

**Title:** A system of control characters for Ancient Egyptian hieroglyphic text

**From:** Mark-Jan Nederhof & Vinodh Rajan & Johannes Lang (University of St Andrews, UK),  
Stéphane Polis & Serge Rosmorduc (Ramses Project, Université de Liege, Belgium & CNAM, Paris),  
Tonio Sebastian Richter & Ingelore Hafemann & Simon Schweitzer (Thesaurus Linguae Aegyptiae, Berlin-Brandenburgische Akademie der Wissenschaften, Berlin)

**To:** UTC

**Date:** 2016-07-25

## 1 Previous Unicode documents

We will refer to:

**L2/16-018R** “Proposal to encode three control characters for Egyptian Hieroglyphs” by Richmond & Glass

**L2/16-90** Three documents criticizing L2/16-018R, by Nederhof & Rajan, Richter & Hafemann & Schweitzer, Polis & Rosmorduc

**L2/16-104** “Observations about L2/16-90” by Richmond

**L2/16-156** “Recommendations to UTC #147 May 2016 on Script Proposals”

**L2/16-177** “A comprehensive system of control characters for Ancient Egyptian hieroglyphic text (preliminary version)” by Nederhof & Rajan & Polis & Rosmorduc & Richter & Hafemann & Schweitzer

For the motivations leading to the present proposal, see L2/16-177.













## 2 The new encoding

In this document we present a proposal on the basis of proven concepts taken from various frameworks for encoding hieroglyphic text, most notably PLOTTEXT [12, 13] and RES [6, 7], which shares some primitives with JSesh [10]. The first was used, among other things, to prepare a grammar [4], the second was used, among other things, for the St Andrews corpus [8], and the third in the Ramses Project [9].

Our starting points were:

- The encoding should rely on a small set of primitives, each of which can be precisely defined in a self-contained manner, in terms of relatively simple geometric principles, without reference to external databases of any kind, nor to any heuristics that might lead to unpredictable behaviour.
- It should be possible to implement the primitives, preferably using off-the-shelf rendering engines, to give satisfactory visual realizations for typical encodings.
- The primitives should be expressive enough to be able to (approximately) reflect relative positions of signs in a wide range of original hieroglyphic texts. Of secondary importance are reproductions of existing type-set editions, as these suffer from limitations of partly outdated printing technology.
- The primitives do *not* specify exact distances between signs nor exact scaling factors.
- The functionality of the encoding should be extensible by formats outside the realm of Unicode, to allow more precise specification of positioning and scaling. However, neither by Unicode nor by the extended formats do we intend to achieve quasi-facsimiles of original texts.

Table 1: The proposed control characters.

default glyph	code point	short name	character name
	13440	<b>HOR</b>	EGYPTIAN HIEROGLYPH HORIZONTAL
	13441	<b>VERT</b>	EGYPTIAN HIEROGLYPH VERTICAL
	13442	<b>END</b>	EGYPTIAN HIEROGLYPH END
	13443	<b>SEP</b>	EGYPTIAN HIEROGLYPH SEPARATOR
	13444	<b>JOIN</b>	EGYPTIAN HIEROGLYPH JOIN
	13445	<b>INSERT</b>	EGYPTIAN HIEROGLYPH INSERT
	13446	<b>INSERT_T_L</b>	EGYPTIAN HIEROGLYPH INSERT TOP LEFT
	13447	<b>INSERT_T_R</b>	EGYPTIAN HIEROGLYPH INSERT TOP RIGHT
	13448	<b>INSERT_B_L</b>	EGYPTIAN HIEROGLYPH INSERT BOTTOM LEFT
	13449	<b>INSERT_B_R</b>	EGYPTIAN HIEROGLYPH INSERT BOTTOM RIGHT
	1344A	<b>STACK</b>	EGYPTIAN HIEROGLYPH STACK
	?	<b>EMPTY</b>	?

A powerful encoding scheme not only relieves font developers of the permanent and unreasonable burden of having to update fonts ad nauseum with new spatial arrangements of signs, it will also avoid proliferation of the sign list by unnecessary composite signs, which would place a permanent and unreasonable burden on Unicode itself to provide frequent updates, as well as a collective burden on the Egyptological community to provide suggestions for such updates.

This proposal introduces the control characters in Table 1. They will be motivated step by step in the following sections, where we will use abbreviated names for these characters.

