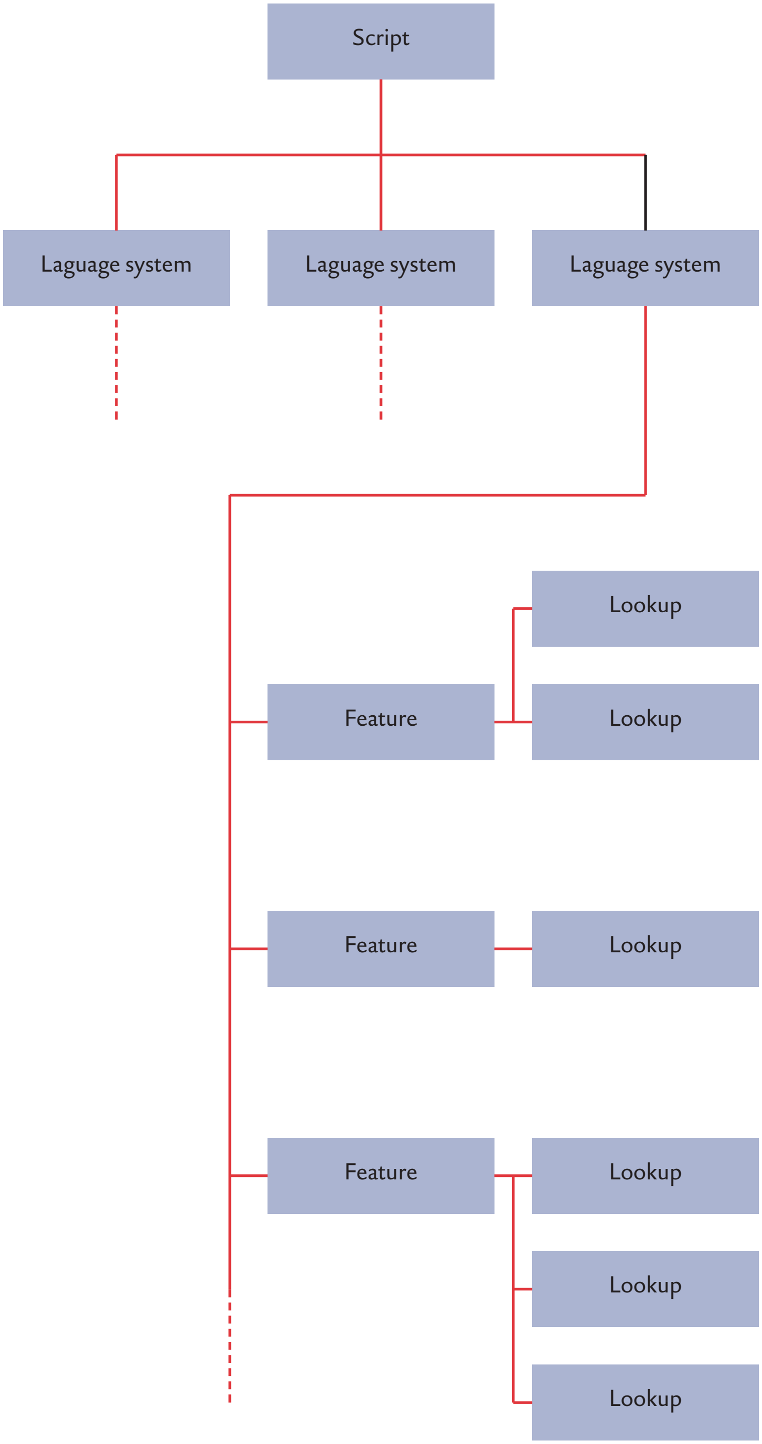
FontMaster™ Utilities

The OpenType Layout Model

OpenType layout data is organized by script, language system, typographic feature and lookup.



Contents

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 data base 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. – Implementation in Windows 3.1 in 1991.
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.

Increasing complexity

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 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 (GLYF table):
– Powerful instructions for suberb 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,

FontMaster™ Utilities

	TrueType (TTF)	Apples TTF (AAT/GX)	OpenType (TTF)	OpenType (OTF)	SFNT-CID (Adobe)
Required	head, hhea, hmtx name OS/2 maxp post cmap	head, hhea, hmtx name OS/2 maxp post cmap	head, hhea, hmtx name OS/2 maxp post cmap **DSIG**	head, hhea, hmtx name OS/2 maxp post cmap	cmap name post
Outline	glyf, loca cvt, fpgm, prep	glyf, loca cvt, fpgm, prep	glyf, loca cvt, fpgm, prep	CFF	CID
Optional	gasp hdmx kern LTSH PCLT VDMX vhea vmtx	gasp hdmx kern vhea vmtx	gasp hdmx kern LTSH PCLT VDMX vhea vmtx	gasp kern vhea vmtx VORG	
Bitmap	EBDT EBLC EBSC	bdat bloc	EBDT EBLC EBSC		bdat bloc
OTF			BASE (baseline data) GDEF (glyph definition) GPOS (glyph positioning) GSUB (glyph substitution) JSTF (Justification)	BASE (baseline data) GDEF (glyph definition) GPOS (glyph positioning) GSUB (glyph substitution) JSTF (Justification)	
AAT		mort, feat, bsln, prop opdb, trak, just … fvar, gvar, Zapf …			faet mort
Adobe					ALMX BBOX FNAM, HFMX, VFMX

The TrueType Font File (Apple's specification AAT)

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

so a textprocessing application can adjust its behavior accordingly.
– OpenType Layout fonts have an open format that allows font developers to define their own typographical features.

