The OpenType layout features file: Structure and editing

Christopher Slye
Adobe Systems, Inc.
ATypI Lisbon, 28 September 2006

Introduction

What am I talking about?

Feature files, not features.

So-called *western* fonts

- Latin, Greek & Cyrillic scripts.
- Both “Pro” and “Std” fonts.
- *Adobe Originals* are Adobe’s most advanced Western fonts, with the most complex feature files.

Adobe type production

Adobe type production

- We write feature code as a plain text file, with a plain text editor.

Adobe type production

- We write feature code as a plain text file,
  with a plain text editor.
  If you like to script, or write HTML or some other kind of code,
  then you might be the right kind of person.

Adobe type production

- We write feature code as a plain text file,
  with a plain text editor.
  If you like to script, or write HTML or some other kind of code,
  then you might be the right kind of person.
- We build our fonts with tools found in our font development kit (FDK).

Adobe type production

- We write feature code as a plain text file,
  with a plain text editor.
  If you like to script, or write HTML or some other kind of code, then you might be the right kind of person.
- We build our fonts with tools found in our font development kit (FDK).
  Such as MakeOTF, which builds an OpenType font from a PFA font and a features file (among others).

Adobe type production

- We write feature code as a plain text file,
  with a plain text editor.
  If you like to script, or write HTML or some other kind of code, then you might be the right kind of person.
- We build our fonts with tools found in our font development kit (FDK).
  Such as MakeOTF, which builds an OpenType font from a PFA font and a features file (among others).
- Of course, there are other ways to do it.

Adobe type production

Adobe type production

- We follow the *OpenType Feature File Specification*.

Adobe type production

- We follow the *OpenType Feature File Specification*.
- We use a “character + feature” model.

Adobe type production

- We follow the *OpenType Feature File Specification*.
- We use a "character + feature" model.
  We used to assign Unicode PUA values to alternates and other glyphs, but we've stopped. All glyphs need access by way of a layout feature.

Adobe type production

- We follow the *OpenType Feature File Specification*.
- We use a "character + feature" model.
  We used to assign Unicode PUA values to alternates and other glyphs, but we've stopped. All glyphs need access by way of a layout feature.
- A lot of our older OpenType fonts have good examples of bad practices.

Our fonts tell the story.

From our first OpenType fonts, our approach to creating the features file has evolved and improved.

Features milestones

- Adobe Garamond Pro
  One of the first OpenType "Pro" fonts.
- Warnock Pro
  One of the first Adobe Originals created in the OpenType era.
- Caflisch Script Pro
  Complex contextual substitution interactions.
- Minion Pro (v2)
  Expanded, edited, fixed.
- Garamond Premier Pro
  A recent (and complex) font family.

Who are you?

Code example

```
feature case {
    sub @PUNCT1_DEFAULT by @PUNCT1_UC;
    sub @PUNCT3_DEFAULT by @PUNCT3_UC;
    sub @ACCENTS_LC by @ACCENTS_UC;
    sub sfthyphen by hyphen.cap;
    sub bulletoperator by periodcentered.cap;
    lookup LNUM;
} case;
```

There are different ways to
make a feature file.

There are also different ways to arrange the parts of the features file.

The big picture

What makes a features file.
Organizing files and text to maximize sanity and minimize errors.

Feature *files*

Feature *files*

- If data can be shared among fonts, then don’t duplicate it — share it.

Feature *files*

- If data can be shared among fonts, then don’t duplicate it — share it.
- Name files appropriately and consistently.

Feature *files*

- If data can be shared among fonts, then don't duplicate it — share it.
- Name files appropriately and consistently.
- Think about others.

Feature *files*

- If data can be shared among fonts, then don't duplicate it — share it.
- Name files appropriately and consistently.
- Think about others.

Maybe.

Feature *files*

- If data can be shared among fonts, then don't duplicate it — share it.
- Name files appropriately and consistently.
- Think about others.

  Maybe.
- Some parts can be:

Feature *files*

- If data can be shared among fonts, then don't duplicate it — share it.
- Name files appropriately and consistently.
- Think about others.

  Maybe.
- Some parts can be:

  family-wide
  style-wide
  font-specific

A simple font family directory

```
GaramondPremierPro/
    features.family
    Roman/
        Bold/
            features
            features.kern
        Regular/
            features
            features.kern
    Italic/
        BoldItalic/
            features
            features.kern
        Italic/
            features
            features.kern
```

A simple ‘features’ file

```
table head {
   FontRevision 2.015;
} head;

table OS/2 {
   FSType 8;
   TypoAscender 727;
   TypoDescender -273;
   Panose 2 4 5 3 5 2 1 2 2 3;
   XHeight 438;
   CapHeight 651;
} OS/2;

include (../../features.family)

feature kern {
   include (features.kern)
} kern;
```

The ‘include’ statement

```
include (../../features.family)
```

Some significant parts

- head table
- name table
- OS/2 table
- class definitions
- Substitution features (GSUB)
- Kerning (GPOS)
- other stuff

The head table

- Holds the font version number.
- Put it at the top — because if you or anyone else is looking for it, that's where they'll start!
- MakeOTF adds more info automatically.

head table

```
table head {
   FontRevision 1.014;
} head;
```

The name table

- Probably something which can be shared across the whole font family.

name table

```
table name {
  nameid 0 "\00a9 2005 Adobe Systems Incorporated. All rights reserved.";
  nameid 0 1 "\a9 2005 Adobe Systems Incorporated. All rights reserved.";
  nameid 11 "http://www.adobe.com/type";
  nameid 11 1 "http://www.adobe.com/type";
  nameid 14 "http://www.adobe.com/type/legal.html";
  nameid 14 1 "http://www.adobe.com/type/legal.html";
  nameid 7 "Garamond Premier is a trademark of Adobe Systems Incorporated
    in the United States and/or other countries.";
  nameid 7 1 "Garamond Premier is a trademark of Adobe Systems
    Incorporated in the United States and/or other countries.";
  nameid 9 "Robert Slimbach";
  nameid 9 1 "Robert Slimbach";
} name;
```

The OS/2 table

- One of the more annoying table.
- Adobe typically includes:
  Vertical metrics
  PANOSE numbers
  Embedding (fsType) value
- PANOSE values are usually font-specific, so each font needs its own OS/2 table.
- For more information, read the *Feature File Specification*, the *OpenType Specification*, or prowl the online discussions.