### 3 Linear text

In the simplest case, hieroglyphic text can consist of a series of signs one after the other, in a horizontal row. For this, no control characters are needed. The signs are separated by a default, font-defined, inter-sign distance. An example is:

Appearance	Unicode	PLOTTEXT	RES
		R8 R8 R8 V30 G43	R8-R8-R8-V30-G43

<sup>a</sup>BM EA 584 [2, p. 122]



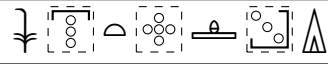
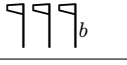
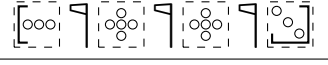

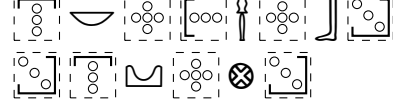
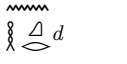
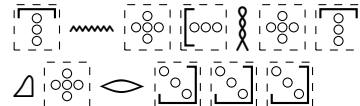



Note that the fourth sign from the left, , is less high than the height of a line, and is therefore vertically centered by default.

Table 2: Groups and boxes.




Appearance	Unicode	PLOTTEXT	RES
 <sup>a</sup>		M23/X1,R4/X8	M23-X1:R4-X8
 <sup>b</sup>		"R8 R8 R8"	R8*R8*R8
 <sup>c</sup>		V30,U23 D58/ N26,O49	V30:U23*D58-N26:O49
 <sup>d</sup>		N35,V28 "N29,D21"	N35:V28*(N29:D21)





<sup>a</sup>BM EA 571 [2, p. 77]<sup>b</sup>BM EA 585 [2, p. 48]<sup>c</sup>BM EA 587 [2, p. 46]<sup>d</sup>BM EA 1783 [2, p. 74]

As we will see later, a sign may be scaled down when it is combined with other signs in a group. The size of a sign before it is scaled down will be called its *natural size*. The natural size is measured in terms of the height of the unscaled ‘sitting man’ sign , which is called the *unit size*. In the above example, the natural height of  is 1.0, while that of  may be closer to 0.4 (in our font). Often, but not always, the height of a line is the same as the unit size.

In the example above, the text is written from left to right. Original hieroglyphic texts, however, are often written from right to left. The signs then appear mirrored, with the hieroglyphs representing living entities facing right. Text may also be written in vertical columns, either left-to-right or right-to-left. We will assume most rendering engines can only realize (hieroglyphic) text horizontally from left to right, but see Section 10.2 for further discussion.


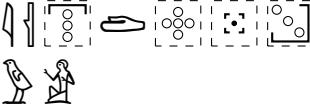

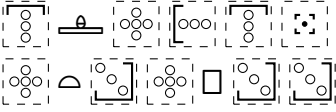
## 4 Groups

For encoding complex groups of signs, we need to be able to compose signs horizontally and vertically, at the very least. This may require repeated composition. An example is the group , which is a vertical arrangement of two groups, of which the bottom one, , is a horizontal arrangement of two groups, of which the second, , is a vertical arrangement of two signs.

We will use control characters to indicate where a horizontal or vertical group begins, and where it ends. We use different markers for the beginning of a horizontal or vertical group, namely **HOR** and **VERT** with default glyphs  and . For the end of a group, we use a common marker **END**, with default glyph . We will put a separator **SEP**, with default glyph , between the elements of a group.<sup>1</sup> Examples are listed in Table 2.


<sup>1</sup>The reason for the **SEP** is to keep closer to the traditions of Unicode. It is strictly speaking superfluous.





Table 3: Empty signs.

Appearance	Unicode	PLOTTEXT	RES
 <sup>a</sup>		M17/Aa28/”D46,”/G43/A1	M17-Aa28-D46:- G43-A1
 <sup>b</sup>		R4,”X1” Q3	R4:(.X1)*Q3


<sup>a</sup>BM EA 584 [2, p. 122]

<sup>b</sup>BM EA 581 [2, p. 59]

The simplest case of grouping is if we have a horizontal arrangement of signs in horizontal text, as in . The intended rendering of this is very similar to what we would get without grouping, except that line breaks are disallowed within a group.

We anticipate that our syntax may be somewhat controversial, as some elements should occur in pairs, such as  and , or  and . In other words, we have the structure of matching brackets. This is arguably uncommon in Unicode, and requires some justification. See further Section 9.

## 5 Empty

It is convenient to have an **EMPTY** sign  with zero width and height. Placing this sign above or below another sign effectively pushes the latter sign down or up, respectively. Examples are listed in Table 3.

