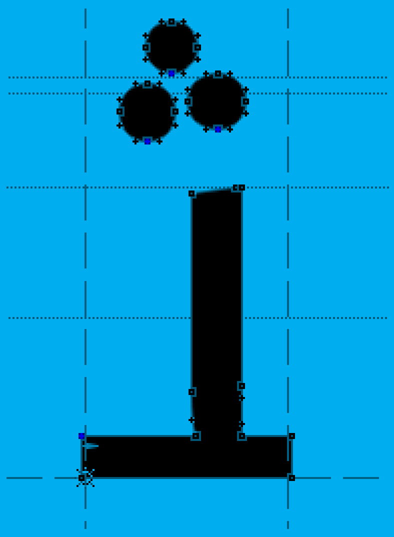
Stéphane Boeuf

Arabic Font Production Tutorial

Part I Typographic Fonts



Khatt Books

Arabic Font Production Tutorial

This tutorial is the result of an interview of Stéphane Boeuf by Edo Smitshuijzen. While working together on the interview the idea arose to produce a much needed tutorial about how to produce an Arabic font. Stéphane Boeuf wrote the text of the tutorial and Edo Smitshuijzen was a sounding board for it.

Stéphane has worked as computer engineeer for more than 15 years. He studied the Arabic language at the University of Grenoble and continued studying with the CNED *(Centre National d'Education à Distance). He joined WinSoft International in 2006, and worked since, he worked on the Middle Eastern versions of FileMaker and Adobe Dreamweaver, and to a lesser extent on InDesign. He was also involved in the design of the shaping engine which is the basis for every Middle Eastern product, where he added to the engine the OpenType support for scripts of India and South East Asia.*

This tutorial is the first of a series of three. The other tutorials will deal with Arabic calligraphic and web fonts.

Copyright © 2011.Stéphane Boeuf

This edition is first published in 2011
© Khatt Books

Khatt Books
Van Tuyll Van Serooskerkenweg
1076 JT Amsterdam
The Netherlands
www.khattbooks.com

Book Design: Huda Smitshuijzen AbiFares,

This book is set in the bilingual font Fedra Serif and Fedra Sans by Peter Bilak (www.typotheque.com).

All rights reserved. No parts of this book may be reproduced in any form or by any means without prior written permission from the publisher.

The information in this book is distributed without warranty. While every effort has been made to insure the accuracy of the information in this book, neither the author nor the publisher are responsible for inadvertently overlooking any copyright holders and will be pleased to include any necessary credits in any subsequent edition.

Introduction

The purpose of this booklet is to explain the basics and provide a step by step guide for the creation of an Arabic OpenType font. Before you start, you must be familiar with the font creation process; in particular you should already have designed the glyphs of the font.

First, we will present the various technologies and tools involved in the process and describe how to use them. Second, we will provide a step by step guide on how to use FontLab and VOLT.

The Basics of Arabic Font Making

The process in which a series of characters typed on the keyboard are transformed into a series of glyphs displayed on the screen is quite complex. It involves three basic components:
1. A character encoding technology (Unicode)
2. A text engine software (there are many, that can be as basic as Notepad or as sophisticated as Adobe's InDesign ME)
3. A font technology (OpenType)

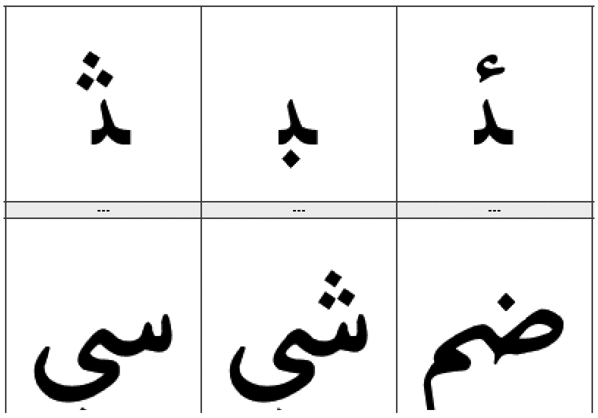
1. UNICODE

Unicode is, among other things, a huge catalog of characters including all the writing scripts in the world. One of the objectives of Unicode is to provide a unique code for every character ever written. The latest version Unicode 6.0 (www.unicode.org) defines tens of thousands of characters ranging from Egyptian hieroglyphs to Braille...
The Arabic script is mainly defined within a range of character starting with the code 0x600 (0x means hexadecimal encoding) and ending with the code 0x6FF. (http://www.unicode.org/charts/PDF/U0600.pdf shows the complete chart). Additional ranges are defined but let's focus on this range since the other ranges (in particular the Arabic extensions A et B) are outdated and can only lead to confusion.

If you want to type for instance فثل (it is not a real Arabic word but it does not matter here) the corresponding Unicode encoding is: 0x641(FA) 0x62B (THA) 0x644 (LAM). When you type these letters on your keyboard, what are actually transmitted to the software you are using are these codes, we'll call it the input code; it is a series of character codes.

