WPF font selection model

Font selection model used by Windows Presentation Foundation: description and guidelines.

Mikhail V. Leonov
mikhail.leonov@microsoft.com
David C. Brown
david.c.brown@microsoft.com

Contents

1. Background

Applications specify fonts they intend to use via a set of attributes that include font family names and typographic properties such as font weight, width and slope. For example, an application may ask for a bold version of "Arial" font. The appearance of glyphs from the requested font depends on multiple factors, such as application itself, operating system and font files installed on the system. The goal of this document is to describe the font selection model used by Windows Presentation Foundation (referred as "WPF" in this document) and to provide guidelines for application developers and font designers to create applications and fonts that work best with WPF.

2. Font matching process

WPF recognizes a *typeface* by its font family, weight, width, and slope properties, which are expressed in the WPF API and XAML markup language as *FontFamily, FontWeight, FontStretch,* and *FontStyle* respectively. WPF groups fonts that have the same name but vary in the above font properties into a logical group of fonts called *font family*. The concept of a font family is valid in the scope of a *font collection*, which is an abstract notion of a font folder. For example, a font collection can denote all fonts installed on a system, the collection of fonts supplied by an application, or the collection of fonts from a local or a network folder.

WPF font matching process performs mapping in the context of a font collection from the logical font description, such as…

{ FontFamily="Arial", FontStyle="Italic", FontStretch="Condensed", FontWeight="Normal" }

to the physical font description, such as…

{ "H:\Windows\Fonts\arialni.ttf", no algorithmic emboldening, no algorithmic slant }.

In addition, WPF computes a *face name* property that can be used to identify a particular combination of FontStyle, FontWeight and FontStretch properties within a given font family. Face name is exposed via Typeface.FaceNames property.

It is worth noting that WPF performs font fallback if the requested characters are not supported by a given typeface. WPF font fallback model is built on top of the WPF font selection model, and detailed description of the fallback logic is outside of the scope of this document.

WPF preprocesses information from all font files in a font collection in order to find the closest font match in a speedy and consistent way. The table below illustrates how this classification process works for faces in the "Arial" font family in the US English environment (please refer to the "

Localization" section for more on how the family and face names are interpreted in different languages.)

Font file name	FontFamily	FaceName	FontStyle	FontWeight	FontStretch
arial.ttf	"Arial"	"Regular"	Normal	Normal	Normal
arialbd.ttf	"Arial"	"Bold"	Normal	Bold	Normal
arialbi.ttf	"Arial"	"Bold Italic"	Italic	Bold	Normal
ariali.ttf	"Arial"	"Italic"	Italic	Normal	Normal
arialn.ttf	"Arial"	"Narrow"	Normal	Normal	Condensed
arialnb.ttf	"Arial"	"Narrow Bold"	Normal	Bold	Condensed
arialnbi.ttf	"Arial"	"Narrow Bold Italic"	Italic	Bold	Condensed
arialni.ttf	"Arial"	"Narrow Italic"	Italic	Normal	Condensed
ariblk.ttf	"Arial"	"Black"	Normal	Black	Normal

Since font matching process is performed multiple times in a typical application lifetime, data similar to what is shown in the above table is stored in the WPF font cache for quick retrieval.

Extracting font matching data from OpenType font files

This section provides information on how WPF computes values of FontFamily, FaceName, FontWeight, FontStyle and FontStretch properties for a given OpenType font file. General familiarity with the OpenType specification is assumed. Please refer to http://www.microsoft.com/typography/otspec/default.htm for more details on the OpenType file format. The document refers to the following new definitions that are not a part of the OpenType 1.4 specification:

- 'OS/2' fsSelection bit 9 to denote oblique font faces.
- 'name' table IDs 21 and 22 to denote WWS family and subfamily names. Please see the "WWS and non-WWS font families" section for more details on WWS families.
- 'OS/2' fsSelection bit 8 to denote a WWS only font face.

Step 1. Extracting family name, face name, style, weight and stretch

FontStyle:

```
If 'OS/2' table is present, use 'fsSelection':
   if fsSelection bit 9 is set, FontStyle is set to Oblique
   else if fsSelection bit 0 is set, FontStyle is set to Italic
   else FontStyle is set to Normal
else use 'macStyle' from 'head' table:
   if macStyle bit 1 is set, FontStyle is set to Italic
   else FontStyle is set to Normal
```

FontStretch:

```
If 'OS/2' table is present, use 'usWidthClass' according to the table below:
```

usWidthClass	WPF font stretch name
1	UltraCondensed
2	ExtraCondensed
3	Condensed
4	SemiCondensed
5	Normal
6	SemiExpanded
7	Expanded
8	ExtraExpanded
9	UltraExpanded

```
else use 'macStyle' from 'head' table:
    if macStyle bit 5 is set, FontStretch is set to Condensed
    else if macStyle bit 6 is set, FontStretch is set to Expanded
else FontStretch is set to Normal
```

FontWeight:

```
If 'OS/2' table is present, use 'usWeightClass':
   if 1 <= usWeightClass && usWeightClass <= 9, usWeightClass = usWeightClass * 100
   if usWeightClass < 1 || usWeightClass > 999 reject the font file as malformed
   set FontWeight to usWeightClass value (WPF supports all values between 1 and 999).
else use 'macStyle' from 'head' table:
   if macStyle bit 0 is set, FontWeight is set to Bold
   else FontWeight is set to Normal
```

WPF directly supports all numeric font weight values between 1 and 999 in both WPF API and markup. In addition, WPF provides a set of named constants for commonly used font weight values, and using such constants is equivalent to using corresponding numeric values.