OS/2 table

```
table OS/2 {
    TypoAscender 725;
    TypoDescender -275;
    Panose 2 2 4 2 6 5 6 2 4 3;
    XHeight 390;
    CapHeight 641;
    FSType 8;
} OS/2;
```

Class definitions

- More important than they seems.
- Well-designed classes can make feature code easier, save you time, and reduce errors.
- Strike a balance between functional, useful classes, and unnecessary clutter.
- Class definitions used for kerning are a somewhat different kind of thing.

  Still important to create carefully, but usually distinct from classes used for substitutions.

Class definitions

```
@FIG_TAB_LINING = [zero one two three four five six
seven eight nine];

@FIG_FIT_OLDSTYLE = [zero.oldstyle one.oldstyle
two.oldstyle three.oldstyle four.oldstyle five.oldstyle
six.oldstyle seven.oldstyle eight.oldstyle
nine.oldstyle];

@PUNCT3_DEFAULT = [exclamdown questiondown];

@PUNCT3_SC = [exclamdown.sc questiondown.sc];

@PUNCT3_UC = [exclamdown.cap questiondown.cap];
```

Substitution features (GPOS)

- Probably what you think of when you think about OpenType features.
- The most fun, but also the most work.

Substitution features (GPOS)

```
feature ornm {
   sub bullet from @ORNAMENTS;
} ornm;

feature zero {
   sub zero by zero.slash;
   sub zero.fitted by zero.slashfitted;
} zero;

feature ordn {
   sub a by ordfeminine;
   sub o by ordmasculine;
} ordn;
```

Kerning (GSUB)

- If there is any feature code that you want to generate automatically or with a script, *this is it*.
- Getting kerning feature code into the font can be a huge task.

Kerning (GSUB)

```
@A_LC = [a aacute acircumflex adieresis agrave aring
atilde abreve amacron aogonek adotbelow ahook
acircumflexacute acircumflexgrave acircumflexhook
acircumflextilde acircumflexdotbelow abreveacute
abrevegrave abrevehook abrevetilde abrevedotbelow
aringacute];

@ROUND_LC_LEFT_CYR = [be.ital o.cyr er ef e.cyr yu fita
schwa];
```

```
enum pos @V_UC [egrave ebreve ecaron emacron] -94;
pos @A_UC_LEFT @VWY_LC -66;
pos @A_UC_CYR @ECYR_LC_RIGHT_CYR 11;
pos @ALPHA_LC_GRK @BETA_ALT_LC_GRK -3;
```

Other stuff

- size feature
- BASE table

size feature

```
feature size {
    parameters
        110 # design size (decipoints)
        3   # subfamily identifier
        84  # range start (exclusive, decipoints)
        130 # range end (inclusive, decipoints)
    ;
    sizemenuname "Regular"; # Windows Unicode menu name
    sizemenuname 1 "Regular"; # Mac Roman menu name
} size;
```

BASE table

```
table BASE {
   HorizAxis.BaseTagList ideo romn;
   HorizAxis.BaseScriptList
      latn romn -177 0,
      cyrl romn -177 0,
      grek romn -177 0;
} BASE;
```

Substitution features

Also known as “GSUB.”

Basic substitution is easy

```
feature salt {
   sub a by a.alt;
   sub b from [b.alt1 b.alt2 b.alt3];
} salt;
```

```
feature liga {
   sub f f i by f_f_i;
   sub f i by f_i;
} liga;

feature calt {
   sub g y' by y.alt;
} calt;
```

Order can be important

```
feature liga {
   sub f f by f_f;
   sub f i by f_i;
   sub f f i by f_f_i;
} liga;
```

Order can be important

```
feature liga {
   sub f f by f_f;
   sub f i by f_i;
   sub f f i by f_f_i;
} liga;
```

The layout engine will "stop" after it matches something.
`f f i` *will become* `f_f i`.

Order can be important

This will work better:

```
feature liga {
   sub f f i by f_f_i;
   sub f f by f_f;
   sub f i by f_i;
} liga;
```

Some feature code is not entirely necessary, but makes you feel good.

And maybe it's safer.

Over-engineering?

```
feature onum {
    sub @FIG_DEFAULT by @FIG_TAB_OLDSTYLE;
    sub @FIG_FIT_LINING by @FIG_FIT_OLDSTYLE;
} onum;

feature lnum {
    sub @FIG_TAB_OLDSTYLE by @FIG_DEFAULT;
    sub @FIG_FIT_OLDSTYLE by @FIG_FIT_LINING;
} lnum;
```

Over-engineering?

```
feature onum {
    sub @FIG_DEFAULT by @FIG_TAB_OLDSTYLE;
    sub @FIG_FIT_LINING by @FIG_FIT_OLDSTYLE;
} onum;

feature lnum {
    sub @FIG_TAB_OLDSTYLE by @FIG_DEFAULT;
    sub @FIG_FIT_OLDSTYLE by @FIG_FIT_LINING;
} lnum;
```

Some substitutions are not necessary, because the input glyphs will not be present before other layout features are applied.

