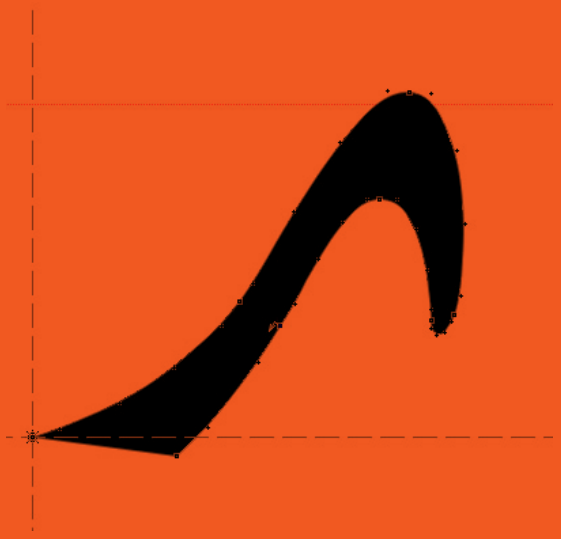
Stéphane Boeuf

Arabic Font Production Tutorial

Part II Calligraphic 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 second of a series of three. The first tutorial deals with Arabic typographic fonts and the third with Arabic web fonts.

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).

Introduction

In the first tutorial we have covered the basics of OpenType. The present tutorial is going one step further and explains more advanced OpenType features and strategies, in particular we will discuss the notions of context, alternates and an alternative approach to dots positioning. In this tutorial we assume that you are familiar with VOLT and are able to create features, lookups, etc.

Arabic Calligraphic Font Making

The kind of fonts we could create using the features described in the first part of the tutorial have many limitations. For instance, each positional form can only be represented by a single glyph. However, when trying to make more calligraphic fonts, we would need more glyph variations for a single positional form.
For example, let's consider the word بحر, the output displayed here is somehow simple, people familiar with the traditional Arabic writing styles would rather see something like this:

Calligraphic styles from left to right: Thuluth, Ruqah and Nastaliq.

The purpose of this tutorial is to explain the techniques that are required to be able to produce such calligraphic results. To do so, we need to introduce the following new notions:

1. The decomposition feature
2. The alternate feature
3. The context feature
4. The <calt> feature
5. The cursive positioning feature

1. THE DECOMPOSITION FEATURE

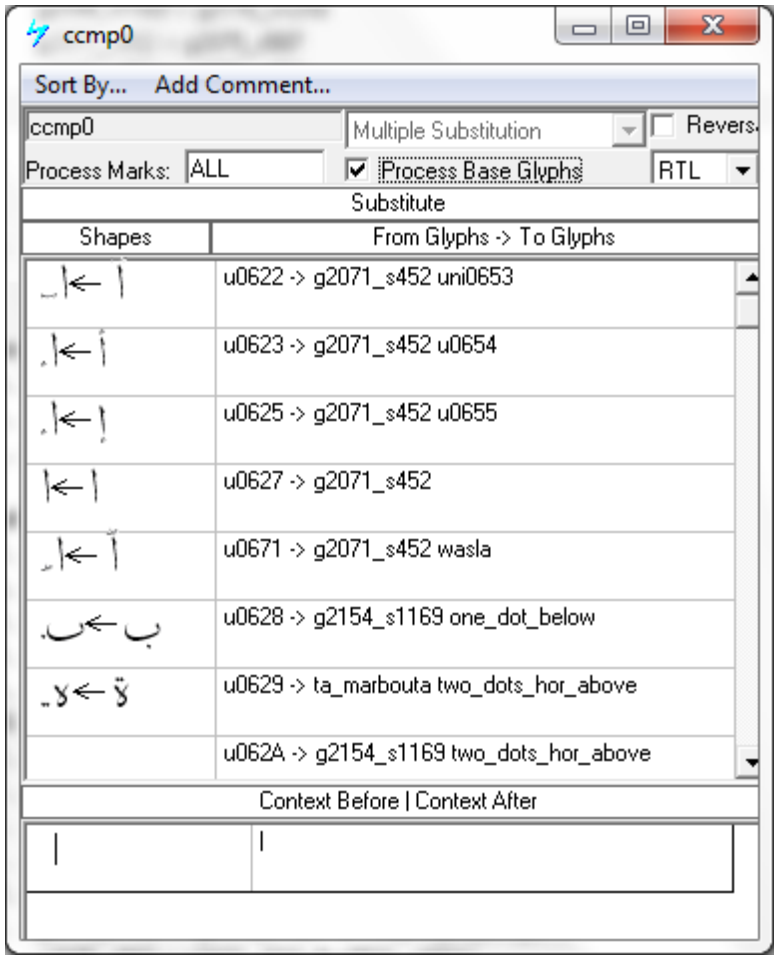
The first thing that comes to mind when looking at the samples above, is that we have to design new glyphs for these new shapes. In our sample, it would require a new glyph for the initial BA that is adapted to the connection with the medial HAH. We know that this new shape can also be used for other characters; using that same shape we can write any of the following words (existing or not, is

