THE BEASTS OF FONTS ARE STILL ALIVE AND KICKING

Bogusław Jackowski
BachoTEX 2025

ENTANGLEMENT

MISCONCEPTION

THE BEASTS OF FONTS ARE STILL ALIVE AND KICKING

IDIOSYNCRASY

RELICS

Bogusław Jackowski, BachoTEX 2025 (30 IV – 4 V)

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts.

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts:

- Latin Modern Math (2011)
- TEX Gyre Bonum Math (2014)
- TEX Gyre Schola Math (2014)
- TEX Gyre Pagella Math (2014)
- TEX Gyre Termes Math (2014)
- TEX Gyre DejaVu Math (2016)

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts:

- Latin Modern Math (2011)
- TEX Gyre Bonum Math (2014)
- TEX Gyre Schola Math (2014)
- TEX Gyre Pagella Math (2014)
- TEX Gyre Termes Math (2014)
- TEX Gyre DejaVu Math (2016)

Interestingly, none of these fonts – except for Latin Modern, which is rightly mentioned as a variant of Computer Modern – is listed on the relevant Wikipedia page:

`https://en.wikipedia.org/wiki/Category:Mathematical_OpenType_typefaces`

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts.

We weren’t exactly thrilled with the options for typesetting mathematical formulas available in OpenType fonts (mostly via the MATH table). Piotr Strzelczyk and I shared our thoughts on the currently available font technology in the publication “How to make more than one math OpenType font, or the Beasts of Fonts”.

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts.

We weren’t exactly thrilled with the options for typesetting mathematical formulas available in OpenType fonts (mostly via the MATH table). Piotr Strzelczyk and I shared our thoughts on the currently available font technology in the publication “How to make more than one math OpenType font, or the Beasts of Fonts”.

I believe that Hans and Mikael agreed (to some extent) with our opinion, as they eventually abandoned the struggle with math fonts and instead implemented the necessary means for typesetting math in LuaTEX.

INTRO – RECOLLECTIONS

In 2011, the GUST e-Foundry team (Piotr Pianowski, Piotr Strzelczyk, and I) released a pilot version of the Latin Modern Math font, which, after five years, resulted in a stable collection of six math fonts.

We weren’t exactly thrilled with the options for typesetting mathematical formulas available in OpenType fonts (mostly via the MATH table). Piotr Strzelczyk and I shared our thoughts on the currently available font technology in the publication “How to make more than one math OpenType font, or the Beasts of Fonts”.

I believe that Hans and Mikael agreed (to some extent) with our opinion, as they eventually abandoned the struggle with math fonts and instead implemented the necessary means for typesetting math in LuaTeX.

And I agree with them, as the beasts of fonts described in our publication apparently still happily dwell in the Realm of Fonts.

WELCOME TO OUR REALM

OUR CURRENT GOAL

The fonts released by GUST e-Foundry are freely available; however, the sources (mainly METAPOST scripts, along with the necessary tools to convert the METAPOST output into a widely accepted format) changed so frequently that we were unable to publish them.

OUR CURRENT GOAL

The fonts released by GUST e-Foundry are freely available; however, the sources (mainly METAPOST scripts, along with the necessary tools to convert the METAPOST output into a widely accepted format) changed so frequently that we were unable to publish them.

Eventually, we decided that before retiring, we should release the sources for generating the fonts, which also meant providing the necessary documentation. Ryszard Kubiak took charge of this task. He rightly suggested that, in order to make the documentation as clear as possible, we needed to understand the semantics of the fonts.

OUR CURRENT GOAL

The fonts released by GUST e-Foundry are freely available; however, the sources (mainly METAPOST scripts, along with the necessary tools to convert the METAPOST output into a widely accepted format) changed so frequently that we were unable to publish them.

Eventually, we decided that before retiring, we should release the sources for generating the fonts, which also meant providing the necessary documentation. Ryszard Kubiak took charge of this task. He rightly suggested that, in order to make the documentation as clear as possible, we needed to understand the semantics of the fonts.

And once again, we encountered our old friends – the beasts of fonts – standing in our way.

OUR CURRENT GOAL

The fonts released by GUST e-Foundry are freely available; however, the sources (mainly METAPOST scripts, along with the necessary tools to convert the METAPOST output into a widely accepted format) changed so frequently that we were unable to publish them.