Over-engineering?

```
feature onum {
   sub @FIG_DEFAULT by @FIG_TAB_OLDSTYLE;
} onum;

feature lnum {
   sub @FIG_TAB_OLDSTYLE by @FIG_DEFAULT;
} lnum;

feature pnum {
   sub @FIG_DEFAULT by @FIG_FIT_LINING;
   sub @FIG_TAB_OLDSTYLE by @FIG_FIT_OLDSTYLE;
} pnum;

feature tnum {
   sub @FIG_FIT_LINING by @FIG_DEFAULT;
   sub @FIG_FIT_OLDSTYLE by @FIG_TAB_OLDSTYLE;
} tnum;
```

Over-engineering?

```
feature onum {
   sub @FIG_DEFAULT by @FIG_TAB_OLDSTYLE;
} onum;

feature lnum {
   sub @FIG_TAB_OLDSTYLE by @FIG_DEFAULT;
} lnum;

feature pnum {
   sub @FIG_DEFAULT by @FIG_FIT_LINING;
   sub @FIG_TAB_OLDSTYLE by @FIG_FIT_OLDSTYLE;
} pnum;

feature tnum {
   sub @FIG_FIT_LINING by @FIG_DEFAULT;
   sub @FIG_FIT_OLDSTYLE by @FIG_TAB_OLDSTYLE;
} tnum;
```

Over-engineering?

```
feature onum {
   sub @FIG_DEFAULT by @FIG_TAB_OLDSTYLE;
} onum;

feature lnum {
   sub @FIG_TAB_OLDSTYLE by @FIG_DEFAULT;
} lnum;

feature pnum {
   sub @FIG_DEFAULT by @FIG_FIT_LINING;
   sub @FIG_TAB_OLDSTYLE by @FIG_FIT_OLDSTYLE;
} pnum;

feature tnum {
   sub @FIG_FIT_LINING by @FIG_DEFAULT;
   sub @FIG_FIT_OLDSTYLE by @FIG_TAB_OLDSTYLE;
} tnum;
```

Over-engineering? Maybe not.

Over-engineering? Maybe not.

- Some applications might expect all four layout features to be present.

Over-engineering? Maybe not.

- Some applications might expect all four layout features to be present.
- The presence of all four features in the font conveys some information about what glyphs are in the font.

Over-engineering? Maybe not.

- Some applications might expect all four layout features to be present.
- The presence of all four features in the font conveys some information about what glyphs are in the font.
- Having all possibilities means you worry less.

Over-engineering? Maybe not.

- Some applications might expect all four layout features to be present.
- The presence of all four features in the font conveys some information about what glyphs are in the font.
- Having all possibilities means you worry less.
- *But* ... you should be able to trust the font and layout engines to do the right thing.

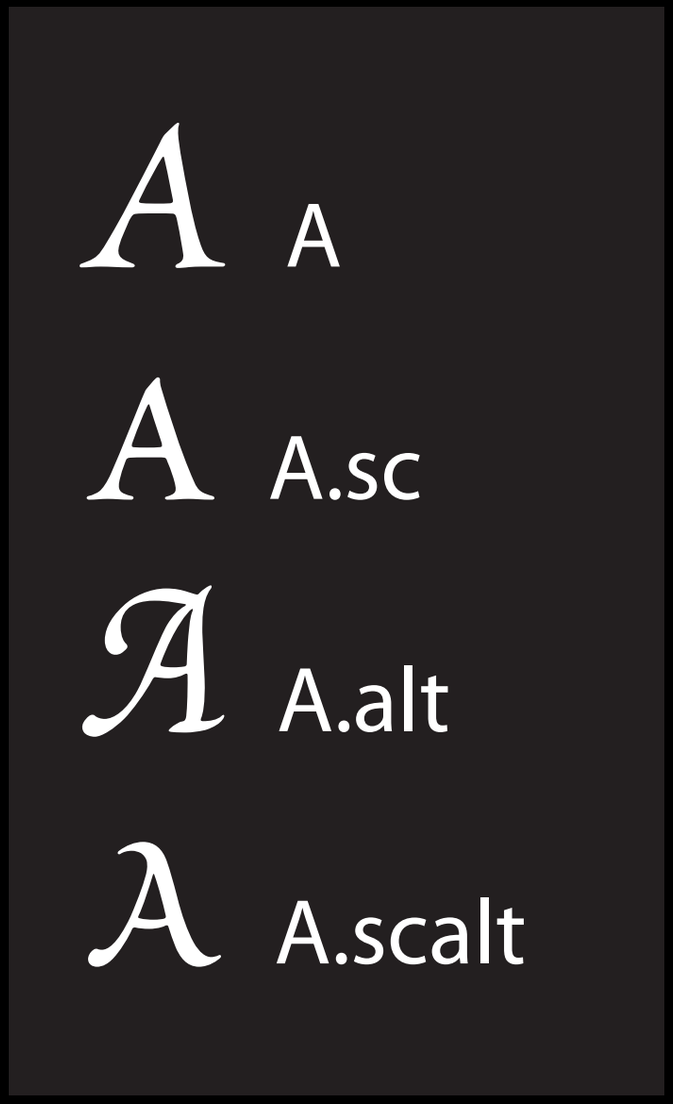
Now here’s an even scarier example...

Feature order

- Dictates the order in which features are applied during OpenType layout.
  At least it should.
- The most perilous part of the feature file.
- Not all answers are obvious.
- Not all problems can be solved without creating new problems.