WPF font weight name	OpenType weight value
Thin	100
ExtraLight	200
UltraLight	200
Light	300
Normal	400
Regular	400
Medium	500
DemiBold	600
SemiBold	600
Bold	700
ExtraBold	800
UltraBold	800
Black	900
Heavy	900
ExtraBlack	950
UltraBlack	950

FontFamily:

Please note that 'name' table ID in the algorithm below refers to the variant of the OpenType 'name' table entry chosen in the following priority order (the top entry is considered first):

1. Platform ID = 3 (Microsoft), valid languageID that corresponds to an existing language.
2. Platform ID = 1 (Macintosh), encoding ID = 0 (Roman), languageID = 0 (English). When decoding such 'name' table entries, 0-127 is identical to Unicode 0-127, 128-255 uses a mapping specified at http://www.unicode.org/Public/MAPPINGS/VENDORS/APPLE/ROMAN.TXT

'name' table entries with properties other than the ones listed above are ignored. If a font contains multiple 'name' table entries with the highest priority, an entry that appears first in the font table is chosen.

```
if 'OS/2 table' is present and 'fsSelection' bit 8 is set
   if 'name' table ID 16 (Preferred Family) is present, assign it to FontFamily
   else if 'name' table ID 1 (Family) is present, assign it to FontFamily
   else reject the font as malformed
else
   if 'name' table ID 21 (WWS Family) is present, assign it to FontFamily
   else[1] if 'name' table ID 16 (Preferred Family) is present, assign it to FontFamily
   else if 'name' table ID 1 (Family) is present, assign it to FontFamily
else reject the font as malformed
```

FaceName:

```
if 'OS/2 table' is present and 'fsSelection' bit 8 is set
   if 'name' table ID 17 (Preferred Subfamily) is present, assign it to FaceName
   else if 'name' table ID 2 (Subfamily) is present, assign it to FaceName
   else reject the font as malformed
```

[1] Algorithm steps highlighted in green are not supported in the first version of WPF. WPF team will consider adding support for these steps in a next release of WPF.

```
else
   if ‘name’ table ID 22 (WWS Subfamily) is present, assign it to FaceName
   else if ‘name’ table ID 17 (Preferred Subfamily) is present, assign it to FaceName
   else if ‘name’ table ID 2 (Subfamily) is present, assign it to FaceName
   else reject the font as malformed
```

Step 2. Font differentiation: resolving conflicts between extracted family name, face name, style, weight and stretch

After extracting family name, face name, style, weight and stretch from an OpenType font file, WPF runs an algorithm that resolves potential conflicts between these values. This process is called *font differentiation*. Since existing fonts may contain weight, style or stretch values that either don't reflect font appearance or conflict with the values from other faces within the same font family, WPF performs font differentiation by default. However, font manufacturers may direct WPF to bypass the font differentiation process by either setting ‘OS/2’ fsSelection bit 8 or providing ‘name’ table IDs 21 and 22 (section “Guidelines for font manufacturers” provides more details on this bypass process.)
All string manipulations in the algorithm below are case insensitive and they are performed using invariant culture. When the term *match* is used, string patterns are matched only when there are *separator characters* (whitespace, ‘.’, ‘-’ or ‘_’) surrounding them. The end of a string is also treated as a separator character for matching purposes, but the beginning of a string is not, therefore string patterns at the beginning of a string are not matched. When a match is removed, the separator characters are removed as well. If the match is surrounded by other words on both sides, a single space is inserted in place of the match to make sure the words that surround it are still separated. Patterns are tested in the order given in the tables, so that, for example, the whole pattern “extra bold” is matched before the pattern “bold”.

A. Build combined family and face name

Remove leading and trailing spaces from FamilyName and FaceName strings. Build a combined family and face name string from the modified FontFamily and FaceName strings as follows:

1. Remove the rightmost occurrence of a regular face name (please refer to the “Regular/upright face patterns” table below) from FaceName, and keep track of it as the correct term for a regular font in case it needs to be concatenated back later.

Regular/upright face patterns
Book
Normal
Regular
Roman
Upright

2. Append FaceName to FontFamily, unless FontFamily already includes the same text as FaceName (case-insensitive) as a substring surrounded by separator characters. A couple of fonts do this, for example:

FontFamily	**FaceName**
Albertus Extra Bold	Bold
DecoType Naskh Swashes	Swashes
BenjaminCaps Caps:001.001	Caps:001.001

The resulting combined family and face name string (*combined string*) is used in the subsequent algorithm steps.