not important here, only the possibility of writing them is what counts):

بحر تحر ثحر نحر يحر پحر

Considering this, it makes sense to separate the dots from the basic shape. It simplifies matters and limits the number of glyphs to be created and manipulated.
To do so, you need to use an OpenType feature called <ccmp> (which stands for character composition/decomposition), this feature is the first one to be applied by the text engine, therefore you can put aside the dots very early in the process and then forget about them until the moment when you need to position them correctly (which is not always simple to do...)
<ccmp> is a substitution feature, it usually replaces one glyph into 2 (or more) glyphs, here is a sample output of a typical ccmp lookup:

A typical <ccmp> lookup. The 'dotless' glyph decompositions don't need to have an Unicode value attached to them, therefore you are free to name these new glyphs as you wish. It is best to devise a system that makes them easily identifiable. Here you see a naming convention using groups (g) and shapes (s) to structure the names of the glyphs. But you are totally free to create your own naming convention for these glyphs.



As you can see, you just need to replace the glyph representing the isolated form of each character into two separate items: a basic shape (sometimes called grapheme) and a dot or mark. Remember that at the end of the process you will need to position the dot, so be sure to separate dots appearing above from below a character even if it means duplicating the same glyph. If you don't do that you won't be able to make the difference between an initial TA and an initial YEH, because both glyphs have the same (glyph) shape. In order to position to dots correctly (above or below) you need to give each a different name although they share an identical (glyph) shape. Following this concept of font production, you just need to create 'dotless' shapes for the glyphs that normally carry one, and make separately a set of isolated dots. Finally, you need to create 'complete' glyphs (that bear the Unicode value) for the isolated glyphs, that act as starting points. But as soon as the <ccmp> feature is applied, all the original glyphs are replaced and the originals won't be used anymore in the process.

NB: It is wise to create a duplicate glyph also for glyphs that don't have dots (like ALEF for instance), in our sample the ALEF (coded u0627) is replaced by a similar glyph called (g2071_s452). The idea behind this action is to completely separate the glyphs used before <ccmp> and after.

2. THE ALTERNATE FEATURE

Now that we have put aside the dots we can focus entirely on the shapes of the glyphs. In this tutorial, we limit the variations to two (glyph) shapes for each character; the 'regular' (or standard one) and a dedicated one that is used before a JIM shape. If you want to create a complete calligraphic font, you certainly will have to create dozens of variations for each glyph shape:

Various shapes of the initial TA grapheme

In the OpenType terminology these various shapes are called alternates; OpenType defines two kinds of alternates:

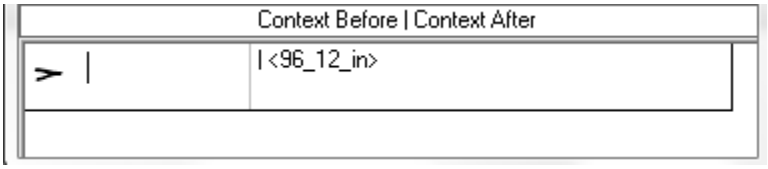
1. The stylistic alternates, where the (glyph) shape can be selected optional by the user for aesthetic reasons.
2. The contextual alternates, where the alternate shape is selected automatically depending on its context (i.e the glyphs appearing before and/or after the glyph).