Eventually, we decided that before retiring, we should release the sources for generating the fonts, which also meant providing the necessary documentation. Ryszard Kubiak took charge of this task. He rightly suggested that, in order to make the documentation as clear as possible, we needed to understand the semantics of the fonts.

And once again, we encountered our old friends – the beasts of fonts – standing in our way. It should be emphasized, however, that not all of these beasts are giant creatures, but even the smaller monsters – affectionately called bugs or, as I'd prefer to call them, little beasts of snags – can still cause significant trouble.

OUR CURRENT GOAL

The fonts released by GUST e-Foundry are freely available; however, the sources (mainly METAPOST scripts, along with the necessary tools to convert the METAPOST output into a widely accepted format) changed so frequently that we were unable to publish them.

Eventually, we decided that before retiring, we should release the sources for generating the fonts, which also meant providing the necessary documentation. Ryszard Kubiak took charge of this task. He rightly suggested that, in order to make the documentation as clear as possible, we needed to understand the semantics of the fonts.

And once again, we encountered our old friends – the beasts of fonts – standing in our way. It should be emphasized, however, that not all of these beasts are giant creatures, but even the smaller monsters – affectionately called bugs or, as I'd prefer to call them, little beasts of snags – can still cause significant trouble.

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION

As you may recall, the GUST e-Foundry font engine uses METAPOST to produce EPS text files. These are processed by a set of Python scripts (Fontplant) and then passed to FontForge to generate binary OpenType and/or Type 1 PostScript fonts.

Sometimes, there’s a need to take a peek inside a font’s contents.

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION

As you may recall, the GUST e-Foundry font engine uses METAPOST to produce EPS text files. These are processed by a set of Python scripts (Fontplant) and then passed to FontForge to generate binary OpenType and/or Type 1 PostScript fonts.

Sometimes, there’s a need to take a peek inside a font’s contents.

For PostScript Type 1 fonts, there’s a pair of tools – a disassembler and an assembler (developed by Lee Hetherington) – that convert the binary form of a font (PFB) into a textual representation and back again. The important thing here is that the text form is fairly readable, and more importantly, the assembler can recreate exactly the same binary file.

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION

As you may recall, the GUST e-Foundry font engine uses METAPOST to produce EPS text files. These are processed by a set of Python scripts (Fontplant) and then passed to FontForge to generate binary OpenType and/or Type 1 PostScript fonts.

Sometimes, there's a need to take a peek inside a font's contents.

For PostScript Type 1 fonts, there's a pair of tools – a disassembler and an assembler (developed by Lee Hetherington) – that convert the binary form of a font (PFB) into a textual representation and back again. The important thing here is that the text form is fairly readable, and more importantly, the assembler can recreate exactly the same binary file.

Such reversible conversions are quite standard in the TeX world – for example, disassembling and assembling tools by DEK for TFM files (tftopl and pltotf), or by Geoffrey Tobin for DVI files (dv2dt and dt2dv).