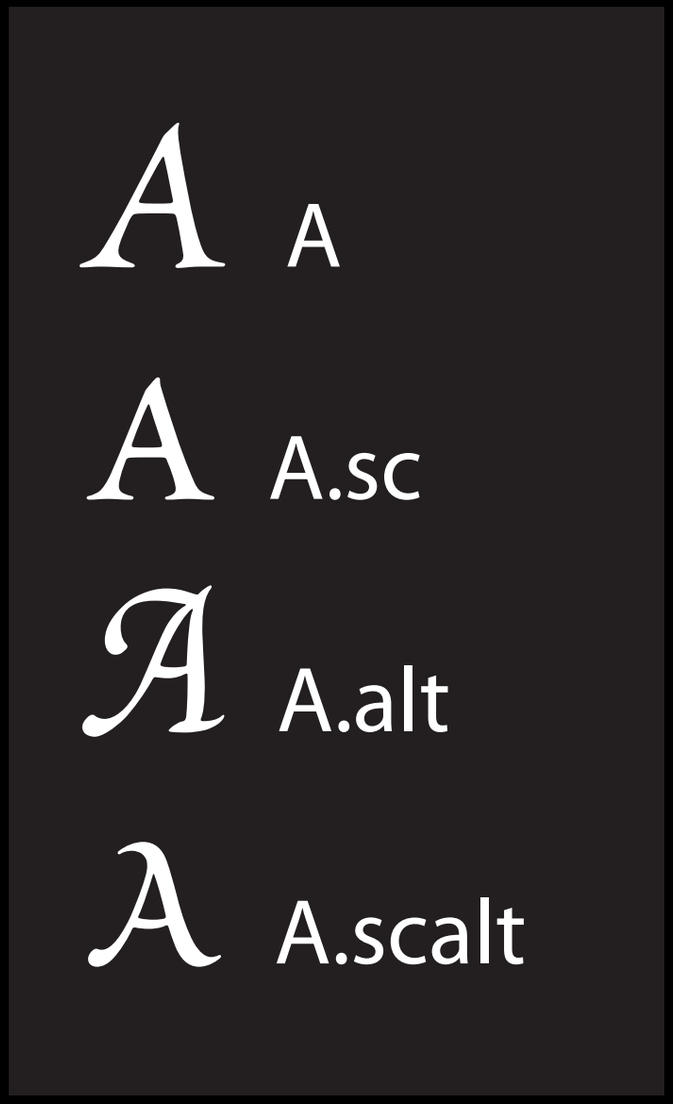
Feature order: an example

Feature order: an example



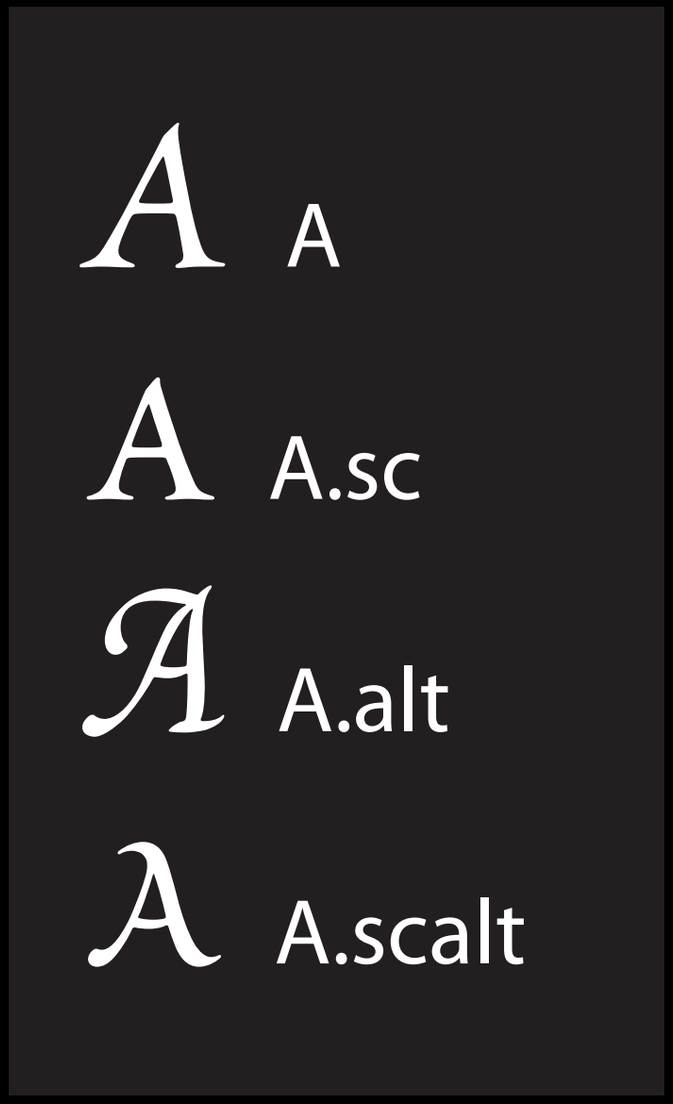
- One character, **A**, represented by four glyphs:

Feature order: an example



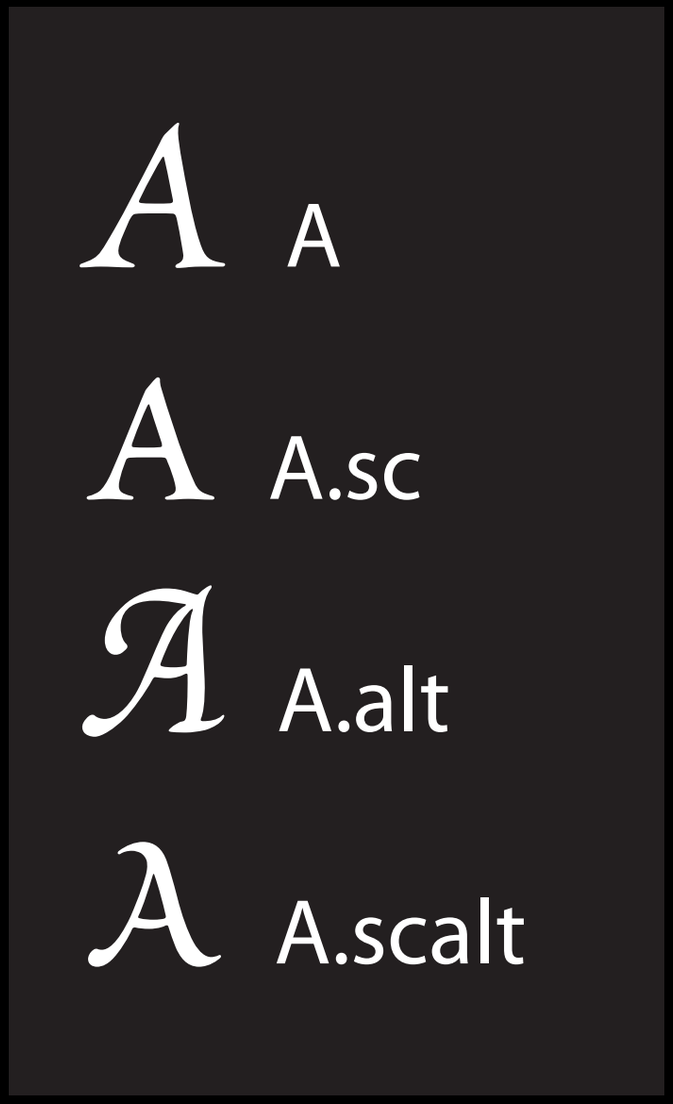
- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt

