

FASTER HORSE

C++ Font Subsetter: Fast, Flexible, and Secure

Behdad Esfahbod | behdad@google.com | TYPO Labs 2018

goo.gl/xVNcSH | Whitepaper goo.gl/Qy3Eqc | Video goo.gl/XbLa4X

What is Font Subsetting?

- Producing new font file from an input font,
- that behaves like the input, but with reduced functionality,
- for the purpose of reducing font file size *or* for compatibility,
- based on a subsetting specification.



...new font file from an input font...

- Input TTF / OTF
- Output TTF / OTF

Not:

- WOFF / WOFF2 / EOT / ...
- UFO / Glyphs / SFD / ...



...behaves like the input...

- Correct.
- Functionally equivalent.



...reducing font file size...

- Reduced glyph set
- Reduced discretionary layout features
- Reduced variation space
- Drop hinting, glyph names, bitmaps, legacy tables, ...



...or for compatibility...

- Generating PDF, XPS, ...
- Older software that does not understand certain features like variations



...based on a spec'n...

- Character set
- Layout feature set
- Name table entries
- Variation sub-space
- Other options (hinting, ...)



Why Subsetting?

- Web-font delivery
- Production of limited fonts
- Optimizing for specific system
- Instantiating a variable font
- Font streaming
- Printing



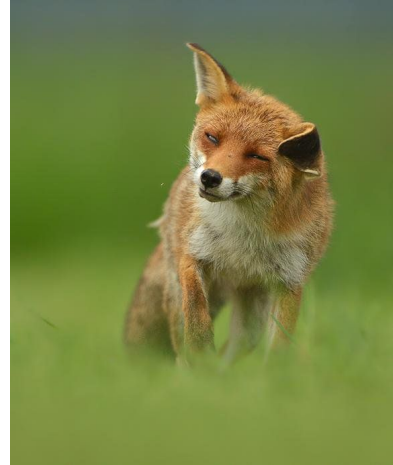
fonttools subset, aka pyftsubset

- Started in 2013
- 3.3k Python lines of code
- Powering Google Fonts (Jython then CPython)
- Major speed-ups implemented (lazy loading)
- TTF optimized more than OTF
- Fine for small fonts; Slooooow for huge fonts (Noto Sans CJK)
- Hit the limits of what's easily doable in Python



```
@_add_method(ttLib.getTableClass('glyf'))  
def subset_glyphs(self, s):  
    self.glyphs = {g:self[g] for g in s.glyphs}  
    return True
```





```
@_add_method(ttLib.getTableClass('glyf'))
def subset_glyphs(self, s):
    self.glyphs = {g:self.glyphs[g] for g in s.glyphs}
    indices = [i for i,g in enumerate(self.glyphOrder) if g in s.glyphs]
    for v in self.glyphs.values():
        if hasattr(v, "data"):
            v.remapComponentsFast(indices)
        else:
            pass # No need
    self.glyphOrder = [g for g in self.glyphOrder if g in s.glyphs]
    # Don't drop empty 'glyf' tables, otherwise 'loca' doesn't get subset.
    return True
```

pyftsubset performance: TTF

```
behdad:fonttools$ \time ./fonttools subset Roboto-Regular.ttf --gids=1-1  
0.24user 0.09system 0:00.34elapsed 98%CPU (0avgtext+0avgdata 142764maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset Roboto-Regular.ttf --gids=1-10  
0.20user 0.07system 0:00.27elapsed 98%CPU (0avgtext+0avgdata 142668maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset Roboto-Regular.ttf --gids=1-100  
0.27user 0.05system 0:00.33elapsed 99%CPU (0avgtext+0avgdata 143316maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset Roboto-Regular.ttf --gids=1-1000  
0.46user 0.07system 0:00.54elapsed 99%CPU (0avgtext+0avgdata 148860maxresident)k
```



pyftsubset performance: GSUB

```
behdad:fonttools$ \time ./fonttools subset IranNastaliq2.ttf --gids=1-1  
0.36user 0.07system 0:00.46elapsed 96%CPU (0avgtext+0avgdata 145900maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset IranNastaliq2.ttf --gids=1-10  
0.39user 0.06system 0:00.46elapsed 99%CPU (0avgtext+0avgdata 145860maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset IranNastaliq2.ttf --gids=1-100  
0.40user 0.07system 0:00.48elapsed 99%CPU (0avgtext+0avgdata 146064maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset IranNastaliq2.ttf --gids=1-1000  
1.56user 0.09system 0:01.65elapsed 99%CPU (0avgtext+0avgdata 169064maxresident)k
```