Snaglog: Notes from the Trenches of OpenType

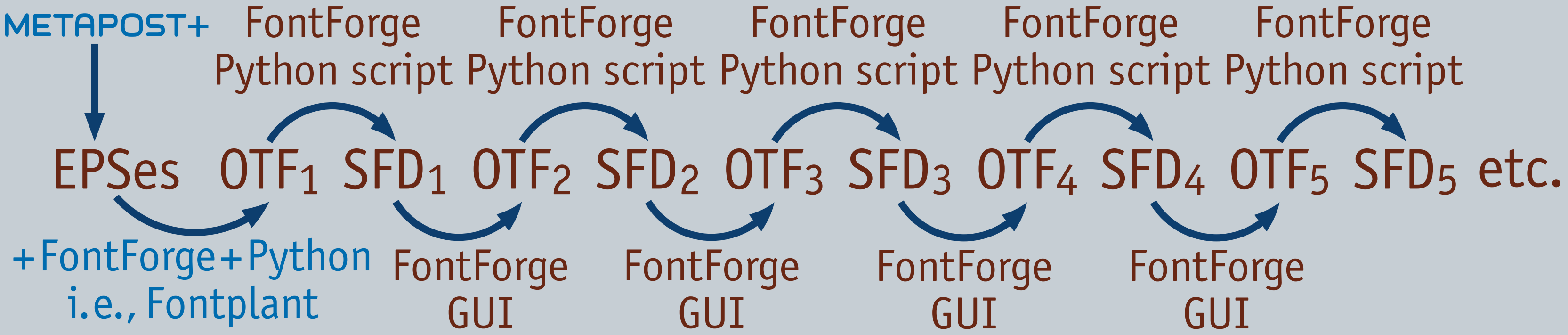
TEXT–BINARY CONVERSION: A SETBACK

OpenType fonts can also be converted into a textual format using FontForge – namely, to the Spline Font Database (SFD) format – which can then be loaded back into FontForge. However, the SFD file isn’t particularly readable for humans, and the round-trip conversion doesn’t exactly meet our expectations.

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION: A SETBACK

OpenType fonts can also be converted into a textual format using FontForge – namely, to the Spline Font Database (SFD) format – which can then be loaded back into FontForge. However, the SFD file isn’t particularly readable for humans, and the round-trip conversion doesn’t exactly meet our expectations.



METAPOST+

FontForge Python script FontForge Python script FontForge Python script FontForge Python script FontForge Python script

EPSes OTF$_1$ SFD$_1$ OTF$_2$ SFD$_2$ OTF$_3$ SFD$_3$ OTF$_4$ SFD$_4$ OTF$_5$ SFD$_5$ etc.

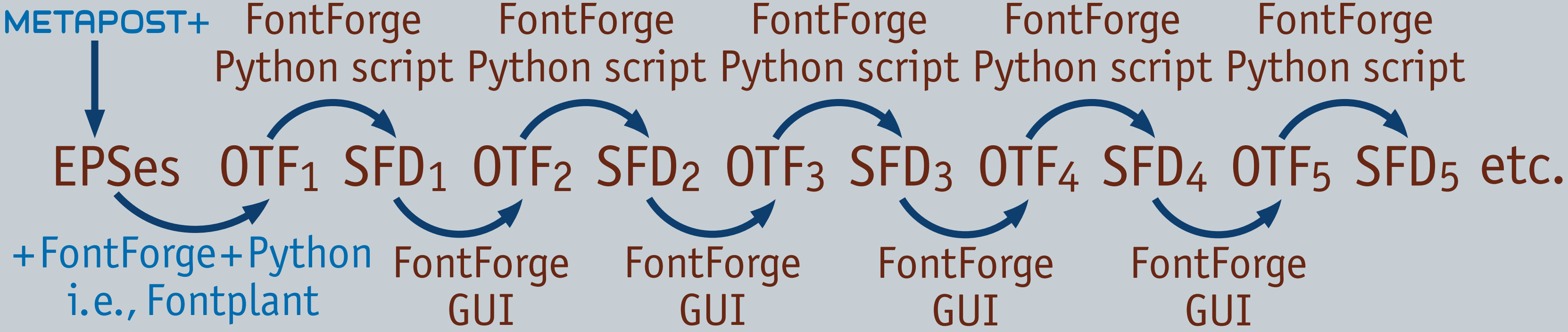
+FontForge+Python i.e., Fontplant

FontForge GUI FontForge GUI FontForge GUI FontForge GUI

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION: A SETBACK

OpenType fonts can also be converted into a textual format using FontForge – namely, to the Spline Font Database (SFD) format – which can then be loaded back into FontForge. However, the SFD file isn't particularly readable for humans, and the round-trip conversion doesn't exactly meet our expectations.


METAPOST+

FontForge Python script FontForge Python script FontForge Python script FontForge Python script FontForge Python script

EPSes OTF_1 SFD_1 OTF_2 SFD_2 OTF_3 SFD_3 OTF_4 SFD_4 OTF_5 SFD_5 etc.