The stylistic alternates (<salt> feature) are simple to use. The only constraint in the design is that they all require the same stroke connection. To make use of stylistic alternates, we need to select the menu option to make use of alternates and thereafter to select a specific glyph of the given options. Text engines do not apply this

feature automatically. Only sophisticated text engines (such as the one of InDesign ME) allow you to explicitly apply a feature on a single character. Therefore, to be able to use the <salt> features is rather discretionary.
In this tutorial we only focus on the contextual alternates and leave stylistic alternates aside. The contextual alternates require the definition of a context and since many alternates can happen, the order in which the alternates are applied can be sometimes tricky. Let's take a closer look at the context feature.

3. THE CONTEXT FEATURE

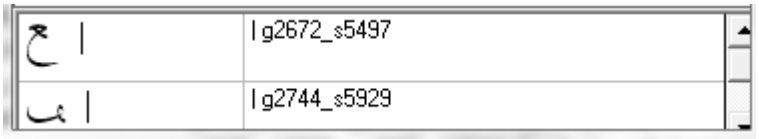
A context as defined by OpenType is a definition stating which glyph (or glyphs) may precede or follow a given glyph. To define a context in VOLT you have a designated area in each lookup:



The context area in a lookup

The vertical bar appearing in the context defines the current position, and since we are dealing with Right-To-Left lookups, what appears on the left side defines what follows and what appears on the right side defines what precedes. In the sample displayed above, the context can be translated as: 'We stand before a medial HEH shape'.

Since this font possesses many medial HEH shapes I used a group with all the similar shapes. You can define a context using a single glyph, a glyph group, many glyphs or many groups.
To limit the number of lookups you need to carefully define your contexts, making them as general as possible.



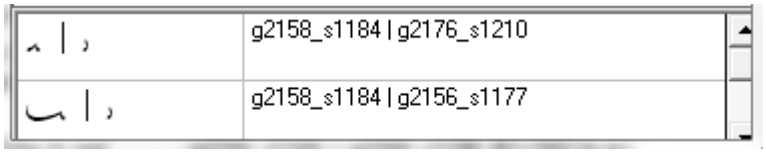
The beginning of a context made of many glyphs

You can also define contexts enclosing the current glyph, for instance the traditional way of writing the word بيت is to use a taller medial form like this:

To do so, you need to create the following context:

بيت

However, since these contexts are very precise try to avoid using them unless you absolutely need them.