pyftsubset performance: GSUB

```
behdad:fonttools$ \time ./fonttools subset NotoNastaliqUrdu-Regular.ttf --gids=1-1  
0.47user 0.06system 0:00.54elapsed 99%CPU (0avgtext+0avgdata 145816maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoNastaliqUrdu-Regular.ttf --gids=1-10  
0.44user 0.09system 0:00.54elapsed 99%CPU (0avgtext+0avgdata 146160maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoNastaliqUrdu-Regular.ttf --gids=1-100  
0.55user 0.06system 0:00.62elapsed 99%CPU (0avgtext+0avgdata 146440maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoNastaliqUrdu-Regular.ttf --gids=1-1000  
4.15user 0.10system 0:04.26elapsed 99%CPU (0avgtext+0avgdata 190884maxresident)k
```



pyftsubset performance: OTF

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.otf --gids=1-1  
0.68user 0.14system 0:00.83elapsed 99%CPU (0avgtext+0avgdata 168448maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.otf --gids=1-10  
0.69user 0.08system 0:00.78elapsed 99%CPU (0avgtext+0avgdata 168392maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.otf --gids=1-100  
0.75user 0.10system 0:00.86elapsed 99%CPU (0avgtext+0avgdata 170636maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.otf --gids=1-1000  
2.16user 0.11system 0:02.27elapsed 99%CPU (0avgtext+0avgdata 190100maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.otf --gids=1-10000  
11.44user 0.16system 0:11.61elapsed 99%CPU (0avgtext+0avgdata 254288maxresident)k
```



pyftsubset performance: TTF

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.ttf --gids=1-1  
1.06user 0.11system 0:01.18elapsed 99%CPU (0avgtext+0avgdata 162564maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.ttf --gids=1-10  
1.07user 0.11system 0:01.18elapsed 99%CPU (0avgtext+0avgdata 163516maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.ttf --gids=1-100  
1.13user 0.10system 0:01.23elapsed 99%CPU (0avgtext+0avgdata 166144maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.ttf --gids=1-1000  
1.54user 0.12system 0:01.68elapsed 99%CPU (0avgtext+0avgdata 182260maxresident)k
```

```
behdad:fonttools$ \time ./fonttools subset NotoSansCJK-Regular.ttf --gids=1-10000  
2.05user 0.15system 0:02.20elapsed 99%CPU (0avgtext+0avgdata 200544maxresident)k
```



Python performance

```
$ ipython3
```

```
In [1]: %timeit object()
```

```
10000000 loops, best of 3: 84 ns per loop
```

```
In [2]: from types import SimpleNamespace
```

```
In [3]: %timeit SimpleNamespace()
```

```
10000000 loops, best of 3: 158 ns per loop
```



Python performance

```
$ ipython3
```



```
In [1]: from fontTools.ttLib import TTFont
```

```
In [2]: font = TTFont('NotoSansCJK-Regular.otf')
```

```
In [3]: %timeit -n1 -r1 font['cmap']
```

```
1 loop, best of 1: 113 ms per loop
```



Criteria

- Fast
- Flexible
- Secure



Plan

- Rewrite in C++
- Ship it!

Design document:

goo.gl/Oy3Eqc



Performance Characteristics

- Initialization:
 - $O(1) + O(N)$ if post-table glyph names needed
- Subsetting:
 - $O(n.\log(N).C) + O(b)$



Alternatives

- sftnly
- ots
- otfcc



Design

- pyftsubset for user interface
- HarfBuzz for implementation



HarfBuzz

- Text shaper
- C++ library
- Zero-cost loading
- Encapsulation
- Memory-safety
- Operator overloading
- Strong type system
- Simple templates




```

if ( fvar_head.instanceCount && !face->blend->avar_loaded )
    ft_var_load_avar( face );

ns = mmvar->namedstyle;
nsc = face->blend->normalized_stylecoords;
for ( i = 0; i < fvar_head.instanceCount; i++, ns++ )
{
    /* PostScript names add 2 bytes to the instance record size
    if ( FT_FRAME_ENTER( ( usePsName ? 6L : 4L ) +
                        4L * fvar_head.axisCount ) )

        goto Exit;

    ns->strid      = FT_GET_USHORT();
    (void) /* flags = */ FT_GET_USHORT();

    c = ns->coords;
    for ( j = 0; j < fvar_head.axisCount; j++, c++ )
        *c = FT_GET_LONG();

    if ( usePsName )
        ns->psid = FT_GET_USHORT();

    ft_var_to_normalized( face,
                        fvar_head.axisCount,
                        ns->coords,
                        nsc );
    nsc += fvar_head.axisCount;
}

