

In spite of the generally high level of acceptance of characters proposed by STIX, the UTC rejected some symbols. The reason for most rejections was that they weren’t “math”. Symbols used by other disciplines (astronomy, meteorology) were not considered to be relevant to the STIX request; it was suggested that an organization involved in those disciplines should make a separate submission, at which time it would be considered on its own merits.

Some symbols were rejected because they were easily constructed as compounds of existing characters and combining diacritics; this includes any negated relations that hadn’t already been encoded.

For some symbols, in particular ones that were identified after the initial proposal, the available documentation was deemed insufficient for acceptance. However, when a suitable in-context published example is found, acceptance of these stragglers is very likely.

Finally, some items in the STIX collection aren’t considered independent symbols; they are partial glyphs used for constructing larger symbols such as multi-line parentheses or braces, or extenders for arrows. These weren’t even submitted to the UTC since they fall into the area that is the responsibility of application software.

7 Okay, Unicodes have been assigned; how can we print them?

Assignment of Unicodes, while necessary, is not sufficient for use of these symbols in electronic or paper communication. It is also necessary to be able to generate images that can be understood by someone trying to read them. Here is where fonts come in.

A popular font for typesetting of math is Times Roman or one of its variants. This font, originally designed for newspaper use, is compact (a lot of material can be squeezed onto a page), and is legible at small sizes. Its adoption for technical material means that a large number of symbols have been designed to be compatible. Times Roman was the overwhelming choice of the STIX organizations as the base font around which the new STIX fonts would be created.

There are some very specific design criteria for a font intended for math:

- Each letter must be unambiguously recognizable in isolation; for Times, this means that a substitute must be provided for the italic v, since the usual Times shape is too easily confused with the Greek letter nu ν.
- Hairlines must be thick enough to keep shapes from breaking up in sub- and superscripts, and to withstand multiple photocopy runs.
- Normal weight must be readily distinguishable from bold.
- An alphabet intended for use as symbols need not be usable for continuous text; in fact, it is often desirable for a math alphabet to look a bit peculiar if used for text.

Implementation of the STIX glyphs was contracted out. The working list was a database in order of provisional ID; assignment of new Unicodes was still in the future. Glyphs were implemented in blocks, which were returned to the STIX Technical Review Committee for comments; any problem glyphs were returned to the contractor for repair.

The random ordering of the glyphs in the working list meant that glyphs intended to be used together, or supposed to be the same shape or weight, often weren't designed in the same batches, and weren't available for review at the same time. This meant that a final design review would be essential.

The random ordering also meant that the fonts couldn't yet be used for anything practical. Among other things, it was necessary to have a well defined naming scheme. Because the fonts were delivered in Adobe Type 1 form, it was decided to assign glyph names according to the Adobe guidelines. Except for a relatively small core of glyphs — essentially those representing the ASCII and Latin 1 blocks and some additional punctuation — the recommended form of a name was based on the Unicode, with extensions to indicate compounding or size and shape variations. This name begins with either "`uni`" or "`U`" for glyphs corresponding to characters in Unicode Plane 0 or Plane 1 respectively, or with "`stix`" for (the fewer than 256) glyphs with no corresponding Unicode.

8 Bookkeeping, bookkeeping

In order to keep track of what was happening, master tables or databases were maintained in several places. Tim Ingoldsby (of AIP, the overall project manager) started with the same database as used by the font contractor. To this he added, as phases were delivered, information about what glyphs were delivered in which phase, and the font and position in the font where each was located.

I maintained a list based on the original collection information, sorted by Unicode or provisional ID. This initially included sources, the names by which the sources refer to each glyph, the number of instances required (for weight, posture, size, etc.), and a glyph description. As new information became available, or was defined, it was added to the table:

- newly assigned Unicodes, with cross-references to and from the provisional ID;
- Type 1 glyph names;
- "TeX names", since several of the STIX organizations use that typesetting system;
- MathML entity names.

When delivery of the glyphs was nearing completion, Tim reprocessed my list, merged the differences into his database, and produced a file for checking. In this process we identified items that had been overlooked, and made a final list for completion of the deliveries.

That left only a few tasks:

- design review;
- shape and content corrections;
- packaging and user documentation;
- beta testing;
- (LA)TeX support;
- final coordination of MathML entity names.

As of today (May 21, 2007), we are nearly ready to release the fonts for beta test.

9 The design review

One more rearrangement was necessary — organizing the glyphs into groups that reflected shape categories, irrespective of identifier value. Since alphabets are ordered logically within Unicode, they had already been reviewed and corrected, and it was not necessary to look at them again. The other categories included

- diacritics;
- punctuation;
- geometric shapes (circles, squares, diamonds and lozenges, triangles, other polygons);
- arrows;
- relations (equals, greater/less, sub/supersets, others);
- binary operators (cups/caps, and/or, plus/times, other);
- large operators (integrals, other);
- delimiters and fences;
- other shapes.

Within each category, glyphs were arranged by similarity of shape and size. Making sure that everything was accounted for involved one more sweep through the entire STIX master table. This turned up some residual errors, which were corrected so that the permanent documentation would be accurate.

For each group of symbols, proof sheets were generated, reviewed, and comments forwarded to the font specialist making the corrections. No category was accepted the first time around, but most required no more than two cycles for approval.

10 The future

We expect that a few more problems will be identified during beta testing, but in general, we believe that our efforts have resulted in a collection of fonts that will make it possible to represent nearly all mathematical expression both on paper and on computer screens. How this is actually done does depend on application developers, but since support of Unicode beyond just Plane 0 is beginning to be viewed as necessary by browser distributors, we are optimistic.

As we mentioned earlier, mathematical notation is open-ended. The mechanism for adding this notation to Unicode is now in place. The only question open is, how will new glyphs become part of these fonts. Presumably the STIX organizations will address that question after they've all had a well deserved rest.

Support for Multiple TEX Distributions in i-Installer and MacTeX

Richard Koch

Abstract

We discuss a data structure by Gerben Wierda and Jérôme Laurens which makes it easy to use multiple TEX distributions on Mac OS X.

1 A Confession

The wonderful data structure described below was designed by Gerben Wierda and Jérôme Laurens and implemented first in Gerben's i-Installer package for gwTeX and then later in the various MacTeX install packages I maintain. When I learned the details of the design, I was repulsed by its complexity, and began opposing it with increasingly vitriolic emails. Then one day, an email from Jérôme led to a religious conversion on my part. The data structure now seems natural, and the Gods have condemned me to give this talk as punishment for my vitriol.

2 The Problem

TEX GUI applications on the Macintosh, like (my program) TeXShop, LaTeXiT, iTeXMac, BibDesk, and others, call command line programs to typeset. A year ago, the standard Mac TEX distribution was teTeX as packaged by Gerben Wierda. Therefore these GUI applications were configured to point to that distribution, and so the applications *worked right out of the box without any configuration.*

But in May of 2006, Thomas Esser announced the end of support for teTeX. This led to a mad scramble in the TEX world — some users switched to the full unmodified TeX Live and others continued to rely on teTeX. At the International TUG meeting in Marrakesh held in November, 2006, Gerben introduced a new distribution named gwTeX based on TeX Live. But alarmingly, he also announced the end of email support for his distribution, and modified i-Installer so the first dialog which appears says **Unsupported Software** in bold letters, followed by a paragraph of text which begins “I regret having to inform you that i-Installer is unsupported software as of Jan 1, 2007.” Gerben continues to maintain gwTeX, which has a substantial following, and “unsupported” seems to mean merely that he has adopted Donald Knuth's policy of not reading email. But his message puts new users into panic mode.

Meanwhile, the Macintosh Working Group in TUG produced three one-button install packages for TEX, all based on TeX Live and differing mainly in size. These packages are available at `www.tug.org/mactex`. The first of these, BasicTeX, is a 39.7 MB package for users with slow download connections; it installs a surprisingly useful subset of TeXLive 2007. The second, gwTeX, is a 321 MB package which installs gwTeX; users can use i-Installer to maintain this distribution. The third, TeXLive-2007, is a 619 MB package which installs the complete 2007 version of TeX Live.

There are also distributions based on teTeX in Fink and in MacPorts.

All of these distributions install in different locations, so installing one does not overwrite the others. For example, BasicTeX is a subset of TeX Live 2007, but installing TeX Live 2007 creates a completely separate installation rather than upgrading the BasicTeX installation.

This proliferation of distributions confuses new users. In early June, a physics graduate student at the University of Oregon called me after switching from Windows to a Mac. He set aside a Saturday to install software. From the web he learned about the MacTeX full TeX Live distribution and installed it. Then his lab partners told him to get scientific applications with Fink, so he installed Fink. Fink asked him which programs to install, so he said “give me everything.” After that, friends suggested learning TEX by starting with LyX, so he installed that. The LyX installer told him that it needed to put style files in /usr/local/gwTeX, so he searched the internet and found gwTeX. In the course of a single afternoon, he had managed to install three complete TEX distributions on his portable. Everything went smoothly until a Fink web page explained how to reconfigure TeXShop for teTeX, and he couldn't figure out how he had obtained teTeX or what it was.

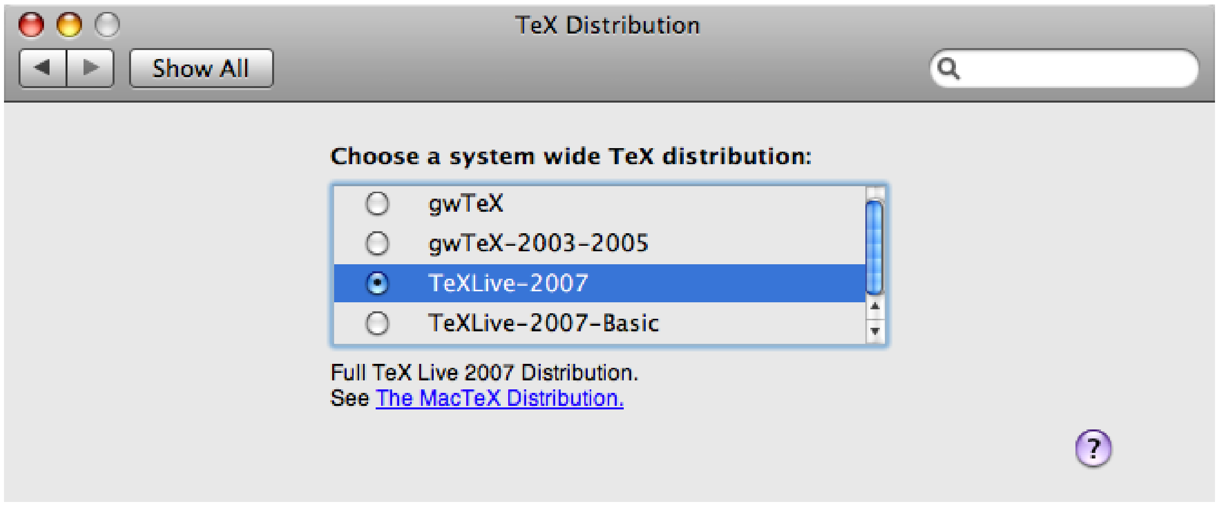
3 The Solution

I'll get to the new data structure in a minute, but first let me tell you what Jérôme Laurens wrote in email which led to my religious conversion. Jérôme told me he had a preference pane for Apple's System Preferences which would list available TEX distributions and allow users to switch from one to another.

Jérôme's pane lists distributions installed on a machine. The active distribution is marked in this list. Click on another distribution to make it active. This action automatically switches the PATH and MANPATH variables so Terminal and other shells use the correct binaries and man pages. It automatically reconfigures *all* GUI programs to use the appropriate distribution. For example, TeXShop, LaTeXiT, iTeXMac, and BibDesk switch instantly to the new distribution. You can open a source file, typeset, switch distributions with the pane, and typeset again with the new distribution, all without restarting the GUI program or even reloading the file.

All of the GUI programs I've listed are configured by default to use the new data structure, so they all *work right out of the box without any configuration.*

Here is the preference pane:



Choose a system wide TeX distribution:

( ) gwTeX
( ) gwTeX-2003-2005
[✓ TeXLive-2007]
( ) TeXLive-2007-Basic

Full TeX Live 2007 Distribution.
See The MacTeX Distribution.

The TeX Live distribution is upgraded once a year and each upgrade is installed in a different location. So TeX Live users can safely upgrade, knowing that it is easy to return to their older distribution. Nevertheless, I'm willing to bet that a very large percentage of TUG members put the TeX Collection DVD aside when it arrives because they are in the middle of a project – and then never retrieve the disk and upgrade.

Do you remember our physics student? I told him "TeXShop is already configured; don't do anything. Go to System Preferences, open the TeX Distribution pane, and you'll see a list of your three distributions. Switch to the one you want to use." Magical, huh?

Jérôme's panel is automatically installed by i-Installer when it installs gwTeX, and by all three MacTeX distributions. So of course the preference pane recognizes these distributions. But it also recognizes legacy distributions written before the pane existed: Fink's teTeX, MacPort's teTeX, Gerben's old teTeX, TeX Live 2004, TeX Live 2005, and – well, there wasn't any TeX Live 2006.

4 Another Problem Solved

Developers want the newest versions of computer software. But many TEX users don't think that way. I've often heard variants of 'I'm in the middle of two book projects and five papers with coauthors; don't talk to me about upgrading TEX." To some extent, interest in upgrading depends on the component of TEX being used; ConTeXt and XeTeX are undergoing rapid development and users of these programs upgrade regularly, while users of LaTeX and pdfTeX are more likely to desire stability over change.

The problem is that retreating to the old distribution is not foolproof. It is all too easy to reconfigure the GUI front end and leave the PATH in Terminal unchanged. Months later such a user might reconfigure TEX with a Terminal command, only to discover that the change didn't "take" in the GUI program. This is a difficult bug to diagnose.

But Jérôme's panel completely solves the problem. Users can retreat to the old distribution with a single panel click and be assured that all programs are configured correctly. Consequently, I'm hoping that Mac users get in the habit of upgrading as soon as the new TUG DVD arrives.

5 Some History

When Thomas Esser made his announcement, there was barely a ripple in the Mac world because "Gerben will take care of it." But when Gerben announced end of support, there was an explosion of concern; several messages on the TeX on Mac OS X mailing list asked "is this the end of TEX on the Mac?"

At that point, several people began experimenting with TeX Live. I am one of them. It is common knowledge that TeX Live is difficult to install; its install script asks obscure questions like "does you compiler use BSD calling conventions." Mind you, nobody ever told me about this difficulty, but I somehow knew. And as confirmation, I often talk to Karl Berry, who maintains the install script, at meetings and he never once asked "why don't you Mac guys use TeX Live?"

But anyway, I dug up an old TUG DVD and ran the script. Gosh. No obscure questions. Very few questions at all, and then poof TeX Live was installed and it ran like a charm.

So I have a complaint about Karl: excessive modesty. Let me push this complaint by quoting a similar complaint about the inventor of the theory of electricity and magnetism. Freeman Dyson's great essay *Missed Opportunities*, Bulletin of the AMS, 1972, is about situations where mathematicians would have made faster progress is they had paid attention to the physicists. The first case he discusses is Maxwell's theory of electricity and magnetism. Dyson begins by quoting Maxwell himself, who said in a lecture "According to a theory of electricity which is making great progress in Germany, two electrical particles act on one another directly at a distance, but with a force which, according to Weber, depends on their relative velocity, and according to a theory developed by Riemann, Lorenz, and Neumann, acts not instantaneously, but after a time depending on the distance. The power with which this theory explains every kind of electrical phenomena must be studied in order to be appreciated. Another theory of electricity which I prefer ..." and then described his own theory.

Dyson writes "It is difficult to read Maxwell's address without being infuriated by his excessive modesty, which led him to refer to his epoch-making discovery of nine years earlier as only 'Another theory of electricity which I prefer.' How different is his style from that of Newton, who wrote at the beginning of the third book of his *Principia* 'It remains that, from the same principles, I now demonstrate the frame of the System of the World.' "

That's my complain about Karl. But to go back to the story –

The three MacTeX install packages mentioned earlier contain unmodified TeX Live installations. In particular, the MacTeX package TeXLive-2007 package installs the full TeX Live exactly as it would appear if installed directly with the TeX Live script. However, there are several reasons that Mac users should install using the MacTeX packages. First, the packages modify PATH and MANPATH, whereas the TeX Live install script asks users to do this themselves. MacTeX modifies these variables exactly as Gerben does from i-Installer and as required by the TEX distribution data structure, so the MacTeX method is compatible with both gwTeX and this structure.