2. TEXT ENGINE

Let's see now how the software (which contains the text engine) transforms the input code (0x641 0x62B 0x644) into the nice looking output فثل. The output is constituted by glyphs. Glyphs are the graphic signs of a font. They can take various forms: they can be linked to a character code or not, they can represent one or more


Some glyphs from the *me_quran* font which can be downloaded from http://arabicfonts.wikispaces.com/. As one can see, some glyph represent one character, those who represent more than one are called *ligatures*.

characters, or even a part of a character.
First, the software will analyze the input and determine the contextual forms of each character, as you know, a letter in Arabic usually has 4 different shapes; the initial, the medial, the final and the isolated. In my example we have an initial *fa*, a medial *tha* and a final *lam*.
The software will "discuss" with the font to get the information it requires.There are many different font technologies and many different ways of achieving this, let's focus on the OpenType font technology which is the dominant one at the moment.

3. OPENTYPE

OpenType is a font technology essentially co-developed by Microsoft and Adobe which allows the creation of 'rules' (or features in the OpenType terminology, indicated by 4-letter tags) inside the font. The idea behind OpenType is that the font designer is the best person to determine how the font should behave. OpenType does not deal with the design of the glyphs of the font, but rather with how these glyphs can interact with one another.

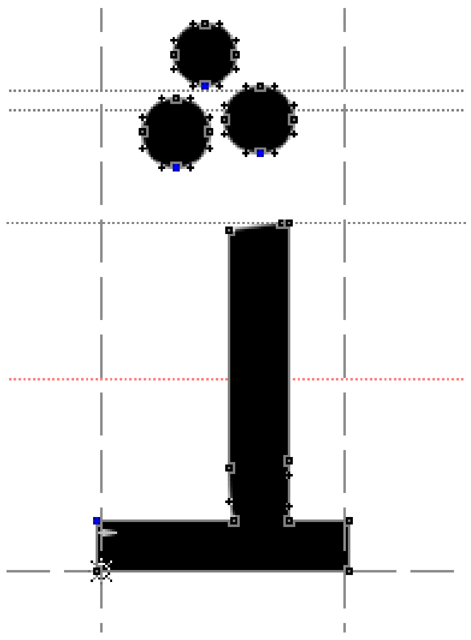
OpenType defines 3 categories (categories are also called font tables):
1. The GDEF (for definitions) which lets you define names and properties for the glyphs.
2. The GSUB (for substitutions) which lets you define replacement rules, i.e. how a glyph can be replaced by another one and under which circumstances.
3. The GPOS (for positioning) which lets you define the position of glyphs against the position of other glyphs.

Microsoft offers a free tool called VOLT (acronym for Visual OpenType Layout Tool) which can be downloaded from http://www.microsoft.com/typography/VOLT.mspx

Using VOLT you will be able to define the rules for your font, but you can't define any rule you want, you need to follow the specifications for the particular script you're aiming to support. For the Arabic script those specifications can be found at:
http://www.microsoft.com/typography/OpenType%20Dev/arabic/intro.mspx

Since these specifications are very technical and not very easy to understand, let’s take a closer look at them. The important thing in these specifications for a font designer is the kind of features that are applied and the order in which they are applied. Depending on the complexity of the font you are designing, you might need to implement all of these features or only a subset of them.

Let’s consider the case of a simple ‘modern’ Arabic font. By this I mean, a font where the connections between the glyphs are fairly regular and linear (on the left and right sides of the glyphs).


A typical ‘modern’ glyph with equal horizontal connections

With these fonts the number of features to implement is restricted, you will need at least:
‘isol’: to define the isolated positional forms
‘init’: to define the initial positional forms
‘medi’: to define the medial positional forms
‘fina’: to define the final positional forms
‘rlig’: to define the required ligatures (usually only *lam-alef* لا is a required ligature)
‘mark’: to define the position of marks against base letters
‘mkmk’: to define the position of marks against other marks

Step by Step Guide

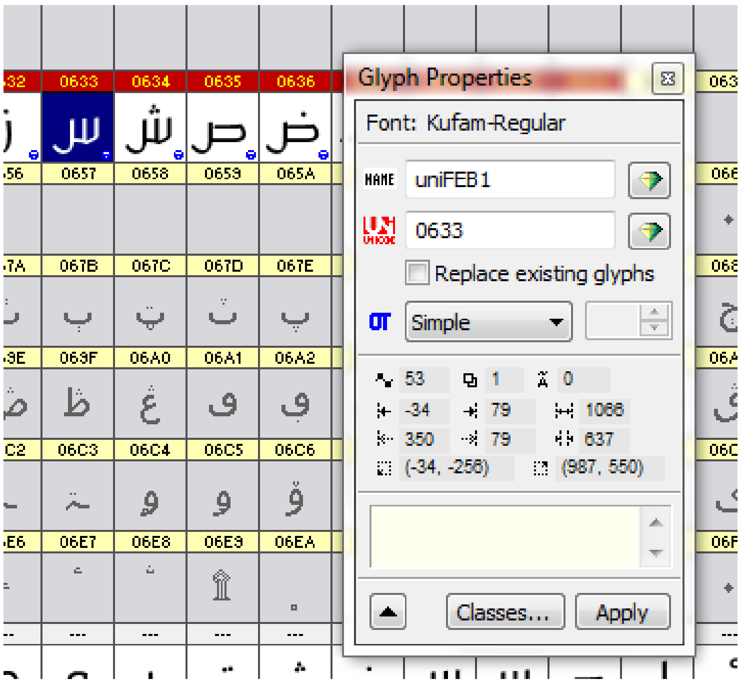
To produce Arabic fonts, two different tools must (still) be used: a font editor, like FontLab and - since FontLab cannot define your font sufficiently - the Microsoft VOLT editor is needed to complete the required definitions.
Adobe Indesign ME will show Arabic fonts properly even without the definitions, because this software has a built-in Arabic font engine that only needs the proper Unicodes to function. All other software, including web browsers will need the definitions made with VOLT.