```



```

if ( fvar_head.instanceCount && !face->blend->avar_loaded )
{
    FT_ULong  offset = FT_STREAM_POS();

    ft_var_load_avar( face );

    if ( FT_STREAM_SEEK( offset ) )
        goto Exit;
}

FT_TRACE5(( "%d instance%s\n",
            fvar_head.instanceCount,
            fvar_head.instanceCount == 1 ? "" : "s" ));

ns = mmvar->namedstyle;
nsc = face->blend->normalized_stylecoords;
for ( i = 0; i < fvar_head.instanceCount; i++, ns++ )
{
    /* PostScript names add 2 bytes to the instance record size
    if ( FT_FRAME_ENTER( ( usePsName ? 6L : 4L ) +
                        4L * num_axes ) )

        goto Exit;

    ns->strid      = FT_GET_USHORT();
    (void) /* flags = */ FT_GET_USHORT();

```



```
class EncodingConverter(SimpleConverter):

    def read(self, parent, value):
        if value == 0:
            return "StandardEncoding"
        elif value == 1:
            return "ExpertEncoding"
        else:
            assert value > 1
            file = parent.file
            file.seek(value)
            log.log(DEBUG, "loading Encoding at %s", value)
            fmt = readCard8(file)
            haveSupplement = fmt & 0x80
            if haveSupplement:
                raise NotImplementedError("Encoding supplements are not yet supported")
            fmt = fmt & 0x7f
            if fmt == 0:
                encoding = parseEncoding0(parent.charset, file, haveSupplement,
                                           parent.strings)
            elif fmt == 1:
                encoding = parseEncoding1(parent.charset, file, haveSupplement,
                                           parent.strings)
            return encoding
```



```

def decompile(self, data, offset):
    # initial offset is from the start of trak table to the current TrackData
    trackDataHeader = data[offset:offset+TRACK_DATA_FORMAT_SIZE]
    if len(trackDataHeader) != TRACK_DATA_FORMAT_SIZE:
        raise TTLibError('not enough data to decompile TrackData header')
    sstruct.unpack(TRACK_DATA_FORMAT, trackDataHeader, self)
    offset += TRACK_DATA_FORMAT_SIZE

    nSizes = self.nSizes
    sizeTableOffset = self.sizeTableOffset
    sizeTable = []
    for i in range(nSizes):
        sizeValueData = data[:SIZE_VALUE_FORMAT_SIZE]
        if len(sizeValueData) < SIZE_VALUE_FORMAT_SIZE:
            raise TTLibError('not enough data to decompile TrackData size subtable')
        sizeValue, = struct.unpack(SIZE_VALUE_FORMAT, sizeValueData)
        sizeTable.append(fi2fl(sizeValue, 16))
        sizeTableOffset += SIZE_VALUE_FORMAT_SIZE

    for i in range(self.nTracks):
        entry = TrackTableEntry()
        entryData = data[:TRACK_TABLE_ENTRY_FORMAT_SIZE]
        if len(entryData) < TRACK_TABLE_ENTRY_FORMAT_SIZE:
            raise TTLibError('not enough data to decompile TrackTableEntry record')
        sstruct.unpack(TRACK_TABLE_ENTRY_FORMAT, entryData, entry)
        perSizeOffset = entry.offset
        for j in range(nSizes):
            size = sizeTable[j]
            perSizeValueData = data[:PER_SIZE_VALUE_FORMAT_SIZE]
            if len(perSizeValueData) < PER_SIZE_VALUE_FORMAT_SIZE:
                raise TTLibError('not enough data to decompile per-size track values')
            perSizeValue, = struct.unpack(PER_SIZE_VALUE_FORMAT, perSizeValueData)
            entry[size] = perSizeValue
            perSizeOffset += PER_SIZE_VALUE_FORMAT_SIZE
        self[entry.track] = entry
        offset += TRACK_TABLE_ENTRY_FORMAT_SIZE

