The OpenType Layout Model

OpenType layout data is organized by script, language system, typographic feature and lookup.
**1. A short history of font technology**

**Before 1980**
Proprietary and hardware dependent font formats (bitmap, vector).

**1974–1978**
Ikarus outline font format (open format, machine independent, font database format).

**Mid 1980s**
Scalable font formats (outline + hints).
- URW vs, BS.
- Type 1 (based on Bezier and URW-like hints).
- F3, Bitstreams Speedo and others …

**Late 1980s**
Development of TrueType by Apple (Unicode based, instructions, flexible and expandable).
- Implementation on the Macintosh in 1990.

**1991**
Opening of Type 1 Format (Adobe) (1-Byte font format).
To font format for 2-Byte fonts.

**1993**
CID font format for CJK (2-Byte).
- Took 5–6 years to appear on the market.

**1994**
TrueType GX (advanced Layout features).
- Failed on the market.

**1995**
TTO (multilingual support, Layout features for Arabic).
TTC (TrueType Collection Files).

**1996**
SFNT-Wrapped CID Fonts (Adobe, Mac platform).

**1997**
OpenType specification.

### 1.1 Conclusion

- Font technology has been rapidly developed during the last 20 years.
- Font technology has become a very important part in the computerized world.
- Parallel to globalization, fonts have been extended to complex scripts like Arabic, Indic, Thai etc. and to large character sets for China, Japan and Korea.
- Fonts are becoming more and more complex, which puts more pressure on the font developer and designer.
- The evolution of the font formats also allows the use of fine typographic features.
2. What is OpenType?
Open Type is more than a simple font format, it is an architecture with building blocks:
– OpenType fonts.
– Operating System support.
– Application support.
– Printer support.

OpenType fonts have four essential ingredients:
– Outline description (Bezier, quadratic splines …).
– Hinting information for screen optimization (hints, instructions).
– Character mapping tables.
– Features (for glyph substitution and positioning).

OpenType fonts come in two flavours:
– Type 1 outlines, hints (.otf)
– TrueType outlines, instructions (.ttf)

There is no standard as to what an OpenType font must contain (this might be difficult for the customer and but also for marketing):
– 256 – >50000 glyphs.
– hundreds of features or none.

2.1 os Support
OpenType fonts should work on different platforms (Windows, Mac os, Linux). Windows 2000 and xp support both otf flavours natively and support many features (not all) through its Uniscribe API and the otls (OpenType Layout Services Library). Mac os 9.2 and os x support for both otf flavours is limited. Glyph access and rendering is supported but there is no os support for layout features. Apple supports instead its own Apple Advanced Technology (aat) technology, which is a renamed version of gx. This means that fonts which should work on both platforms must support both OpenType layout tables as well as the aat tables. Linux should support OpenType through Freetype.

2.2 Applications
Applications are using the outlines, hints and feature tables. Adobe has implemented the feature font support into the applications such as InDesign, PhotoShop, etcetera. These programs are platform independent, and os independent).
3. The structure of OpenType fonts

OpenType fonts have a common table structure like TTFs (also called SFNT on the Macintosh). OpenType Fonts may use Type 1-like outlines and hints or TrueType-like outlines and hints. The reason for that was probably that neither Microsoft nor Adobe wanted to throw away the considerable amount of work which had been done on the Type 1 and TrueType architectures.

Advantages of Type 1-like outlines (CFF table):
– Simple hinting structure, intelligence in the rasterizer.
– Thousands of existing Type 1 fonts can be converted without quality loss.
– Bezier outlines are familiar to (type) designers.

Advantages of TrueType outlines (GLYP table):
– Powerful instructions for superb screen quality.
– Quadratic spline outlines.

Other information is stored in common tables, such as:
– cmap for the mapping of glyphs —> Unicode code points.
– head, hhea for header information.
– os/2 for general font information.
– Gasp for greyscaling.

Essential for OpenType are the following tables:
– GPOS  glyph positioning
– GSUB  glyph substitution
– GDEF  glyph definition
– BASE  baseline table for different scripts
– JSTF  justification
– DSIG  digital signature

The main difference with simple TrueType fonts is the presence of some of the above listed tables which allow access to glyphs which have no direct Unicode codepoint. For complex scripts, i.e. writing systems that require some degree of character reordering and/or glyph processing to display, print or edit text (such as Arabic or Indic) Open Type tables are absolutely necessary.