FONTLAB STUDIO

Let’s see how to implement these features using FontLab Studio and Microsoft VOLT.
The first thing to do is to define the glyphs; you need to give each glyph a unique name, affect a Unicode value if needed and specify the OpenType value.

A very important thing is that only a few glyphs in the font should have a Unicode value, most of the glyphs are only used through the features that we will define and therefore don’t need a Unicode value.

All this information can be set from the Glyph Properties panel of FontLab Studio.



Glyph Properties

Font: Kufam-Regular

NAME uniFEB1

0633

Replace existing glyphs

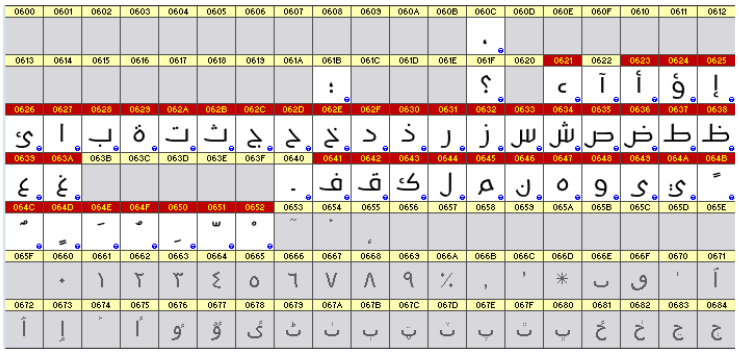
OT Simple

53 1 0
-34 79 1066
350 79 637
(-34, -256) (987, 550)

Classes... Apply

The ‘NAME’ field is where you specify the name you have chosen for the glyph; it is up to you to choose a name, the only constraint is that the name must be unique and that every glyph must have a name.

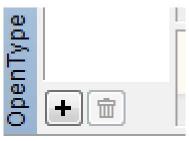
The 'UNI' field is where you specify the Unicode value (code) associated with the glyph. As I said earlier, it is not necessary for all the glyphs to have a Unicode value. For a typical modern font you would need to define at least the basic character set for Arabic. On the picture below, you see the typical set of Unicode values that need to be defined:

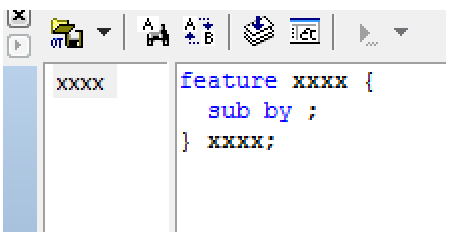

0600 0601 0602 0603 0604 0605 0606 0607 0608 0609 060A 060B 060C 060D 060E 060F 0610 0611 0612
0613 0614 0615 0616 0617 0618 0619 061A 061B 061C 061D 061E 061F 0620 0621 0622 0623 0624 0625
0626 0627 0628 0629 062A 062B 062C 062D 062E 062F 0630 0631 0632 0633 0634 0635 0636 0637 0638
0639 063A 063B 063C 063D 063E 063F 0640 0641 0642 0643 0644 0645 0646 0647 0648 0649 064A 064B
064C 064D 064E 064F 0650 0651 0652 0653 0654 0655 0656 0657 0658 0659 065A 065B 065C 065D 065E
065F 0660 0661 0662 0663 0664 0665 0666 0667 0668 0669 066A 066B 066C 066D 066E 066F 0670 0671
0672 0673 0674 0675 0676 0677 0678 0679 067A 067B 067C 067D 067E 067F 0680 0681 0682 0683 0684

The 'OT' field is where you specify the OpenType value (tag) associated with the glyph. You have the choice between:
—Unassigned: the default value which needs to be changed
—Simple: the value to use for most glyphs
—Ligature: the value for ligature glyphs, when you select this value, a numeric field is activated to let indicate how many characters are present in the ligature glyph.
—Mark: the value to use for marks like *fatha*, *shadda*, etc.
—Component: the value to use if the glyph represents only a part of a character.

NB. FontLab creates automatically a numeric glyph index. Each glyph is given a sequential number following the order of introduction of the glyph.
All these values constitute the GDEF part defined by OpenType that was presented earlier.

Once all the glyphs are defined and therefore the GDEF part is completed, it is time to export these definitions to VOLT. To do so, you need to open the OpenType panel in FontLab (using the menu entry Windows | Panels | OpenType).
You need to create a 'dummy' feature to get things started; to do so, click on the + button in the OpenType panel (below, right), a feature called 'xxxx' (below, left) is created.





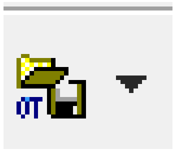
xxxx

```
feature xxxx {
  sub by ;
} xxxx;
```

You need to delete the sub by; text and leave just:

```
feature xxxx {
} xxxx;
```

Now that a feature is created, click on the arrow on the ‘Save’ button of the OpenType panel and select from the menu ‘Save Features’. In the ‘Save As...’ dialog, select VOLT project files (*.vtp).



The final thing to do in FontLab is to generate the font as a TTF file (and NOT as a OTF file).