+FontForge+Python
i.e., Fontplant

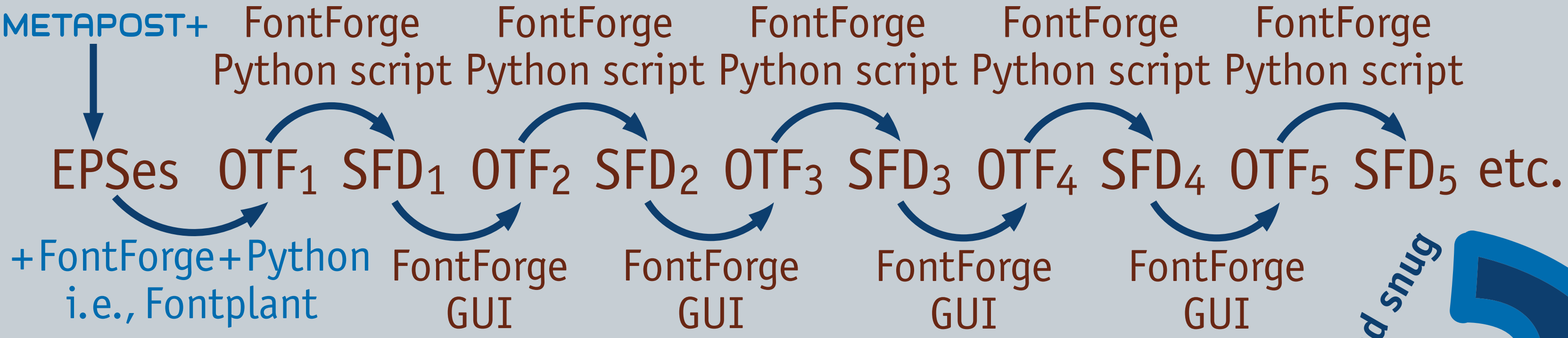
FontForge GUI FontForge GUI FontForge GUI FontForge GUI

Somewhat surprisingly, $\text{OTF}_1 \neq \text{OTF}_2 \neq \text{OTF}_3 = \text{OTF}_4 = \text{OTF}_5 = \ldots$, and $\text{SFD}_1 \neq \text{SFD}_2 \neq \text{SFD}_3 \neq \text{SFD}_4 \neq \text{SFD}_5 \neq \ldots$ It should be noted that the files SFD_3, SFD_4, SFD_5, etc., differ only in a comment regarding the XUID. Incidentally, Adobe stopped using UniqueIDs and XUIDs in their OpenType CFF fonts at the latest around 2005.

Snaglog: Notes from the Trenches of OpenType

TEXT–BINARY CONVERSION: A SETBACK

OpenType fonts can also be converted into a textual format using FontForge – namely, to the Spline Font Database (SFD) format – which can then be loaded back into FontForge. However, the SFD file isn’t particularly readable for humans, and the round-trip conversion doesn’t exactly meet our expectations.

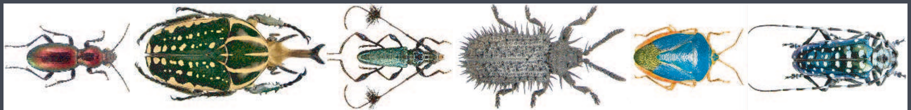


Somewhat surprisingly, $\mathrm{OTF}_1 \neq \mathrm{OTF}_2 \neq \mathrm{OTF}_3 = \mathrm{OTF}_4 = \mathrm{OTF}_5 = \ldots$, and $\mathrm{SFD}_1 \neq \mathrm{SFD}_2 \neq \mathrm{SFD}_3 \neq \mathrm{SFD}_4 \neq \mathrm{SFD}_5 \neq \ldots$ It should be noted that the files SFD_3, SFD_4, SFD_5, etc., differ only in a comment regarding the XUID. Incidentally, Adobe stopped using UniqueIDs and XUIDs in their OpenType CFF fonts at the latest around 2005.

a little beast od snug

FEATURES

I’ll now turn to a crucial – yet still poorly documented – component of OpenType fonts: features. These structures are specific to OpenType and have no counterpart in the Type 1 format.



FEATURES

I’ll now turn to a crucial – yet still poorly documented – component of OpenType fonts: features. These structures are specific to OpenType and have no counterpart in the Type 1 format.

FontForge, our main open-source tool for OpenType generation, reads feature definitions written in a somewhat quirky declarative language. Introduced in 1998 – two years after the launch of OpenType – this syntax is also used by Adobe’s Font Development Kit for OpenType (AFDKO), among others.

FEATURES

I’ll now turn to a crucial – yet still poorly documented – component of OpenType fonts: features. These structures are specific to OpenType and have no counterpart in the Type 1 format.