In addition, the MacTeX packages automatically configure paper size, while the TeX Live scripts ask users to do this manually.

Finally, the MacTeX packages install Jérôme's preference pane and the associated data structures, while the TeX Live script knows nothing about these features.

6 A Small Suggestion

When I made MacTeX packages for TeX Live, I ran into a small problem. TeXShop and the other GUI programs no longer worked out of the box; they had to be configured first. To make this easy, I stole an idea from TeX Live and created a symbolic link /usr/local/texprograms pointing to the binary directory of the installed distribution. The idea was that GUI programs would have to be configured just once to use this link, and all installation packages on the Mac would set the link during installation. I wrote Gerben and casually asked him to support this link in i-Installer.

7 and Gerben Runs With It

Instead of making this change, Gerben began to look at the problem from a larger perspective. There are many things that a GUI application might like to know about the active TEX distribution. Certainly it will want to know where the binaries are. But each TEX distribution contains a lot of documentation, and the GUI app might well like to find and display that documentation. Perhaps the GUI app wants to display man pages too. And recall the LyX installer, which wanted to install style files in the distribution; it needed to know the location of the distribution's main texmf tree.

Gerben proposed a data structure which would organize and locate all of this information and more.

Let's return to the simple issue of binary location. The trouble is that each distribution handles this in a slightly different way. In gwTeX, there are two binary directories, one for Intel binaries and one for PowerPC binaries. The intel directory is i386-apple-darwin-current and the powerpc directory is powerpc-apple-darwin-current. TeX Live works similarly, but the binary directories are named differently: i386-darwin and the powerpc-darwin. Fink contains universal binaries which work with both processors, located in /sw/bin. A GUI app might want to find the binary directory for the processor of the machine on which it is running; this GUI app shouldn't have to know these various naming conventions to do its job.

Once we get this far, the basic idea of Gerban and Jérôme's data structure will be clear. For each distribution, they create a small folder of symbolic links pointing to actual locations in the distribution. The names chosen for these links are common for all distributions rather than names used by a particular distribution. Thus a GUI app need only know the common names assigned by Gerban and Jérôme rather than the actual names chosen by a distribution. In addition, there is a link named DefaultTeX pointing to the folder of symbolic links for the active distribution.

Here is how this works in practice. In the folder for each distribution there is a subfolder named Programs and inside that folder are two symbolic links named i386 and powerpc. These point to the actual directories containing the i386 binaries and the powerpc binaries for that distribution. Putting this altogether, a GUI program running on an Intel processor will find the binaries for the currently active distribution in DefaultTeX/Programs/i386. If the active TeX is TeX Live 2007, this will yield /usr/local/texlive/2007/bin/i386-darwin. It will be /sw/bin for Fink's teTeX. That is the simple idea in a nutshell. In this explanation I have left out a few details, so the paths to the links aren't exactly these.

Incidentally, the path to binaries for the active TeX using the architecture of the current machine is of such importance that Gerben and Jérôme provide a shortcut. They create a link /usr/texbin which points to DefaultTeX/Programs/i386 on Intel machines and DefaultTeX/Programs/powerpc on PowerPC machines. Thus GUI apps can use /usr/texbin as a path to TeX binaries. All of the standard GUI apps on Mac OS X already use this as their default path. That is why they run right out of the box.

When the structure is installed by i-Installer or MacTeX, a technical description of the structure is installed in /Library/TeX/Distributions. So it isn't necessary for me to give full details here. Let me say just a little more to indicate the range of possibilities opened up by the structure. The folder of links for a particular distribution contains subfolders named Doc, Info, Man, Root, AllTexmf, and others. Doc contains symbolic links to actual folders in the distribution containing documentation; the number of such links varies with distribution. For instance, for TeX Live 2007 the links are called texmf-dist-doc, texmf-doc, texmf-doc-doc, and texmf-var-doc. Thus a GUI application could point to DefaultTeX/Doc and discover the entire documentation tree for the active distribution.

Info contains links to the various info files; Man contains links to the various man pages, and Root contains a link to the folder containing the entire distribution. AllTexmf contains links to the various texmf trees of the distribution; for example, the TeX Live 2007 AllTexmf contains links named texmf, texmf-dist, texmf-doc, texmf-local, and then texmf-var.

The Macintosh file system has a folder named /Library containing various Apple and Third Party data files which apply system wide. The TeX distribution data structure is installed in /Library/TeX, where TeX is a subfolder created by i-Installer or MacTeX. At the moment this folder only contains information about the TeX distribution data, but I understand that some developers are eyeing it for other uses. So I don't recommend throwing it away cavalierly. The actual distribution symbolic links are in folders in /Library/TeX/Distributions. There will be one such folder for each distribution installed on the machine, together with folders for legacy distributions even if those aren't installed.

It is not necessary to clean up the symbolic links in the data structure if the corresponding distribution is thrown away, because Jérôme Laurens' preference pane is smart and only shows distributions when the data structure points to something concrete. Thus, for instance, it is quite legal to entirely remove TeX Live 2007 using the command

```
sudo rm -R /usr/local/texlive/2007
```

even though this will leave a stranded data structure behind. The data structure itself is tiny.

I have now explained all of the key ideas. The structure and preference pane are quite simple, but I use them virtually every day. Because of this structure,

- GUI apps work right out of the box
- TeX can safely be upgraded because is trivial to revert back if necessary
- GUI TeX applications may display documentation in the future

8 But Give Us the Dirt

You may be saying "Look, I really don't care about the data structure. I don't even use a Mac. I'm here because I want to hear about the vitriolic email." If so, this is the section for you.

Let me say from the start that the disagreements I'm going to describe are in the past — there was a battle, Gerben and Jérôme were right and I was wrong. But for your amusement –

When Gerben first designed the data structure, there was no hint of a preference pane. I didn't exactly know how users would switch the default distribution, but it looked like they were expected to directly open /Library/TeX/Distributions and manipulate the folders and data inside it. This made it urgent that this location contain straightforward data.

At first the design was simple, but then it took an unexpected turn. On the Macintosh, it is possible to trick the Finder so that it displays a folder as just another file. The program XCode, which developers use to write applications, saves projects in this way; each saved project appears to be a simple file, but if you hold down the control key while clicking on the file name, a menu will appear offering to open up the file and display its contents.

For some reason that I still do not understand, Gerben and Jérôme decided that the folders in /Library/TeX/Distributions should be such magic folders. To make this happen, they added an extension ".texdist" to each folder name, put a subfolder named "Contents" in the folder, and then put the various symbolic links inside this Contents folder. The curious consequence is that as long as no application claims to understand .texdist, these will appear as ordinary folders. But as soon as an application – any application – says that it can edit .texdist files, all of those folders get icons supplied by the application and become files which a casual user cannot open. This behavior became the central point of the email controversy, and I'll return to it at the end of this article. But first, let me sketch some of the remaining design complexities.

The magic folder scheme required that the link DefaultTeX become instead a folder containing a link named Contents pointing to the Contents folder of the active distribution. Since this was rather confusing for a casual user, this DefaultTeX folder was made invisible by adding an initial dot to its name. Additional folders which I will not describe here were also made invisible, so a developer attempting to decipher /Library/TeX/Distributions will face a daunting task unless they read the documentation. It is not enough to write DefaultTeX/Programs/i386 to get to the binary directory of the currently active TeX distribution. Instead, the correct path is

```
.DefaultTeX/Contents/Programs/i386
```

and since the data is in /Library/TeX/Distributions, this is really

```
/Library/TeX/Distributions/
.DefaultTeX/Contents/Programs/i386
```

In particular, /usr/texbin is a symbolic link to this location. But there is a final complication. Gerben was worried that users would install TeX on a second hard disk. In that case, absolute paths would point to the primary disk rather than this second disk, so /usr/texbin really needs to be a symbolic link to

```
../Library/TeX/Distributions/
.DefaultTeX/Contents/Programs/i386
```

and most of the other symbolic links must also become relative. For example, the symbolic link for Programs/i386 in TeXLive-2007 is actually

```
../../../../../../../usr/local/
texlive/2007/bin/i386-darwin
```

Around this point, I realized that I could no longer explain the data structure in a few sentences and became alarmed. I have never met Gerben or Jérôme or most of my other collaborators, but Gerben and I have been in email contact for years — really since the start of OS X — and all of this email has been pleasant or better. As I realized the baroque nature of the developing data, I began complaining.

In growing frustration, I wrote a few people who weren't involved in the design and must have wondered what the heck I was talking about. One of those folks was Jonathan Kew, and here's what I wrote him, titling the email "Kvetch":

```
I shouldn't be writing you, but I need
someone to "kvetch" to and I don't want
to pollute the mailing lists. I have just
seen Gerben's final "link" design. I think
it is a mess. Am I supposed to support
the design in MacTeX?

BACKGROUND:

Gerben will end support of his packages
in January, so we face the possibility of
dealing with multiple TeX distributions on
the Mac.

With multiple packages, we need an
easy way to configure GUI applications.
A couple of weeks ago I introduced MacTeX
packages for three distributions and had
each package set a symbolic link
/usr/local/texprograms.  When I asked
Gerben to set /usr/local/texprograms in
i-Installer packages, he decided to design
a more complete solution. Fair enough.
Here is his ingenious idea:
```

```
Gerben proposes that each distribution
set a "link directory" in
/Library/TeX/Distributions. This
directory  would be a very small
collection of  standard links for the
distribution.  There needs to be a way
to select the "active" distribution. and
Gerben proposes creating an extra file
alled "Default"  which is just a link to
the active distribution. I support this
design.

KVETCH:

But when it came time to implement
these changes, Gerben lost all sense
of proportion. For reasons I don't
understand (related to bundles,
but why these small link directories
need to be bundles is beyond me) he uses
TeXLive2005.texdist/Contents/... instead
of TeXLive2005/... Even worse,

  (and etc.; let me skip to the end)

Am I expected to support this baroque
design in MacTeX? When I make a package
for TeX Live, should I use Gerben's
TeXLive-2005, or just give up and make my
own. Am I taking this too personally? Do I
need to get a life?
```

Jonathan is a very intelligent fellow. He didn't answer this email.

9 I Fight and Come To My Senses

In the days after this email, I gradually realized that everything in the design, while complicated, improved the user experience except the decision to use those strange magic folders. What program was intended to claim .texdist and come to possess those folders? The documentation didn't say, but I had my suspicions. Probably a rival front end!

I wrote impassioned email arguing that the data structure ought to be front end neutral, but got no response. Then I had an inspiration. By adding a single line to the TeXShop code, I got *it* to claim .texdist. Suddenly all those folders changed to files with TeXShop's icon. Victory!

Except that I didn't want to claim those folders; no front end should do that. I wrote a final email saying "here's what I did, and clearly it makes no sense. Why can't you see this?"

At that point, a magical thing happened. I had been getting email from Gerben, but nothing from Jérôme. Now Jérôme managed to get in contact by an indirect route and it turned out that the University of Oregon mail system had been rejecting his email. Jérôme agreed that front ends shouldn't claim .texdist, and instead had written a small application which claimed those files. When one of the files was opened in this small application, the application called the TeX distribution preference pane to select that distribution.

So in a single email, I learned that users could avoid /Library/TeX/Distributions, that the magic folders were benign, and that a preference pane existed which met Apple's high standards for simplicity and usability. It was an exciting day.

◇ Richard Koch
2740 Washington St
Eugene, Oregon
USA
koch@math.uoregon.edu
http://uoregon.edu/~koch/

CTAN Package Sourcing

Jim Hefferon
`ftpmaint@tug.ctan.org`

I have some experimental software to improve the way in which packages are added to the Comprehensive TEX Archive Network (CTAN).

1 Now

CTAN is run at three different sites, one in Germany, one in Britain, and one in the US. "Run" means that any adding, deleting, or moving of files happens at one of these three. Custom software, written by Rainer Schöpf, ensures that a change at one is reflected at the other two quickly, within fifteen minutes. (The other hundred CTAN mirrors go at a different pace, usually syncing nightly.)

New or updated material can reach CTAN in three ways. Most often it is uploaded through a web form. However, some authors use FTP uploads and some packages are mirrored over from their home site. I will focus on the web uploads.

In the present system, receipt of a web upload triggers an email to the CTAN maintainer's mailing list. On reading that, the maintainer at the site that received the material will handle the upload. This means unpacking the `.zip` or `.tar.gz` bundle in which the package was sent, examining the resulting files, and checking details such as license and placement. It means emailing the author and the other maintainers. It may mean asking for some adjustments such as adding a `README` file, or some more technical work such as changing text file line endings. After that, the maintainer places the material into the archive at their machine and triggers mirroring by the other two core sites, and so ultimately by the additional mirrors.

But placing a package's files into the archive does not end its processing. For one thing, information about the package such as description and license — the package's metadata — needs to go into the *Catalogue* (this is done by Robin Fairbairns). Finally, distributions such as MiKTEX and TEX Live repackage the material to meet the TEX Directory Standard (TDS) and deliver this to end users.

Here are some worries about the current process. (1) Package's authors cannot directly edit the metadata, so that must be done by the maintainers. (2) There are delays: one example is that package metadata often gets into the database only after the files are in the archive so there is a period where the description does not match the package, and another example is that the web pages for the archive at `http://www.ctan.org/tex-archive` are often only regenerated nightly so information about new materials is not current. (3) To be a core maintainer a person needs to run a server and there are people who could help with the archive but who oughtn't be administering a system that is exposed to the Internet (this includes me). (4) The package gets installed by the maintainer whose site happened to receive the upload, so if that person is unavailable then there is a delay. (5) At the time the package is put in the archive and announced, that package should be easy for end users to get into their distribution tree.

Before describing the proposed upload process, let me first counterbalance the prior paragraph by noting a strength of the current process. At an archive such as SourceForge, where package authors are responsible for what is put out to the public, many authors do not do a good job. The fact that CTAN's maintainers are involved in all installations ensures that packages meet some standards. That is, the current process is a wide-mouthed funnel, catching submissions and narrowing them to a reasonably uniform offering.

2 Developments

Users groups, notably Dante, have sponsored very helpful discussions of CTAN issues. In response, I have been working on software that is now being tested. Some people have expressed interest and the outline below will explain its current state.

If you are not keen on CTAN internals then probably the most interesting feature is also the most experimental: the attempt to provide installable bundles. The TEX Live team has a script to bring most packages from the CTAN tree over to a TEX Directory Standard layout, that is, over to a format that could be dropped by an end user into an existing installation. My software wraps that to make available the TDS-ready material as a bundle `<`*package id*`>.tds.zip` at the same time that the bundle's source is put up. Users can drop this `.tds.zip` file right into their distribution, without much need for installation instructions. (This does not integrate with any package manager but it does allow users to easily place desired material.)

I will explain the software by walking through the steps that a typical package takes to get from author to archive.

1. The author puts the package into a `.zip` or `.tar.gz` bundle. They point their browser to the upload page.

   There, they choose from two upload forms. The simple one, the default, asks only for name, license, and description. The complex form allows the author to enter obscure attributes such as package home page.

If this is an update of an existing package then the form comes up with the metadata now in the database, and the author just makes any changes. Also, the author is asked separately for additional information such as any handling instructions (now, the description and handling information all goes in one box).

2. The system accepts the uploaded package and metadata. It places the metadata in the database, in a pool of not-yet-processed uploads. It emails a list of people who can install uploads, called here "editors."
3. The uploaded bundle is unpacked to a file tree by a program that runs periodically. This program also does some processing of the unpacked files. For example, it ensures that text file line endings are the same throughout the archive.
4. One of the editors sees the email notification and logs into a web site listing the pool. They click on a form to claim responsibility for this package.
5. This editor examines, possibly edits, and then approves the metadata left by the contributor. (Requiring that metadata be approved ensures that the descriptions meet some standard of comprehensibility, and also keeps us out of the password business while reassuring authors that people they don't know cannot change the package's description.)

   The editor can read, add, delete, or rename files. For instance, they can add a `README` file if one is not already there.

   The system will warn the editor of some problems, for instance if the metadata says that a documentation file exists but there is not one in the upload, or if installing a package update will leave the archive with dangling links.
6. The uploader may have included a `.tds.zip` bundle. If so, the editor can see its contents and compare with what TEX Live now has for this package. The editor can also push a button to make a new `.tds.zip`, using TEX Live's script. If the package is suitable then it can be queued for placement into the TEX Live repository.
7. The editor then pushes a button to install the package.