NB. If we take a look under the hood at the contents of the VTP file, we see definitions that look like these:

```
DEF_GLYPH “uniFE9B” ID 285 UNICODE 65179 TYPE BASE END_
GLYPH
```

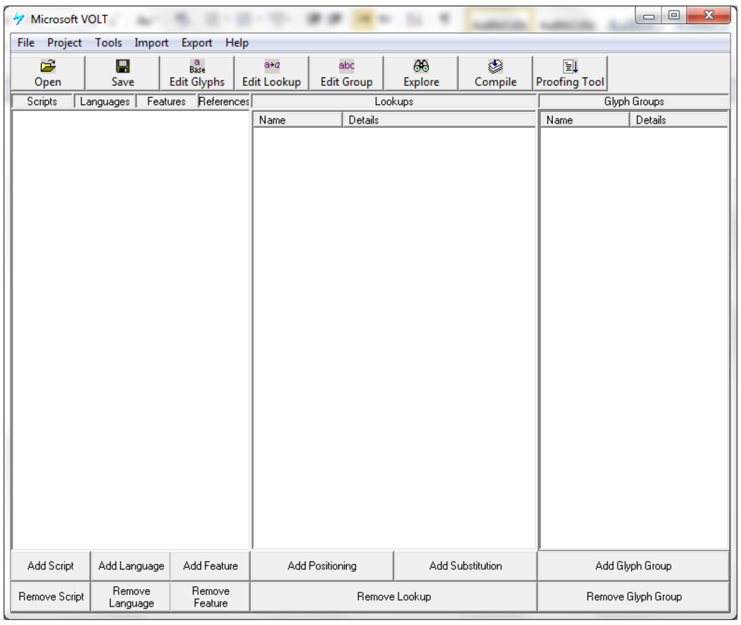
We recognize the various elements already defined, the name (« uniFE9B »), the glyph index (285), the Unicode value (65179), the OpenType value (BASE).

MICROSOFT’S VOLT

Now let’s start VOLT and open the TTF file (menu File | Open font), then in the Import menu, select *Import Project* and select the VTP file which we just created.
Now if you click on the *Edit Glyphs* button, a new dialog appears and you can see that the glyph name, code, etc. are preserved from the FontLab file.

That part was somehow fastidious but necessary to start working in VOLT.
Please note: if you make changes in your FontLab file you need to export a new VTP and do it all over again. Therefore it is recommended to start the VOLT part of the font when the design of the font is completely done. However, you can modify existing glyphs but if you add or delete glyphs you would need to export a new Volt Project File.

The VOLT interface is very plain and split into three parts to work on:
1. features
2. lookups
3. groups

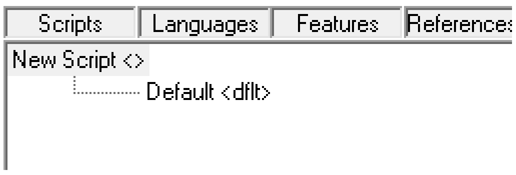


The VOLT interface

1. Features

The first step is to define the script (or scripts) that the font will address. A script is basically a writing system, it is important to distinguish writing system from languages; for instance the Arabic script is used to write Arabic, Farsi, Urdu, Sindhi, Uyghur etc., the Latin script is used to write English, French, German etc.
A font can address many scripts, usually Arabic fonts also support the Latin script, and many Latin fonts also support the Cyrillic script or the Greek script.
Here we are focusing on the Arabic script, so we will just define this one. To do so, click on the “Add Script” button on the bottom left of the interface.

An item appears, labeled ‘New Script<>’:



You need to change it to ‘arab’. It is very important to define the script tag.

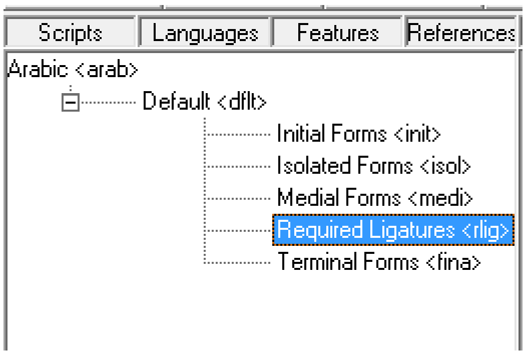
As you can see another item was created automatically (Default <dflt>). This item indicates the language that will be associated with the script.
The default value applies to any language; if you need to create specific rules for Urdu for instance you will need to define an Urdu language tag here, the 4-letter tags for the various languages are defined here :
http://www.microsoft.com/typography/otfntdev/arabicot/appen.aspx
For our purpose *Default* will be enough.

The next step is to implement the features.
We will start with the 'init' feature which indicates what the initial form for the letters is. Just click on the 'Add Feature' button, and type <init> in the new item that was created.

NB: for some strange reason, when you type a script name you don't need to type the brackets (< >), but these brackets are required for the features...

To complete our font, we need to define the following features : <init>, <isol>, <medi>, <fina> and <rlig>. Add them the same way we did for <init>.