FontForge, our main open-source tool for OpenType generation, reads feature definitions written in a somewhat quirky declarative language. Introduced in 1998 – two years after the launch of OpenType – this syntax is also used by Adobe’s Font Development Kit for OpenType (AFDKO), among others.

```
feature liga {
  lookup liga_f_f_l {
    sub f f l by f_f_l;
    sub f f by f_f;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

FEATURES

I’ll now turn to a crucial – yet still poorly documented – component of OpenType fonts: features. These structures are specific to OpenType and have no counterpart in the Type 1 format.

FontForge, our main open-source tool for OpenType generation, reads feature definitions written in a somewhat quirky declarative language. Introduced in 1998 – two years after the launch of OpenType – this syntax is also used by Adobe’s Font Development Kit for OpenType (AFDKO), among others.

```
feature liga {
  lookup liga_f_f_l {
    sub f f l by f_f_l;
    sub f f by f_f;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

uffln ⇒ uﬄn

uffn ⇒ uﬀn

ufln ⇒ uﬂn

FEATURES

I'll now turn to a crucial – yet still poorly documented – component of OpenType fonts: features. These structures are specific to OpenType and have no counterpart in the Type 1 format.

FontForge, our main open-source tool for OpenType generation, reads feature definitions written in a somewhat quirky declarative language. Introduced in 1998 – two years after the launch of OpenType – this syntax is also used by Adobe's Font Development Kit for OpenType (AFDKO), among others.

```
feature liga {
  lookup liga_f_f_l {
    sub f f l by f_f_l;
    sub f f by f_f;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

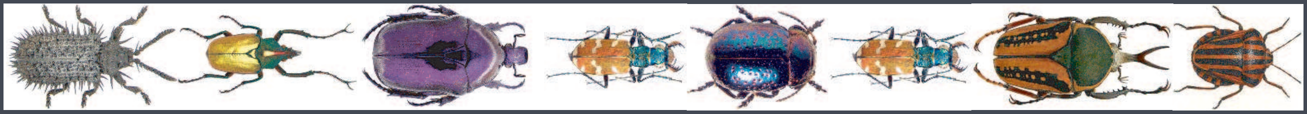
uffln ⇒ uﬄn
uffn ⇒ uﬀn
ufln ⇒ uﬂn

In LuaTeX, one activates a feature by writing the name of the feature preceded by a plus in a declaration of a font, e.g.:

```
\font\F="[Antykwa-regular]:mode=node;+liga" at 20pt
```

FEATURES: A NEXT SETBACK

I wasn’t able to figure out how the ‘liga’ feature is represented in SFD files. Fortunately, FontForge allows you to export a feature file that uses the syntax mentioned earlier. The result is formally correct, but pretty unfriendly to humans.



FEATURES: A NEXT SETBACK

I wasn’t able to figure out how the ‘liga’ feature is represented in SFD files. Fortunately, FontForge allows you to export a feature file that uses the syntax mentioned earlier. The result is formally correct, but pretty unfriendly to humans. After some manual cleaning, we end up with a somewhat surprising result (the same one you get from another tool for manipulating TrueType and OpenType fonts, namely, ttx by Just van Rossum).

Snaglog: Notes from the Trenches of OpenType

FEATURES: A NEXT SETBACK

I wasn’t able to figure out how the ‘liga’ feature is represented in SFD files. Fortunately, FontForge allows you to export a feature file that uses the syntax mentioned earlier. The result is formally correct, but pretty unfriendly to humans. After some manual cleaning, we end up with a somewhat surprising result (the same one you get from another tool for manipulating TrueType and OpenType fonts, namely, ttx by Just van Rossum):

```
feature liga {
  lookup liga_f_f_l {
    sub f f l by f_f_l;
    sub f f by f_f;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

TO OTF

```
feature liga {
  lookup liga_f_f_l {
    sub f f by f_f;
    sub f f l by f_f_l;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

FROM OTF

FEATURES: A NEXT SETBACK

I wasn't able to figure out how the 'liga' feature is represented in SFD files. Fortunately, FontForge allows you to export a feature file that uses the syntax mentioned earlier. The result is formally correct, but pretty unfriendly to humans. After some manual cleaning, we end up with a somewhat surprising result (the same one you get from another tool for manipulating TrueType and OpenType fonts, namely, ttx by Just van Rossum):