   That puts the source files to the archive, say at `/macros/latex/contrib/<`*package id*`>`. Installation is done using the metadata, so the database and the archive tree are consistent as to the location, whether a `.zip` file exists of the directory contents, etc. The system places the files using CTAN's current custom script, ensuring that these web-based installations are consistent with command-line installations.

   The installation system also tends the database: it updates the metadata and the searchable documentation.

   If the package now contains a TDS bundle then the system places it in `/install/macros/latex/contrib/<`*package id*`>.tds.zip`.
8. The system sends an email to the editors list, informing people that the upload was handled and also for possible forwarding to the CTAN announcement mailing list.
9. If a `.tds.zip` bundle was queued for transmission to the TEX Live repository then it gets sent at some later time. (One advantage of using a queue over having it happen at the same time as the install is in case of problems getting to the TEX Live site.)

Material that comes in via FTP goes through the same process, starting at step 3. (The editor can associate existing metadata with the upload).

3 To do

Not every feature of the experimental system is described above (for instance, there is a way to send changes to the metadata alone). And, because it is experimental, probably some of what is described will get changed. In particular, while the TDS feature appears promising, it is quite experimental.

So the upload process still faces a fair number of hurdles, both technical and nontechnical. For one thing, where the current system is like a wide-mouthed funnel, the process described above has not been subject to any real-world testing for the same property. However, all the features described above exist, are now being developed and tested, and seem to solve at least some of the problems with the current package process.

4 Acknowledgement

I thank Karl Berry for helping me with the TEX Live material.

PDF Overview

Everything you wanted to know about PDF, but were afraid to ask!

Leonard Rosenthol
PDF Standards Evangelist
Adobe Systems, Inc.

PDF - Portable Document Format

- File format designed specifically for electronic distribution of “final form documents”
- Created by Adobe in 1992-1993, as part of their Acrobat product.
- PDF is an open public format with specs available from Adobe at <http://partners.adobe.com/public/developer/en/pdf/PDFReference17.pdf>
- Coming soon as an ISO Standard (ISO 32000)!!

PDF - What’s in there?

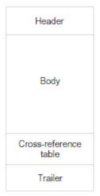
- PostScript/Adobe imaging model
  - Text & graphics in a device & resolution independent manner
- Bitmap Images
- Prepress Features (trapping, bleed, etc.)
- Navigation Tools (Bookmarks, Hyperlinks, etc.)
- Annotations
  - Text notes, “MarkUp”, Movies, Sounds
- Forms
- Security & Authentication
- And more…

PDF - What’s NOT in there?

- PDF is NOT Postscript!
  - Transparency
  - Non-printable elements (hyperlinks, etc.)
  - Interactive elements (multimedia, 3D, forms)
  - Modern compression technology (JBIG2, JPEG2000)
  - Color Management
  - No programming language constructs
  - Strict file structure allowing for random access
  - Presence of font metrics for viewing fidelity
- A PDF file can not be directly interpreted by a PS interpreter, though conversion of PDF page descriptions to PS is simplified.

PDF Document Layout

- Header
  - Specifies PDF version
- Body
  - Sequence of objects
- XREF
  - Where to find each object
- Trailer
  - Tells where to find XREF



Properties of PDF

- Adobe Imaging Model
- Portability
- Compression/Encryption
- Font Independence
- Random Access
- Incremental Update
- Extensibility

Adobe Imaging Model

- Same model as Postscript, where a page is drawn by “placing paint” on a selected area
  - “figures” can be letter shapes, regions defined by lines and curves or sampled images
  - Paint can be any color (specified in variable color spaces)
  - Figures can be clipped to any other figure/shape
  - Figures are “overlaid” on each other, in the order they exist in the page description.
- Plus Transparency/Opacity (PDF 1.4)

Portability

- PDF files are binary, all 8 bits can be used - though support for 7 bit files exists
  - ASCII-85 when needing to encode to 7 bits
- Single document format regardless of platform
- Non-Roman language support via standard encodings as well as Unicode

Compression/Encryption

- Support for a number of industry standard algorithms
  - JPEG (for color & grayscale images)
  - CCITT Group 3 & 4, and RLE for monochrome images
  - LZW & Flate (ZIP) for text, graphics, etc.
  - JBIG2, JPEG2000
  - RC4
    - 40 bits (4.0)
    - 56 bits (4.0.5)
    - 128 bits (5.0 and later)
  - RSA Public Key Cryptography – Digital Signatures
  - AES (7.0)

Font Independence

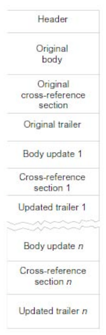
- Use of Font Descriptors
  - Name, character metrics (width, height, etc.)
- Support for font embedding
  - full & subset embedding
- To ensure correct display on all platforms and PDF viewers, you should embed.
  - Adobe Acrobat, however, provides font substitution facilities.

Random Access

- Cross reference table maintains lists of pages, objects on a page, etc.
- XRef is stored at the end of the document, allowing for single pass creation and ease of location
  - Except in the case of linearized documents designed for byte-serving (ie. dynamic serving via the web)

Incremental Update

- Modifications are written to the end of the file, leaving the original data intact
- A new xref table is written containing the new/modified data, and a link back to the old xref.
- Since original data is still present, support for multiple undos across save boundaries can be supported.
  - However Acrobat only provides a UI for this feature when Digital Signatures are used.

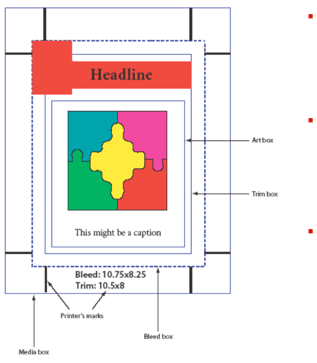


Extensibility

- As seen by the features added to PDF since 1.0, you can see that new features can easily be added to PDF w/o breaking backwards compatibility.
- A viewer will simply ignore an object that it doesn't understand.

Adobe Imaging Model

Page "Layout"



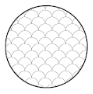
- Page Boxes
  - MediaBox is the physical size of the page
  - CropBox is the visible size of the page
  - BleedBox, TrimBox & ArtBox for prepress
- Content
  - All "marks" on a page are collectively referred to as content.
  - Content can only be placed on a canvas, which is defined as a "Page".
- User Space
  - A coordinate system that stays the same regardless of the output device
  - The Current Transformation Matrix (CTM) specifies the transformation from user space to device space
  - Default user space is 72 units per inch (aka "a point") with the origin at bottom left

PDF 1.0-> 1.5: 1/72 of an inch == 14,400 units == 200 inches.

PDF 1.6 & later: 15,000,000 inches == 1,250,000 feet = ~236.74 miles.

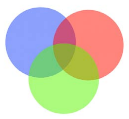
Color Spaces

- PDF Supports 11 color spaces
  - 3 device dependant
    - DeviceGray, DeviceRGB & DeviceCMYK
  - 4 device independent
    - CalGray, CalRGB, Lab & ICCBased
  - 4 special
    - Indexed
    - Pattern
      - Tiling & Smooth Shading
    - Separation (aka Spot colors)
    - DeviceN
      - NChannel (PDF 1.6) is a backwards compatible extension of DeviceN to enable richer and more accurate handling of color blending with dot gain and mixing hints



Transparency

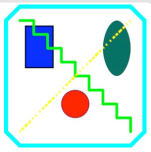
- Simple alpha channel on all objects
  - Separate fill vs. stroke opacity for vector/text
  - Images support a separate "alpha mask"
- Blending Modes
  - Specify how the blending of colors should take place between objects
  - Same choices as in Photoshop/Illustrator
    - Normal, Darken, Dodge, Burn, etc.
- Blending Groups
  - Allows multiple objects to be grouped together and their blending settings applied to the group as a whole
    - Similar to layers in Photoshop



Whenever shadows are dropped...
Wherever modes are blended...
...I'll be there!
–Transparency Gal
TG
TRANSPARENCY GAL

Graphic Objects

- Paths
  - Arbitrary shape made of straight lines, rectangles, and cubic curves.
  - May intersect itself and may have disconnected sections and holes.
  - Filled, stroked, and/or a clipping path.
- Text
  - One or more character strings
  - Filled, stroked, and/or a clipping path
- Raster Image
  - "bitmap" using a specified color model
  - May be rendered with a hard or soft mask
- Form XObjects
  - Reusable elements
- Postscript language fragments



Fonts

- The following types of fonts are supported
  - TrueType
  - Type 1
  - Type 3
  - Type 0 (Composite Font)
    - CIDFontType0 (Type 1)
    - CIDFontType2 (TrueType)
  - OpenType (PDF 1.6)
- There are 14 fonts that **USED TO BE** "built-in", including the Courier, Helvetica and Times families, Symbol & ITC Zapf Dingbats.
  - However, embedding them is still a good idea!

Font Embedding

- Two types of embedding restrictions
  - Technical
    - Fonts in TrueType or OpenType format contain a flag word (in the OS/2 table) that specifies whether embedding is allowed and if so what kind (installable, view & print, view, print & edit).
  - Legal
    - Fonts, like software programs, will include an End User License Agreement (EULA) that stipulates the uses that the creator permits the licensee/user. These may include the ability to embed, and if so, for what purposes.

Form XObject

- A form XObject is kind of like a "PICT" or WMF/EMF.
  - It's a named self-contained description of text, graphics or sample images that can be drawn more than once on a single page, or on multiple pages.
- Useful for things like company logos, imposition, etc.

Postscript Fragments

- Pieces of raw postscript that are ignored for screen render and MAY be included in any Postscript printing stream.
  - Can be used for device-specific features
    - Output tray
    - Duplexing
  - Not all PDF viewers will process these, and even those that do, offer choices to ignore them.
    - Acrobat 6 and later default "Enable Passthrough Postscript when Printing" to **OFF**
- Just say **NO**!

Optional Content

- Goal: Allow document content to be selectively viewed or hidden
- Called "Layers" in Acrobat UI
- Can apply to viewing, printing, exporting, etc.
- Graphical content can be declared as optional
  - e.g. Language, Zoom, Export, Print, View, User
- When opened in Acrobat 5 (or printed), ALL layers/content are visible!
- Has NOTHING to do with rendering order!

Prepress

- Transfer functions
  - Used to adjust color between an output device and the human eye
- Halftones
  - Used to approximate continuous-tone colors on a device with limited discrete colors
- Trapping
  - Used to address physical misalignment of plates on a high end printing device
- Separations
  - Composite - Standard PDF where a page refers to all colors that are used on it
  - Separated - PDF where each page of the documents represents the areas for a specific color/plate.
- OPI - Open Prepress Interface
  - Standard for providing low resolution "proxies" in a document that will be substituted at print time with the original (hi-res) versions.
    - Images
    - Form XObjects

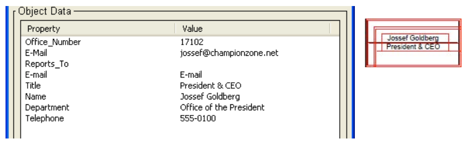
What else you got?

Metadata

- Document Info
  - Simple set of defined “keys”
  - Ability to add additional keys
  - Simple strings
- In PDF 1.4, any object can now have a block of XML metadata associated with it.
  - Uses an RDF-based grammar called XMP
  - Can include both standard and user-extended elements

User Properties
aka Object-level Metadata

- Introduced in Acrobat 7/PDF 1.6
- Enables selected groupings of content elements to have an associated set of data
  - Uses Marked Content within the stream
  - Uses Structure elements for the data
  - Consists of name/value pairs


Object Data

Property	Value
Office_Number	17102
E-Mail	jossef@championzone.net
Reports_To	
E-mail	E-mail
Title	President & CEO
Name	Jossef Goldberg
Department	Office of the President
Telephone	555-0100

Destinations and Actions

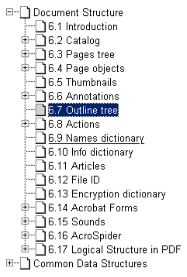
- A particular view of a document
  - Page to be displayed
  - Portion of the page to be displayed
  - Magnification (zoom) factor to use
- Can either be explicit or named
  - GoTo [1 /Fit]
  - GoTo [/MyName]
    - /MyName [1 /Fit]
- These are things that can be attached to certain objects/events in a document
  - Open & Close doc, close doc
  - Enter & Leave page
  - Mouse enter/leave/down/up
  - Bookmarks, annotations, form elements, etc
- They come in a few types:
  - GoTo, GoToR(emote), GoToE(mbedded)
  - URI (web link)
  - Launch
  - Sound, Movie
  - JavaScript
  - And more…

JavaScript

- Based on Netscape’s “SpiderMonkey”
  - Version 1.5
  - ECMAScript compliant
  - Includes all standard objects
    - Date, RegEx, Math, etc.
- Based around a different Document Object Model (DOM) than present in browsers
  - App, doc, field, annot, etc.
- Sandboxed
  - No access to file system, network, user info
  - Not all commands are available in all versions
    - Reader vs. Full
  - Not all commands available from all places
    - Console, menu, field, batch, etc.
- No integration with external/browser JS
  - But can be called FROM applications via COM or Apple events.
  - NOTE: Acrobat/Reader 7.0.5 (for Windows) now supports this ability!

Bookmarks

- Also called the “Outline Tree”
- Hierarchical List of destinations or actions through which the user can navigate.



```
Document Structure
  6.1 Introduction
  6.2 Catalog
  6.3 Pages tree
  6.4 Page objects
  6.5 Thumbnails
  6.6 Annotations
  6.7 Outline tree
  6.8 Actions
  6.9 Names dictionary
  6.10 Info dictionary
  6.11 Articles
  6.12 File ID
  6.13 Encryption dictionary
  6.14 Acrobat Forms
  6.15 Sounds
  6.16 AcroSpider
  6.17 Logical Structure in PDF
Common Data Structures
```

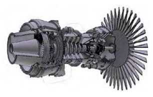
Transitions

- The visual transition between pages of a PDF can be specified for use in presentations
  - Split, Blinds, Box, Wipe, Dissolve, Glitter
  - PDF 1.5 adds another dozen varieties
- You can also specify a duration for how long the transition should last
- Using OCG's, it is also possible to achieve "inter-page" transitions, such as the PowerPoint "fade in" or "typewriter" effects.

Annotations

- Text Notes
- "Mark Up"
  - Underline, strikeout, highlight
  - Circle, Rect, Pen/Scribble
  - Callout, Dimensioning
- Hypertext Links
  - Inter-document, intra-document
  - URI's (web links)
- Stamps
- File Attachments
- Multimedia
  - AVI, MPEG, QT, Real, WMP, etc.
- Sounds
- 3D

This is a simple text annotation
Information Age Insecurity
Federal Government Information Security and the National Information Infrastructure

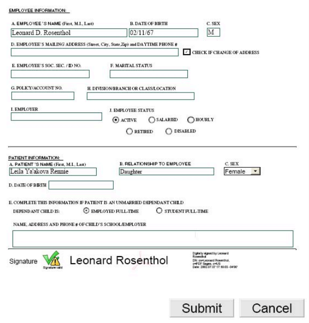


Embedded Files & Packages

- You can embed an entire file (or any type!) inside a PDF document
  - Useful for having a single file that contains everything related.
  - A Launch command, for example, might reference the embedded doc rather than an external reference.
- Embedded PDFs can refer to one another and offer "interdocument links" (PDF 1.6)
  - JavaScript can also interact with the embedded documents for data exchange
- Some file types may end up like a "roach motel" (what goes in, never comes out
  - Black ("bad") & White ("safe") Lists of file types
  - Stored in Registry. No UI in Acrobat for editing
- PDF Packages/Collections (1.7)
  - Extension to Embedded Files
  - Provides a cleaner UI on accessing the embedded content
  - Enables "spreadsheet view" of associated metadata

AcroForms

- A PDF file may contain AT MOST ONE AcroForm, though that form may contain any number of fields located on any page.
- You can also dynamically import/export sets of fields from the file
- There a number of predefined field types
  - Button (checkbox/radio/push)
  - Text
  - Choice (popup, combo or list)
- Fields can be typed (integer, string, boolean) and marked read-only.
- Fields can be "calculated", such that a JavaScript will be autoexecuted when a "related" field is modified.
- Form data can be "submitted" via either FDF (Forms Data Format), XFDF (an XML grammar) or via an HTTP get or post.