The interface should now look like this:

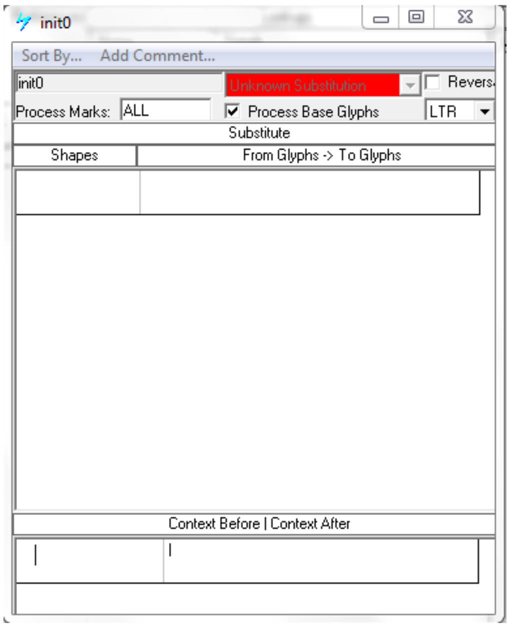


Arabic <arab>
Default <dflt>
Initial Forms <init>
Isolated Forms <isol>
Medial Forms <medi>
Required Ligatures <rlig>
Terminal Forms <fina>

2. Lookup
It is now time to concentrate on the 'lookups' section. A lookup is a set of basic rules that are applied when a feature is activated. A lookup usually applies to only one feature, but a feature can be represented by numerous lookups.
OpenType defines two kinds of lookups, the substitution lookups (to build the GSUB part) and the positioning lookups (to build the GPOS part).
The features we have defined so far all require substitution lookups so let's create a substitution lookup by clicking on the 'Add Substitution' button.

A lookup called 'New substitution' is created, the first thing to do is to give it a better name. Since we are working on the <init> feature, let's call it <init0>. If we need more lookups for the <init> feature, we'll call them <init1>, <init2>...
The names by themselves have no importance; it is just easier to give them a name that relates to the feature.

A double-click on the lookup opens the lookup dialog.



init0

Process Marks: ALL

☑ Process Base Glyphs

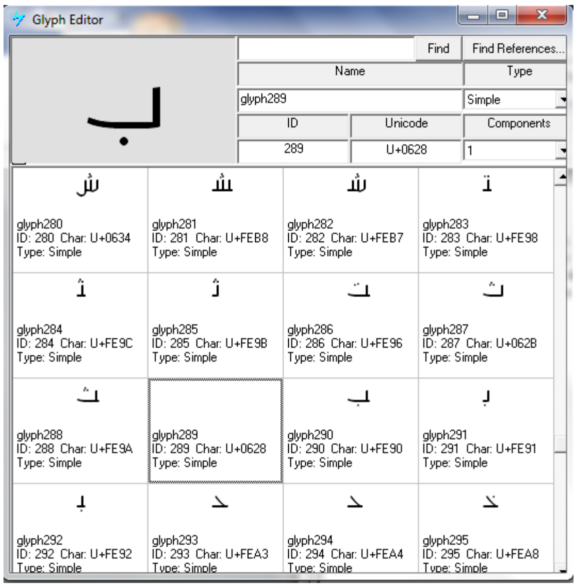
☐ Revers

LTR

The substitution lookup dialog

There are a few things to set before starting to work on the glyphs; first the direction of the script, since we are dealing with Arabic, we need to change the LTR (left to right) value to RTL (right to left) in the right upper corner.
Then we need to change the value of the 'Process Marks' field. By default it is set to ALL which signifies that all the glyphs that we have set as MARKS (vocalisation marks and other diacritics) earlier will be taken into account. Since the <init> feature only deals with the form of (unvocalised) base letters we don't want to consider marks here, so let's set it to NONE.
This being done, it is time to enter our first rule, I recommend to work alphabetically, it is easier. We are working on the <init> feature, so we need to indicate how a letter is transformed when it is in the initial position (beginning of a word, or after a non-connecting letter such as *alef, dal*...).

Let's open the glyph panel (by clicking on the Edit Glyphs button), let's search for the *ba*.



Name	Type
glyph289	Simple

ID	Unicode	Components
289	U+0628	1

glyph280
ID: 280 Char: U+0634
Type: Simple

glyph281
ID: 281 Char: U+FEB8
Type: Simple

glyph282
ID: 282 Char: U+FEB7
Type: Simple

glyph283
ID: 283 Char: U+FE98
Type: Simple

glyph284
ID: 284 Char: U+FE9C
Type: Simple

glyph285
ID: 285 Char: U+FE9B
Type: Simple

glyph286
ID: 286 Char: U+FE96
Type: Simple

glyph287
ID: 287 Char: U+062B
Type: Simple

glyph288
ID: 288 Char: U+FE9A
Type: Simple

glyph289
ID: 289 Char: U+0628
Type: Simple

glyph290
ID: 290 Char: U+FE90
Type: Simple

glyph291
ID: 291 Char: U+FE91
Type: Simple

glyph292
ID: 292 Char: U+FE92
Type: Simple

glyph293
ID: 293 Char: U+FEA3
Type: Simple

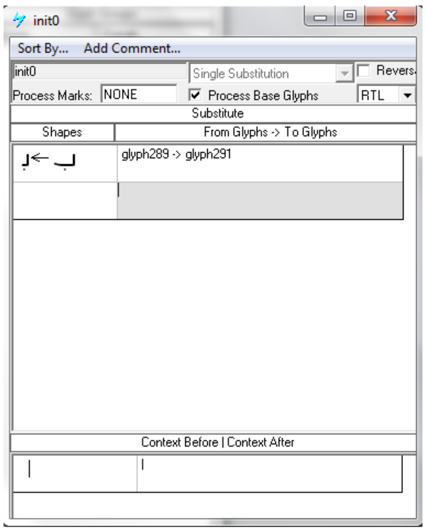
glyph294
ID: 294 Char: U+FEA4
Type: Simple