Feature order: an example



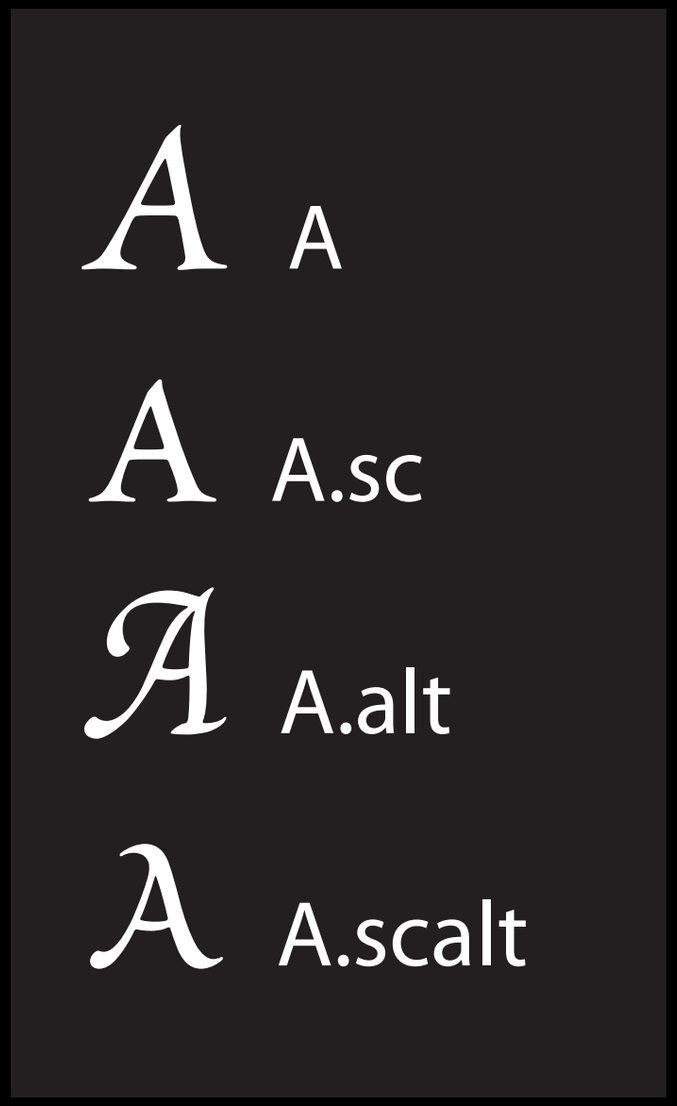
- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt
- **A.sc** is the small-cap version of **A**.

Feature order: an example



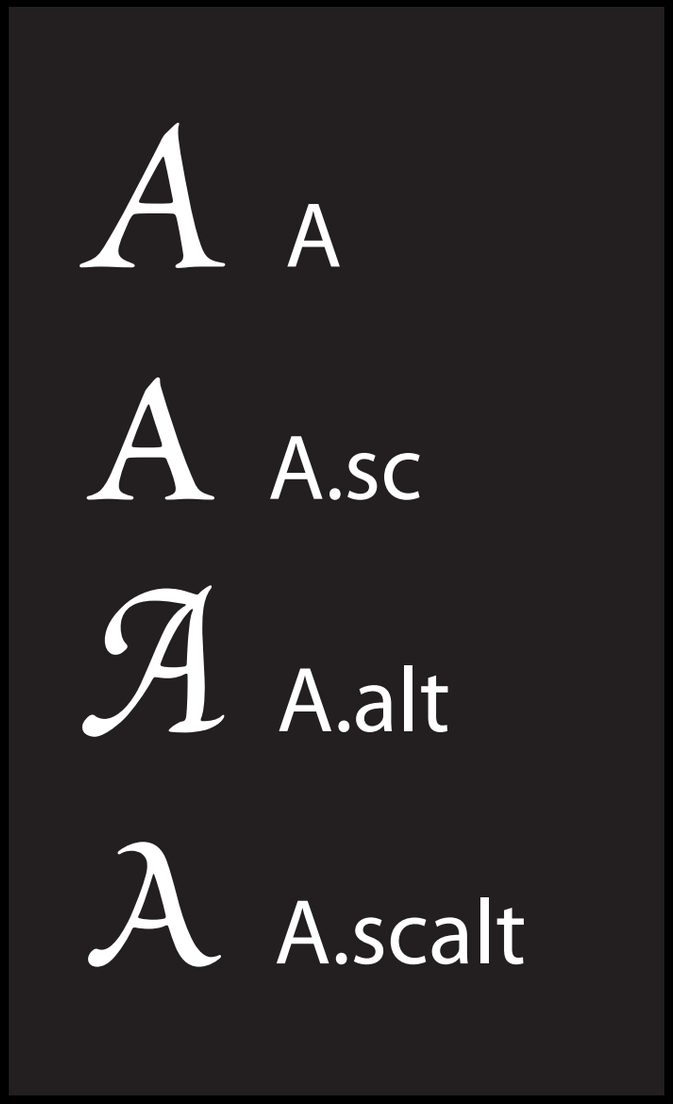
- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt
- **A.sc** is the small-cap version of **A**.
- **A.alt** is an alternate of **A**.