EMPLOYEE INFORMATION:

A. EMPLOYEE'S NAME (First, M.I., Last): Leonard D. Rosenthol
B. DATE OF BIRTH: 02/11/67
C. SEX: M
D. EMPLOYEE'S MAILING ADDRESS (Street, City, State,Zip) and DAYTIME PHONE #:
[✓] CHECK IF CHANGE OF ADDRESS
E. EMPLOYEE'S SOC. SEC. / ID NO.:
F. MARITAL STATUS:
G. POLICY/ACCOUNT NO.:
H. DIVISION/BRANCH OR CLASS/LOCATION:
I. EMPLOYER:
J. EMPLOYEE STATUS: [✓ ACTIVE] SALARIED HOURLY RETIRED DISABLED

PATIENT INFORMATION:

A. PATIENT'S NAME (First, M.I., Last): Leila Ya'akova Rennie
B. RELATIONSHIP TO EMPLOYEE: Daughter
C. SEX: Female
D. DATE OF BIRTH:
E. COMPLETE THIS INFORMATION IF PATIENT IS AN UNMARRIED DEPENDANT CHILD
DEPENDANT CHILD IS: [✓ EMPLOYED FULL-TIME] STUDENT FULL-TIME
NAME, ADDRESS AND PHONE # OF CHILD'S SCHOOL/EMPLOYER:

Signature Leonard Rosenthol

Digitally signed by Leonard Rosenthol
DN: cn=Leonard Rosenthol, o=PDF Sages, c=US
Date: 2002.07.07 17:18:03 -04'00'

XFA – XML Forms

- Introduced with PDF 1.5
- Created by Adobe LiveCycle Designer (Acrobat 7 Pro/Win)
- New and improved architecture for forms based around XML & industry standards
- Supports a variety of cool new features for PDF forms
  - Dynamic/Resizable forms
  - Automatic integration with databases & web services
  - Calendars for date fields
  - Digital Signatures on specific fields (not yet)
  - Etc.

Structured/Tagged PDF

- Introduced as part of the AcroSpider/Web Capture project (Acrobat 4).
- Allows for storing information about the logical structure of a document along side the layout.
  - An XML-like approach to marking sections of a PDF to provide for text reflow, data extraction, etc.
- Improved searching & indexing type operations
- Enables high fidelity export of PDFs to other formats (eg. Word/RTF)

Security & Digital Signatures

- Digital Rights Management (DRM)
  - Disallow Copy, Print, etc.
  - Low Resolution printing
  - Password protection
- Ability to use X.509 compatible signatures that can then be applied to a document.
  - PKCS1
  - PKCS7 (with or without a CA)
  - PKCS12
  - MSCAPI/Windows Certificate
  - Verisign
- TimeStamping (PDF 1.6)
  - Compatible with PKCS7 & RFC 3161
- Revocation (PDF 1.6)
  - Certificate Revocation Lists (CRLs) – RFC 3280
  - Online Certificate Status Protocol (OCSP) – RFC 2560

Certified Documents

- Digitally Signed
  - Invisible/blind signature
- Specifies (limited) Digital Rights
  - "recommend not required"
- Specifies set of objects (byte range) that are covered by the certification
  - So that some things (eg. form fields) can be modified w/o breaking certification
- NO ENCRYPTION!

Q & A

MathType 6.0's TEX input for MS Word and Wikipedia

Paul Topping
Design Science, Inc.
140 Pine Ave.
Long Beach, CA
USA
pault@dessci.com
http://www.dessci.com

Abstract

MathType is well-known for its point-and-click user interface for editing math. However, some users feel more comfortable typing math using TEX, so in MathType 6.0 we have added a TEX input mode. This provides the user with the best of both worlds: TEX for initial entry, point-and-click and drag and drop for easy editing and manipulation. Since MathType can save equations in several graphics formats and objects, it provides a direct path from TEX to Microsoft Word, PowerPoint, and virtually any document or application. Since many blogs and wikis accept a variant of TEX math syntax and expose it in their web pages, we are now able to support both authoring and reuse of equations in these environments. In particular, MathType users can now copy equations out of the thousands of Wikipedia pages containing equations for use in educational and research authoring. In addition, MathType users can create equations and paste them directly into new Wikipedia content.

1 Introduction

Throughout MathType's 20-year history, it has always been firmly in the point-and-click camp of equation editing. Because a trimmed-down version has shipped with Microsoft Office since 1991, it has been used to type a lot of math. Of course, TEX remains popular and is heavily used in some scientific communities. Once a person's hands "know" TEX, it is hard for them to imagine typing math any other way. And, because TEX is free and easy to integrate into web servers as an equation image generator, many blog and wiki applications support it. Unfortunately, many people that don't know TEX struggle with authoring math in these environments. With MathType 6.0, we tried to bridge both of these gaps, bringing TEX input to MathType and allowing people that don't know TEX to more easily work with wikis and blogs. Since Wikipedia is so popular and contains many equations authored in TeX, we have made working with its equations especially easy.

2 Typing TEX in a MathType window

Typing equations using TEX and LATEX math syntax in MathType is very easy. At any point in building up an equation, the user can type text starting with any one of these TEXmode starting characters: $ _ ^ (see Figure 1). TEX language will appear in dark grey as opposed to the normal black of converted math. Hit Enter and the TEX language is converted into normal MathType equation content (see Figure 2). Any errors appear in red. Corrections can be made using MathType's normal point-and-click editing facilities, or the conversion can be undone to allow corrections to be made in the original TEX. TEX can also be pasted into MathType via the clipboard.

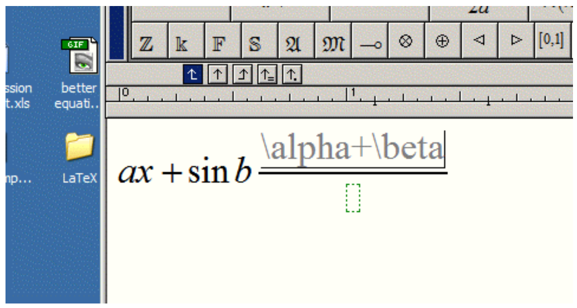
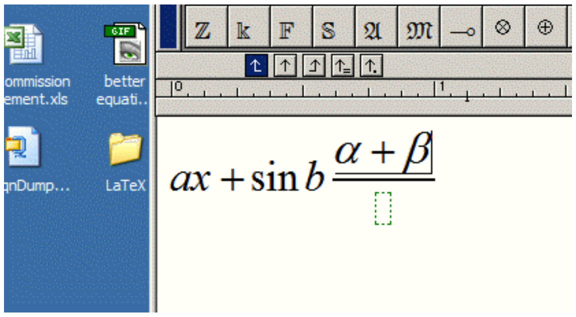


Figure 1: User types some TEX

3 Copying equations out of Wikipedia

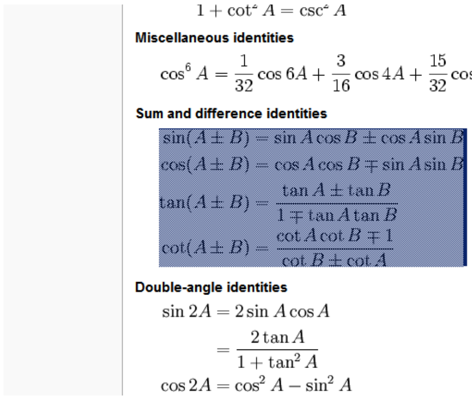
Wikipedia, the popular online encyclopedia, contains thousands of pages with equations represented in both HTML and as images. The math is authored



$$ax + \sin b\,\frac{\alpha + \beta}{\square}$$

Figure 2: After typing ENTER

using the Texvc subset of TEX markup with some extensions in LATEX and AMSLATEX. If simple enough, TEX input is converted to HTML markup. Otherwise, an image is generated with the TEX input stored in the image's ALT attribute. When an equation is copied from the web page onto the Windows clipboard, the math notation is made available to MathType where it is converted to MathType's own equation representation. This works with Internet Explorer, Firefox, and any browser that supports Microsoft's HTML clipboard format (see Figures 3 and 4). A stack of HTML and/or TEX equations can be copied in a single operation. We have also added a Texvc output translator allowing equations to be created in MathType, or copied from equations in Microsoft Word documents, and pasted into Wikipedia pages.



$$1 + \cot^2 A = \csc^2 A$$

Miscellaneous identities

$$\cos^6 A = \frac{1}{32}\cos 6A + \frac{3}{16}\cos 4A + \frac{15}{32}\cos 2A + \frac{5}{16}$$

Sum and difference identities

$$\begin{aligned}\sin(A \pm B) &= \sin A\cos B \pm \cos A\sin B\\ \cos(A \pm B) &= \cos A\cos B \mp \sin A\sin B\\ \tan(A \pm B) &= \frac{\tan A \pm \tan B}{1 \mp \tan A\tan B}\\ \cot(A \pm B) &= \frac{\cot A\cot B \mp 1}{\cot B \pm \cot A}\end{aligned}$$

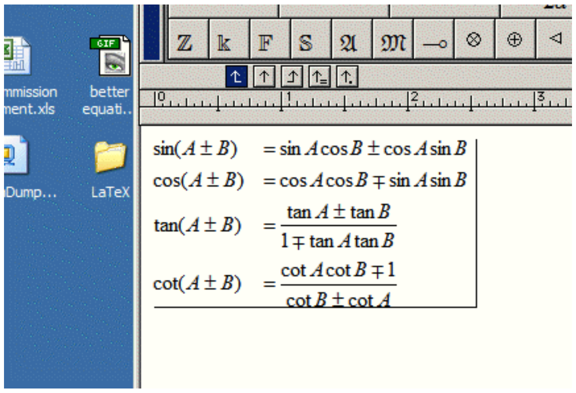
Double-angle identities

$$\begin{aligned}\sin 2A &= 2\sin A\cos A\\ &= \frac{2\tan A}{1 + \tan^2 A}\\ \cos 2A &= \cos^2 A - \sin^2 A\end{aligned}$$

Figure 3: User copies from Wikipedia's Trignometry page, `http://en.wikipedia.org/wiki/Trignometry`.

4 Implementation

MathType's translation process involves two separate translators working together, one for HTML math and one for TEX. The translators are written in a home-grown, rule-based language named Sevilla after the Spanish Restaurant on the lower



$$\begin{aligned}\sin(A \pm B) &= \sin A\cos B \pm \cos A\sin B\\ \cos(A \pm B) &= \cos A\cos B \mp \sin A\sin B\\ \tan(A \pm B) &= \frac{\tan A \pm \tan B}{1 \mp \tan A\tan B}\\ \cot(A \pm B) &= \frac{\cot A\cot B \mp 1}{\cot B \pm \cot A}\end{aligned}$$

Figure 4: After pasting into MathType.

floors of Design Science's Long Beach, California offices. Since the translation is defined using rules in text files, the existing translators can be customized and entire new translators may be created.

5 Current status

As of this writing, MathType 6.0 for Windows is in beta with release expected in July, 2007. The same technology will also be brought into a future Macintosh version. In the future, we expect to be looking at creating smooth interfaces between MathType and other web-based math environments.

Dual Presentation with Math from One Source using GELLMU

Preliminary Version of Talk for the TUG Meeting in San Diego, July 2007

William F. Hammond

Department of Mathematics & Statistics
University at Albany
Albany, New York 12222 (USA)
Email: `hammond@math.albany.edu`
Web: `http://www.albany.edu/~hammond/`

1 Introduction

A contemporary author writing an article for "dual presentation" has in mind both the classical printed presentation of an article and the modern web form of an article based on HTML.

There are two main approaches for achieving dual presentation that are relevant to the TEX community.[1]

- Write a LATEX article, and use a program that translates to HTML.
- Write an article in a suitable XML document type, such as *DocBook* or *TEI*, and use standard software for generating LATEX and HTML.

Both methods present challenges to authors who have been accustomed to using LATEX.

Since mid-2002 the second-generation form of HTML that supports mathematical content has been supported by the two most widely deployed web browsers, but not many articles seem to have appeared on the web in this form so far. The most likely reason is difficulty of production.

This talk will address the use of "generalized LATEX" to produce dual content from a single LATEX-like source. This method combines the reliability of XML document transformation with many of the conveniences available when writing LATEX markup.

2 Writing Source Markup

Here's the source for a relatively simple example.

[1] *Texinfo*, the language of the GNU Documentation System, also provides a route for dual presentation of articles without mathematical markup.

```
The following identity may be regarded as a formulation of the
Weierstrass product for the Gamma function.
\[ \int_{0}^{\infty} t^x e^{-t} \frac{dt}{t} \int:
   = \frac{1}{x}
     \prod_{k=1}^{\infty}
       \frac{\bal{1 + \frac{1}{k}}^x}{\bal{1 + \frac{x}{k}}}
     \prod: \]
Understanding the derivation of this identity is reasonable for
a bright student of first year undergraduate calculus in the
United States.
```

It compiles to this:

> The following identity may be regarded as a formulation of the Weierstrass product for the Gamma function.
> $$\int_{0}^{\infty} t^x e^{-t} \frac{dt}{t} \;=\; \frac{1}{x} \prod_{k=1}^{\infty} \frac{\left(1 + \frac{1}{k}\right)^x}{\left(1 + \frac{x}{k}\right)}$$
> Understanding the derivation of this identity is reasonable for a bright student of first year undergraduate calculus in the United States.

The markup looks like LaTeX. In fact, except for the use of the zone closers `\int:` and `\prod:`, it would be LaTeX. I call it *generalized LaTeX*.

Aside from the zone closers, which are required for GELLMU, there are a few other things to notice. Most of these have to do with the fact that, as part of the overall system design, markup semantics are independent of the command vocabulary. Thus, for example:

1. Command arguments must be explicitly braced.
2. Braces for the argument of a superscript or subscript may be omitted only if the argument is a single character.
3. The semi-colon at the end of a command name (such as `\latex;` above) indicates that the command does not introduce content. Often this type of semi-colon may be omitted, and, beyond that for most purposes `\foo;` may be regarded as shorthand for `\foo{}`.
4. The command vocabulary differs somewhat from that of LaTeX.

3 Another Example

There follows a GELLMU version of an example posted to the UseNet newsgroup `sci.math.research` on 29 October 2002, message id: `<apmpvn$bpb$1.repost@nef.ens.fr>`[2], by David Madore of ENS for the purpose of comparing TeX markup to MathML.

In a letter to Godfrey Harold Hardy, Sṟīṉivāsa Rāmāṉujaṉ Aiyaṅkār asserts that

$$\frac{1}{1+\frac{e^{-2\pi\sqrt{5}}}{1+\frac{e^{-4\pi\sqrt{5}}}{1+\frac{e^{-6\pi\sqrt{5}}}{\cdots}}}} \;=\; \left(\frac{\sqrt{5}}{1+\sqrt[5]{5^{3/4}\left(\frac{\sqrt{5}-1}{2}\right)^{5/2}-1}} - \frac{\sqrt{5}+1}{2} \right) e^{2\pi/\sqrt{5}}$$

[2]URI: http://groups.google.com/group/sci.math.research/msg/3bebf506912cf426

The markup for this:

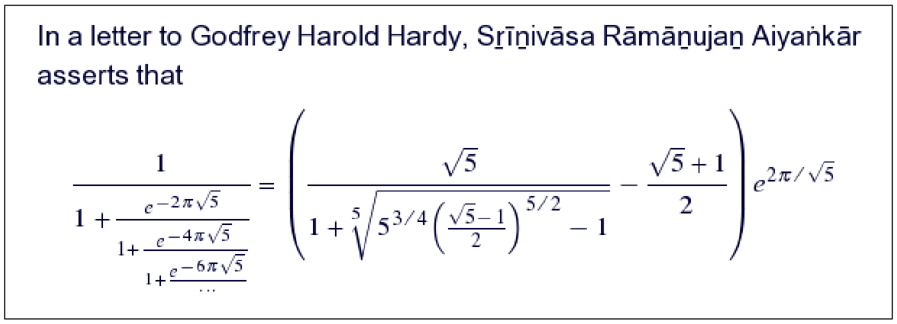
```
\macro{\=}{\ovbar}
\macro{\.}{\ovdot}
\newcommand{\b}[1]{\unbar{#1}}
There follows a \abbr{GELLMU} version of an example posted to the
UseNet newsgroup \quostr{sci.math.research} on 29 October 2002, message id:
\href{http://groups.google.com/group/sci.math.research/msg/3bebf506912cf426}{
\quostr{<apmpvn\$bpb\$1.repost@nef.ens.fr>}}, by David
Madore of ENS for the purpose of comparing \tex; markup to
\abbr{MathML}.

\begin{quotation}
In a letter to Godfrey Harold Hardy, S\b{r}\={\i}\b{n}iv\={a}sa
R\={a}m\={a}\b{n}uja\b{n} Aiya\.{n}k\={a}r asserts that
\[
\frac{1
}{1+\frac{e^{-2\pi\sqrt{5}}
}{1+\frac{e^{-4\pi\sqrt{5}}
}{1+\frac{e^{-6\pi\sqrt{5}}
}{\ldots}}}}
=
\bal{\frac{\sqrt{5}
}{
1+\sqrt[5]{5^{3/4}\bal{\frac{\sqrt{5}-1}{2}}^{5/2}-1}}
-\frac{\sqrt{5}+1}{2}}
e^{2\pi/\sqrt{5}}
\]
\end{quotation}
```

Note the use of `\bal{ ... }` instead of the LaTeX usage `\left( ... \right)`. GELLMU has various balancers of this type and will eventually have more. This is related not only to the fact that the markup is simply a "front" for an SGML document type but also to its notion of sound mathematical semantics. The processor for XHTML+MathML output will not tolerate unbalanced balancing characters in a math zone except as provided through these balancers and also through the list generator

`\vect[...]{...}{...} ... {...}` .