Using this technology permits the font developer to implement:
– OpenType Layout fonts allow a rich mapping between characters and glyphs, which supports ligatures, positional forms, alternates, and other substitutions.
– OpenType Layout fonts include information to support features for two-dimensional positioning and glyph attachment.
– OpenType Layout fonts contain explicit script and language information,
<table>
<thead>
<tr>
<th>Font Format</th>
<th>TrueType (ttf)</th>
<th>OpenType (ttf)</th>
<th>OpenType (otf)</th>
<th>SFNT-CID (Adobe)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Required</strong></td>
<td>head, hhea, hmtx</td>
<td>head, hhea, hmtx</td>
<td>head, hhea, hmtx</td>
<td>cmap</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>name</td>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td>os/2</td>
<td>os/2</td>
<td>os/2</td>
<td>post</td>
</tr>
<tr>
<td></td>
<td>maxp</td>
<td>maxp</td>
<td>maxp</td>
<td>post</td>
</tr>
<tr>
<td></td>
<td>post</td>
<td>post</td>
<td>post</td>
<td>cmap</td>
</tr>
<tr>
<td></td>
<td>cmap</td>
<td>cmap</td>
<td>cmap</td>
<td></td>
</tr>
<tr>
<td><strong>Outline</strong></td>
<td>glyf, loca</td>
<td>glyf, loca</td>
<td>glyf, loca</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cvt, fpgm, prep</td>
<td>cvt, fpgm, prep</td>
<td>cvt, fpgm, prep</td>
<td></td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>gasp</td>
<td>gasp</td>
<td>gasp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hdmx</td>
<td>hdmx</td>
<td>hdmx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kern</td>
<td>kern</td>
<td>kern</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LTSH</td>
<td>LTSH</td>
<td>LTSH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vhea</td>
<td>vhea</td>
<td>vhea</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vmtx</td>
<td>vmtx</td>
<td>vmtx</td>
<td></td>
</tr>
<tr>
<td><strong>Bitmap</strong></td>
<td>EBDT</td>
<td>EBDT</td>
<td>BASE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EBLC</td>
<td>EBLC</td>
<td>BASE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EBSC</td>
<td>EBSC</td>
<td>BASE</td>
<td></td>
</tr>
<tr>
<td><strong>OTF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>AAT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Adobe</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The TrueType Font File (Apple’s specification AAT)

<table>
<thead>
<tr>
<th>acnt</th>
<th>accent attachment table</th>
<th>hhea</th>
<th>horizontal header table</th>
</tr>
</thead>
<tbody>
<tr>
<td>avar</td>
<td>axis variation table</td>
<td>hmtx</td>
<td>horizontal metrics table</td>
</tr>
<tr>
<td>bdat</td>
<td>bitmap data table</td>
<td>hsty</td>
<td>horizontal style table</td>
</tr>
<tr>
<td>bhed</td>
<td>bitmap font header table</td>
<td>just</td>
<td>justification table</td>
</tr>
<tr>
<td>bloc</td>
<td>bitmap location table</td>
<td>kern</td>
<td>kerning table</td>
</tr>
<tr>
<td>bsln</td>
<td>baseline table</td>
<td>lcar</td>
<td>ligature caret table</td>
</tr>
<tr>
<td>cmap</td>
<td>character code mapping table</td>
<td>loca</td>
<td>glyph location table</td>
</tr>
<tr>
<td>cvar</td>
<td>cvt variation table</td>
<td>maxp</td>
<td>maximum profile table</td>
</tr>
<tr>
<td>cvt</td>
<td>control value table</td>
<td>mort</td>
<td>metamorphosis table</td>
</tr>
<tr>
<td>EBSC</td>
<td>embedded bitmap scaling control table</td>
<td>morx</td>
<td>extended metamorphosis table</td>
</tr>
<tr>
<td>fdsc</td>
<td>font descriptor table</td>
<td>name</td>
<td>name table</td>
</tr>
<tr>
<td>feat</td>
<td>layout feature table</td>
<td>opbd</td>
<td>optical bounds table</td>
</tr>
<tr>
<td>fmtx</td>
<td>font metrics table</td>
<td>os/2</td>
<td>compatibility table</td>
</tr>
<tr>
<td>fpgm</td>
<td>font program table</td>
<td>post</td>
<td>glyph name PostScript</td>
</tr>
<tr>
<td>fvar</td>
<td>font variation table</td>
<td>prep</td>
<td>control value program table</td>
</tr>
<tr>
<td>gasp</td>
<td>gridfitting and scanconversion procedure table</td>
<td>prop</td>
<td>properties table</td>
</tr>
<tr>
<td>glyf</td>
<td>glyph outline table</td>
<td>trak</td>
<td>tracking table</td>
</tr>
<tr>
<td>gvar</td>
<td>glyph variation table</td>
<td>vhea</td>
<td>vertical header table</td>
</tr>
<tr>
<td>hdmx</td>
<td>horizontal device metrics table</td>
<td>vmtx</td>
<td>vertical metrics table</td>
</tr>
<tr>
<td>head</td>
<td>font header table</td>
<td>Zapf</td>
<td>glyph reference table</td>
</tr>
</tbody>
</table>