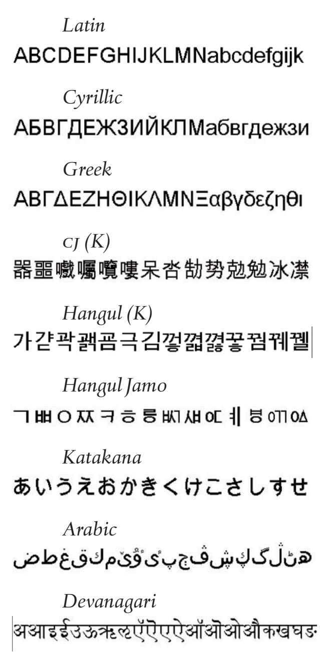
4. The OpenType Layout model

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



*OpenType fonts with CFF outlines and **AAT** support tables.*

```
****************************************
*********** Table Directory ************
****************************************
version:       20308.33
numTables:          22
searchRange:       256
entrySelector:       4
rangeShift:         96
```

tag	offset	length	checksum
BASE	364	456	6962C672
CFF	820	6720412	D234DEBC
DSIG	10240852	5788	EADEC4BC
EBDT	6721232	1636487	32BDCD3
EBLC	8357720	67148	883E371E
GPOS	8424868	14600	DD21703D
GSUB	8439468	185706	7F930AE3
OS/ 2	8625176	96	3814B65D
VORG	8625272	812	2BE8ACA
Zapf	**8626084**	**442236**	**2736C019**
cmap	9068320	276664	E31BA3BF
feat	**9344984**	**340**	**81CD4A53**
head	9345324	54	D3061EC9
hhea	9345380	36	8B5416B
hmtx	9345416	72546	D255AEAD
maxp	9417964	6	4F485000
morx	**9417972**	**739840**	**496DB24**
name	10157812	5060	3F369656
post	10162872	32	FFB80032
prop	**10162904**	**3758**	**DA5761FF**
vhea	10166664	36	74F5311
vmtx	10166700	74152	8EFBA4CC

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

辯 *Chinese traditional*

辯 *Chinese simplified*

辯 *Japanese*

FontMaster™ Utilities

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 Japanes.

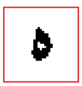
(漢字のテスト。)

(漢字のテスト。)

The substitution of vertical glyphs in Japanese (MS Mincho).

– **init, medi, fina**

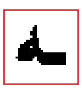
A language system feature for the Arabic script substitutes initial, medial, and final glyph forms based on a glyph's position in a word.



Standalone 'ha'

Initial 'ha'

Medial 'ha'



Final 'ha'

– **liga**

Feature for using ligatures in place of separate glyphs.

Ligature in backing store (left) and ***liga*** *form (right):*

f+i fi

– **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). Fpr example: the ligature glyph 'ft' replaces the sequence f t, except when preceded by an ascending letter.

Ligature in backing store (top) and ***clig*** *form (bottom):*

a+f+t

aft

– **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.

– Vertical.
– Horizontal.
– Size dependent kerning (via device tables).
– cross-stream kerning in the Y text direction.
– adjustment of glyph placement independent of the advance adjustment.
– adjustments for pairs of glyphs (GPOS lookup type 2 or 8).
– Support for left and right classes, and/or as individual pairs.

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(*crcy*).
4. *Ligature*
   Replaces multiple glyphs with one glyph (*liga …*).
5. *Context*
   Replaces one or more glyphs in context (*clig …*).
6. *Chaining context*
   Replaces one or more glyphs in chained context (*Swash alternates*).

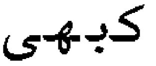
4.6 *GPOS table*

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:

Other examples for GPOS features: Urdu layout requires glyph positioning control, as well as contextual substitution.

Correct: 

Incorrect: 

– 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.

Contextual positioning lowered the accent over a vowel glyph that followed an overhanging uppercase glyph.

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.

```
feature liga {
    sub f f i by ffi;
    sub f i by fi;
  lookup NOFI {
    sub f f l by ffl;
    sub f f by ff;
    sub f l by fl;
    sub f f j by f_ f_ j;
    sub f j by f_ j;
  } NOFI;
  language TUR excludeDFLT;
  lookup NOFI;
} liga;
```

A small part of the feature file which excludes the fi ligature for the Turkish language.

	Feature	Feature function	Layout operation	Required
Language based forms	ccmp	Character composition/ decomposition substitution	GSUB	
Typographical forms	liga	Standard ligature substitution	GSUB	
	clig	Contextual ligature substitution	GSUB	
Positioning features	kern	Pair kerning	GPOS	
	mark	Mark to base positioning	GPOS	x
	mkmk	Mark to mark positioning	GPOS	x

Features for standard scripts (Windows Uniscribe/OTLS). More features are supported by InDesign and other Adobe applications.

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. Its 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.