ufo2fdk Documentation

Release 0.1

Tal Leming

Mar 20, 2019

Contents

ufo2fdk is a package that handles the creation of OpenType-CFF fonts from UFO source files with the aid of the Adobe Font Development Kit for OpenType (AFDKO aka FDK). This package does not embedd the FDK, rather it `bridges` to it on the user's system with the expectation that the user has installed the FDK in the normal way.

The fonts created by the package should be considered only for beta usage and should be used with care.

CHAPTER 1

Basic Usage

Two main things are exposed for public use: *haveFDK* and *OTFCompiler*.

1.1 haveFDK

This function tests the availability of the FDK and returns a boolean indicating what was found. Example:

```
from ufo2fdk import haveFDK

if haveFDK():
  print "I found the FDK!"
else:
  print "I'm sorry, I could not find the FDK."
```

1.2 OTFCompiler

Example:

```
from ufo2FDK import OTFCompiler

compiler = OTFCompiler()
reports = compiler.compile(font, "/MyDirectory/MyFont.otf", checkOutlines=True,␣
↪autohint=True)
print reports["checkOutlines"]
print reports["autohint"]
print reports["makeotf"]
```

That's all there is to it.

1.3 Naming Data

As with any OpenType compiler, you have to set the font naming data to a particular standard for your naming to be set correctly. In ufo2fdk, you can get away with setting *two* naming attributes in your font.info object for simple fonts:

- familyName: The name for your family. For example, "My Garamond".
- styleName: The style name for this particular font. For example, "Display Light Italic"

ufo2fdk will create all of the other naming data based on thse two fields. If you want to use the fully automatic naming system, all of the other name attributes should be set to `None` in your font. However, if you want to override the automated system at any level, you can specify particular naming attributes and ufo2fdk will honor your settings. You don't have to set *all* of the attributes, just the ones you don't want to be automated. For example, in the family "My Garamond" you have eight weights. It would be nice to style map the italics to the romans for each weight. To do this, in the individual romans and italics, you need to set the style mapping data. This is done through the `styleMapFamilyName` and `styleMapStyleName` attributes. In each of your roman and italic pairs you would do this:

My Garamond-Light.ufo

- familyName = "My Garamond"
- styleName = "Light"
- styleMapFamilyName = "My Garamond Display Light"
- styleMapStyleName = "regular"

My Garamond-Light Italic.ufo

- familyName = "My Garamond"
- styleName = "Display Light Italic"
- styleMapFamilyName = "My Garamond Display Light"
- styleMapStyleName = "italic"

My Garamond-Book.ufo

- familyName = "My Garamond"
- styleName = "Book"
- styleMapFamilyName = "My Garamond Display Book"
- styleMapStyleName = "regular"

My Garamond-Book Italic.ufo

- familyName = "My Garamond"
- styleName = "Display Book Italic"
- styleMapFamilyName = "My Garamond Display Book"
- styleMapStyleName = "italic"

etc.

The full details of how the names are created is available in the `fontInfoData documentation`. The usage of the names is detailed in the `makeotfParts documentation`.

Additionally, if you have defined any naming data, or any data for that matter, in table definitions within your font's features that data will be honored.

1.4 Kerning Data

If your font's features, and any files included in your font's features, do not contain a kerning feature, ufo2fdk will create one based on your font's kerning data. Do to the complexities in how raw kerning data is translated into a kerning feature, it is safest to use your preferred kerning editor to create the kerning feature if possible.

1.5 Internals

If you want to know how the bridging works, what information is needed in the source UFO, how to modify the internal behavior to fit your needs, etc. read on:

1.5.1 fdkBridge

1.5.2 makeotfParts

1.5.3 outlineOTF

1.5.4 kernFeatureWriter

1.5.5 fontInfoData

Main Functions

Static Fallbacks

Most of the fallbacks have static values. To see what is set for these, look at the source code's *staticFallbackData* definition.

Special Fallbacks

In some cases, the fallback values are dynamically generated from other data in the info object. These are handled internally with functions.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

u

Index

U