so a text processing application can adjust its behavior accordingly.

– OpenType Layout fonts have an open format that allows font developers to define their own typographical features.

4. The OpenType Layout model

4.1 Scripts

Scripts are defined at the top level. A script is a collection of glyphs used to represent one or more languages in writing. For instance, a single script—Latin—is used to write English, French, German, and many other languages. In contrast, three scripts—Hiragana, Katakana, and Kanji—are used to write Japanese. With OpenType Layout, multiple scripts may be supported by a single font.

4.2 Language system

A language system may modify the functions or appearance of glyphs in a script to represent a particular language. For example, the eszet ligature
is used in the German language system, but not in French or English. And the Arabic script contains different glyphs for writing the Farsi and Urdu languages. In the absence of language-specific rules, default language system features apply to the entire script.

Another example is the hani script which supports China, Korea and Japan. Here we have different glyphs for the same Unicode codepoint for different language systems as can be seen for example in the ms Arial Unicode font:

Script Tag: hani
Language Tag: zht, zhs, kor

traditional
simplified
Japanese
4.3 Features
A language system defines features, which are typographic rules for using glyphs to represent a language. The typographic features define the functionality of an OpenType Layout font and are registered in the OpenType Layout tag registry at the Microsoft Typography homepage. Font developers can use these features, as well as create their own (if they find an application which uses them!)

Some examples of typographic features are:

 – vert
This substitutes vertical glyphs in Japanese.

 – init, medi, fina
A language system feature for the Arabic script substitutes initial, medial, and final glyph forms based on a glyph’s position in a word.

  - Standalone ‘ha’
  - Initial ‘ha’
  - Medial ‘ha’
  - Final ‘ha’

 – liga
Feature for using ligatures in place of separate glyphs.

 – clig
Unlike other ligature features, clig specifies the context in which the ligature is recommended. This capability is important in some script designs and for swash ligatures. The clig table maps sequences of glyphs to corresponding ligatures in a chained context (gsub lookup type 8). For example: the ligature glyph ‘ft’ replaces the sequence f t, except when preceded by an ascending letter.

 – kern
The kern feature is an example of a gpos feature, i.e. it modifies the positioning of the glyphs. The kern feature is used to adjust the amount of space between glyphs, generally to provide optically consistent spacing between glyphs.
Other examples for gpos features: Urdu layout requires glyph positioning control, as well as contextual substitution.

Correct:

Incorrect:

4.4 Lookups
Features are implemented with lookup data that the text processing client uses to substitute and position glyphs. Lookups describe the glyphs affected by an operation, the type of operation to be applied to these glyphs, and the resulting glyph output.

4.5 gsub table
The gsub table contains substitution lookups that map GIDs to GIDs and associate these mappings with particular OpenType Layout features. The OpenType specification currently supports six different gsub lookup types:
1. Single
   Replaces one glyph with one glyph. (vert, salt, …).
2. Multiple
   Replaces one glyph with more than one glyph (ligature decomposition).
3. Alternate
   Replaces one glyph with one of many glyphs (crct).
4. Ligature
   Replaces multiple glyphs with one glyph (liga …).
5. Context
   Replaces one or more glyphs in context (cliq …).
6. Chaining context
   Replaces one or more glyphs in chained context (Swash alternates).