The kinds of weak enforcement of mathematical semantics represented by such balancing provisions and by the requirement for explicit ending of *sum*, *int*, and *prod* containers is prelude to future optional incorporation of stronger mathematical semantics in the markup.

Madore is correct in suggesting that one doesn't want to look at the MathML markup for this, but the rendering by *Firefox*, somewhat enlarged, is captured in this screenshot:


In a letter to Godfrey Harold Hardy, Sṟīṉivāsa Rāmāṉujaṉ Aiyaṅkār asserts that

$$\frac{1}{1+\frac{e^{-2\pi\sqrt{5}}}{1+\frac{e^{-4\pi\sqrt{5}}}{1+\frac{e^{-6\pi\sqrt{5}}}{\cdots}}}} = \left(\frac{\sqrt{5}}{1+\sqrt[5]{5^{3/4}\left(\frac{\sqrt{5}-1}{2}\right)^{5/2}-1}} - \frac{\sqrt{5}+1}{2}\right)e^{2\pi/\sqrt{5}}$$

4 The Importance of XHTML+MathML

There are several reasons why it is important to have articles and course materials with mathematical content online in modern HTML, i.e., XHTML+MathML.

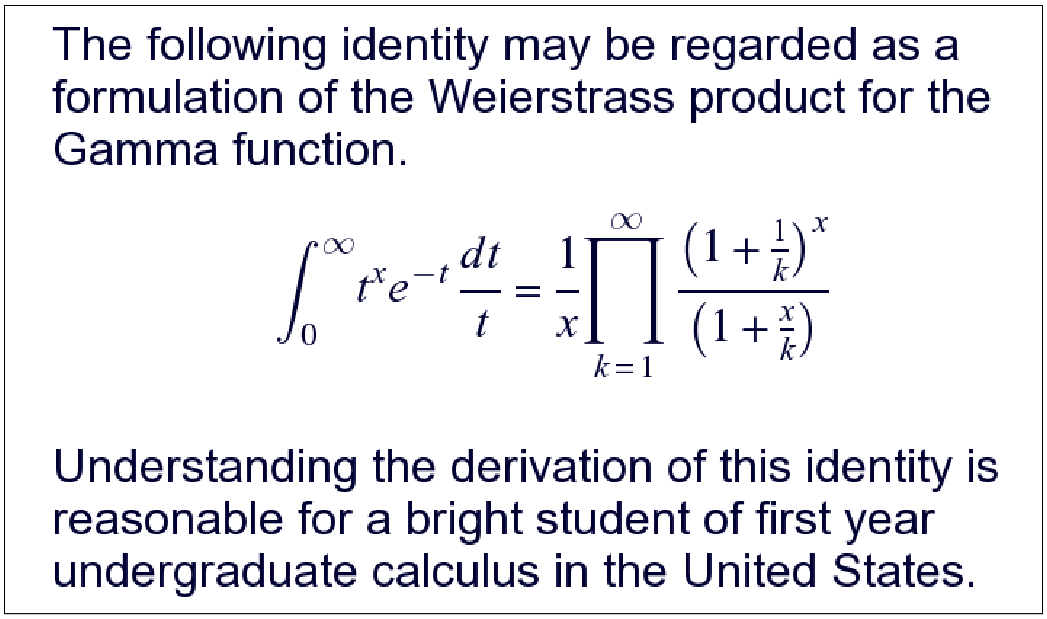
4.1 Public relations for Mathematics

- To a young person XHTML+MathML represents "math on the web".
- It fits the paradigm of web browsing.
- It's more flexible and more convenient for online reading than PDF — doubly so by comparison with double-column online PDF.
- The journals are disappearing from our libraries.

4.2 Special Needs

- It's great for proof reading. (Enlarge and shorten the lines.)
- Articles presented in XHTML+MathML comply with web accessibility guidelines. (PDF with math does not.)
- **Large print editions** at no cost.

  The small gamma bit presented earlier may easily be made to look like this in *Firefox* (a screenshot):


The following identity may be regarded as a formulation of the Weierstrass product for the Gamma function.

$$\int_0^\infty t^x e^{-t}\,\frac{dt}{t} = \frac{1}{x}\prod_{k=1}^{\infty}\frac{\left(1+\frac{1}{k}\right)^x}{\left(1+\frac{x}{k}\right)}$$

Understanding the derivation of this identity is reasonable for a bright student of first year undergraduate calculus in the United States.

5 Compiling an Article

5.1 Acquiring the Software

GELLMU is based on cross-platform free software licensed under the GNU GPL. Its package is available from CTAN ([2]) and from the GELLMU web site ([6]). The package requires a number of other cross-platform free software packages including *GNU Emacs*, *Perl*, and some standard items of SGML/XML software, chiefly *Open SP*.

Linux: The required packages are generally part of a full GNU/*Linux* distribution. The package should be installed in `/usr/local/gellmu` and symlinks to driver scripts should be made from a suitable place in one's command path.

MacOS-X and other Unix variants: The only difference from GNU/*Linux* is that some of the supporting packages may need to be acquired and installed.

MS Windows: The best strategy is to acquire and install a **full** *Cygwin*[3] distribution. Then proceed as with *Linux*.

5.2 Procedure

Let's package the preceding markup segment as an article. Because it is GELLMU, not LaTeX, it begins with `\documenttype` rather than with `\documentclass`.

```
\documenttype{article}
\title{}
\begin{document}
```

[3]URI: http://www.cygwin.com/

```
The following identity may be regarded as a formulation of the
Weierstrass product for the Gamma function.
\[ \int_{0}^{\infty} t^x e^{-t} \frac{dt}{t} \int:
   = \frac{1}{x}
     \prod_{k=1}^{\infty}
       \frac{\bal{1 + \frac{1}{k}}^x}{\bal{1 + \frac{x}{k}}}
     \prod: \]
Understanding the derivation of this identity is reasonable for
a bright student of first year undergraduate calculus in the
United States.

\end{document}
```

Beyond early-stage syntatic processing the system requires that there be a *title* in the preamble of every *article*. An empty *title* is allowed.

Text normally must be in paragraphs. (There are exceptions.) Therefore, the blank line after `\begin{document}` is essential.

It is sometimes said about LaTeX that a blank line *ends* a paragraph. However, in GELLMU a blank line *begins* a paragraph.

We place this article text in a file named `gammabit.glm`, with `".glm"` the canonical suffix for a GELLMU source file, enter the command

```
mmkg gammabit
```

and prepare to read the scroll[4]. At the end when all goes well there are the following outputs:

- XHTML+MathML – the best online version[5]
- PDF – for widely distributable print[6]
- DVI – for TeXies[7]
- classical HTML – for challenged browsing[8]

Additionally one might note that some level of rendering based on *cascading style sheets (*CSS*)* is possible for the author-level XML[9].

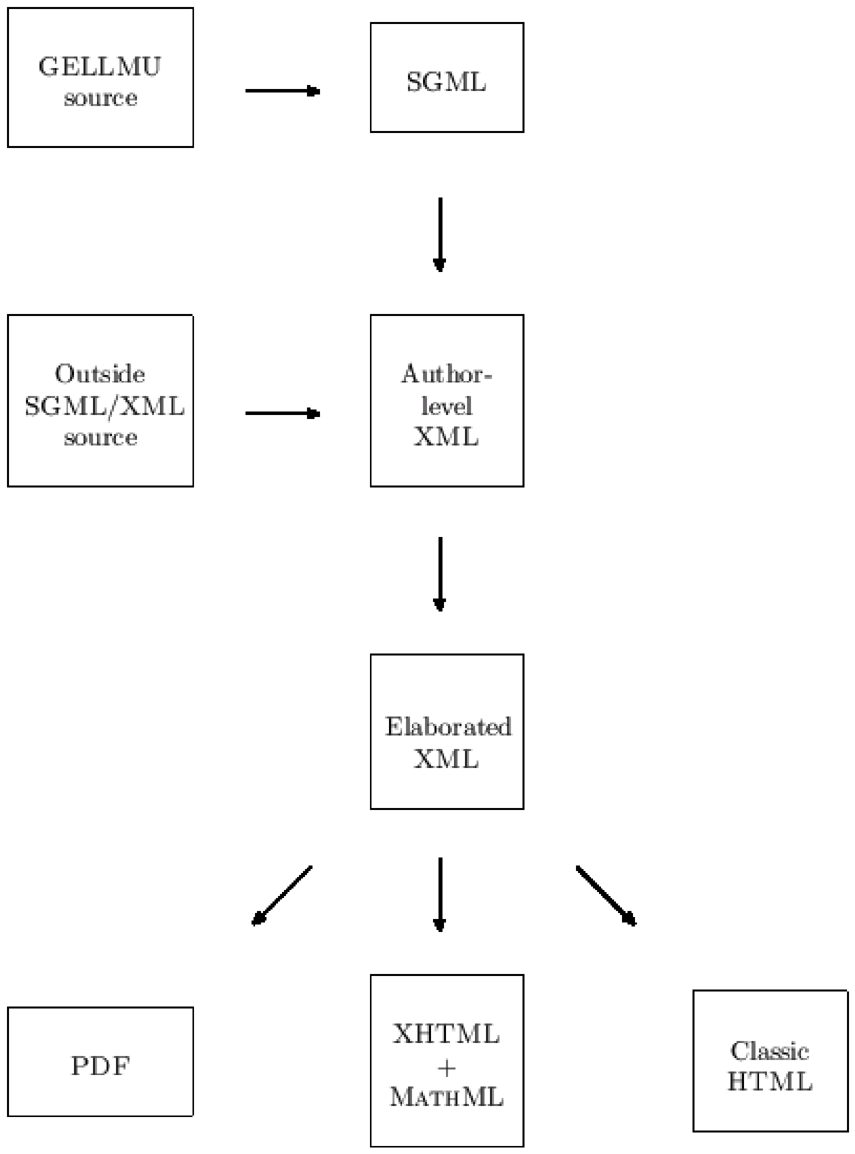
In order to understand the scroll one needs to understand the system design.

6 System Components

Regular GELLMU is a system assembled from modular components. Each step along the way produces an intermediate stage output that has its own sense and that, when things go wrong, provides opportunity both for diagnosis and intervention.

[4]URI: gammabit.out
[5]URI: gammabit.xhtml
[6]URI: gammabit.pdf
[7]URI: gammabit.dvi
[8]URI: gammabit.html
[9]URI: gammabit.xml

Here's the flow chart:



What I call the "side door" is the second row entry showing the possibility of translation from source languages other than the markup of regular GELLMU into the author-level XML documenttype corresponding to the source markup of regular GELLMU.

One will note from the scroll[10] that SGML/XML validation is done at several stages. This validation can be important for catching the author's mistakes. When there are error messages, it is possible and important to consult the scroll's last message regarding the stage of processing. Note in this regard that the translation from elaborated XML to XHTML+MathML takes place in 3 stages that are not shown on the chart but that may be seen in the example scroll.

[10]URI: gammabit.out

7 Further Information

Much more information may be found in the *User Guide* ([3]), the GELLMU *Manual* ([4]), and the GELLMU web site, `http://www.albany.edu/~hammond/gellmu/`. A link to an online version of this preliminary document should be available at the GELLMU web site by July 6.

References

[1] William F. Hammond, "GELLMU: A Bridge for Authors from LaTeX to XML", *TUGBoat: The Communications of the TeX Users Group*, vol. 22 (2001), pp. 204–207; also available online at `http://www.tug.org/TUGboat/Contents/contents22-3.html`.

[2] GELLMU at CTAN:
`http://www.tex.ac.uk/tex-archive/help/Catalogue/entries/gellmu.html`

[3] William F. Hammond, "Introductory User's Guide to Regular GELLMU", `http://www.albany.edu/~hammond/gellmu/igl/userdoc.xhtml` (PDF[11]).

[4] William F. Hammond, "The GELLMU Manual", `http://www.albany.edu/~hammond/gellmu/glman/glman.xhtml` (PDF[12]).

[5] "New York Journal of Mathematics Articles in Mathematically-Capable HTML"[13]; demonstration versions of past articles from *The New York Journal of Mathematics* ported from classical LaTeX using GELLMU.

[6] The GELLMU web:
`http://www.albany.edu/~hammond/gellmu/`

[11] URI: http://www.albany.edu/~hammond/gellmu/igl/userdoc.pdf
[12] URI: http://www.albany.edu/~hammond/gellmu/glman/glman.pdf
[13] URI: http://math.albany.edu/demos/nyj/

Design Decisions for a Structured Front End to LaTeX Documents

Barry MacKichan
MacKichan Software, Inc.

July 19, 2007

1 Logical design

Scientific WorkPlace and Scientific Word are word processors that have been designed from the start to handle mathematics gracefully. Their design philosophy is descended from Brian Reid's Scribe,[1] which emphasized the separation of content from form and was also an inspiration for LaTeX.[2] This *logical design* philosophy holds that the author of a document should concern him- or herself with the content of the document, and with identifying the *role* that each bit of text plays, such as a header, a footnote, or a quote. The details of formatting should be ignored by the author, and handled instead by a predefined (or custom) style specification.

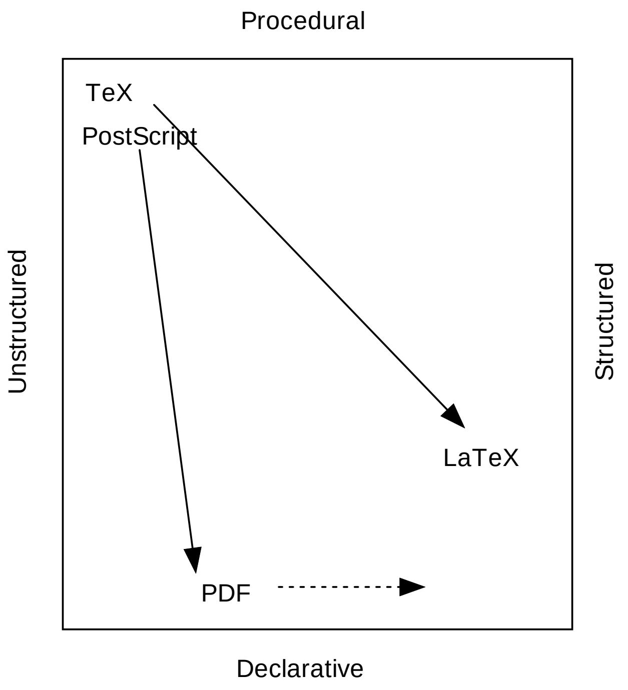
There are several very compelling reasons for the separation of content from form.

- The expertise of the author is in the content; the expertise of the publisher is in the presentation.
- Worrying and fussing about the presentation is wasted effort when done by the author, since the publisher will impose its own formatting on the paper.
- Applying format algorithmically is the easiest way to assure consistency of presentation.
- When a document is re-purposed it can be reformatted automatically for its new purpose. This can happen when a document is put on the Web in addition to being published, or even when the author sends the document to a new publisher.

The most powerful typesetting programs tend to be programming languages themselves. The two most prominent examples are PostScript and TeX. Although these are extremely powerful, they are not always simple, and they do not separate content from form. Consequently, there is a migration on this plot from the top to the bottom, and from the left to the right.

