

# About JSesh 6 sign placement

*Serge Rosmorduc*

## 1 Introduction

JSesh was developed as an implementation of the *Manuel de Codage*, with the explicit goal of being as compatible as possible with the de-facto standard, which was represented by Winglyph on one hand and MacScribe on the other hand, with a strong focus on Winglyph compatibility. Actually, there is a specific converter from MacScribe format to JSesh, which is external to the current software.

In Winglyph, the user could define custom groups, which could be used for specific and precise formatting. One problem with such groups was that the group definition was not saved in the Winglyph document. A number of ligatures were also defined, but without a precise list.

JSesh approach is thus a bit convoluted:

- the first approach we used was to have a “ligature editor”, as we used to have in our tksesh software;
- however, we were reluctant to define manually each possible group, so we tried to propose a more general system;
- Mac Scribe had a powerful system, which allowed to define groups like  $\overline{\overline{R}}$ . We decided to provide such a system (the RES system was also an inspiration, even though our system is less versatile);
- both user feedback and knowledge of the needs of the Egyptological community led us to propose a free-placement system, which allowed the user to place the signs exactly where he wanted. This system is heavily used (and often overused).

## 2 Simple quadrant

For simple quadrant, the sign placement depends on the text orientation.

In this discussion, we will call “hbox” a list of hieroglyphs to be composed side by side (typically separated by “\*”), and “vbox”, a list of hboxes separated by “:”.

### 2.1 Horizontal text

- If the quadrant contains only one hbox (as in  $nw^*nw^*nw^*nw^*nw^*$ ), the elements are placed side by side, as if separated by “-”.
- if the quadrant contains more than one hbox:


- For each hbox in the quadrant (separated by “:”), the “natural width” of the hbox is computed.
- The hbox is scaled so that it doesn't exceed the *maximum quadrant width* (a parameter of the program).
- The natural width of the quadrant is then computed. If it is greater than the *maximum quadrant height*, the whole quadrant is scaled.

## 2.2 Vertical text

The principles are the same for vertical text, except the maximum quadrant width is larger, and that there is no vertical scaling (the *maximum quadrant height* is ignored).

(this part needs more details).

## 3 System for absolute positioning




Example :  `anx\R30{{0,357,51}}**G5{{194,0,97}}`

In absolute positioning, signs are separated by “\*\*”. Each sign is followed by a triplet of coordinates between {{ and }}. The first coordinate is the sign absciss, the second is the sign ordinate, and the third is the sign scale (as a percent of the sign size in the font).

Coordinates are given in a system where the top-start of the group is (0, 0), and the ordinate axis is oriented downwards. The scale is 1 unit = 1/1000 of sign A1 natural height.

If no coordinate is given, they default to {{0,0,100}}.

## 4 Advanced ligature system




In the advanced ligature system, we define two operators which combine a **single glyph** and a **group**. For instance, in , the group  is inserted in the glyph .

The idea is that the glyph will not fill the quadrant completely (else, there would be little reason for a ligature), and that some spaces in the bounding box of the glyph are empty, and can be fill with the group.

Those spaces are called “zones”, and can be defined explicitly in the SVG description of the glyph, or be automatically computed if the sign has no such explicit zone.

There are two zones: zone1 and zone2. Their exact placement depend on the sign. They are rectangles, and their extend is not limited to the limits of the original sign. When combining a group and a hieroglyph, the group will be placed in one of the two zones, and bounded by the limits of the zone. If the group is too large for the zone, it will be scaled accordingly.

The ligatured group will go somewhere in the ligature zone. But where exactly ? It can stand in the middle of the area, or stick to one of its sides. In fact, the behaviour of the layout algorithm is

not always the same. In ,  the “t” tends to fit on the bottom left of the rectangular area. In , the U36 () sign is horizontally centered, and its base rests on the bottom of the ligature zone.

The definition of each zone contains :

- its geometry, the rectangle which will contain the group
- its *gravity*, which describes the way the subgroup will be placed.

If there is only one zone, it will be used for both S1 and S2 (it's the case for )

## 4.1 Ligature group with hieroglyph construct

The construct, written  $G1^{^^}S2$  ( $G1$  is a group,  $S2$  is a single glyph), will try to insert  $G1$  in zone 1 of  $S2$ .

## 4.2 Ligature hieroglyph with group construct

The construct, written  $S1\&\&G2$  ( $G2$  is a group,  $S1$  is a single glyph), will try to insert  $G2$  in zone 2 of  $S1$ .