B. Extract terms for style

Match backward from the end of the combined string any known name for italic or oblique styles (please refer to the “Font style patterns” table below) and remove it. The patterns are matched in order they are listed in the table, and once a pattern is matched, the search for subsequent patterns is not performed.

Font style patterns	FontStyle
ita	Italic
ital	
italic	
cursive	
kursiv	
inclined	Oblique
oblique	
backslanted	
backslant	
slanted	

C. Extract terms for stretch

Match backward from the end of the combined string any known name for a font stretch (please refer to the “Font stretch patterns” table below) and remove it. The patterns are matched in order they are listed in the table, and once a pattern is matched, the search for subsequent patterns is not performed. Spaces inside the patterns correspond to zero or more separator characters.

Font stretch patterns	FontStretch
extra compressed	UltraCondensed
ext compressed	
ultra compressed	
ultra condensed	
ultra cond	
compressed	ExtraCondensed
extra condensed	
ext condensed	
extra cond	
ext cond	
narrow	SemiCondensed
compact	
semi condensed	
semi cond	
wide	SemiExpanded
semi expanded	
semi extended	
extra expanded	ExtraExpanded
ext expanded	
extra extended	
ext extended	
ultra expanded	UltraExpanded
ultra extended	
condensed	Condensed

cond	
expanded	Expanded
extended	

D. Extract terms for weight

Match backward from the end of the combined string any known name for a font weight (please refer to the "Font weight patterns" table below) and remove it. The patterns are matched in order they are listed in the table, and once a pattern is matched, the search for subsequent patterns is not performed. Spaces inside the patterns correspond to zero or more separator characters. The patterns are allowed to have a "face" suffix, also optionally separated by separator characters from the pattern name. For example, WPF will treat "Bold Face" the same way as "Bold", and "Heavyface" the same way as "Heavy".

Font weight patterns	**FontWeight**
extra thin	Thin (100)
ext thin	
ultra thin	
extra light	ExtraLight (200)
ext light	
ultra light	
semi bold	DemiBold (600)
demi bold	
extra bold	ExtraBold (800)
ext bold	
ultra bold	
extra black	ExtraBlack (950)
ext black	
ultra black	
bold	Bold (700)
thin	Thin (100)
light	Light (300)
medium	Medium (500)
black	Black (900)
heavy	
nord	
demi	DemiBold (600)
ultra	ExtraBold (800)

Note that the terms 'demi' and 'ultra' are matched alone at the end of the list. While these are most often used in conjunction with another weight or stretch name, if used alone they are assumed to refer to a weight. For example:

FontFamily	**FaceName**
Eurostile	Demi
Franklin Gothic Demi	Italic

Some fonts use abbreviated suffixes to denote font weight characteristics, such as "H" for "Heavy". Since matching short words and numbers like "M", "H" and "W3" can yield many false positives, WPF restricts the matching to the set listed in the "Abbreviated weight patterns" table below. In addition, the following conditions must be satisfied:

1. Suffixes must match the FaceName string.
2. FontWeight value extracted in **Step 1. Extracting family name, face name, style, weight and stretch** must closely match the suffix meaning. The weight difference in OpenType weight units must not exceed 100.
3. Font file must have 'name' table IDs 1, 2, 16, and 17 present, and combined IDs 16 and 17 should equal the value of ID 1.
4. No other weight matches have been detected in this font.

Abbreviated weight patterns	**FontWeight**
EL	ExtraLight
EB	ExtraBold
SB	SemiBold
B	Bold
L	Light
M	Medium
R	Regular
H	Heavy
UH	ExtraBlack
U	UltraBold
W*nnn*	Weight value *nnn*. If *nnn* is between 1 and 9, *nnn* is multiplied by 100.

E. Determine resolved weight

If no weight was extracted in the step D above, the resolved weight is the font weight extracted in **Step 1. Extracting family name, face name, style, weight and stretch**. Otherwise, the weight value in the step D (*extracted weight*) may not match the weight value specified by the font (*font specified weight*). The following observations were made about existing fonts to determine how to resolve such mismatches:

- Incorrect font weight values are often set to 400 (Normal), 500 (Medium) or 700 (Bold) to make fonts work with existing applications and operating systems, and conversely, if a font weight value is not set to one of these three values, it is likely to be correct.
- Values of 400 (Normal) and 500 (Medium) are often used interchangeably.
- Correct weight values are usually on the same side of Normal weight axis as the corresponding descriptive names from the 'name' table.
- Correct weight values are usually not very far apart from the corresponding descriptive names from the 'name' table.