glyph295
ID: 295 Char: U+FEA8
Type: Simple

VOLT's glyph panel

There can be many *ba* in the font, the one we are looking for, is the isolated shape which has the Unicode U+0628, it is the starting point for the whole process.
In our example its name in the glyph index is glyph289, so in the first line of the 'From glyphs -> To glyphs' section of the lookup let's type glyph289. You can also drag the glyph from the glyph panel to the lookup dialog.
Then you need to type the symbol '->' (if you click outside the lookup dialog it usually is automatically written). And then type (or drag) the name of the initial form of the letter *ba*, in our case 'glyph291', then press Enter.

The first rule for the <init0> lookup



init0

Sort By... Add Comment...

init0 | Single Substitution | Revers | Process Marks: NONE | ☑ Process Base Glyphs | RTL

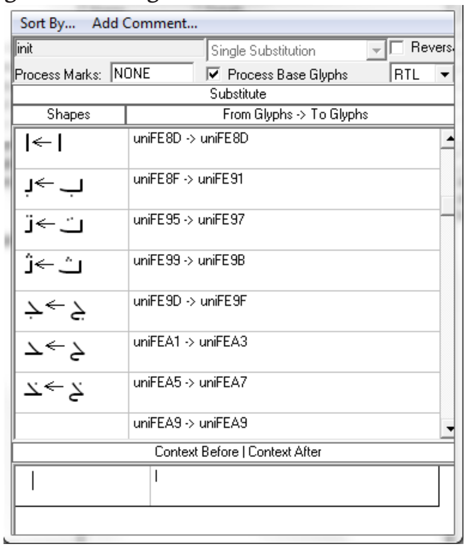
Substitute

Shapes	From Glyphs -> To Glyphs
	glyph289 -> glyph291

Context Before | Context After

If all went well you should see:
The rule we have entered roughly translates:
'When you encounter the glyph named glyph289 you replace it with the glyph glyph291'.

You just need to do the same for all the other letters. When done, you should get something like:



Sort By... Add Comment...

init | Single Substitution | Revers | Process Marks: NONE | ☑ Process Base Glyphs | RTL

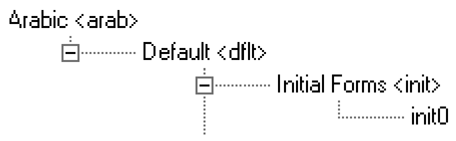
Substitute

Shapes	From Glyphs -> To Glyphs
	uniFE8D -> uniFE8D
	uniFE8F -> uniFE91
	uniFE95 -> uniFE97
	uniFE99 -> uniFE9B
	uniFE9D -> uniFE9F
	uniFEA1 -> uniFEA3
	uniFEA5 -> uniFEA7
	uniFEA9 -> uniFEA9

Context Before | Context After

The *alef* should not appear here, since it does not connect with the next letter, so it can be only be isolated or final. Yet, we must enter a 'silly' rule where the *alef* will be transformed into itself, this way we prevent potential problems with some text engines.

The final step is to link the lookup with the <init> feature; to do so close the lookup dialog and drag the name of the lookup in the middle column and drop it on the feature <init> in the left column.



The lookup will appear under the feature, like this:

To make sure everything is fine, click on the 'Compile' button, and if no error message appears, you're fine! Otherwise you need to try and figure out what the error message indicates, which is not always easy!

You should hit 'Compile' quite often just to make sure that you didn't insert an incorrect rule. Once 'Compile' has successfully run you can test your font, after saving the font and putting it into the font library of your system. In case you were to test the font after implementing only the <init> feature, you should see the beginning of words displayed correctly but the rest of the characters still displayed in the isolated form.

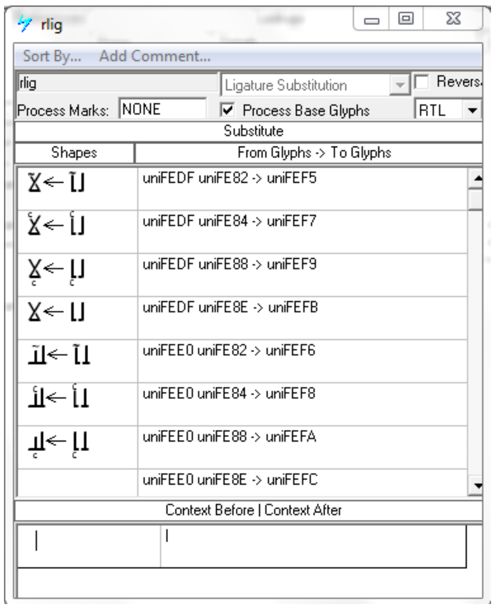
For the other positional features (<medi>, <fina>, <isol>) the same process applies ; the <isol> feature seems useless, since it transforms glyphs into themselves, but for more complicated fonts it can be useful, so you need to implement it anyhow.

All these features involve very simple substitutions (one glyph is replaced by another glyph), yet you can define more complicated substitutions (one glyph into many glyphs, many glyphs into one or more glyphs). Let's illustrate this with the feature <rlig>.
<rlig> signifies required ligatures, in Arabic there is one required ligature, the *lam-alef* (لا). Many fonts also define a required ligature for the word *Allah* (الله) since this usually has a precise graphic form.

