```
# coding: utf-8
"""
    weasyprint.fonts
    ----------------

    Interface with external libraries managing fonts installed on the syst

    :copyright: Copyright 2011-2016 Simon Sapin and contributors, see AUTH
    :license: BSD, see LICENSE for details.

"""

from __future__ import division

import os
import sys
import tempfile
import warnings

from .compat import FILESYSTEM_ENCODING
from .logger import LOGGER
from .text import (
    cairo, dlopen, ffi, get_font_features, gobject, pango,
pangocairo)
from .urls import fetch

# XXX No unicode_literals, cffi likes native strings


class FontConfiguration:
    """Font configuration"""

    def __init__(self):

"""Create a font configuration before rendering a document."""
        self.font_map = None

    def add_font_face(self, rule_descriptors, url_fetcher):
        """Add a font into the application."""

    def clean(self):

"""Clean a font configuration after rendering a document."""


if sys.platform.startswith('win'):
    warnings.warn(
```

```
'@font-face is currently not supported on Windows')
elif pango.pango_version() < 13800:
    warnings.warn('@font-face support needs Pango >= 1.38')
else:
    ffi.cdef('''
        // FontConfig

        typedef int FcBool;
        typedef struct _FcConfig FcConfig;
        typedef struct _FcPattern FcPattern;
        typedef unsigned char FcChar8;

        typedef enum {
            FcResultMatch, FcResultNoMatch, FcResultTypeMismatch, FcResult
            FcResultOutOfMemory
        } FcResult;

        typedef enum {
            FcMatchPattern, FcMatchFont, FcMatchScan
        } FcMatchKind;

        FcConfig * FcInitLoadConfigAndFonts (void);
        FcPattern * FcConfigDestroy (FcConfig *config);
        FcBool FcConfigAppFontAddFile (
            FcConfig *config, const FcChar8 *file);
        FcConfig * FcConfigGetCurrent (void);
        FcBool FcConfigSetCurrent (FcConfig *config);
        FcBool FcConfigParseAndLoad (
            FcConfig *config, const FcChar8 *file, FcBool complain);

        void FcDefaultSubstitute (FcPattern *pattern);
        FcBool FcConfigSubstitute (
            FcConfig *config, FcPattern *p, FcMatchKind kind);

        FcPattern * FcPatternCreate (void);
        FcPattern * FcPatternDestroy (FcPattern *p);
        FcBool FcPatternAddString (
            FcPattern *p, const char *object, const FcChar8 *s);
        FcResult FcPatternGetString (
            FcPattern *p, const char *object, int n, FcChar8 **s);
        FcPattern * FcFontMatch (
            FcConfig *config, FcPattern *p, FcResult *result);


        // PangoFT2

        typedef ... PangoFcFontMap;
```

```
        void pango_fc_font_map_set_config (
            PangoFcFontMap *fcfontmap, FcConfig *fcconfig);
        void pango_fc_font_map_shutdown (PangoFcFontMap *fcfontmap);


        // PangoCairo

        typedef ... PangoCairoFontMap;

        void pango_cairo_font_map_set_default (PangoCairoFontMap *fontmap);
        PangoFontMap * pango_cairo_font_map_new_for_font_type (
            cairo_font_type_t fonttype);
    ''')

    fontconfig = dlopen(ffi, 'fontconfig', 'libfontconfig',
                        'libfontconfig.so.1',
'libfontconfig-1.dylib')
    pangoft2 = dlopen(ffi, 'pangoft2-1.0', 'libpangoft2-1.0-0',
                      'libpangoft2-1.0.so',
'libpangoft2-1.0.dylib')

    FONTCONFIG_WEIGHT_CONSTANTS = {
        'normal': 'normal',
        'bold': 'bold',
        100: 'thin',
        200: 'extralight',
        300: 'light',
        400: 'normal',
        500: 'medium',
        600: 'demibold',
        700: 'bold',
        800: 'extrabold',
        900: 'black',
    }

    FONTCONFIG_STYLE_CONSTANTS = {
        'normal': 'roman',
        'italic': 'italic',
        'oblique': 'oblique',
    }

    FONTCONFIG_STRETCH_CONSTANTS = {
        'normal': 'normal',
        'ultra-condensed': 'ultracondensed',
        'extra-condensed': 'extracondensed',
        'condensed': 'condensed',
        'semi-condensed': 'semicondensed',
        'semi-expanded': 'semiexpanded',
```

```
        'expanded': 'expanded',
        'extra-expanded': 'extraexpanded',
        'ultra-expanded': 'ultraexpanded',
    }

    class FontConfiguration(FontConfiguration):
        def __init__(self):
            """Create a FT2 font configuration.