The following algorithm takes the observations above into account to determine which value should be used by WPF.

```
If both the font specified weight and the extracted weight are lighter than Normal
(400), use the font specified weight.

Otherwise, if both the font specified weight and the extracted weight are heavier
than Medium (500) and the font specified weight is not Bold, use the font specified
weight. Example:
```

FamilyName	**FaceName**	**Extracted weight (ignored)**	**Font file weight (used)**
Bookman Old Style	Bold	700 (Bold)	600 (Semi Bold)
Segoe	Black	900 (Black)	800 (Extra Bold)

```
Otherwise, if both the font specified weight and the extracted weight are either of
Normal or Medium, use the font specified weight. Example:
```

FamilyName	**FaceName**	**Extracted weight (ignored)**	**Font file weight (used)**

Helvetica	Medium	500 (Medium)	400 (Normal)
OCRA	Medium	500 (Medium)	400 (Normal)

```
Otherwise, if the font specified weight and the extracted weight differ by no more
than 150, and the font specified weight is neither normal, nor medium, nor bold, then
use the font specified weight. Example:
```

FamilyName	**FaceName**	**Extracted weight (ignored)**	**Font file weight (used)**
Lucida Sans Typewriter	Bold	700 (Bold)	600 (Semi Bold)
Hobo ATT	Medium	500 (Medium)	600 (Semi Bold)

```
Otherwise, use the extracted weight. Example:
```

FamilyName	**FaceName**	**Font file weight (ignored)**	**Extracted weight (used)**
Arial	Black	400 (Normal)	900 (Black)
Garamond	Light	400 (Normal)	300 (Light)

F. Determine resolved stretch

If no stretch was extracted in the step C above, the resolved stretch is the font stretch extracted in **Step 1. Extracting family name, face name, style, weight and stretch**. Otherwise, the stretch value in the step C (*extracted stretch*) may not match the stretch value specified by the font (*font specified stretch*). The following algorithm determines which value should be used by WPF.

If both the font specified stretch and the extracted stretch are narrower than normal, use the font specified stretch. Example:

FamilyName	**FaceName**	**Extracted stretch (ignored)**	**Font file stretch (used)**
Arial	Narrow	4 (Semi condensed)	3 (Condensed)
Univers	Condensed	3 (Condensed)	1 (Ultra condensed)

If both the font specified stretch and the extracted stretch are heavier than normal, use the font specified stretch. For example:

FamilyName	**FaceName**	**Extracted stretch (ignored)**	**Font file stretch (used)**
Microstyle ATT	Extended	7 (Expanded)	8 (Extra expanded)

In all other cases, use the extracted stretch. For example:

FamilyName	**FaceName**	**Font file stretch (ignored)**	**Extracted stretch (used)**
Segoe	Condensed	5 (Normal)	3 (Condensed)
Monotype Modern	Wide	5 (Normal)	6 (Semi expanded)

G. Determine resolved style

If no style was extracted in the step B above, the resolved style is the font style extracted in **Step 1. Extracting family name, face name, style, weight and stretch**. Otherwise, the style value in the step B (*extracted style*) is used instead of the style value specified by the font (*font specified style*). Example:

FontFamily	**FaceName**	**Font file style (ignored)**	**Extracted style (used)**
Lucida Handwriting	Italic	Normal	Italic
Lucida Typewriter	Oblique	Italic	Oblique

H. Extract numbers that describe font style, weight and stretch from face names.

Some popular font families use special numbering schemes to identify their faces. The face numbers typically correspond to style, weight and stretch combinations. WPF matches a possible face number (furthermore denoted as *candidateFaceNumber*) from the beginning of the FaceName string, and requires the number to be followed by either a separator character or the end of the string. When *candidateFaceNumber* corresponds to font style, weight and stretch determined by steps E – G, *candidateFaceNumber* is removed from the combined string, so that the resulting font family matches the font designer intent. WPF supports the following two numbering schemes:

1. Double digit numbering system used in fonts such as http://www.linotype.com/6-1823-6/neuehelvetica.html. The face number consists of two or three digits that describe the font weight, style and stretch. The last digit to the right (*candidateFaceNumber % 10*) describes font style and stretch, and WPF interprets its value the following way:

candidateFaceNumber % 10	**Accepted FontStretch and FontStyle values**
3	fontStretch > Normal
4	fontStretch > Normal && fontStyle != Normal
5	fontStretch == Normal
6	fontStretch == Normal && fontStyle != Normal
7	fontStretch < Normal
8	fontStretch < Normal && fontStyle != Normal
9	fontStretch < Condensed

The remaining digits to the left correspond to *candidateFaceNumber / 10*, and denote font weight. WPF interprets the latter as shown in the table below:

candidateFaceNumber / 10	**Accepted fontWeight values**
2, 3, 4	fontWeight < Normal
5	fontWeight >= Normal && fontWeight <= Medium
6,7,8,9,10	fontWeight > Normal

2. Three digit numbering system described at http://www.linotype.com/6-1805-6-15548/numeration.html. The face number consists of three digits that describe the font weight, style and stretch. The first digit (*candidateFaceNumber / 100*) denotes weight and is interpreted as shown in the table below:

candidateFaceNumber / 100	**Accepted fontWeight values**
1, 2, 3	fontWeight < Normal
4, 5	fontWeight >= Normal && fontWeight <= Medium
6,7,8,9	fontWeight > Normal