[1]Brian K. Reid, "Scribe: A Document Specification Language and its Compiler," Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA, Oct. 1980.

[2]Leslie Lamport, LaTeX: a document preparation system, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1989



Thus, PostScript is a powerful programming language, but it was later supplemented by PDF, which is not a programming language, but contains declarations of where individual characters are placed. PDF is not structured, but Adobe has been adding a structural overlay. LaTeX is quite structured, but it still contains visible signs of the underlying programmability of TeX, so I haven't quite placed it at the bottom of the plot. The pattern is that power and flexibility generally get supplemented or replaced in some circumstances with structured and declarative alternatives.

The original design philosophy for Scientific WorkPlace and Scientific Word was to make visual word processors that live at the bottom right of the diagram, and produce their output by generating LaTeX using one of over a hundred typesetting styles. This is the optimal solution for publishing, at least when we support a publisher's style, or when a publisher's style uses the same tags as one of the standard LaTeX document types.

2 Enter the customer

Although this philosophy works very well for publishing, many of our customers want to have greater control over the appearance of their documents. The truth is that not all mathematical documents are written for publication in a journal. The author might want to post a document on the Web or to send out preprints, or to prepare reports that will not be published, or to prepare handouts for students. The cold hard truth is that programs like Microsoft Word – despite its intellectual roots being also in *Scribe* – have over the years encouraged users to fiddle and futz with formatting. The experts may all agree that the result is ugly, but the customer is the one who pays our salaries.

In the past, Scientific Word users had a hard time if they wanted to change or add to a style. The advice of our tech support staff has been:

- You don't want to do that
- You shouldn't do that
- You can use package X to do that
- You can rewrite the style file

We no longer give the first two responses, and our users are not going to be able to use the fourth bit of advice. Due to the large number of useful packages, we now encourage users to start with a standard LaTeX document type and to use packages. This works, but it is not the most elegant way to solve the problem, since you shouldn't have to write options for the `geometry` package in order to change a margin.

We also allow the user to enter snippets of raw TeX or LaTeX code in what we call a "TeX button" (which is how we enter "TeX" and "LaTeX") but this runs counter to the design philosophy, and can't address problems when a user wants to change, for example, how list items are generated (since the code to be added would be in the middle of code we have generated).

3 A statement of the problem

This discussion now allows a statement of the problem we are solving.

1. We want an internal form for our documents that is both rich and extensible, and a rendering engine that is rich enough to render a LaTeX document and which is extensible.
2. We want to convert a LaTeX document to our internal form in a way that is extensible and preferably uses standard, well-documented tools, and in particular does not require access to our source code.
3. We want to convert our internal form to LaTeX in a way that is extensible and uses standard tools, and does not require access to our source code.

Part of the motivation for not needing access to our source code is that extending these operations will be easier for us if it is not necessary to change and re-build C++ code in order to support a new tag or to change the behavior of a standard tag. The other part of the motivation is that if the tools are standard and well-documented, then advanced users can make their own changes.

3.1 Internal form of a document

Scientific Word has an internal form that is not LaTeX but looks superficially like LaTeX, and we have an adequate rendering engine for it. However, it is not extensible – that is, to extend it means rewriting C++ code and extending the rendering engine. To avoid this problem, to get the extensibility we need, we choose an internal form that is rich enough and extensible (and it must also be declarative and structured). The obvious candidate (at least in this century) is XML. We are basing future versions of our software on the Mozilla Gecko rendering engine for HTML and XML. Tags can be introduced at will, and CSS (Cascading Style Sheets) are used to determine how these tags appear on the screen.

Some of the features of Gecko that are very useful to us are:

- The rendering engine is open-source under a license that allows us to extend it if necessary.
- The rendering engine is rich and powerful (the program user-interface is in fact a Gecko document).
- XML is a standard that is easily converted to and from LaTeX.
- A powerful scripting language is integrated into Gecko.
- A technology (XBL – XML Binding Language) allows attaching behavior to (new) XML tags.
- A system of broadcasters and observers simplifies coordinating the behaviors of objects.
- Support for infinite undo and redo is built into the document-modifying functions.

3.2 Conversion from LaTeX

Scientific Word does not process TeX or LaTeX files with TeX. It simply determines the structure of the file by recognizing tags such as \section and \subsection. In the past, it has caused problems when users define their own macros: we did not recognize them and loaded the macro invocation as a TeX button. Beginning with version 5.5 (two years ago) we now run a version of the TeX macro processor, and we evaluate macros defined by the user, but we do not evaluate macros defined in LaTeX or any of the standard packages. The result should be a document that contains only the standard macros, and which can be read by Scientific Word.

We continue with this same approach in our new architecture, except that the definitions of the standard LaTeX macros converts them to XML. The resulting files are complicated, but most of the complication is in some utility macros that make the final macros quite easy to understand. Some sample code from one of these files is:

```
\def\out@begin@abstract{%
 \msitag{^^0a}\msiopentag{abstract}{<abstract>}
}
\def\out@end@abstract{%
```

```
 \msitag{^^0a}\msiclosetag{abstract}{</abstract>}
}
```

This is all that is required to convert the `abstract` environment to XML.

3.3 Conversion to LaTeX

The conversion to LaTeX is done using XSLT (XML Stylesheet Language Transformations). As the name implies, XSLT was designed as part of a method of applying styles to XML objects, which sometimes requires making some transformations or re-ordering the XML elements. It has evolved into a powerful standalone transformation language for XML documents. It can be used to transform XML into XML, or XML into text, which includes TeX.

Here is the XSLT rule that generates the `abstract` environment:

```
<xsl:template match="abstract">
  \begin{abstract}
    <xsl:apply-templates/>
  \end{abstract}
</xsl:template>
```

When XSLT finds the `<abstract>` tag, it generates `\begin{abstract}`, applies whatever rules are needed for the content of the tag, and generates `\end{abstract}` when it gets to the end of the `abstract` node. The tag may have attributes, which might affect the TeX generated, and the rules can depend on the context of the tag.

The point here is that it is relatively easy to add support for new tags, or to change the TeX that gets produced by a tag. In older versions of our products, these operations took place in compiled code, but now they are controlled by text files that can be replaced or modified without rewriting or recompiling C++ code. It is now feasible to support different flavors of TeX for Math Reviews, or to support something like Context.

The next section addresses the question of how you can tailor the on-screen presentation of a tag.

4 Some examples

4.1 Displaying 'LaTeX' on screen

This is a brief discussion of how you can display a new tag, such as `<latex/>` on the screen. This is done by using XBL. We'll skip lightly over the details.

In a CSS file there is a line that tells Gecko that special rules apply to this tag:

```
latex {
 -moz-binding: url("resource://app/res/xbl/latex.xml#latex");
}
```

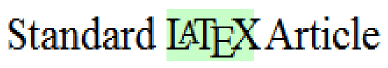
In the file `latex.xml`, there is a section that says how to display the tag:

```
<xbl:content>
  <sw:invis><xbl:children/></sw:invis>
 <sw:latex2>L<sw:latexa>A</sw:latexa><sw:latext>T</sw:latext>
  <sw:latexe>E</sw:latexe>X</sw:latex2>
</xbl:content>
```

Each letter in LaTeX (almost) is in a separate tag, which allows us to change the style for each letter. Here is the style rule for the 'A' (the tag `latexa`):

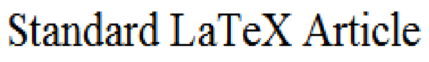
```
latexa {
  font-size: smaller;
  position: relative;
  bottom: .15em;
  left: -0.20em;
}
```

This rule shrinks the 'A' and moves it up and left. A style rule for the `latex2` rule changes the letter spacing to squeeze them together a bit. The final result on the screen is:



Standard LaTeX Article

Actually, what appears in the internal format is `<latex>LaTeX</latex>`. The content of the tag ('LaTeX') is thrown away, except when the XML document is viewed by some other browser, such as Internet Explorer, or even Firefox. Internet Explorer, when it sees the `-moz-binding` statement in the CSS file will ignore it completely. Firefox will understand it, but will be unable to find the `latex.xml` file, which is internal to our program. As a result, they will ignore the `latex` tag and will simply display the contents. Thus, the above displayed on Firefox will appear as:



Standard LaTeX Article

Of course, the LaTeX generated by this tag, no matter what its content, will be `\LaTeX`.

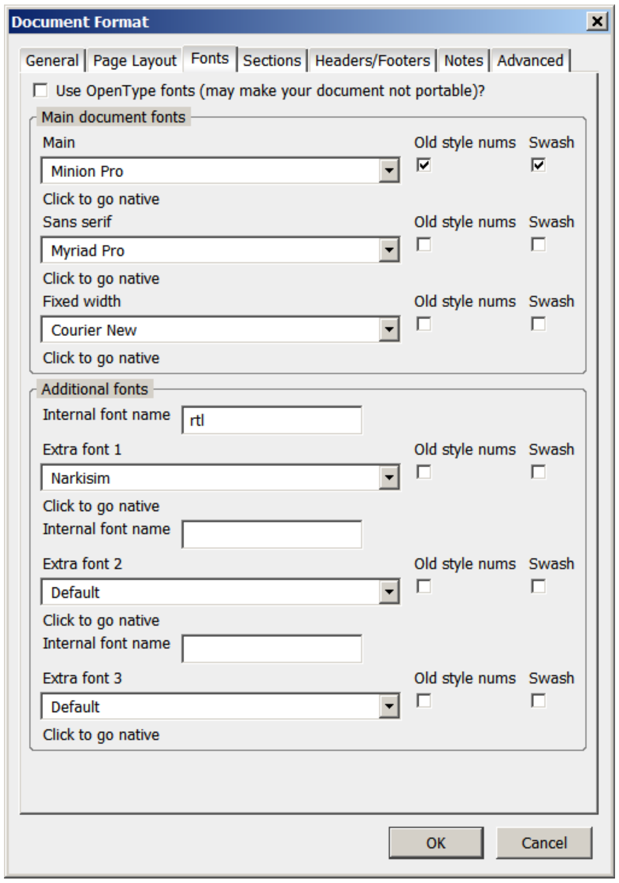
4.2 Spaces

LaTeX provides a wide choice of spaces, both horizontal and vertical. It is possible to make them visible by selecting a menu item "Show Invisibles." This is accomplished in the same way as the above example, with special CSS rules to apply in the case when "Show Invisibles" is on.

5 User interface enhancements

The next two items are not particularly related to the new architecture for Scientific Word; rather, they can be looked on as one solution to the problem of converting a rich keyword-value interface to a friendlier (to the novice) dialog-based interface. The result is marginally less powerful, but still allows the advanced user to get access to almost all the features of the keyword-value interface.

The dialog shown is for selecting OpenType fonts. Before the user gets to this point, he will be warned that if he proceeds, his document will have to be compiled with XƎTEX and therefore will not be completely portable. Here is the dialog:



Document Format

General | Page Layout | Fonts | Sections | Headers/Footers | Notes | Advanced

☐ Use OpenType fonts (may make your document not portable)?

Main document fonts

		Old style nums	Swash
Main	Minion Pro	✓	✓
Click to go native			
Sans serif	Myriad Pro	☐	☐
Click to go native			
Fixed width	Courier New	☐	☐
Click to go native			

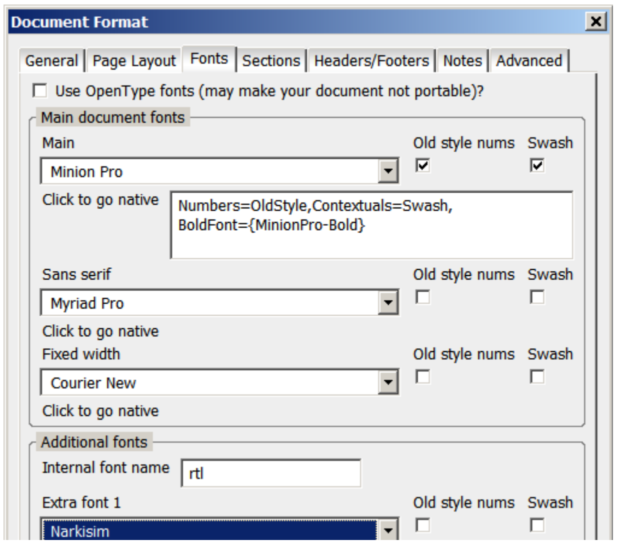
Additional fonts

		Old style nums	Swash
Internal font name	rtl		
Extra font 1	Narkisim	☐	☐
Click to go native			
Internal font name			
Extra font 2	Default	☐	☐
Click to go native			
Internal font name			
Extra font 3	Default	☐	☐
Click to go native			

OK   Cancel

This allows the user to pick the three main fonts: the main (roman) font, the sans serif font, and the monospaced font. He can also choose other fonts and give them names. We have a `rtlpara` tag for right-to-left text, and this uses the `rtl` font, for which the user has chosen Narkisim.

There are many font attributes, and many are not supported for many fonts, so we have chosen only two for access by checkboxes: old-style numerals and swash italics. Other attributes are accessible, but only by falling back on the keyword-value interface and clicking on the "Go native" link.


Document Format

☐ Use OpenType fonts (may make your document not portable)?

Main document fonts

Main: Minion Pro — Old style nums ✓ — Swash ✓

Click to go native:
```
Numbers=OldStyle,Contextuals=Swash,
BoldFont={MinionPro-Bold}
```

Sans serif: Myriad Pro — Old style nums ☐ — Swash ☐

Click to go native

Fixed width: Courier New — Old style nums ☐ — Swash ☐

Click to go native

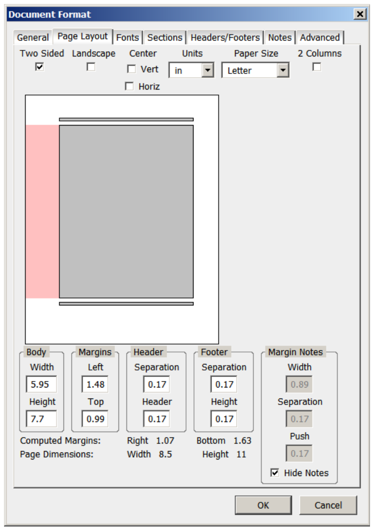
Additional fonts

Internal font name: rtl

Extra font 1: Narkisim — Old style nums ☐ — Swash ☐

The first line in the “Go native” box was provided automatically since the user had clicked on “Old style nums” and “Swash”. The user added the next line to use the MinionPro-Bold font as his bold Roman font rather than the default Semibold. This interface allows almost complete access to the power of the Fontspec package but gives more casual users the ability to choose basic fonts easily.

Another dialog interface to package options is the page layout dialog:


Document Format

Two Sided ✓ — Landscape ☐ — Center: ☐ Vert ☐ Horiz — Units: in — Paper Size: Letter — 2 Columns ☐

Body: Width 5.95, Height 7.7

Margins: Left 1.48, Top 0.99

Header: Separation 0.17, Header 0.17

Footer: Separation 0.17, Height 0.17

Margin Notes: Width 0.89, Separation 0.17, Push 0.17, ✓ Hide Notes

Computed Margins: Right 1.07, Bottom 1.63

Page Dimensions: Width 8.5, Height 11

Here the user is adjusting the left margin by pressing or holding the up or down arrow key in the left margin width field.

6 Conclusion

Scientific WorkPlace and Scientific Word are designed to make it easy for authors to write a beautiful LaTeX document with skills they already have. To keep its simplicity from becoming a limitation, we have to provide ways for more advanced users to override the default decisions that Scientific Word makes. This paper has covered a few of the new technologies we are using to make a more modular system, with the interconnections provided by stable and well-documented standards in a way that we, or a knowledgeable user, can easily customize. We expect this new platform to allow us to be more nimble than before in responding to the changing needs of our customers, and to serve as a solid base for the next ten years of development.

LuaTEX going beta

by Hans Hagen & Taco Hoekwater

This is Chapter XI from ConTEXt, from MkII to MkIV, a document that describes our explorations, experiments and decisions made while we develop LuaTEX.

introduction

We're closing in on the day that we will go beta with LuaTEX (end of July 2007). By now we have a rather good picture of its potential and to what extend LuaTEX will solve some of our persistent problems. Let's first summarize our reasons for and objectives with LuaTEX.