We leave open the question whether **EMPTY** can be an existing empty character from Unicode, or whether a dedicated character for hieroglyphic encoding is appropriate. We suspect that if we do not explicitly introduce the possibility of using **EMPTY** in hieroglyphic encoding here, then the use will sneak in some time in the future, using this or using another existing empty character, as encoders will frequently want to flush the position of a sign to the top or to the bottom of a line, to describe the appearance on original manuscripts.

## 6 Insertion

A fair number of signs have empty space in one of the corners of their bounding box. Often this empty space is used for placement of a smaller sign, especially, but not exclusively, if the two signs are in a special relationship, for example, if the two signs together are the writing of (a part of) a morpheme or a direct genitive. The empty space may also be occupied by several signs. Our encoding includes a number of primitives for such a composition of signs.










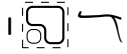

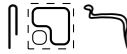

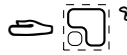



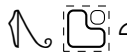









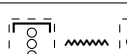


In our syntax, the center element is the ‘main’ sign in which smaller signs are inserted. To its left is, optionally, a sign or group to be inserted in the top-left corner followed by , then, optionally, a sign or group to be inserted in the bottom-left corner followed by . To the right of the main sign is, optionally,  followed by a sign or group to be inserted in the upper-right corner, and optionally,  followed by a

Table 4: Insertions.

Appearance	Unicode	PLOTTEXT	RES
 <sup>a</sup>		D60;/X1/;	insert(D60,X1)
		F4;X1//;	insert[ts](F4,X1)
 <sup>b</sup>		F20;Z1/;	insert[b](F20,Z1)
 <sup>c</sup>		I10;S29/;	insert[b](I10,S29)
 <sup>d</sup>		I10;D46/;	insert[s](I10,D46)
 <sup>e</sup>		I10;X1,N17;	insert[bs](I10,X1:N17)
 <sup>f</sup>		D17;/X1;	insert[t](D17,X1)
 <sup>g</sup>		G39;/N5;	insert[te](G39,N5)
 <sup>h</sup>		G17;/%B4+D36;	insert[te](G17*,D36)
 <sup>i</sup>		A17;/X1;	insert[be](A17,X1)
		G39;X1/;;/N21;	insert[te](insert[s](G39,X1),N21)
 <sup>i</sup>		N35,X1, "V28 E6;/X1;"	N35:X1:V28*insert[te](E6,X1)

<sup>a</sup>BM EA 143 [2, p. 110], Meir I, pl. 9 [2, p. 45]<sup>b</sup>BM EA 581 [2, p. 59]<sup>c</sup>BM EA 1783 [2, p. 74]<sup>d</sup>BM EA 581 [2, p. 59]<sup>e</sup>BM EA 101 [2, p. 58]<sup>f</sup>URK IV,373,12<sup>g</sup>BM EA 117 [2, p. 31]<sup>h</sup>P. Turin Cat. 2070<sup>i</sup>Karnak (KRI II,226,6)

sign or group to be inserted in the bottom-right corner.

There is also evidence suggesting use of an insertion inside another sign. Here the main sign is followed by  and the sign to be inserted. Such insertion is particularly common in the writing of *w<sup>eb</sup>t* ('priestess', 'pure thing', etc.) as ; the inserted sign is the feminine ending, and an analysis of the group as an atomic sign would be highly unsatisfactory.



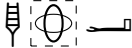

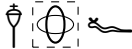

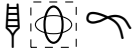

Examples are listed in Table 4. In some cases, the inserted sign or group fits entirely within the bounding box of the main sign, possibly after scaling it down. There are cases however where the inserted sign or group may spill over to outside the bounding box, as in the case of . It is for the font to decide whether the inserted group is small enough to fit within the bounding box, possibly after some scaling down, or whether



Table 5: Stacking.

Appearance	Unicode	PLOTTEXT	RES
 <sup>a</sup>		P6=D36	stack(P6,D36)
 <sup>b</sup>		U34=I9	stack(U34,I9)
 <sup>c</sup>		P6=V12	stack(P6,V12)

<sup>a</sup>BM EA 581 [2, p. 59]<sup>b</sup>BM EA 584 [2, p. 122]<sup>c</sup>[11, p. 758]




it should extend to beyond the bounding box. In PLOTTEXT and RES, additional control characters would be required to artificially extend the bounding box, whereas in our Unicode encoding, we sacrifice precision for ease of use.

The inserted groups may be arbitrarily complex. For example in  we have a vertical group within a horizontal group within a vertical group, which is inserted into another sign.<sup>2</sup> Note that the inserted group here spills over to below the bounding box.