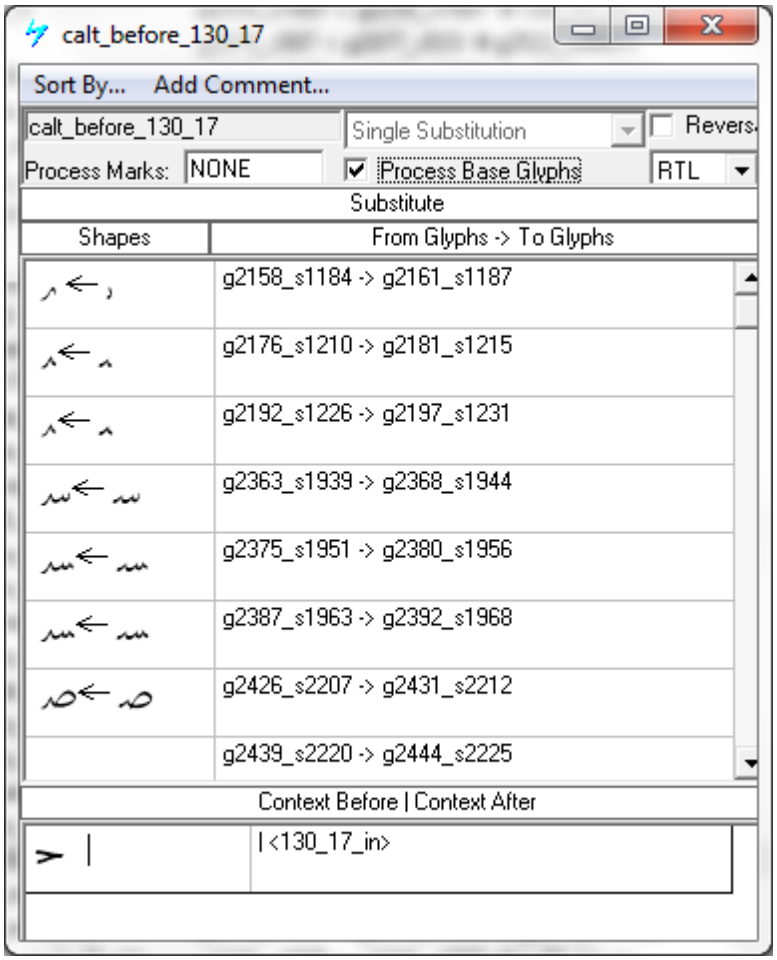
An 'enclosing' context

Now that the notion of context is clearly established (at least I hope...) let's see how we are going to use them in combination with contextual alternates.

4. THE <CALT> FEATURE

The <calt> feature (<calt> stands for contextual alternates) is the feature you need to use to create the substitutions we need for our purposes.
In the text engine the <calt> feature is called in this order: first, the decomposition (<ccmp>) feature is applied, second, the positional features (<isol>,<init>,<medi>,<fina>), and third, the required ligatures (<rlig>) are put into place.

A <calt> lookup for our sample. As you will notice, all Unicodes have disappeared from the list now. It is important to eliminate all Unicode during the first reading (call) of the font by the text engine, which happens when the <ccmp> features are applied.



The standard positional features substitute the isolated letter forms into the 'standard' positional form. What we need to do in the <calt> feature is to substitute the 'standard' positional form into the appropriate (glyph) shape depending on the context in which this glyph must appear.
In our original sample (the word بحر) we need to create the <calt> lookup appearing on the previous page.

Let's sum up what this lookup describes:
—The context states that we currently stand before a medial HEH
—The substitute describes what to do in that context (substitute the 'standard' shape with the appropriate one)
—In this sample we do that for every character and positional forms, as you can see we also substitute the SIN, SAD... and their medial shapes.

The <calt> feature is very powerful and allows many substitutions. One very important thing to remember is that the lookups are applied from the start of the word and are applied following the order of appearance in the VOLT lookup list, from top to bottom. So if you create many lookups be very careful, once you have applied a lookup, the glyphs have changed, and so all the lookups that follow must take into account these changes and anticipate on the substituted glyphs. This can become very complicated when you deal with many lookups and may require some time to sort things out properly. Test your font regularly during the process of writing <calt> lookups.
It is not uncommon for a complex font to require dozens of <calt> lookups.
But before you can test your font, there is one more feature we need to take a look at, which deals with positioning of the glyphs.