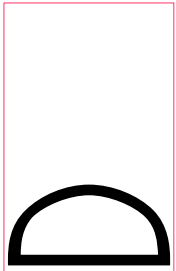
## 4.3 Zones

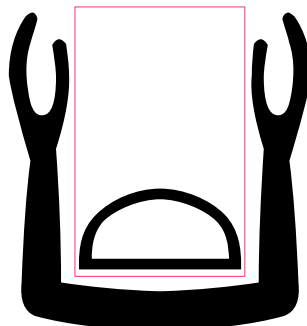
### 4.3.1 Gravity

Each zone has a gravity, which define how the group will be placed in the zone. The gravity has two components, an horizontal one and a vertical one.

where gravity specifications contain up to two letters:

- “s” or “e” to ask the group to stick to the start side or to the end side of the zone). If neither “s” nor “e” is specified, the group will be horizontally centred.
- “t” or “b” to ask the group to stick to the top or bottom of the zone. If neither “t” nor “b” is specified, the group will be vertically centred.

For example, in the following drawing, the  glyph is placed on the center bottom of the red zone.



### 4.3.2 Explicit zones

JSesh glyphs are defined as SVG pictures. In those pictures, only the black outlines are considered for the glyph drawing. Anything drawn in another colour is discarded. The sign bounding box itself is defined by the bounding box of the actual black drawing, not by the bounding box of the SVG file. It allows us to use drawings in other colours as control/metadata. It's still a trick, and not a clean way to do things. A cleaner system would be to use XML classes (or to extend the SVG language).

So, the zones are defined by adding rectangles to the SVG file. The rectangles have “zone1” or “zone2” as ids.

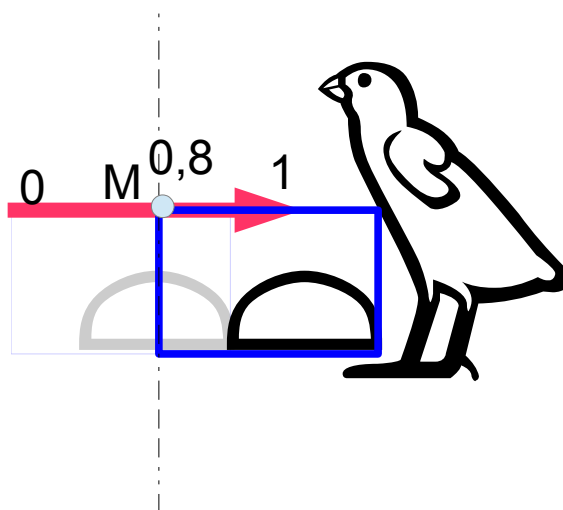
### 4.3.3 Computed zones

When no explicit zone information is available for a sign, JSesh will try to guess them. It will search in three places: in bottom/front of the sign, behind the top of the sign, and below the sign (as below ↷).

As only two zones are used, the last two possibilities can be used for zone2. If a reasonable fit for the third option is found, it prevails.

#### 4.3.3.1 Computation algorithm

To chose a zone coordinates, the algorithm will move a rectangle along a line segment, and try to move the rectangle as close as possible to the sign without touching it.



The line segment defines the possible position of the top-start point of the rectangle (in left-to-right orientation, the top-left point of the rectangle).

In the present example, the barycentric coordinate of point M is 0,8.

The size of the rectangle depends on the size of the sign, and on the zone.

The result will give a position for the rectangle (actually, a barycentric coordinate, 0 being the start of the segment, and 1 being the end of it). It is possible, in theory, that there is no result (if

all rectangles along the line overlap the sign). But, in practice, we have chosen our parameters so that there is always a solution. It is also possible in theory that there are more than one solution, but it's not a practical case.

However, the solution may give poor results (for instance, may avoid any sign overlapping). In this case, the corresponding zone is not defined. The precise conditions to keep a zone depends on the type of zone.

This algorithm is used to compute all three zones. The parameters which differ are the rectangle size, the line segment, and the rule to keep the zone.

#### 4.3.3.2 *bottom/front (zone 1)*

- rectangle  $W/2 \times H/3$
- from  $(-W/2, H/2)$  to  $(0, H/2)$

with  $W, H$  being the size of the sign. The origin point for the rectangle is top-start.

Coordinate system with  $Y$  axis going downward  $(0,0)$  being top-start coordinate.

If the result has very little overlap with the original sign (barycentric coordinate  $t < 0.2$ ), try again with a rectangle of size  $W/4 \times H/4$ .

If no good zone can be found (barycentric coordinate  $t < 0.1$ ), there is no zone 1.

#### 4.3.3.3 *top/behind (zone 2)*

- rectangle  $W/2 \times H/2.5$
- from  $(W,0)$  to  $(W/2,0)$

If  $t < 0.2$ , try with the following fallback:  $W/2 \times H/3$

If  $t$  is still lesser than 0.1, then the result is null.