Feature order: an example



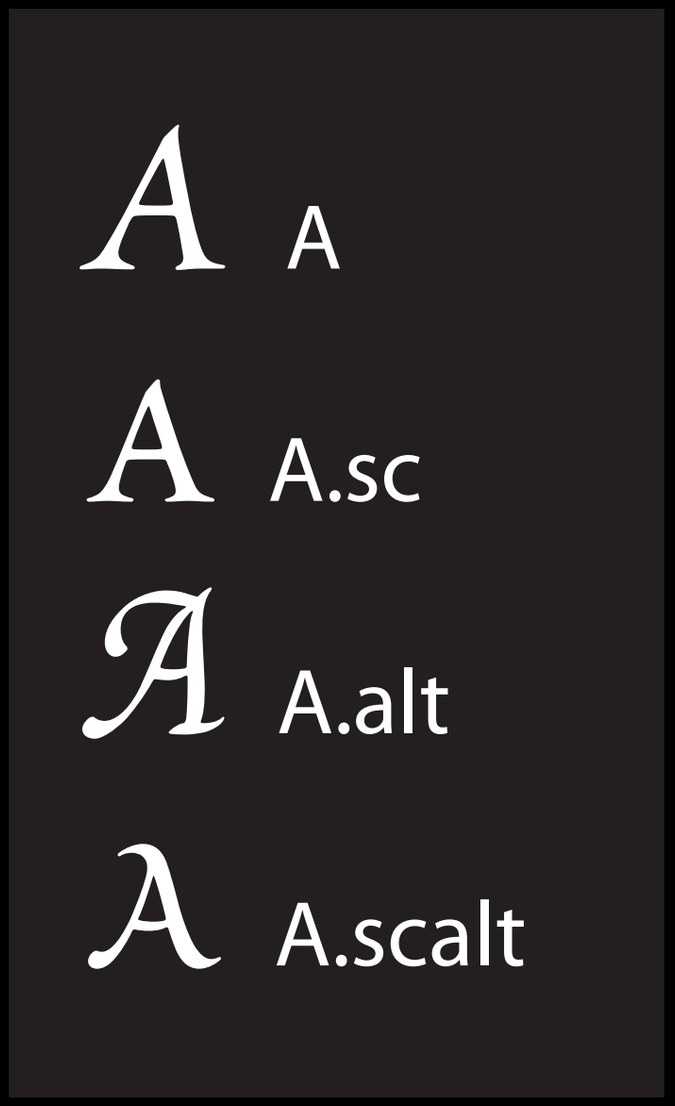
- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt
- **A.sc** is the small-cap version of **A**.
- **A.alt** is an alternate of **A**.
- **A.scalt** is an alternate of **A.sc**.

Feature order: an example



- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt
- **A.sc** is the small-cap version of **A**.
- **A.alt** is an alternate of **A**.
- **A.scalt** is an alternate of **A.sc**.
- **A.scalt** is *not* the small-cap version of **A.alt**.

Feature order: an example



- One character, **A**, represented by four glyphs:
  A, A.alt
  A.sc, A.scalt
- **A.sc** is the small-cap version of **A**.
- **A.alt** is an alternate of **A**.
- **A.scalt** is an alternate of **A.sc**.
- **A.scalt** is *not* the small-cap version of **A.alt**.
  *It is only the small-cap version of plain **A**.*

Feature order: an example

```
feature salt {
   sub A by A.alt;
   sub A.sc by A.scalt
} salt;

feature c2sc {
   sub [A A.alt] by A.sc;
} c2sc;
```

Feature order: an example

```
feature salt {
   sub A by A.alt;
   sub A.sc by A.scalt
} salt;

feature c2sc {
   sub [A A.alt] by A.sc;
} c2sc;
```

No way to get `A.scalt`.
If both 'salt' and 'c2sc' are applied,
`A` *first becomes* `A.alt`,
then `A.alt` *becomes* `A.sc`.

Feature order: an example

```
feature c2sc {
   sub [A A.alt] by A.sc;
} c2sc;

feature salt {
   sub A by A.alt;
   sub A.sc by A.scalt;
} salt;
```

Feature order: an example

```
feature c2sc {
   sub [A A.alt] by A.sc;
} c2sc;

feature salt {
   sub A by A.alt;
   sub A.sc by A.scalt;
} salt;
```

`A.alt` does not become `A.sc`, as desired.
If 'c2sc' is applied to `A.alt`, then 'salt' must already be applied to `A`.
`A` becomes `A.sc`, and then `A.sc` becomes `A.scalt`.

Language systems and lookups

Controlling what happens in different scripts and languages.

Language system

- A *language system* is a combination of a script and language tag.
- Specifying language systems ensures that all features behave properly in the various scripts and languages in which they are used.
- Two special tags, *DFLT* and *dflt*, are used to identify a default language system, in environments where a script and language is unspecified.

‘languagesystem’

- The simplest use of a language system is the ‘DFLT’ script and ‘dflt’ language, specified in a single statement, before all layout features:

```
languagesystem DFLT dflt;
```

- This tells the layout engine, “Unless specified otherwise, use DFLT/dflt for everything.”

‘languagesystem’

```
languagesystem DFLT dflt;
languagesystem latn dflt;
languagesystem cyrl dflt;
languagesystem grek dflt;
languagesystem latn AZE;
languagesystem latn CRT;
languagesystem latn DEU;
languagesystem latn FRA;
languagesystem latn MOL;
languagesystem latn ROM;
languagesystem latn TRK;
languagesystem cyrl SRB;
```

All language systems used elsewhere must also be specified here.

Script and language

Script and language

- One or more language systems can be specified within a feature.

Script and language

- One or more language systems can be specified within a feature.
- This tells the layout engine, “Never mind, I am going to specify everything in this feature manually.”

Script and language

- One or more language systems can be specified within a feature.
- This tells the layout engine, “Never mind, I am going to specify everything in this feature manually.”
- This is how you change OpenType layout for different languages and scripts.

Script and language

- One or more language systems can be specified within a feature.
- This tells the layout engine, “Never mind, I am going to specify everything in this feature manually.”
- This is how you change OpenType layout for different languages and scripts.
  But your OS or layout engine has to help.

Script and language

- One or more language systems can be specified within a feature.
- This tells the layout engine, “Never mind, I am going to specify everything in this feature manually.”
- This is how you change OpenType layout for different languages and scripts.

  But your OS or layout engine has to help.
- When you specify a new language system, it becomes one of the possible language systems for the entire font.