Relative to PLOTTEXT, our notation is simplified in that ‘insert just above the feet of a bird’ and ‘insert into the lower-left corner’ have been merged. This sacrifice of precision for simplicity seems justifiable. Relative to RES, our notation is simplified in that we have only five insertion primitives, rather than nine. The functionality of the extra four in RES, i.e. ‘insert into the right/left/bottom/top side’, can partly be taken over by the **JOIN**, to be discussed in Section 8. Further, the appearances of for example  and , encoded using ‘insert into the left side’ and ‘insert into the bottom-left corner’ in RES, could be confused without causing significant problems for typical users.













JSesh has two primitives for insertion, each corresponding to one of up to two rectangular *zones* per sign. These primitives are used as  $G^{\wedge\wedge}S$  and  $S\&\&G$ , respectively, where  $S$  is a sign and  $G$  is a group. In the first case,  $G$  is inserted in zone 1 of  $S$  and in the second case,  $G$  is inserted in zone 2 of  $S$ . The zones can be defined in the font or can be computed automatically through heuristics. Typically, zone 1 is at the bottom or in front of a sign and zone 2 is at the top of or behind a sign. The zones may extend beyond the bounding box. Moreover, a zone of a sign is associated with a *gravity*, which indicates towards which of the four sides of the rectangle the inserted group is to be flushed. If a sign has two zones, the two insertions may be combined in the form of  $G1^{\wedge\wedge}S\&\&G2$ .

## 7 Stacking

Sometimes two signs or groups are superimposed. Table 5 presents examples. The first two happen to also exist as individual Unicode characters, while the third is not part of any established sign list as far as we know. There are also many examples of whole groups being stacked, such as , which is the stacking of horizontal group  and vertical group . In JSesh, stacking of two signs is expressed using binary operator  $\#\#$ .

<sup>2</sup>Stela Cairo, JE 60539, l. 8

Table 6: Stacking as compositional operation.

Stacking	Attested alternatives	Transliteration
 <i>a</i>		<i>h<sup>c</sup></i>
 <i>b</i>		<i>h<sup>cc</sup></i>
 <i>c</i>		<i>h<sup>c</sup></i>
 <i>d</i>	 <i>e</i>	<i>cbb</i>
 <i>f</i>	 <i>g</i>	<i>b<sup>c</sup>b<sup>c</sup></i>
 <i>h</i>		<i>bš</i>
 <i>i</i>	 <i>j</i>	<i>b<sub>t</sub></i>

<sup>a</sup>WB III p. 40

<sup>b</sup>Dendera VII, 148.4

<sup>c</sup>Stacked form and non-stacked alternative: WB III p. 40

<sup>d</sup>ASAE 43, p. 254

<sup>e</sup>WB I p. 178

<sup>f</sup>BIFAO 43, p. 118 and WB I p. 447

<sup>g</sup>WB I p. 446




<sup>h</sup>Stacked form and non-stacked alternative: WB I p. 477

<sup>i</sup>MIFAO 16, p. 49

<sup>j</sup>Both non-stacked alternatives: WB I p. 485

It cannot be emphasized enough that stacking is part of how the Ancient Egyptian writing system works. Much like horizontal and vertical grouping and insertion, it was one of the mechanisms the ancient scribes had to their disposal to position signs relative to one another. In other words, stacking is to a large extent productive. The fact that some frameworks in the past (most notably the Manuel de Codage [1]) resorted to introducing separate codepoints for stacked sign combinations, even for those that are hapax, may be blamed on shortcomings of the used technology more than anything else.


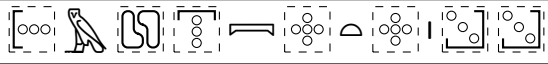

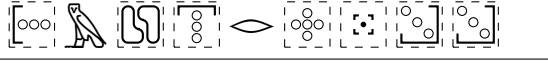





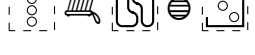
Another selection from the many thousands of known stacked signs is given in Figure 6. Some have attested non-stacked alternatives, while for others we cannot immediately verify whether non-stacked alternatives might have existed. In the overwhelming majority of cases, the meaning of the stacked signs is completely compositional. For example, a stacked sign may represent a sequence of phonemes, each of which corresponds to one of the constituent signs.