The second digit ((*candidateFaceNumber % 100) / 10*) denotes stretch and is interpreted as shown in the table below:

(candidateFaceNumber % 100) / 10	**Accepted fontStretch values**
1, 2	fontStretch < Normal
3	fontStretch == Normal
4	fontStretch > Normal

The third digit (*candidateFaceNumber % 10*) denotes style and is interpreted as shown in the table below:

candidateFaceNumber % 10	**Accepted fontStyle values**
0	fontStyle == Normal
1	fontStyle != Normal

If *candidateFaceNumber* doesn't match font style, weight and stretch values according to any of the supported numbering schemes, the face number remains in the combined string.

I. *Determine final FontFamily and FaceName*

Compare the original FontFamily value with the final combined string.

If they are the same, consider the original FontFamily value to be correct, and use the font provided values for FontFamily and FamilyName.

If they differ, instead use the combined string as a FontFamily instead, and construct a new FaceName string as the concatenation of strings representing the extracted stretch, weight and style as described below:

a) Determine resolved weight name.

```
if the extracted weight was used as a resolved weight in lieu of the font weight
      use the weight name pattern that was matched
else if the font weight value is Normal
      use an empty string
else if the weight value corresponds to a known FontWeight value from the "Font
weight patterns" table
      use the font weight name
else
      use the string conversion of a decimal representation of the font weight value
```

b) Determine resolved stretch name.

```
if the extracted stretch was used as a resolved stretch in lieu of the font stretch
      use the stretch name pattern that was matched
else if the font stretch value is Normal
      use an empty string
else
      use a FontStretch name from the "Font stretch patterns" table
```

c) Determine resolved style name.

```
if the extracted style was used as a resolved style in lieu of the font style
      use the style name pattern that was matched
else if the font style value is Normal
      use an empty string
else
      use a FontStyle name from the "Font style patterns" table
```

d) If weight, style and stretch names are all empty, then the FaceName is set to the regular face name extracted in the step A-1. If no regular face was extracted, FaceName is set to "Regular".

Resolving conflicts across multiple font faces

It is possible for a single font collection to contain multiple font files that define the same values of font family, face name, weight, style and stretch, therefore creating duplicate font faces.

Where two faces in the same font family have the same stretch, weight and style, one is discarded by WPF. The following tests are made in order to determine which is kept:

- If the versions differ, the highest version is kept. WPF determines the font version using the following algorithm:

i. Select the version string from the 'name' table:

```
Obtain the list of localized strings with 'name' table ID 5. The 'name' table entries
are chosen using the priority logic described in "Step 1. Extracting family name,
face name, style, weight and stretch".
If an entry with LanguageId 0x409 (en-US) exists, use that entry as the version
string.
Otherwise, use an entry with the highest language ID.
```

ii. Convert the version string to the version number:

```
Starting from the beginning of the version string, find the first floating point
number in the decimal, en-US, mantissa-only format.
Convert this number to a floating point value to be used as a version number.
```

- If the version string doesn't contain a valid floating point number, the version number is set to 0.

- If the file last modification times differ, the latest modified file is kept.
- If the file Uri's differ, the alphabetically greater file is kept. String comparison is done using invariant culture and ignoring the case.
- If the face indices (in a .ttc file) differ, the higher face index is kept.

Adding simulated font faces

Adding simulated bold faces to a font family

When a font family doesn't contain bold font faces for all combinations of style and stretch, WPF adds simulated bold faces to the family using the following process:

1. The faces in the family that have the same style and stretch are combined into individual collections and sorted by weight.
2. For each such collection, find the heaviest weight present.
3. If the heaviest weight is at least 350, and no more than 550, add a new face based on the face with the heaviest weight, with a weight of Bold (700), and FaceName the same as the FaceName of the identified heaviest face with the word "Bold" added.

Before adding "Bold" to the face name, any existing weight or term for regular in the face name is removed using the pattern matching and extraction rules from the "

Extracting font matching data from OpenType font files" section.

First, the "Regular/upright face patterns" list is scanned in order and the first match removed from the FaceName.

Then, the "Weight simulation patterns" list below is scanned in order and the first match removed from the face name:

Weight simulation patterns
extra light
ext light
ultra light
semi bold
demi bold
bold
thin
light
medium
demi

Adding simulated oblique faces to a font family

Where a font family doesn't contain matching oblique font faces for all regular font faces, WPF adds simulated oblique faces to the family using the following process.

1. The faces in the family that have the same weight and stretch are combined into individual collections and sorted by style.
2. For each such collection, find a regular style.
3. If a regular style is available and an oblique style is not available, add a new face based on the regular style with a style of Oblique, and a face name the same as the regular style with the word "Oblique" added.