The OpenType norm also defines the <liga> feature (standard ligature) and <dlig> feature (discretionary ligature), where you can define other ligatures, it is up to you to decide the classification.
It is important to remember that most of the text engines don't apply the <liga> or <dlig> features, only sophisticated engines (such as the one in Adobe's InDesign ME for instance) lets you choose precisely which features you want to apply to a given text.

Let’s take a look at the implementation of the <rlig> feature, the principle is very simple, you just need to define the sequence of glyphs that is transformed, your lookup should look like:

A typical rlig lookup



rlig | Ligature Substitution | ☐ Revers
Process Marks: NONE | ✓ Process Base Glyphs | RTL

Substitute

Shapes	From Glyphs -> To Glyphs
	uniFEDF uniFE82 -> uniFEF5
	uniFEDF uniFE84 -> uniFEF7
	uniFEDF uniFE88 -> uniFEF9
	uniFEDF uniFE8E -> uniFEFB
	uniFEE0 uniFE82 -> uniFEF6
	uniFEE0 uniFE84 -> uniFEF8
	uniFEE0 uniFE88 -> uniFEFA
	uniFEE0 uniFE8E -> uniFEFC

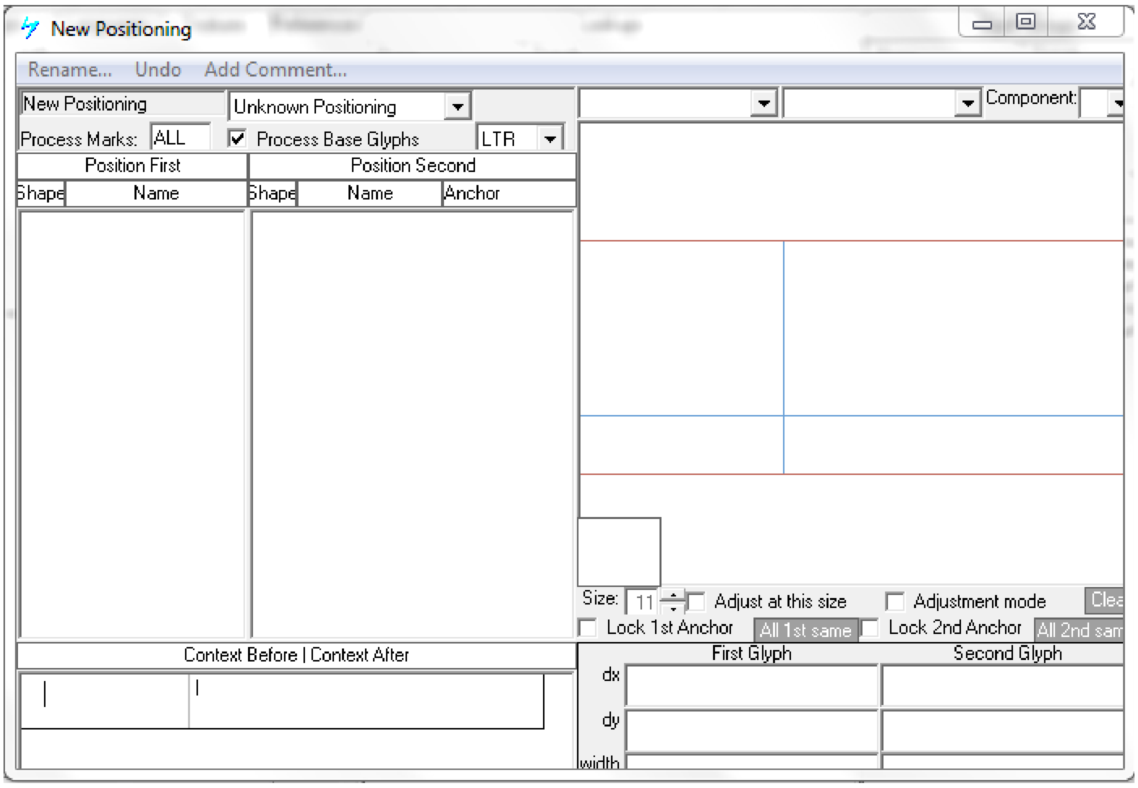
Context Before | Context After

As you can see in the lookup, you just need to indicate that an initial or a medial LAM followed by a final ALEF should be transformed into a single glyph representing the LAM-ALEF ligature.
Here we also see various possible ALEF (ALEF MADDA, ALEF HAMZA ...) if you have defined such possibilities in your font; you need to cover them all.

With this feature we have completed the substitutions. It is now time to define the positioning features.

2. Positioning
The positioning features are useful if you want your font to support the ‘harakat’ (or short vowels). For each glyph you need to define where the vowels (or other marks) should appear.
To do so, OpenType defines two features that are interesting for the Arabic script, the feature <mark> (mark positioning) and <mkmk> (mark to mark positioning).