- The world has moved from 8 bits to 32 bits and more, and this is quite noticeable in the arena of fonts. Although Type1 fonts could host more than 256 glyphs, the associated technology was limited to 256. The advent of OpenType fonts will make it easier to support multiple languages at the same time without the need to switch fonts at awkward times.

- At the same time Unicode is replacing 8 bit based encoding vectors and code pages (input regimes). The most popular and rather efficient utf8 encoding has become a de factor standard in document encoding and interchange.

- Although we can do real neat tricks with TEX, given some nasty programming, we are touching the limits of its possibilities. In order for it to survive we need to extend the engine but not at the cost of base compatibility.

- Coding solutions in a macro language is fine, but sometimes you long to a more procedural approach. Manipulating text, handling io, interfacing . . . the technology moves on and we need to move along too.

Hence LuaTEX: a merge of the mainstream traditional TEX engines, stripped from broken or incomplete features and opened up to an embedded Lua scripting engine.

We will describe the impact of this new engine by starting from its core components reflected in the specific Lua interface libraries. Missing here is embedded support for MetaPost, because it's not yet there (apart from the fact that we use Lua to convert MetaPost graphics into TEX). Also missing is the interfacing to the pdf backend, which is also on the agenda for later. Special extensions, for instance those dealing with runtime statistics are also not discussed. Since we use ConTEXt as testbed, we will refer to the LuaTEX aware version of this macro package, MkIV, but most conclusions are rather generic.

tex internals

In order to manipulate TEX's data structures, we need access to all those registers. Already early in the development, dimension and counters were accessible and when token and node interfaces were implemented, those registers also were interfaced.

Those who read the previous chapters will have noticed that we hardly discussed this option. The reason is that we didn't yet needed that access much in order to implement font support and list processing. After all, most of the data that we need to access and manipulate is not in the registers at all. Information meant for LUA can be stored in LUA data structures. In fact, the basic call

```
\directlua 0 {some lua code}
```

has shown to be a pretty good starting point and the fact that one can print back to the TEX engine overcomes the need to store results in shared variables.

```
\def\valueofpi{\directlua0{tex.sprint(math.pi()}}
```

The number of such direct calls is not that large anyway. More often a call to LUA will be initiated by a callback, i.e. a hook into the TEX machinery.

What will be the impact of access on CONTEXT MKIV? This is yet hard to tell. In a later stage of the development, when parts of the TEX machinery will be rewritten in order to get rid of the current global nature of many variables, we will gain more control and access to TEX's internals. Core functionality will be isolated, can be extended and/or overloaded and at that moment access to internals is much more needed. But certainly that will be beyond the current registers and variables.

callbacks

These are the spine of LUATEX: here both worlds communicate with each other. A callback is a place in the TEX kernel where some information is passed to LUA and some result is returned that is then used along the road. The reference manual mentions them all and we will not repeat them here. Interesting is that in MKIV most of them are used and for tasks that are rather natural to their place and function.

```
callback.register("tex_wants_to_do_this",
    function but_use_lua_to_do_it_instead(a,b,c)
        -- do whatever you like with a, b and c
        return a, b, c
    end
)
```

The impact of callbacks on MkIV is big. It provides us a way to solve persistent problems or reimplement existing solutions in more convenient ways. Because we tested realistic functionality on real (moderately complex) documents using a pretty large macro package, we can safely conclude that callbacks are quite efficient. Stepwise Lua kicks in in order to:

- influence the input medium so that it provides a sequence of UTF characters
- manipulate the stream of characters that will be turned into a list of tokens
- convert the list of tokens into another list of tokens
- enhance the list of nodes that will be turned into a typeset paragraph
- tweak the mechanisms that come into play when lines are constructed
- finalize the result that will end up in the output medium

Interesting is that manipulating tokens is less useful than it may look at first sight. This has to do with the fact that it's (mostly) an expanded stream and at that time we've lost some information or need to do quite some coding in order to analyze the information and act upon it.

Will ConTEXt users see any of this? Chances are small that they will, although we will provide hooks so that they can add special code themselves. Users activating a callback has some danger, since it may overload already existing functionality. Chaining functionality in a callback also has drawbacks, if only that one may be confronted with already processed results and/or may destroy this result in unpredictable ways. So, as with most low level TEX features, ConTEXt users will work with more abstract interfaces.

in- and output

In MkIV we will no longer use the KPSE library directly. Instead we use a reimplementation in Lua that not only is more efficient, but also more powerful: it can read from ZIP files, use protocols, be more clever in searching, reencodes the input streams when needed, etc. The impact on MkIV is large. Most TEX code that deals with input reencoding has gone away and is replaced by Lua code.

Although it is not directly related with reading from the input medium, in that stage we also replaced verbatim handling code. Such (often messy) catcode related situations are now handled more flexible, thanks to fast catcode table switching (a new LuaTEX feature) and features like syntax highlighting can be made more neat.

Buffers, a quite old but frequently used feature of ConTEXt, are now kept in memory instead of files. This speeds up runs. Auxiliary data, aka multi–pass information, will no longer be stored in TEX files but in Lua files. In ConTEXt we have one such auxiliary file and in MkII this file is selectively filtered, but in MkIV we will be less careful with memory and load all that data once. Such speed improvements compensate the fact that LuaTEX is somewhat slower than it's ancestor PDFTEX. (Actually, the fact that LUATEX is a bit slower that PDFTEX is mostly due to the fact that it has ALEPH code on board.)

Users often wonder why there are so many temporary files, but these mostly relate to METAPOST support. These will go away once we have METAPOST as a library.

In a similar way support for XML will be enriched. We already have experimental loaders, filters and other code, and integration is on the agenda. Since CONTEXT uses XML for some sub systems, this may have some impact.

Other IO related improvements involve debugging, error handling and logging. We can pop up helpers and debug screens (MkIV can produce XHTML output and then launch a browser). Users can choose more verbose logging of IO and ask for log data to be formatted in XML. These parts need some additional work, because in the end we will also reimplement and extend TEX's error handling.

Another consequence of this will be that we will be able to package TEX more conveniently. We can put all the files that are needed into a ZIP file so that we only need to ship that ZIP file and a binary.

font readers

Handling OPENTYPE involves more that just loading yet another font format. Of course loading an OPENTYPE file is a necessity but we need to do more. Such fonts come with features. Features can involve replacing one representation of a character by another one of combining sequences into other sequences and finaly resolving them to one or more glyphs.

Given the numerous options we will have to spend quite some time on extending CONTEXT with new features. Instead of defining more and more font instances (the traditional TEX way of doing things) we will will provides feature switching. In the end this will make the often confusing font mechanisms less complex for the user to understand. Instead of for instance loading an extra font (set) that provides old style numerals, we will decouple this completely from fonts and provide it as yet another property of a piece of text. The good news is that much of the most important machinery is alresady in place (ligature building and such). Here we also have to decide what we let TEX do and what we do by processing node lists. For instance kerning and ligature building can either be done by TEX or by LUA. Given the fact that TEX does some juggling with character kerning while determining hyphenation points, we can as well disable TEX's kerning and let LUA handle it. Thereby TEX only has to deal with paragraph building. (After all, we need to leave TEX some core functionality to deal with.)

Another everlasting burden on macro writers and users is dealing with character representations missing from a font. Of course, since we use named glyphs in CONTEXT MkII already much of this can be hidden, but in MkIV we can create virtual fonts on the fly and keep thinking in terms of characters and glyphs instead of dealing with boxes and other structures that don't go well with for instance hyphenating words.

This brings us to hyphenation, historically bound to fonts in traditional TEX. This dependency will go away. In MkII we already ship UTF8 based patterns fore some time and these can be conveniently used in MkIV too. We experimented with using hyphenated word lists and this looks promising. You may expect more advanced ways of dealing with words, hyphenation and paragraph building in the near future. When we presented the first version of LuaTEX a few years ago, we only had the basic `\directlua` call available and could do a bit of string manipulation on the input. A fancy demo was to color wrongly spelled words. Now we can do that more robustly on the node lists.

Loading and preparing fonts for usage in LuaTEX or actually MkIV because this depends on the macro package takes some runtime. For this reason we introduces caching into MkIV: data that is used frequently is written to a cache and converted to Lua bytecode. Loading the converted files is incredibly fast. Of course there is aprice to pay: disk space, but that comes cheap these days. Also, it may as well be compensated by the fact that we can kick out many redundant files from the core TEX distributions (metric files for instance).

tokens handlers

Do we need to handle tokens? So far in experimental MkIV code we only used these hooks to demonstrate what TEX does with your characters. For a while we also constructed token lists when we wanted to inject `\pdfliteral` code in node lists, but that became obsolete when automatic string to token conversion was introduced in the node conversion code. Now we inject literal whatsit nodes. It may be worth noticing that playing with token lists gave us some good insight in bottlenecks because quite some small table allocation and garbage collections goes on.

nodes and attributes

These are the most promissing new features. In itself, nodes are not new, nor are attributes. In some sense when we use primitives like `\hbox`, `\vskip`, `\lastpenalty` the result is a node, but we can only control and inspect their properties within hard coded bounds. We cannot really look into boxes, and the last penalty may be obscured by a whatsit (a mark, a special, a write, etc.). Attributes could be fakes with marks and macro bases stacks of states. Native attributes are more powerful and each node can cary a truckload of them.

With LuaTEX, out of a sudden we can look into TEX's internals and manipulate them. Although I don't claim to be a real expert on these internals, even after over a decade of TEX programming, I'm sometimes surprised what I found there. When we are playing with these interfaces, we often run into situations where we need to add much print statements to the LUA code in order to find out what TEX is returning. It all has to do with the way TEX collects information and when it decides to act. In regular TEX much goes unnoticed, but when one has for instance a callback that deals with page building there are many places where this gets called and some of these places need special treatment.

Undoubtely this will have a huge impact on CONTEXT MKIV. Instead of parsing an input stream, we can now manipulate node lists in order to achieve (slight) inter–character spacing which is often needed in sectioning titles. The nice thing about this new approach is that we no longer have interference from characters that need multiple tokens (input characters) in order to be constructed, which complicates parsing (needed to split glyphs in MKII).

Signaling where to letterspace is done with the mentioned attributes. There can be many of them and they behave like fonts: they obey grouping, travel with the nodes and are therefore insensitive for box and page splitting. They can be set at the TEX end but needs to be handled at the LUA side. One may wonder what kind of macro packages would be around when TEX has attributes right from its start.

In MKII letterspacing is handled by parsing the input and injecting skips. Another approach would be to use a font where each character has more kerns or space around it (a virtual font can do that). But that would not only demand knowledge of what fonts need that that treatment, but also many more fonts and generating them is no fun for users. In PDFTEX there is a letterspace feature, where virtual fonts are generated on the fly, and with such an approach one has to compensate for the first and last character in a line, in order to get rid of the left- and rightmost added space (being part of the glyph). The solution where nodes are manipulated does put that burden upon the user.

Another example of node processing is adding specific kerns around some punctuation symbols, as is custom in French. You don't want to know what it takes to do that in traditional TEX, but if I mention the fact that colons become active characters you can imagine the nightmare. Hours of hacking and maybe even days of dealing with mechanisms that make these active colons workable in places where colons are used for non text are now even more wasted time if you consider that it takes a few lines of code in MKIV. Currently we let CONTEXT support both good old TEX (represented by PDFTEX), XETEX (a UNICODE and OPENTYPE aware variant) and LUATEX by shared and dedicated MKII and MKIV code.

Vertical spacing can be a pain. Okay, currently MKII has a rather sophisticated way to deal with vertical spacing in ways that give documents a consistent look and feel, but every now and then we run into border cases that cannot be dealt with simply because we cannot look back in time. This is needed because TEX adds content to the main vertical list and then it's gone from our view. Take for instance section titles. We don't want them dangling at the bottom of a page. But at the same time we want itemized lists to look well, i.e. keep items together in some situations. Graphics that follow a section title pose similar problems. Adding penalties helps but these may come too late, or even worse, they may obscure previous skips which then cannot be dealt with by successive skips. To simplify the problem: take a skip of 12pt, followed by a penalty, followed by another skip of 24pt. In ConTEXt this has to become a penalty followed by one skip of 24pt.

Dealing with this in the page builder is rather easy. Ok, due to the way TEX adds content to the page stream, we need to collect, treat and flush, but currently this works all right. In ConTEXt MkIV we will have skips with three additional properties: priority over other skips, penalties, and a category (think of: ignore, force, replace, add).

When we experimented with this kind of things we quickly decided that additional experiments with grid snapping also made sense. These mechanisms are among the more complex ones on ConTEXt. A simple snap feature took a few lines of Lua code and hooking it into MkIV was not that complex either. Eventually we will reimplement all vertical spacing and grid snapping code of MkII in Lua. Because one of ConTEXt column mechanism is grid aware, we may as well adath that and/or implement an additional mechanism.

A side effect of being able to do this in LuaTEX is that the code taken from pdfTEX is cleaned up: all (recently added) static kerning code is removed (inter–character spacing, pre- and post character kerning, experimental code that can fix the heights and depths of lines, etc.). The core engine will only deal with dynamic features, like hz and protruding.

So, the impact on MkIV of nodes and attributes is pretty big! Horizontal spacing isues, vertical spacing, grid snapping are just a few of the things we will reimplement. Other things are line numbering, multiple content streams with synchronization, both are already present in MkII but we can do a better job in MkIV.

generic code

In the previous text MkIV was mentioned often, but some of the features are rather generic in nature. So, how generic can interfaces be implemented? When the MkIV code has matured, much of the Lua and glue–to–TEX code will be generic in nature. Eventually ConTEXt will become a top layer on what we internally call MetaTEX, a collection of kernel modules that one can use to build specialized macro packages. To some extent MetaTEX can be for LuaTEX what plain is for TEX. But if and how fast this will be reality depends on the amount of time that we (and other members of the ConTEXt development team) can allocate to this.

I Zapfing fonts

features

In previous chapters we've seen support for OpenType features creep into LuaTeX and ConTeXt MkIV. However, it may not have been clear that so far we were just feeding the traditional TeX machinery with the right data: ligatures and kerns. Here we will show what so called features can do for you. Not much Lua code will be shown, if only because relatively complex code is needed to handle this kind of trickery with acceptable performance.

In order to support features in their full glory more is needed than TeX's ligature and kern mechanisms: we need to manipulate the node list. As a result, we have now a second mechanism built into MkIV and users can choose what method they like most. The first method, called `base`, is less powerful and less complete than the one named `node`. Eventually ConTeXt will use the node method by default.

There are two variants of features: substitutions and positioning. Here we concentrate on substitutions of which there are several. Positioning is for instance used for specialized kerning as needed in for instance typesetting Arab.

One character representation can be replaced by one or more fixed alternatives or alternatives chosen from a list of alternatives (substitutions or alternates). Multiple characters can be replaces by one character (substitutions, alternates or a ligature). The replacements can depend on preceding and/or following glyphs in which case we say that the replacement is driven by rules. Rules can deal with single glyphs, combinations of glyphs, classes (defined in the font) of glyphs and/or ranges of glyphs.

Because the available documentation of OpenType is rather minimalistic and because most fonts are relatively simple, you can imagine that figuring out how to implement support for fonts with advanced features is not entirely trivial and involves some trial and error. What also complicate things is that features can interfere. Yet another complicating factor is that in the order of applying a rule may obscure a later rule. Such fonts don't ship with manuals and examples of correct output are not part of the buy.

We like testing LuaTeX's open type support with Palatino Regular and Palatino Sans and good old Type1 support with Optima Nova. So it makes sense to test advanced features with Zapfino Pro. This font has many features, which happen to be implemented by Adam Twardoch, a well known font expert and familiar with the TeX community. We had the feeling that when LuaTeX can support Zapfino Pro, designed by Hermann Zapf and enhanced by Adam, we have reached a crucial point in the development.