#### 4.3.3.4 *below (zone 2)*



- rectangle  $W, H$
- from  $(W/2, H)$  to  $(W, 0)$

if  $t > 0.2$ , try with a different line: from  $(0, H)$  to  $(W, 0)$

## 4.4 Shortcomings of the system

The system is too rigid. Let's consider for instance those cases:

- : typically, we would like the “t” sign to be glued to the *top-right* of the  $b_3$ -sign bounding box.
- : the Z1 sign is somewhere, more or less in the middle of the space.


- : the “k” sign does take advantage of the place behind the b;bird's neck, but it does significantly overlap the sign. It's glued to the left of the box. To obtain this result, we would also need a very large box, which would give a weird result with the first group (something like ).

So, the actual position seems to depend on the group shape, a point which is currently not taken into account.

The RES system deals with this problem by proposing two primitives: insert and fit (incidentally, the hand-made kerning of HieroTeX lead to some similar solutions, see § 4.10 of HieroTeX documentation).

The last point (which is correctly dealt with by RES, for instance), is that the box system is but a simple approximation of the actual geometry of the signs. Using more precise placement primitives might allow to deal with round or irregular signs.

#### 4.4.1 Need for horizontal and vertical axis ?

This paragraph does not deal with the current system, but indeed with all existing systems. When a sign is centred relatively to another one, the actual centre is probably not the mathematical centre (although the centre of gravity might be close to it). Consider for instance the  basket. If I want to center something relatively to this sign, the correct center will probably the middle of the basket, *ignoring the handle*.

It might be reasonable to enrich signs definitions with precise notation of horizontal and vertical centers. This might also help when defining overwriting primitives (it won't work for all cases, though).

## 5 Simple Ligature

Two or more signs may be ligatured with the “&” operator. In the worst case, JSesh will currently place them all at position 0,0. A better default is needed (placing them side-by-side, or stacking them, for instance).

### 5.1 Signs placement

For a few groups, JSesh still use predefined placements for ligatures. This is the case for `stp&n&ra`. However, in most cases, JSesh tries to be clever and guess the correct way of ligaturing.

It doesn't always succeed, though.

Simple ligature are built by trying to find the “best” corresponding complex ligature. This can be done with up to three signs.

### 5.1.1 Three-signs ligature

A ligature of the type  $S1 \& S0 \& S2$  is considered as a ligature around  $S0$  (topical example :  $t \& w \& t$ ). It is rendered as  $S1 \wedge \wedge S0 \& \& S2$

### 5.1.2 Two-signs ligature

The first step is to find the “main” sign. For instance, in theory,  $t \& w$  could be resolved as  $t \wedge \wedge w$  (main sign is  $w$ ,  $t$  plays the role of the inserted group) or as  $t \& \& w$  (main sign is  $t$ ;  $w$  plays the role of the inserted group).

The idea is that the main sign is the largest one. In  $S1 \& S2$ , we consider  $S2$  as the main sign *iff*  $0.8 \text{ height}(S2) > \text{height}(S1)$ . This favours the first sign:  $S2$  will be chosen only if it is significantly larger than  $S1$ . If it's not,  $S1$  will be chosen.

The rationale for this rule is that, among the ligatures, we have for instance  $D \& i$  (and all ligatures which will fill a space in the bottom of a sign). In this category of ligatures, the first sign is the main one, and the second sign is not always smaller than the first.

Then once the main sign is found, the system is straightforward:

Let's call  $S0$  the main sign, and  $S$  the other sign.

- if there are two signs, and we have
  - $S \& S0$ : use  $S \wedge \wedge S0$
  - $S0 \& S$ : use  $S0 \& \& S$

## 6 Grammar

We give here a *simplified* extract of the grammar of JSesh MdC format. The idea is to show the possible interactions between the operators. Note that some of them can't be combined.

The weird aspect of some operators is due to JSesh history.

`horizontalList ::= horizontalListElement (“*” horizontalListElement)*`

`horizontalListElement ::=`

- `innerGroup`
- `| complexLigature`
- `| cartouche`

`innerGroup ::=`

- `ligature`
- `| hieroglyph`
- `| overwrite`

| (“ subgroup “)”

| absoluteGroup

ligature ::= hieroglyph (“&” hieroglyph)+

overwrite ::= hieroglyph “##” hieroglyph

complexLigature ::=

innerGroup “^^^” hieroglyph “&&&” innerGroup

| innerGroup “^^^” hieroglyph

| hieroglyph “&&&” innerGroup

absolutegroup ::= hieroglyph “{{“INT”, “INT”, “INT”}}” (“\*\*” hieroglyph “{{“INT”, “INT”, “INT”}}”)+