Before adding "Oblique" to the face name, any existing term for regular in the face name is removed using the pattern matching and extraction rules from the "

Extracting font matching data from OpenType font files" section. This is done by scanning the "Regular/upright face patterns" list from the subsection "Build combined family and face name" above in order and removing the first match from the FaceName.

Finding the closest font face match for an input type face

This section describes how WPF selects the closest matching font face from a font collection given the logical font description.

User input:

- FontFamily string, e.g. "Arial"
- FontWeight value, e.g. SemiBold.
- FontStyle value, e.g. Italic.
- FontStretch value, e.g. Condensed.
- Font collection identifier (e.g., Uri of a font folder)

Internal font collection:

- A pre-created list of font face structures in the following format { FontFamily, FaceName, FontWeight, FontStyle, FontStretch } that corresponds to all font faces in a font collection.

Output:

- A font face from the internal data that is the best match for the user input.
- Implied font face flag that describes whether the input FontFamily string identifies a complete type face and not just a font family.

Step 1. Finding the font family

If the input FontFamily exactly matches a FontFamily value in the internal font collection, identify all font faces that have this FontFamily value and save them as a *candidate face list*. Proceed to **Step 2. Matching a face from the candidate face list.**

Otherwise, find the longest substring beginning from the start of the input FontFamily string that matches an existing face in the internal font collection. Identify all font faces whose FontFamily value exactly matches this substring. Compare the non-matched remainder of the input FontFamily string with FaceName values of such font faces. If the remainder exactly matches the FaceName value of a font face, such font face is considered to be an exact match for the input request, the implied font face flag is set to true and the algorithm completes. If the remainder doesn't match any FaceName values, the font face algorithm terminates with no matching face found.

Step 2. Matching a face from the candidate face list.

The task at this step is reduced to finding the best { FontWeight$_i$, FontStyle$_i$, FontStretch$_i$ } combination from the candidate face list to match the input { FontWeight, FontStyle, FontStretch }. The algorithm uses a notion of *font attribute vector*, which for a given combination of { weight, style, stretch } is computed using the following formulae,

```
FontAttributeVector.X = (stretch - 5) * 11.0;
FontAttributeVector.Y = style * 7.0;
FontAttributeVector.Z = (weight – 400) / 100.0 * 5.0;
```

where weight is measured in the same units as the OpenType 'OS/2' usWeightClass value, and stretch is measured in the same units as the OpenType 'OS/2' usWidthClass value.

WPF uses the following method to determine whether a candidate font attribute combination matches an input font attribute combination better than another font attribute combination:

```
bool IsCandidateBetterMatchThanOther(
    FontAttributeVector candidate,
    FontAttributeVector other,
    FontAttributeVector input)
{
    double distanceBetweenCandidateAndInput = VectorDistance(candidate, input);
    double distanceBetweenOtherAndInput = VectorDistance(other, input);

    // Shorter distance from the input vector is a better match.
    if (distanceBetweenCandidateAndInput < distanceBetweenOtherAndInput)
        return true;
    if (distanceBetweenCandidateAndInput > distanceBetweenOtherAndInput)
        return false;

    double dotProductBetweenCandidateAndInput = VectorDotProduct(candidate, input);
    double dotProductBetweenOtherAndInput = VectorDotProduct (other, input);

    // Stronger projection onto the input vector is a better match.
    if (dotProductBetweenCandidateAndInput > dotProductBetweenOtherAndInput)
        return true;
    if (dotProductBetweenCandidateAndInput < dotProductBetweenOtherAndInput)
        return false;

    // Stronger X component is a better match.
    if (candidate.X > other.X)
        return true;
    if (candidate.X < other.X)
        return false;

    // Stronger Y component is a better match.
    if (candidate.Y > other.Y)
        return true;
    if (candidate.Y < other.Y)
        return false;

    // Stronger Z component is a better match.
    if (candidate.Z > other.Z)
        return true;
    if (candidate.Z < other.Z)
        return false;

    // All components are equal, the candidate is no better match than other.
    return false;
}
```

This matching algorithm has the following properties:

1. Matching of weight and stretch are monotonic, for example if the requested weight is increased from ultra light to extra bold, the faces matched will get progressively heavier.
2. Priority order is: stretch, style, weight. Stretch is considered most important in choosing an alternate face because it has most effect on overall layout. Style is next because algorithmic italicization is typically less desirable in terms of appearance than algorithmic emboldening. Therefore, the error distance between requested and available stretch is counted worse than the difference between requested and available styles, and in turn than requested and available weights.
3. Regular style on each axis acts as an “anti-magnet”. For example, if a bold weight is requested, WPF will choose any weight heavier than regular before considering regular or a lighter weight.

3. Localization

Generally, WPF supports font family and face names specified in multiple languages independent of system language settings.