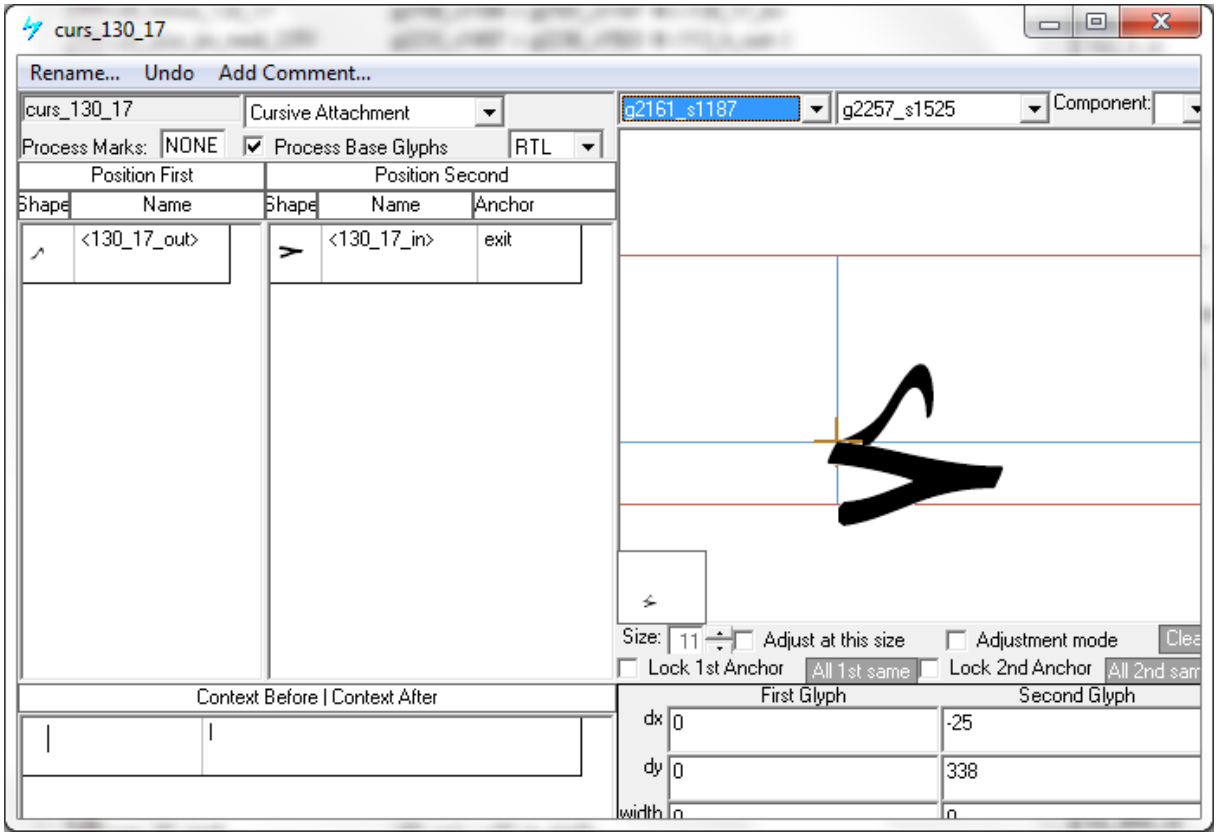
5. THE CURSIVE POSITIONING FEATURE

With the simple 'typographic' fonts, base glyph positioning is not an issue since all the glyphs have very linear connections. To position the glyphs, you just need to put them next to one another on a baseline.
However, if we consider the kind of connections that are required to write our sample word:

We notice that these connections involve very precise positioning of each individual glyph.

To achieve this, we need to introduce a new feature called <curs> (for cursive positioning). It is a positioning feature using exclusively a special kind of attachment (called 'Cursive Attachment') to describe how to position glyphs.

You should be familiar with the anchor positioning to make use of this feature, if not, please see the first tutorial. I am not going to get into the details of this lookup.



A typical <curs> lookup describing the relative positioning of glyphs.
The <curs> feature uses the internal units of the font (the EM), so no additional 'metric' information is needed.

One important thing to remember when you are using these kinds of attachment is that the font is no longer sitting on one baseline. This can create problems with the leading of text lines. For instance if we consider this silly fake word:

In this case, the line feed (leading) must be very large.
Some text engines will not handle this feature properly, so you probably have to use a sophisticated text engine (like Adobe's InDesign ME for instance) to get the best out of your font.

Conclusion

Designing with these new features, you are now able to expand the possibilities at hand and create very sophisticated fonts.
I didn't cover in this tutorial another important feature of positioning which is called kerning. The kerning (feature <kern>) lets you position non-connecting glyphs next to one another. For instance, you want to get a nice position for the combination دعيد like this:

Arriving at this level of sophistication, you need to create a very large amount of kerning lookups, contexts, and so on. I am not sure OpenType is powerful enough to manage all these related situations properly, but that is another subject. What is sure is that if you want to explore these directions, you are likely to face the limitations of the OpenType technology and you might have to use more powerful tools than VOLT, you might have to write your own tools using the Adobe Font Development Kit AFDKO for OpenType which is available from:

https://www.adobe.com/devnet/opentype/afdko.html

When we start to use AFDKO, we are leaving the subject of font development are getting into software development which is out of the scope of this tutorial.

Anyway, making use of the features described in this tutorial you can create very powerful and interesting fonts, but remember that when you start to use many alternate glyphs, the positioning of dots becomes increasingly complicated, and sometimes you would need to use contextual dot positioning to get the best position for the dots in various contexts.

Also, the finalization of a font is a very extensive work which requires a lot of time and skills, but the result is worth the work!

www.khattbooks.com