            See Behdad's blog:
            https://mces.blogspot.fr/2015/05/
                    how-to-use-custom-application-fonts.html

            """
            self._fontconfig_config = ffi.gc(
                fontconfig.FcInitLoadConfigAndFonts(),
                fontconfig.FcConfigDestroy)
            self.font_map = ffi.gc(
                pangocairo.pango_cairo_font_map_new_for_font_type
(
                    cairo.FONT_TYPE_FT),
                gobject.g_object_unref)
            pangoft2.pango_fc_font_map_set_config(
                ffi.cast('PangoFcFontMap *', self.font_map),
                self._fontconfig_config)

# pango_fc_font_map_set_config keeps a reference to config
            fontconfig.FcConfigDestroy(self._fontconfig_config)
            self._filenames = []

        def add_font_face(self, rule_descriptors, url_fetcher):
            for font_type, url in rule_descriptors['src']:
                if font_type in ('external', 'local'):
                    config = self._fontconfig_config
                    if font_type == 'local':
                        font_name = url.encode('utf-8')
                        pattern = ffi.gc(
                            fontconfig.FcPatternCreate(),
                            fontconfig.FcPatternDestroy)
                        fontconfig.FcConfigSubstitute(
                            config, pattern, fontconfig.
FcMatchFont)
                        fontconfig.FcDefaultSubstitute(pattern)
                        fontconfig.FcPatternAddString(
                            pattern, b'fullname', font_name)
                        fontconfig.FcPatternAddString(
                            pattern, b'postscriptname', font_name
)
```

```
                        family = ffi.new('FcChar8 **')
                        postscript = ffi.new('FcChar8 **')
                        result = ffi.new('FcResult *')
                        matching_pattern = fontconfig.FcFontMatch
(
                            config, pattern, result)

# TODO: do many fonts have multiple family values?
                        fontconfig.FcPatternGetString(
                            matching_pattern, b'fullname', 0,
family)
                        fontconfig.FcPatternGetString(
                            matching_pattern, b'postscriptname',
0, postscript)
                        family = ffi.string(family[0])
                        postscript = ffi.string(postscript[0])
                        if font_name.lower() in (
                                family.lower(), postscript.lower
()):
                            filename = ffi.new('FcChar8 **')
                            matching_pattern = fontconfig.
FcFontMatch(
                                config, pattern, result)
                            fontconfig.FcPatternGetString(
                                matching_pattern, b'file', 0,
filename)
                            url = (
                                u'file://' +
                                ffi.string(filename[0]).decode(
'utf-8'))
                        else:
                            LOGGER.warning(
                                'Failed to load local font "%s"',
                                font_name.decode('utf-8'))
                            continue
                    try:
                        with fetch(url_fetcher, url) as result:
                            if 'string' in result:
                                font = result['string']
                            else:
                                font = result['file_obj'].read()
                    except Exception as exc:
                        LOGGER.warning(
                            'Failed to load font at "%s" (%s)',
url, exc)
                        continue
                    font_features = {
                        rules[0][0].replace('-', '_'): rules[0][1
```

```
] for rules in
                        rule_descriptors.get('font_variant', [])}
                    if 'font_feature_settings' in
rule_descriptors:
                        font_features['font_feature_settings'] =
(
                            rule_descriptors[
'font_feature_settings'])
                    features_string = ''
                    for key, value in get_font_features(
                            **font_features).items():
                        features_string += '<string>%s %s
</string>' % (
                            key, value)
                    fd, filename = tempfile.mkstemp()
                    os.write(fd, font)
                    os.close(fd)
                    self._filenames.append(filename)
                    xml = '''<?xml version="1.0"?>
                    <!DOCTYPE fontconfig SYSTEM "fonts.dtd">
                    <fontconfig>
                      <match target="scan">
                        <test name="file" compare="eq">
                          <string>%s</string>
                        </test>
                        <edit name="family" mode="assign_replace">
                          <string>%s</string>
                        </edit>
                        <edit name="slant" mode="assign_replace">
                          <const>%s</const>
                        </edit>
                        <edit name="weight" mode="assign_replace">
                          <const>%s</const>
                        </edit>
                        <edit name="width" mode="assign_replace">
                          <const>%s</const>
                        </edit>
                      </match>
                      <match target="font">
                        <test name="file" compare="eq">
                          <string>%s</string>
                        </test>
                        <edit name="fontfeatures"
                              mode="assign_replace">%s</edit>
                      </match>
                    </fontconfig>''' % (
                        filename,
                        rule_descriptors['font_family'],
```

```
                            FONTCONFIG_STYLE_CONSTANTS[
                                rule_descriptors.get('font_style',
'normal')],
                            FONTCONFIG_WEIGHT_CONSTANTS[
                                rule_descriptors.get('font_weight',
'normal')],
                            FONTCONFIG_STRETCH_CONSTANTS[
                                rule_descriptors.get('font_stretch',
'normal')],
                            filename, features_string)
                        fd, conf_filename = tempfile.mkstemp()

# TODO: coding is OK for <test> but what about <edit>?
                        os.write(fd, xml.encode(FILESYSTEM_ENCODING))
                        os.close(fd)
                        self._filenames.append(conf_filename)
                        fontconfig.FcConfigParseAndLoad(
                            config, conf_filename.encode(
FILESYSTEM_ENCODING),
                            True)
                        font_added = fontconfig.
FcConfigAppFontAddFile(
                            config, filename.encode(
FILESYSTEM_ENCODING))
                        if font_added:

# TODO: we should mask local fonts with the same name
                            # too as explained in Behdad's blog entry
                            return filename
                        else:
                            LOGGER.warning('Failed to load font at "
%s"', url)
                LOGGER.warning(
                    'Font-face "%s" cannot be loaded',
                    rule_descriptors['font_family'])

            def clean(self):
                """Clean a font configuration for a document."""
                for filename in self._filenames:
                    os.remove(filename)
```