Script and language

```
languagesystem DFLT dflt;
languagesystem latn dflt;
languagesystem latn DEU;

feature dlig {
   script DFLT;
      language dflt;
         sub c t by c_t;
         sub s t by s_t;
   script latn;
      language dflt include_dflt;
      language DEU include_dflt;
         sub c h by c_h;
         sub c k by c_k;
} dlig;
```

Script and language

```
languagesystem DFLT dflt;
languagesystem latn dflt;
languagesystem latn DEU;

feature dlig {
   script DFLT;
      language dflt;
         sub c t by c_t;
         sub s t by s_t;
   script latn;
      language dflt include_dflt;
      language DEU include_dflt;
         sub c h by c_h;
         sub c k by c_k;
} dlig;
```

Script and language

```
languagesystem DFLT dflt;
languagesystem latn dflt;
languagesystem latn DEU;

feature dlig {
   script DFLT;
      language dflt;
         sub c t by c_t;
         sub s t by s_t;
   script latn;
      language dflt include_dflt;
      language DEU include_dflt;
         sub c h by c_h;
         sub c k by c_k;
} dlig;
```

Script and language

```
languagesystem DFLT dflt;
languagesystem latn dflt;
languagesystem latn DEU;

feature dlig {
   script DFLT;
      language dflt;
         sub c t by c_t;
         sub s t by s_t;
   script latn;
      language dflt include_dflt;
      language DEU include_dflt;
         sub c h by c_h;
         sub c k by c_k;
} dlig;
```

Script and language

	DFLT/dflt	latn/dflt	latn/DEU
liga	●	●	●
dlig	●	●	●
salt	●	●	●

Saving the i-dot

- What if you don’t want that dot on the i to disappear in a particular language?
  This often happens with fi ligatures.
- Turkish is one such language. (There are others.)

Saving the i-dot

```
feature liga {
   script DFLT;
      language dflt;
      sub f i by f_i;
   script latn;
      language dflt include_dflt;
      language TRK exclude_dflt;
} liga;
```

Saving the i-dot

```
feature liga {
   script DFLT;
      language dflt;
      sub f i by f_i;
   script latn;
      language dflt include_dflt;
      language TRK exclude_dflt;
} liga;
```

Declare special handling for script ‘latin’ and language ‘TRK’.
Default subsitutions will be excluded.

With many ligatures and language systems, this approach can get messy.

Saving the i-dot

```
feature locl {
   script latn;
      language TRK;
      sub i by i.dot;
} locl;

feature liga {
   sub f i by f_i;
} liga;
```

Saving the i-dot

```
feature locl {
   script latn;
      language TRK;
      sub i by i.dot;
} locl;

feature liga {
   sub f i by f_i;
} liga;
```

Create a duplicate of `i` *named* `i.dot`*, and substitute it in the 'locl' feature.*

Saving the i-dot

```
feature locl {
   script latn;
      language TRK;
      sub i by i.dot;
} locl;

feature liga {
   sub f i by f_i;
} liga;
```

Later, a ligature will be substituted for `f` *and* `i`*, but not* `i.dot`*.*

Lookups

- In a feature file, a lookup is a labeled group of rules which can be referenced later in the features.
- Like a glyph class, it saves space and prevents errors by helping keep things identical where they need to be.

Lookups

```
feature ordn {
    lookup ORDN {
        sub @LETTERS_LC by @LETTERS_SUPERIOR;
    } ORDN;
} ordn;
```

```
feature sups {
    sub @FIG_DEFAULT by @FIG_SUPERIOR;
    sub @PUNCT_DEFAULT by @PUNCT_SUPERIOR;
    lookup ORDN;
} sups;
```

Lookups

```
feature ordn {
    lookup ORDN {
        sub @LETTERS_LC by @LETTERS_SUPERIOR;
    } ORDN;
} ordn;
```

```
feature sups {
    sub @FIG_DEFAULT by @FIG_SUPERIOR;
    sub @PUNCT_DEFAULT by @PUNCT_SUPERIOR;
    lookup ORDN;
} sups;
```

Lookups

Another advantage of lookups is that they allow you to “match” more than one substitution rule within a feature.

Lookups

```
feature calt {
   sub [h m n]' [c d o] by [h.alt m.alt n.alt];
   sub n.alt' d by n.alt2;
} calt;
```

With the code above, the second substitution will never happen.

```
feature calt {
   lookup HMN {
      sub [h m n]' [c d o] by [h.alt m.alt n.alt];
   } HMN;
   sub n.alt' d by n.alt2;
} calt;
```

Separated by a lookup, the layout engine will continue after matching the first substitution.

Kerning

Something that could be an entirely separate talk. (Oh, wait, it is.)

Two different kinds of kerning

- A table for “specific” kerns holds single glyphs kerned to single glyphs.
- A table for class kerns holds glyph classes kerned to other glyph classes.

Specific kerns

```
feature kern {
   pos A V -20;
   pos A W -20;
} kern;
```

Or, this does the same thing:

```
feature kern {
   enum pos A [V W] -20;
} kern;
```

Class kerns

```
feature kern {
   pos @A_UC @VW_UC -20;
} kern;
```

Use both to make exceptions

```
feature kern {
   enum pos [Aacute Acircumflex] @VW_UC -10;
   pos @A_UC @VW_UC -20;
} kern;
```

If two glyphs match in the first line, the kern in the second line will not be applied.

That was stupid

Some examples from old and ill-conceived feature code.

Adobe Garamond Pro (c. 1999)

```
# Adobe Garamond Regular Pro Feature File
# Version 99.03.10.00
```

This feature file from an early version of Adobe Garamond Pro, circa March 1999, shows some interesting and amusing approaches to OpenType layout.

Adobe Garamond Pro (c. 1999)

```
# --- FEATURE SUMMARY
# --- Substitution
#     aalt: access all alternates
#     smcp: lc to smallcaps
#     c2sc: caps to smallcaps
#     case: uppercase, math and punct
#     titl: replaces default glyphs
#     fina: replaces glyphs at ends of
#     onum: changes to oldstyle
#     lnum: changes to lining figures
#     pnum: changes to proportional
#     tnum: changes to tabular figures
#     crcy: replaces any currency char
#     sups: makes numbers superior
#     sinf: makes numbers inferior
#     numr: replaces selected figures
#     dnom: replaces selected figures
#     frac: substitutes the existing
#     dpng: dipthong subs
#     liga: standard lig replacement
#     salt: replaces glyphs with
#     dlig: discretionary ligatures
#     ordn: subs ordinal glyphs after
#     ornm: allows access to ornaments
#     zero: slashed zero
```