```
feature liga {
  lookup liga_f_f_l {
    sub f f l by f_f_l;
    sub f f by f_f;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

TO OTF

```
feature liga {
  lookup liga_f_f_l {
    sub f f by f_f;
    sub f f l by f_f_l;
    sub f l by f_l;
  } liga_f_f_l;
} liga;
```

FROM OTF

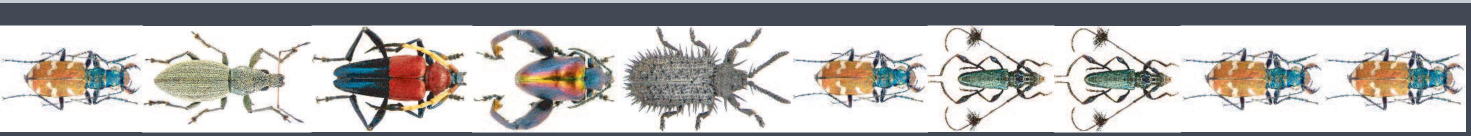
Why on earth was the order of rules messed up? Misconception? Misimplementation? Ayway, the rule '`sub f f l by f_f_l;`' is applied first – as shown in the freshly presented example). But why?

FEATURES: MISDOCUMENTING?

This brings up several key questions: what is the actual order in which rules are applied? How are the lookups – the sets of rules – ordered? And finally, in what order are features applied?

FEATURES: MISDOCUMENTING?

This brings up several key questions: what is the actual order in which rules are applied? How are the lookups – the sets of rules – ordered? And finally, in what order are features applied? Perhaps Hans can shed some light on the “order of application” algorithm implemented in in LuaTeX?



FEATURES: MISDOCUMENTING?

This brings up several key questions: what is the actual order in which rules are applied? How are the lookups – the sets of rules – ordered? And finally, in what order are features applied? Perhaps Hans can shed some light on the “order of application” algorithm implemented in in LuaTEX?

Microsoft states on their page “Developing OpenType Fonts for Standard Scripts” that the standard order for applying OpenType features is as follows:

- ccmp – Character composition / decomposition substitution
- liga – Standard ligature substitution
- clig – Contextual ligature substitution
- dist – Distances
- kern – Pair kerning
- mark – Mark-to-base positioning
- mkmk – Mark-to-mark positioning

https://learn.microsoft.com/pl-pl/typography/script-development/standard

FEATURES: MISDOCUMENTING?

This brings up several key questions: what is the actual order in which rules are applied? How are the lookups – the sets of rules – ordered? And finally, in what order are features applied?

The order Microsoft seems to recommend is likely only partially accurate. For instance, the ‘dlig’ (discretionary ligatures) feature may be applied either before or after the ‘liga’ feature, although one must admit that the position of ‘dlig’ is not explicitly defined – actually, ‘dlig’ is not mentioned in Microsoft’s note at all.

```
feature liga {
  sub a a by x;
} liga;
feature dlig {
  sub a a by z;
} dlig;
```

```
feature dlig {
  sub a a by z;
} dlig;
feature liga {
  sub a a by x;
} liga;
```

FEATURES: MISDOCUMENTING?

This brings up several key questions: what is the actual order in which rules are applied? How are the lookups – the sets of rules – ordered? And finally, in what order are features applied?

The order Microsoft seems to recommend is likely only partially accurate. For instance, the ‘dlig’ (discretionary ligatures) feature may be applied either before or after the ‘liga’ feature, although one must admit that the position of ‘dlig’ is not explicitly defined – actually, ‘dlig’ is not mentioned in Microsoft’s note at all.

```
feature liga {
  sub a a by x;
} liga;
feature dlig {
  sub a a by z;
} dlig;
```

uaan ⇒ uxn

```
feature dlig {
  sub a a by z;
} dlig;
feature liga {
  sub a a by x;
} liga;
```

uaan ⇒ uzn

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

Snaglog: Notes from the Trenches of OpenType

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

uaan ⇒ uxn

Snaglog: Notes from the Trenches of OpenType

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

uaan ⇒ uxn

uxn⇒ uxn

Snaglog: Notes from the Trenches of OpenType

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

uaan ⇒ uxn

uxn⇒ uxn

uqn ⇒ uxn

Snaglog: Notes from the Trenches of OpenType

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

TO OTF

```
feature liga {
    sub q by x;
    sub a a by x;
} liga;
```

FROM OTF

uaan ⇒ uxn

uxn ⇒ uxn

uqn ⇒ uxn

The only plausible explanation for this riddle is a FontForge bug.