The first thing that you will observe when using this font is that the files are larger than normal, especially the cached versions in MkIV. This made me extend some of the serialization code that we use for caching font data so that it could handle huge tables better but at the cost of some speed. Once we could handle the data conveniently and as a side effect look into the font data with an editor, it became clear that implementing for the `calt` and `clig` features would take a bit of coding.

example

Before some details will be discussed, we will show two of the test texts that ConTeXt users normally use when testing layouts or new features, a quote from E.R. Tufte and one from Hermann Zapf. The TeX code shows how features are set in ConTeXt.

```
\definefontfeature
   [zapfino]
   [language=nld,script=latn,mode=node,
    calt=yes,clig=yes,liga=yes,rlig=yes,tlig=yes]

\definefont
    [Zapfino]
    [ZapfinoExtraLTPro*zapfino at 24pt]
    [line=40pt]
\Zapfino
\input tufte \par
```

We thrive in information--thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, *flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.*

You don't even have to look too closely in order to notice that characters are represented by different glyphs, depending on the context in which they appear.

```
\definefontsynonym
[Zapfino]
[ZapfinoExtraLTPro]
[features=zapfino]
\definedfont
[Zapfino at 24pt]
\setupinterlinespace
[line=40pt]
\input zapf \par
```

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely--praised program, called up on the screen, will make everything automatic from now on.

obeying rules

When we were testing node based feature support, the only way to check this was to identify the rules that lead to certain glyphs. The more unique glyphs are good candidates for this. For instance

- there is s special glyph representing ℅
- in the input stream this is the character sequence `c/o`
- so there most be a rule that tells us that this sequence becomes that ligature

As said, in this case, the replacement glyph is supposed to be a ligature and indeed there is such a ligature: `c_slash_o`. Of course, this replacement will only take place when the sequence is surrounded by spaces.

However, when testing this, we were not looking at this rule but at the (randomly chosen) rule that was meant to intercept the alternative `h.2` followed by `z.4`. Interesting was that this resolved to a ligature indeed, but the shape associated with this ligature was an `h`, which is not right. Actually, a few more of such rules turned out to be wrong. It took a bit of an effort to reach this conclusion because of the mentioned interferences of features and rules. At that time, the rule entry (in raw LuaTeX table format) looks as follows:

```
[44] = {
    ["format"] = "coverage",
    ["rules"] = {
        [1] = {
            ["coverage"] = {
                ["ncovers"] = {
                    [1] = "h.2",
                    [2] = "z.4",
                }
            },
            ["lookups"] = {
                [1] = {
                    ["lookup_tag"] = "L084",
                    ["seq"] = 0,
                }
            }
        }
    }
    ["script_lang_index"] = 1,
    ["tag"] = "calt",
    ["type"] = "chainsub"
}
```

Instead of reinventing the wheel, we used the FontForge libraries for reading the OpenType font files. Therefore the LuaTEX table is resembling the internal FontForge data structures. Currently we show the version 1 format.

Here ncovers means that wen the current character has shape (h.2) and the next one is (z.4) (a sequence) then we need to apply the lookup internally tagged L084. Such a rule can be more extensive, for instance instead of h.2 one can have a list of characters, and there can be bcovers and fcovers as well, which means that preceding or following character need to be taken into account.

When this rule matches, it resolves to a specification like:

```
[6] = {
    ["flags"] = 0,
    ["lig"] = {
        ["char"] = "h",
        ["components"] = "h.2 z.4",
    },
    ["script_lang_index"] = 65535,
    ["tag"] = "L084",
    ["type"] = "ligature",
}
```

Here tag and script_lang_index are kind of special and are part of an private feature system, i.e. they make up the cross reference between rules and glyphs. Watch how the components don't match the character, which is even more peculiar when we realize that these are the initials of the author of the font. It took a couple of Skype sessions and mails before we came to the conclusion that this was probably a glitch in the font. So, what to do when a font has bugs like this? Should one disable the feature? That would be a pitty because a font like Zapfino depends on it. On the other hand, given the number of rules and given the fact that there are different rule sets for some languages, you can imagine that making up the rules and checking them is not trivial.

We should realize that Zapfino is an extraordinary case, because it used the OpenType features extensively. We can also be sure that the problems will be fixed once they are known, if only because Adam Twardoch (who did the job) has exceptionally high standards but it may take a while before the fix reached the user (who then has to update his or her font). As said, it also takes some effort to run into the situation described here so the likelihood of running into this rule is small. This also brings to our attention the fact that fonts can now contain bugs and updating them makes sense but can break existing documents. Since such fonts are copyrighted and not available on line, font vendors need to find ways to communicate these fixes to their customers.

Can we add some additional checks for problems like this? For a while I thought that it was possible by assuming that ligatures have names like `h.2_z.4` but alas, sequences of glyphs are mapped onto ligatures using mappings like the following:

`three fraction four.2`	`threequarters`	¾
`three fraction four`	`threequarters`	¾
`d r`	`d_r`	dr
`e period`	`e_period`	e.
`f i`	`fi`	fi
`f l`	`fl`	fl
`f f i`	`f_f_i`	ffi
`f t`	`f_t`	ft

Some ligature have no _ in their names and there are also some inconsistencies, compare the `fl` and `f_f_i`. Here font history is painfully reflected in inconsistency and no solution can be found here.

So, in order to get rid of this problem, MkIV implements a method to ignore certain rules but then, this only makes sense if one knows how the rules are tagged internally. So, in practice this is no solution. However, you can imagine that at some point ConTEXt ships with a database of fixes that are applied to known fonts with certain version numbers.

We also found out that the font table that we used was not good enough for our purpose because the exact order in what rules have to be applies was not available. Then we noticed that in the meantime FontForge had moved on to version 2 and after consulting the author we quickly came to the conclusion that it made sense to use the updated representation.

In version 2 the snippet with the previously mentioned rule looks as follows:

```
["ks_latn_l_66_c_19"]={
 ["format"]="coverage",
 ["rules"]={
  [1]={
   ["coverage"]={
    ["current"]={
     [1]="h.2",
     [2]="z.4",
    }
   },
   ["lookups"]={
    [1]={
     ["lookup"]="ls_l_84",
     ["seq"]=0,
    }
   }
  }
 },
 ["type"]="chainsub",
},
```

The main rule table is now indexed by name which is possible because the order of rules is specified somewhere else. The key `ncovers` has been replaced by `current`. As long as LuaTEX is in beta stage, we have the freedom to change such labels as some of them are rather FontForge specific.

This rule is mentioned in a feature specification table. Here specific features are associated with languages and scripts. This is just one of the entries concerning `calt`. You can imagine that it took a while to figure out how best to deal with this, but eventually the MkIV code could do the trick. The cryptic names are replacements for pointers in the FontForge datastructure. In order to be able to use FontForge for font development and analysis, the decision was made to stick closely to its idiom.

```
["gsub"]={
 ...
 [67]={
  ["features"]={
   [1]={
    ["scripts"]={
     [1]={
      ["langs"]={
       [1]="AFK ",
       [2]="DEU ",
       [3]="NLD ",
       [4]="ROM ",
       [5]="TRK ",
       [6]="dflt",
      },
      ["script"]="latn",
     }
    },
    ["tag"]="calt",
   }
  },
  ["name"]="ks_latn_l_66",
  ["subtables"]={
   [1]={
    ["name"]="ks_latn_l_66_c_0",
   },
   ...
   [20]={
    ["name"]="ks_latn_l_66_c_19",
   },
   ...
  },
  ["type"]="gsub_context_chain",
 },
```

practice

The few snapshots of the font table probably don't make much sense if you haven't seen the whole table. Well, it certainly helps to see the whole picture, but we're talking of a 14 MB file (1.5 MB bytecode). When resolving ligatures, we can follow a straightforward approach:

- walk over the nodelist and at each character (glyph node) call a function
- this function inspects the character and takes a look at the following ones
- when a ligature is identified, the sequence of nodes is replaced

Substitutions are not much different but there we look at just one character. However, contextual substitutions (and ligatures) are more complex. Here we need to loop over a list of rules (dependent on script and language) and this involves a sequence as well as preceding and following characters. When we have a hit, the sequence will be replaced by another one, determined by a lookup in the character table. Since this is a rather time consuming operation, especially because many surrounding characters need to be taken into account, you can imagine that we need a bit of trickery to get an acceptable performance. Fortunately LUA is pretty fast when it comes down to manipulating strings and tables, so we can prepare some handy datastructures in advance.

When testing the implementation of features one need to be aware of the fact that some appearance are also implemented using the regular ligature mechanisms. Take the following definitions:

```
\definefontfeature
   [none]
   [language=dflt,script=latn,mode=node,liga=no]
\definefontfeature
   [calt]
   [language=dflt,script=latn,mode=node,liga=no,calt=yes]
\definefontfeature
```

```
    [clig]
    [language=dflt,script=latn,mode=node,liga=no,clig=yes]
\definefontfeature
    [dlig]
    [language=dflt,script=latn,mode=node,liga=no,dlig=yes]
\definefontfeature
    [liga]
    [language=dflt,script=latn,mode=node]
```

This gives:

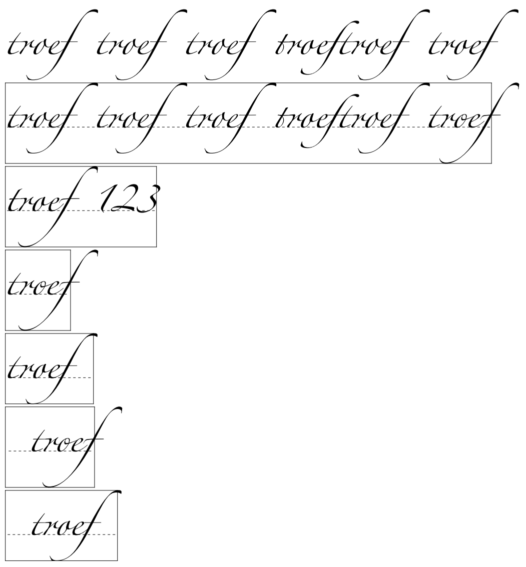
`none`	*on the synthesis winnow the wheat*
`calt`	*on the synthesis winnow the wheat*
`clig`	*on the synthesis winnow the wheat*
`dlig`	*on the synthesis winnow the wheat*
`liga`	*on the synthesis winnow the wheat*

Here are Adam's recommendations with regards to the `dlig` feature: The `dlig` feature is supposed to by use only upon user's discretion, usually on single runs, words or even pairs. It makes little sense to enable `dlig` for an entire sentence or paragraph. That's how the OpenType specification envisions it.

When testing features it helps to use words that look similar so next we will show some examples that used. When we look at these examples, we need to understand that when a specific character representation is analyzed, the rules can take preceding and following characters into account. The rules take characters as well as their shapes, or more precisely: one of their shapes since Zapfino has many variants, into account. Since different rules are used for languages (okay, this is limited to only a subset of languages that use the latin script) not only shapes but also the way words are constructed are taken into account. Designing te rules is definitely non trivial.

When testing the implementation we ran into cases where the initial `t` showed up wrong, for instance in the the Dutch word `troef`. Because space can be part of the rules, we need to handle the cases where words end and start and boxes are then kind of special.

```
troef troef troef troeftroef troef  \par
\ruledhbox{troef troef troef troeftroef troef} \par
\ruledhbox{troef 123} \par
\ruledhbox{troef} \ruledhbox{troef } \ruledhbox{ troef} \ruledhbox
{ troef } \par
```


troef troef troef troeftroef troef
troef troef troef troeftroef troef
troef 123
troef
troef
troef
troef

Unfortunately, this does not work well with punctuation, which is less prominent in the rules than space. In our favourite test quote of Tufte, we have lots of comma's and there it shows up:

```
review review review, review \par
itemize, review \par
itemize, review, \par
```

review review review, review

itemize, review

itemize, review,

Of course we can decide to extend the rule base at runtime and this may well happen when we experiment more with this font.

The next one was one of our first test lines, Watch the initial and the Zapfino ligature.

```
Welcome to Zapfino
```

Welcome to Zapfino

For a while there was a bug in the rule handler that resulted in the variant of the `y` that has a very large descender. Incidentally the word `synthesize` is also a good test case for the `the` pattern which gets special treatment because there is a ligature available.

```
synopsize versus synthesize versus
synthase versus sympathy versus synonym
```

synopsize versus synthesize versus synthase versus sympathy versus synonym

Here are some examples that use the `g`, `d` and `f` in several places.

```
eggen groet ogen hagen \par
dieren druiven onder aard  donder modder \par
fiets effe flater triest troef \par
```

eggen groet ogen hagen

dieren druiven onder aard donder modder

fiets effe flater triest troef

Let's see how well Hermann has taken care of the `h`'s representations. There are quite some variants of the lowercase one:

`h`	h
`h.2`	h
`h.3`	h
`h.4`	h
`h.5`	h
`h.init`	h
`h.sups`	h
`h.sc`	H
`orn.73`	☞

How about the uppercase variant, as used in his name:

```
M Mr Mr. H He Her Herm Herma Herman Hermann Z Za Zap Zapf \par
Mr. Hermann Zapf
```

M Mr Mr. H He Her Herm Herma Herman Hermann Z Za Zap Zapf

Mr. Hermann Zapf

Of course we have to test another famous name:

```
D Do Don Dona Donal Donald K Kn Knu Knut Knuth \par
Don Knuth Donald Knuth Donald E. Knuth DEK \par
Prof. Dr. Donald E. Knuth \par
```

D Do Don Dona Donal Donald K Kn Knu Knut Knuth

Don Knuth Donald Knuth Donald E. Knuth DEK

Prof. Dr. Donald E. Knuth

Unfortunately the LUA and TEX logo's don't come out that well:

`L Lu Lua l lu lua t te tex TeX luatex luaTeX LuaTeX`

L Lu Lua l lu lua t te tex TeX luatex luaTeX LuaTeX

This font has quite some ornaments and there is an `ornm` feature that can be applied. We're still not sure about its usage, but when one keys in text in lowercase, `hermann` comes out as follows:

As said in the beginning, dirty implementation details will be kept away from the reader. Also, you should not be surprised if the current code had some bugs or does some things wrong. Also, if spacing looks a bit weird to you, keep in mind that we're still in the middle of sorting things out.

Taco Hoekwater & Hans Hagen

The MPlib project
MetaPost as a reusable component

As you probably know, MetaPost development has restarted approximately two years ago. After a period of investigating user demands, it has now become obvious that MetaPost is showing its age.

The problems lie not so much in the actual drawing language that is used, but in the 1980s metafont legacy that is very noticable in the way the program interacts with the user and in how it deals with the computing environment in general.

Some of the big user-side problems that resurface on a regular basis are:

- The model used for the handling of external labels is outdated.

  Running a per-file preprocessor to create the labels was already problematic before, but it is getting worse now that both TEX and Troff are moving away from their traditional output formats.
- All number handling is based on fractions of a 32-bit integer.

  User input often hits one of the many boundaries that are a result of that. For instance, no numbers can be any larger than 16384, and there is a noticeable lack of precision in the intersection-point calculations.
- MetaPost cannot be used as a system-level service.

  In fact, MetaPost cannot even be used as a system-wide library, because the many global variables make it non-shareable.
- Lack of 3-D support.

  Even technical drawings that are nominally considered to be two-dimensional, like the ones in highschool math and physics books, often need to handle projections of 3-D objects to a plane.

Much of the needed development to fix these issues can be done in the normal course of events, because the needed extensions or changes to the program are isolated to a small section of the source code (this is for instance true for 3-D projection support), or because the needed changes are so well understood that it is trivial to make many changes (this is true for upgrading the 32-bit internal calculus).

But the handling of labels and the lack of system integration require massive changes to the source code as well as to the build system, and therefore it was very unlikely that this would ever get done without extra incentives: a significant amount of time and effort that has to be dedicated to those particular problems.

An estimate of the needed programming hours to turn MetaPost into a modern, re-entrant system library with a modern form of inter-process communication was created:

Converting from the use of hunderds of global variables into a data structure that is passed on from one function to another	200
Adding a unified redirection layer for the input and output, allowing files as well as buffers to be used	100
Designing and implementing a new subsystem for label typesetting	150
Adding an interface for configurable default error responses	50
Total	500

If writing documentation is included in that estimate, it makes for six months of full-time (40 hours a week) programming, time that simply could be alloted within anybody's free hours in any way. It was clear to us that, to get these tasks done within a reasonable time frame, at least some of the work would have to be done during office hours. And that requires money.

So, a funding proposal was written and at the Dante 2007 meeting Hans proposed this new project for funding. Dante immediately stepped in for 50% of the requested amount (6.000 euro) and within a week other user groups joined in as well: TUGIndia (1000 euro), TUG (1500 euro), NTG (2000 euro) and CSTug (1000 euro). Currently 500 euro is still missing, but we are confident that this gap wil be bridged or overcome.

Work will start in the autumn of this year, and it is our current estimate that the project will be complete by the summer of 2008. The actual programming will be carried out by Taco. Hans Hagen will lead the project, and Bogusław Jackowski will be in charge of quality control.

Hans Hagen & Taco Hoekwater