Adobe Garamond Pro (c. 1999)

```
# --- FEATURE SUMMARY
# --- Substitution
#     aalt: access all alternates
#     smcp: lc to smallcaps
#     c2sc: caps to smallcaps
#     case: uppercase, math and punct
#     titl: replaces default glyphs
#     fina: replaces glyphs at ends of
#     onum: changes to oldstyle
#     lnum: changes to lining figures
#     pnum: changes to proportional
#     tnum: changes to tabular figures
#     crcy: replaces any currency char
#     sups: makes numbers superior
#     sinf: makes numbers inferior
#     numr: replaces selected figures
#     dnom: replaces selected figures
#     frac: substitutes the existing
#     dpng: dipthong subs
#     liga: standard lig replacement
#     salt: replaces glyphs with
#     dlig: discretionary ligatures
#     ordn: subs ordinal glyphs after
#     ornm: allows access to ornaments
#     zero: slashed zero
```

These commented lines at the top of the feature file show an early “table of contents” which we included for our own information.

Adobe Garamond Pro (c. 1999)

```
# --- FEATURE SUMMARY
# --- Substitution
#     aalt: access all alternates
#     smcp: lc to smallcaps
#     c2sc: caps to smallcaps
#     case: uppercase, math and punct
#     titl: replaces default glyphs
#     fina: replaces glyphs at ends of
#     onum: changes to oldstyle
#     lnum: changes to lining figures
#     pnum: changes to proportional
#     tnum: changes to tabular figures
#     crcy: replaces any currency char
#     sups: makes numbers superior
#     sinf: makes numbers inferior
#     numr: replaces selected figures
#     dnom: replaces selected figures
#     frac: substitutes the existing
#     dpng: dipthong subs
#     liga: standard lig replacement
#     salt: replaces glyphs with
#     dlig: discretionary ligatures
#     ordn: subs ordinal glyphs after
#     ornm: allows access to ornaments
#     zero: slashed zero
```

These commented lines at the top of the feature file show an early “table of contents” which we included for our own information.

Such notes were not accurately maintained and quickly lost their usefulness.

Adobe Garamond Pro (c. 1999)

```
# --- FEATURE SUMMARY
# --- Substitution
#     aalt: access all alternates
#     smcp: lc to smallcaps
#     c2sc: caps to smallcaps
#     case: uppercase, math and punct
#     titl: replaces default glyphs
#     fina: replaces glyphs at ends of
#     onum: changes to oldstyle
#     lnum: changes to lining figures
#     pnum: changes to proportional
#     tnum: changes to tabular figures
#     crcy: replaces any currency char
#     sups: makes numbers superior
#     sinf: makes numbers inferior
#     numr: replaces selected figures
#     dnom: replaces selected figures
#     frac: substitutes the existing
#     dpng: dipthong subs
#     liga: standard lig replacement
#     salt: replaces glyphs with
#     dlig: discretionary ligatures
#     ordn: subs ordinal glyphs after
#     ornm: allows access to ornaments
#     zero: slashed zero
```


Notice that we were using two features, now deprecated: ‘crcy’ and ‘dpng’.

Adobe Garamond Pro (c. 1999)

```
feature frac {
  lookup numerator {
    sub @FIG_ONE' @FRACTION @FIG_TWO by [one one one one one one one];
    sub @FIG_ONE' @FRACTION @FIG_FOUR by [one one one one one one one];
    sub @FIG_THREE' @FRACTION @FIG_FOUR by [three three three three three
    sub @FIG_ONE' @FRACTION @FIG_THREE by [one one one one one one one];
    sub @FIG_TWO' @FRACTION @FIG_THREE by [two two two two two two two];
    sub @FIG_ONE' @FRACTION @FIG_EIGHT by [one one one one one one one];
    sub @FIG_THREE' @FRACTION @FIG_EIGHT by [three three three three three
    sub @FIG_FIVE' @FRACTION @FIG_EIGHT by [five five five five five five
    sub @FIG_SEVEN' @FRACTION @FIG_EIGHT by [seven seven seven seven seven
    sub @FIG_ZERO' @FRACTION @FIG_ZERO @FIG_ZERO by [zero zero zero zero
  } numerator;

  lookup denominator {
    sub @FIG_ONE @FRACTION @FIG_TWO' by [two two two two two two two];
    sub @FIG_ONE @FRACTION @FIG_FOUR' by [four four four four four four
```

The ‘frac’ feature contains lookups which change different numbers in fraction contexts into default numbers, then ...

Adobe Garamond Pro (c. 1999)

```
lookup fractions {
  sub one @FRACTION two by onehalf;
  sub one @FRACTION four by onequarter;
  sub three @FRACTION four by threequarters;
  sub one @FRACTION three by onethird;
  sub two @FRACTION three by twothirds;
  sub one @FRACTION eight by oneeighth;
  sub three @FRACTION eight by threeeighths;
  sub five @FRACTION eight by fiveeighths;
  sub seven @FRACTION eight by seveneighths;
  sub zero @FRACTION zero zero by perthousand;
  sub zero @FRACTION zero by percent;
  } fractions;
}frac;
```

... changes those default numbers into pre-built fractions.

A better way...

```
feature frac {
    sub @FIG_ONE @SLASH @FIG_TWO by onehalf;
    sub @FIG_ONE @SLASH @FIG_FOUR by onequarter;
    sub @FIG_THREE @SLASH @FIG_FOUR by threequarters;
    sub @FIG_ONE @SLASH @FIG_THREE by onethird;
    sub @FIG_TWO @SLASH @FIG_THREE by twothirds;
    sub @FIG_ONE @SLASH @FIG_EIGHT by oneeighth;
    sub @FIG_THREE @SLASH @FIG_EIGHT by threeeighths;
    sub @FIG_FIVE @SLASH @FIG_EIGHT by fiveeighths;
    sub @FIG_SEVEN @SLASH @FIG_EIGHT by seveneighths;
    sub @FIG_ZERO @SLASH @FIG_ZERO @FIG_ZERO by
        perthousand;
    sub @FIG_ZERO @SLASH @FIG_ZERO by percent;
} frac;
```

There’s a lot more
I didn’t mention.

The end

Thanks.