Snaglog: Notes from the Trenches of OpenType

FEATURES: A CONUNDRUM

Because the available documentation is unsatisfactory and unreliable, the only way to understand how OpenType feature processing actually works is through testing. Of course, for testing we use trivial yet fanciful (artificial) features – but even then, we ended up with a result that, to us, was an inexplicable conundrum.

The following example was prepared to test (using FontForge) whether the output of one feature rule can be picked up by a subsequent rule.

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

TO OTF

```
feature liga {
    sub a a by x;
    sub x by q;
} liga;
```

FROM OTF

uaav ⇒ uqv

uxv⇒ uqv

uqv ⇒ uqv

Fortunately, the newest FontForge (January 1, 2023) fixed it.

CONCLUSIONS

Given the number of problematic cases we've encountered (and I'm discussing only some of them here), despite our best efforts, we cannot guarantee that no bugs have slipped into the fonts we generated.

CONCLUSIONS

Given the number of problematic cases we've encountered (and I'm discussing only some of them here), despite our best efforts, we cannot guarantee that no bugs have slipped into the fonts we generated.

What's worse, we lack tools that would allow for a thorough inspection of the fonts. By far the most reliable and handy tool we've come across is LuaTeX – but unfortunately, even that is sometimes not enough.

CONCLUSIONS

Given the number of problematic cases we've encountered (and I'm discussing only some of them here), despite our best efforts, we cannot guarantee that no bugs have slipped into the fonts we generated.

What's worse, we lack tools that would allow for a thorough inspection of the fonts. By far the most reliable and handy tool we've come across is LuaTeX – but unfortunately, even that is sometimes not enough.

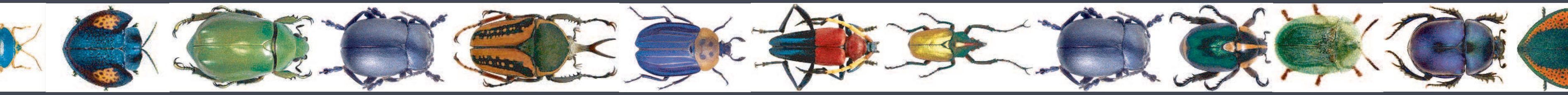
What we can certainly promise is that any reported issues with our fonts will be carefully analyzed, and we'll do our best to find an appropriate solution.

CONCLUSIONS – CONTINUED

The fonts we’ve mentioned will be available on the GUST website shortly after the meeting in Bachotek, and not long after that, they will also appear in the CTAN repository.

The published set will include:

- the Antykwa Półtawskiego family
- the Latin Modern family
- and the TeX Gyre collection.



CONCLUSIONS – CONTINUED

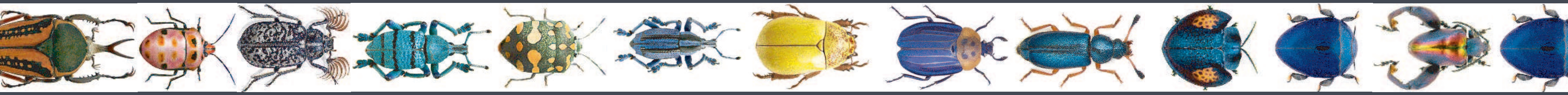
The fonts we've mentioned will be available on the GUST website shortly after the meeting in Bachotek, and not long after that, they will also appear in the CTAN repository.

The published set will include:

- the Antykwa Półtawskiego family
- the Latin Modern family
- and the TeX Gyre collection.

This is a set generated for the GUST e-Foundry using the latest version of the Fontplant software. We have not introduced any significant modifications to the fonts themselves, as our main goal was to test whether the rapidly evolving Fontplant is functioning correctly.

We hope to release a stable version of Fontplant in the near future – and that will be the time for polishing and fine-tuning the fonts.



LET'S MEET AT BACHOTEX NEXT YEAR