One important exception is font differentiation. Since the font differentiation process is based on English name parsing, it requires the primary language name from the font 'name' table ID 1 (Family) to have its main language ID (http://www.microsoft.com/typography/otspec/name.htm#lang3) set to English, i.e. (LCID & 0x3ff) == 0x09. If this is not the case, the font differentiation process is bypassed. If there are multiple entries with main language ID set to English, the language is obtained in the following priority order:

```
if there is a name table entry for LanguageId 0x409 (en-US), use that entry
else sort the name table names by LanguageId in numerical order, and use the first
language.
```

The chosen language is called *primary language*. If 'name' table doesn't contain IDs 16 and 17 in the primary language, the font is treated as not having IDs 16 and 17.

If the font differentiation process is performed, the generated FamilyName affects the font name localization in the following way:

1. If the resulting FamilyName is the same as the FamilyName per 'name' table ID 16 (Preferred Family), the FamilyName localizations from 'name' table ID 16 and FaceName localizations from 'name' table ID 17 (Preferred Subfamily) or, in case 'name' table ID 17 is not present in the font, 'name' table ID 2 (Subfamily), are supported by WPF.
2. Otherwise, if the resulting FamilyName is the same as the FamilyName per 'name' table ID 1 (Family), the FamilyName localizations from 'name' table ID 1 and FaceName localizations from 'name' table ID 2 (Subfamily) are supported by WPF.
3. Otherwise, since FamilyName is modified from its original value, all original localizations of FamilyName and FaceName are discarded and only primary language versions are supported.

4. Guidelines for font manufacturers

This section outlines recommendations for font manufacturers to create fonts that work best with the WPF font selection model.

1. Fonts should set typographically appropriate and unambiguous stretch, weight and style values in the 'OS/2' table. A font family should increase usWeightClass values for its faces monotonically as the font weight gets heavier, and a font family should increase usWidthClass values for its faces monotonically as the font stretch gets wider. There should be no duplicate combinations of font weight, style and stretch values within the same font family (please see the "WWS and non-WWS font families" subsection below for an important clarification.) Fonts are encouraged to differentiate between italic and oblique styles by setting fsSelection bits 0 and 9 respectively.
2. Fonts should support the same set of name table localization languages for all faces in the same font family.
3. Fonts are encouraged to include native bold and oblique faces to achieve better appearance compared to the simulated bold and oblique.

WWS and non-WWS font families

WPF definition of a font family is different from the preferred family definition from the OpenType specification. WPF font families are composed of font faces that differ only in weight, style and stretch, whereas fonts that have the same value of the preferred family property can have the same weight, style and stretch, but differ in other traditional attributes, such as "handwriting", "Caption", "Subheading", "Display", "Optical" etc.

Font families that conform to the former definition of a font family are called *WWS font families* (WWS stands for traditional typographic terms 'weight', 'width' and 'slope.')

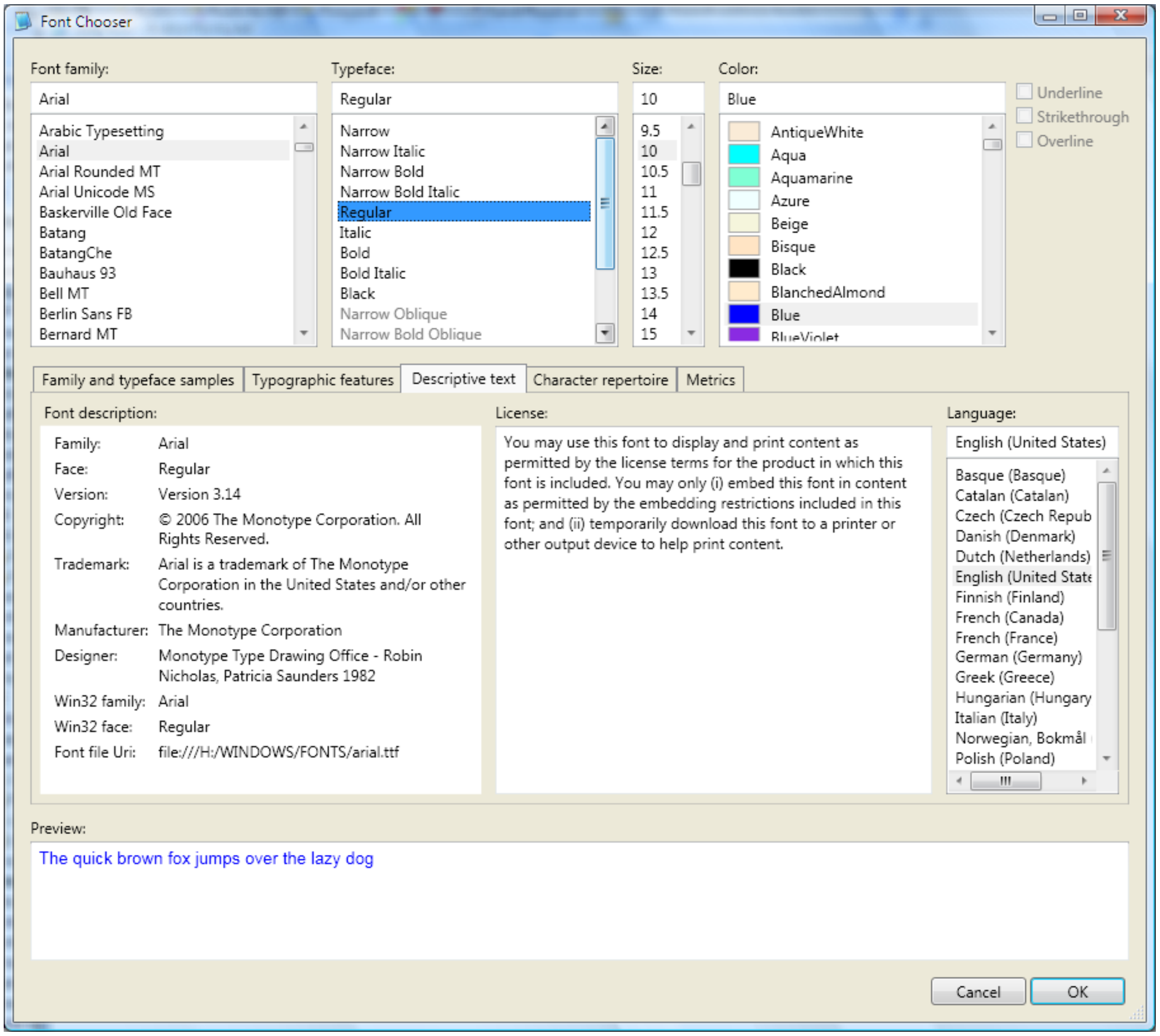
Font families that use the latter, broader, definition of a font family are called *non-WWS font families*.