Let’s add them to our font, and create a new positioning lookup; the positioning lookup dialog is quite different from the substitution lookup dialog, it looks like this:

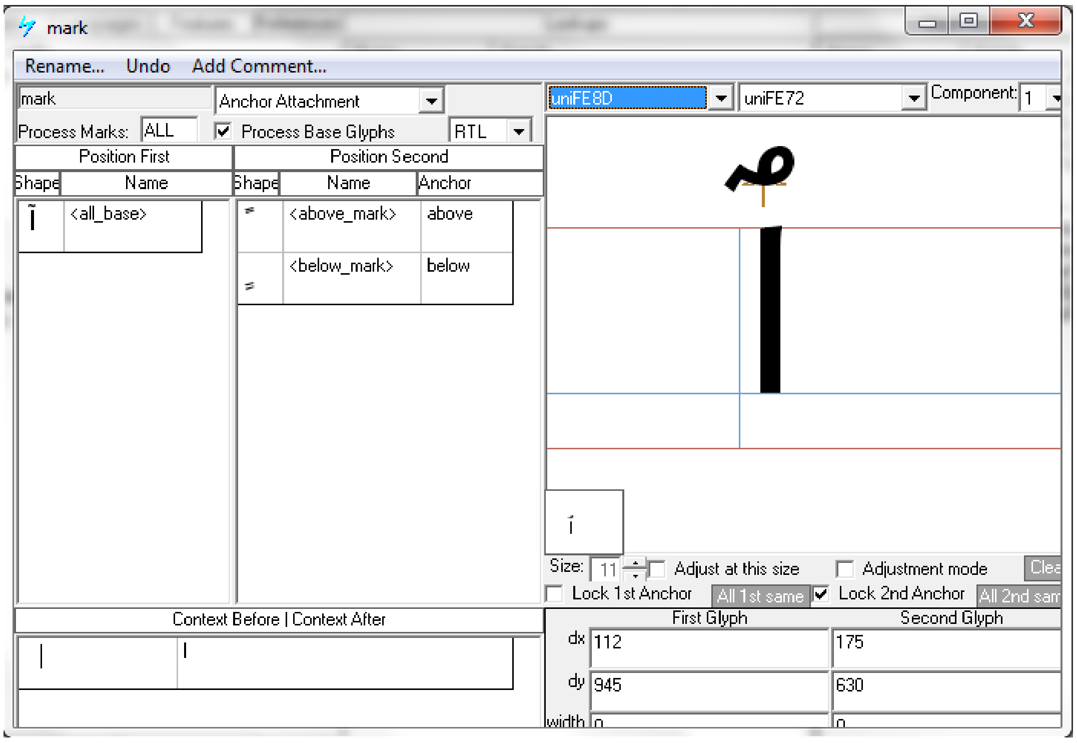


New Positioning | Unknown Positioning
Process Marks: ALL ✓ Process Base Glyphs LTR
Size: 11

As usual, we need to switch from LTR to RTL, and select the kind of positioning we want to do, the choices are:
—Unknown Positioning: default value, it has no meaning and should never be used
—Single adjustment
—Pair adjustment
—Caret positioning
—Cursive attachment
—Anchor attachment

I am not going to describe each item at this point, for a simple font the only one interesting is ‘Anchor attachment’. So let’s select this one.
Finally, the value of ‘Process Marks’ should be ‘ALL’, because we are dealing with marks here and we want them to appear.
To populate the two columns (Position First, Position Second) there are two possibilities; either you type or drag the glyphs in the columns, or you create one or more groups containing all the glyphs that we want to use.

In this sample, I have created three groups; a group called <all_base> containing all the possible forms for the base letters, and two groups containing marks, one called <above_mark> and the other <below_mark>.



mark — Anchor Attachment
Process Marks: ALL — [✓] Process Base Glyphs — RTL
uniFE8D — uniFE72 — Component: 1

Position First Name	Position Second Name	Anchor
<all_base>	<above_mark>	above
	<below_mark>	below

Size: 11 — [ ] Adjust at this size — [ ] Adjustment mode
[ ] Lock 1st Anchor — [✓] Lock 2nd Anchor

	First Glyph	Second Glyph
dx	112	175
dy	945	630
width	0	0

There are a few important things to know here;
—First, always name your anchors. Here you see in the Anchor column the names 'above' and 'below', if you don't name the anchors they will merge and you won't be able to define different positions for above marks and below marks.
—To reference a group you need to add the < > brackets around its name
—If you are working with groups you can select the various elements using the combo boxes on the upper right side of the dialog.
—You need to create 2 separate lookups for the single base letters and the ligatures. You can't mix them.

The process is quite simple, each glyph possesses one anchor, symbolized by and + sign on the right panel and you simply need to position the anchors. To help you, you can lock one anchor using the checkboxes 'Lock 1st Anchor' or 'Lock 2nd Anchor'. The best practice is to position the mark anchor first, lock it and then adjust the other anchor.
A very instructive video is available on Microsoft's Typography website http://www.microsoft.com/typography/VOLT.mspx then click on 'New VOLT Training video'.

This is it! Press the 'Compile' button, and you can now test your font.

Once you are satisfied with the result, you should create the final state of the font, with the menu entry ‘Ship Font’ in the ‘File’ menu. *Please note, the result of ‘Ship Font’ is a font which you can’t edit with VOLT, so always keep a VOLT editable version which you get using ‘Save’ or ‘Save As...’ for future update.*

CONCLUSION

We have reached the end of this tutorial; you should now have the basic knowledge to make modern Arabic fonts. If you want to go further and build more complex fonts, there are a few more features you need to be familiar with, I’ll cover them in the next tutorial that will present and explain more advanced functionalities.

www.khattbooks.com