```



```

def decompile(self, data, offset):
    # initial offset is from the start of trak table to the current TrackData
    trackDataHeader = data[offset:offset+TRACK_DATA_FORMAT_SIZE]
    if len(trackDataHeader) != TRACK_DATA_FORMAT_SIZE:
        raise TTLibError('not enough data to decompile TrackData header')
    sstruct.unpack(TRACK_DATA_FORMAT, trackDataHeader, self)
    offset += TRACK_DATA_FORMAT_SIZE

    nSizes = self.nSizes
    sizeTableOffset = self.sizeTableOffset
    sizeTable = []
    for i in range(nSizes):
        sizeValueData = data[sizeTableOffset:sizeTableOffset+SIZE_VALUE_FORMAT_SIZE]
        if len(sizeValueData) < SIZE_VALUE_FORMAT_SIZE:
            raise TTLibError('not enough data to decompile TrackData size subtable')
        sizeValue, = struct.unpack(SIZE_VALUE_FORMAT, sizeValueData)
        sizeTable.append(fi2fl(sizeValue, 16))
        sizeTableOffset += SIZE_VALUE_FORMAT_SIZE

    for i in range(self.nTracks):
        entry = TrackTableEntry()
        entryData = data[offset:offset+TRACK_TABLE_ENTRY_FORMAT_SIZE]
        if len(entryData) < TRACK_TABLE_ENTRY_FORMAT_SIZE:
            raise TTLibError('not enough data to decompile TrackTableEntry record')
        sstruct.unpack(TRACK_TABLE_ENTRY_FORMAT, entryData, entry)
        perSizeOffset = entry.offset
        for j in range(nSizes):
            size = sizeTable[j]
            perSizeValueData = data[perSizeOffset:perSizeOffset+PER_SIZE_VALUE_FORMAT_SIZE]
            if len(perSizeValueData) < PER_SIZE_VALUE_FORMAT_SIZE:
                raise TTLibError('not enough data to decompile per-size track values')
            perSizeValue, = struct.unpack(PER_SIZE_VALUE_FORMAT, perSizeValueData)
            entry[size] = perSizeValue
            perSizeOffset += PER_SIZE_VALUE_FORMAT_SIZE
        self[entry.track] = entry
        offset += TRACK_TABLE_ENTRY_FORMAT_SIZE

```



```

struct TrackData
{
    inline bool sanitize (hb_sanitize_context_t *c, const void *base) const
    {
        TRACE_SANITIZE (this);
        return_trace (c->check_struct (this) &&
            sizeTable.sanitize (c, base, nSizes) &&
            trackTable.sanitize (c, nTracks, base, nSizes));
    }
}

```



protected:

```

HBUINT16          nTracks;          /* Number of separate tracks included in this table. */
HBUINT16          nSizes;           /* Number of point sizes included in this table. */
LOffsetTo<UnsizeArrayOf<Fixed> >    /* Offset to array[nSizes] of size values. */
    sizeTable;
UnsizeArrayOf<TrackTableEntry>
    trackTable;                      /* Array[nTracks] of TrackTableEntry records. */

```

public:

```

DEFINE_SIZE_ARRAY (8, trackTable);
};

```


pyftsubset structure

```
font = TTFont(...)
```

```
for table in font:  
    table.prune_pre_subset(s)
```

```
for table in font:  
    table.subset_glyphs(s)
```

```
for table in font:  
    table.prune_post_subset(s)
```

```
font.save(...)
```



Status

- TrueType implemented
- GSUB/GPOS/GDEF WIP
- CFF/CFF2 later
- Variable fonts laaater



hb-subset performance

```
behdad:util$ \time ./hb-subset Roboto-Regular.ttf -o out.ttf a  
0.00user 0.00system 0:00.01elapsed 25%CPU
```

```
behdad:util$ \time ./hb-subset Roboto-Regular.ttf -o out.ttf \  
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789  
0.00user 0.01system 0:00.02elapsed 54%CPU
```



Fast

1000x



Flexible

Modern,
healthy,
codebase.



Secure

oss-fuzz



Cutesies



Foxes courtesy of
twitter.com/hourlyfox