Fonts should clearly specify whether they are a part of a WWS or a non-WWS font family:

1. Font that is a part of a WWS font family should set 'OS/2' 'fsSelection' bit 8.
2. Font that is a part of a non-WWS font family should unset 'OS/2' 'fsSelection' bit 8, and provide 'name' table IDs 21 and 22 that include other differentiating attributes, such as "handwriting", "Caption", "Subheading", "Display" and "Optical", as a part of the 'name' table ID 21. 'name' table ID 22 should reflect only style, weight and stretch attributes of the font.

5. Font chooser guidelines

It is recommended that applications provide font choosing user interface built on top of the WPF System.Windows.Media.Fonts object. Using this method guarantees that all available typefaces will be presented to the user as a part of the appropriate font family, and the selected type face will be represented unambiguously in the XAML markup. Here is an example of possible font chooser dialog:



Font Chooser

Font family: Arial

Arabic Typesetting
Arial
Arial Rounded MT
Arial Unicode MS
Baskerville Old Face
Batang
BatangChe
Bauhaus 93
Bell MT
Berlin Sans FB
Bernard MT

Typeface: Regular

Narrow
Narrow Italic
Narrow Bold
Narrow Bold Italic
Regular
Italic
Bold
Bold Italic
Black
Narrow Oblique
Narrow Bold Oblique

Size: 10

9.5
10
10.5
11
11.5
12
12.5
13
13.5
14
15

Color: Blue

AntiqueWhite
Aqua
Aquamarine
Azure
Beige
Bisque
Black
BlanchedAlmond
Blue
BlueViolet

Underline
Strikethrough
Overline

Font description:

Family:	Arial
Face:	Regular
Version:	Version 3.14
Copyright:	© 2006 The Monotype Corporation. All Rights Reserved.
Trademark:	Arial is a trademark of The Monotype Corporation in the United States and/or other countries.
Manufacturer:	The Monotype Corporation
Designer:	Monotype Type Drawing Office - Robin Nicholas, Patricia Saunders 1982
Win32 family:	Arial
Win32 face:	Regular
Font file Uri:	file:///H:/WINDOWS/FONTS/arial.ttf

License:

You may use this font to display and print content as permitted by the license terms for the product in which this font is included. You may only (i) embed this font in content as permitted by the embedding restrictions included in this font; and (ii) temporarily download this font to a printer or other output device to help print content.

Language: English (United States)

Basque (Basque)
Catalan (Catalan)
Czech (Czech Repub
Danish (Denmark)
Dutch (Netherlands)
English (United State
Finnish (Finland)
French (Canada)
French (France)
German (Germany)
Greek (Greece)
Hungarian (Hungary
Italian (Italy)
Norwegian, Bokmål
Polish (Poland)

Preview:

The quick brown fox jumps over the lazy dog

Please note that the dialog combines all Arial faces into a single font family. In addition, the following techniques used in the dialog above are encouraged:

1. Once a font family is selected, any type face from it can be selected using a single mouse click. This exposes the full range of available typefaces to the customer.
2. Simulated font faces appear visually distinct from font faces that correspond to font files. This enables customers to choose higher quality font faces for their applications and documents.
3. Physical font location and version available for diagnostic purposes.
4. Font license information is available.
5. It is possible to display preview text in the variety of languages, sizes and colors.