4.6 gpos table
The gpos table contains a powerful set of lookup types to reposition glyphs relative to their normative positions and to each other. Glyph positioning lookups work in two ways: by adjusting glyph positions relative to their metrical space or by linking predefined attachment points on different glyphs.

These two methods are further divided into specific adjustment and attachment lookup types that can be used to control positioning of diacritics relative to single or ligatured characters and even to enable chains of contextual positioning operations. The OpenType specification currently supports eight different gpos lookup types:
– A single adjustment positions one glyph, such as a superscript or subscript.
– A pair adjustment positions two glyphs with respect to one another; kerning is an example of pair adjustment.
– A cursive attachment describes cursive scripts and other glyphs that are connected with attachment points when rendered.
– A MarkToBase attachment positions combining marks with respect to base glyphs, as when positioning vowels, diacritical marks, or tone marks in Arabic, Hebrew and Vietnamese.
– A MarkToLigature attachment positions combining marks with respect to ligature glyphs. Because ligatures may have multiple points for attaching marks, the font developer needs to associate each mark with one of the ligature glyph’s components.
– A MarkToMark attachment positions one mark relative to another, as when positioning tone marks with respect to vowel diacritical marks in Vietnamese, for example.
– Contextual positioning describes how to position one or more glyphs in context.
– Chaining Contextual positioning describes how to position one or more glyphs in a chained context.

4.7 Processing of features and lookups

After choosing which features to use, the client assembles all lookups from the selected features. Multiple lookups may be needed to define the data required for different substitution and positioning actions, as well as to control the sequencing and effects of those actions. To implement features, a client applies the lookups in the order the lookup definitions occur in the LookupList. As a result, within the gsub or gpos table, lookups from several different features may be interleaved during text processing. A lookup is finished when the client locates a target glyph or glyph context and performs a substitution (if specified) or a positioning (if specified). The substitution (gsub) lookups always occur before the positioning (gpos) lookups. The lookup sequencing mechanism in TrueType relies on the font to determine the proper order of text-processing operations.

4.8 Ordering lookups (within the future tag)

The order of the lookup within the feature tag is critical. The lookup you define first will take priority. For example: if you have two ligatures TA + AE defined in your lookup table, with the AE listed first, and you type ‘TAE’, you would only get the AE ligature and not the TA, because the A is already converted into the AE ligature.
4.9 Ordering ligatures and conjuncts (within the lookup)
To ensure that ligatures and conjuncts are formed properly, one has to order substitutions so that the ones with higher priority precede others those with lower priority. It is also important to form the longer lookups before the shorter ones.

When forming ligatures, the lookups need to be encoded as follows:
– The first substitution in a lookup maps the longest string of component characters to the appropriate glyph; the next substitution provides the glyph corresponding to the next longest string of characters; and so forth. This is important because the search process through the lookups terminates with the first match.
– For consonant conjuncts, full form conjuncts must precede half forms.

For the fi & ffi ligatures, feature tag liga, if you order f + i → fi before f + f + i → ffi the ffi ligature would not be formed, because the search process stopped with the fi. When the ‘longer’ lookup is listed first, the ffi ligature is formed correctly.

Language dependency of features and lookups:
On the right is a (well-known) example for the language dependent glyph substitution. It shows a small part of the feature file which excludes the fi ligature for the Turkish language; in Turkish it is not allowed to form an fi ligature because the dotless i has a different meaning than the normal dotted i.
5. **OpenType production with DTL FontMaster**

As can be seen from the previous sections, OpenType is a rich specification which allows thousands of possible combinations of language lookups and features. It's quite obvious that writing a GUI for the OpenType tables is a huge task. The DTL FontMaster approach is trying to make it quite easy to generate an OpenType font.

- The Opentype production is based on Adobe’s SDK.
- Currently only the OTF production is supported (via Type 1 and CFF).
- DTL DataMaster automatically generates as many features as possible.
- Advanced users can create their own set of features.
- No fancy graphic user interface.

In DTL DataMaster the OTF production is essentially governed by two files:
- The Character Layout File, which is described in Appendix III.
- The OpenType Feature File.