One may naively object that stacked signs are not entirely compositional, because they not only represent the constituent signs themselves, but also the order in whether these are to be read. This objection is weakened, if not invalidated altogether, by the many known cases where in fact all conceivable orders are valid, as long as an existing word is written. For example,  may be used both in words starting with *c<sub>b</sub>*, where the non-stacked alternative  may be used, and in words starting with *b<sup>c</sup>*, where the non-stacked alternative  may be used.<sup>3</sup>

We foresee that the UTC will raise objections against generic stacking, as some existing signs among the

<sup>3</sup>See WB I p. 173-178 and p. 446-450, respectively.

Table 7: Joining.

Appearance	Unicode	RES
 <i>a</i>		G17-[fit]N1:X1:Z1
 <i>b</i>		G17-[fit]D21:.
 <i>c</i>		G43-[fit]D46:.*O49
 <i>d</i>		U23*N26*[fit]D58
 <i>e</i>		F39:[fit]Aa1

<sup>a</sup>BM EA 581 [2, p. 59]<sup>b</sup>BM EA 584 [2, p. 122]<sup>c</sup>BM EA 585 [2, p. 48]<sup>d</sup>BM EA 143 [2, p. 110]<sup>e</sup>BM EA 587 [2, p. 46]

1071 hieroglyphic signs currently in Unicode are stacked combinations of constituent signs, and some may argue that there is an issue with compatibility. Our rebuttal is three-fold:

- Mistakes from the past should not dictate that more mistakes ought to be made in the future.
- A central principle is that we should have reasonable confidence that we can encode a text not seen before. Unless we are willing to sacrifice this principle, and thereby the practical value of the encoding as a whole, then generic stacking is a necessity.
- We understand there is a set procedure within the Unicode framework to deal with such compatibility issues, namely by describing existing characters in terms of a combining character plus constituent characters. In the case of stacking, this is clearly the only viable way forward.

## 8 Joining



A natural consequence of the tendency to make efficient and esthetically pleasing use of available space was to squeeze groups together. It would be highly undesirable to have a rendering engine do this indiscriminately for all groups. Therefore, we introduce **JOIN**, with default glyph , which can be used instead of  to indicate that neighbouring signs of groups may, but need not, move towards each other, to have their bounding boxes overlap. The signs should preferably not touch each other however.

Table 7 presents examples. The second example is similar, but not quite identical, to an insertion in the top-right corner. (As a rule, insertions try to scale down inserted groups before they spill over to outside the bounding box, which is not what happens here. Admittedly, there are no absolute criteria when to use **JOIN** and when to use insertion.)

## 9 Brackets or infix operators



As for the justification for our use of brackets, let us first recognize that there is true recursion in the structure of groups of signs, and Ancient Egyptian may be one of the very few writing systems with this property.




Concretely, we can have a vertical group containing a horizontal group containing a vertical group, etc. The deeper the nesting, the more rare such groups become, but we see no obvious reason why a nesting of say 4 or 5 levels deep could not be found if one looked hard enough. It seems quite reasonable that some implementations (fonts) could only handle up to a bounded depth, but it does not seem warranted to put a hard restriction on the depth by choosing a syntax that is inherently limited.


Secondly, it seems likely that any notation for cartouches (enclosures) will involve pairs of matching brackets; see also Section 10.1. So we have brackets one way or another. (One way out would be to distinguish between a cartouche enclosing one single group, and one enclosing several groups. For the former, a prefix or suffix operator would be used. For the latter, infix operators could be used between each pair of consecutive groups within the cartouche.)

Thirdly, the alternative to having matching brackets is to have infix operators with varying operator precedence. Note however that we need several operators (for horizontal groups, for vertical groups, for insertion, stacking, etc.) Because of the discussed recursion, we might in fact need different versions of each operator depending on the intended operator precedence. This might easily lead to many levels of operator precedence, even if we restricted the depth of recursion.

For example, a group such as  suggests that an infix operator for vertical grouping should have a lower binding value than that for insertion.<sup>4</sup> But now consider again . There would be two infix operators for vertical grouping for the inserted group, with different binding values, both higher than that of the insertion operator.

Concretely, let us introduce levels of operator precedence, indicated by numbers 0, 1, 2, . . . . Each operator can occur with a certain level, which we indicate between brackets after operator occurrences. For any given level, there is fixed precedence between stacking, insertion, horizontal grouping and vertical grouping. Any operator occurring with a higher level has higher operator precedence than any operator occurring with a lower level. For  we could have:

$$\ominus \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(0)} \quad \text{---} \quad \text{---} \quad \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(0)} \quad \text{---}$$

if we assume that insertion has higher binding value than the infix operator  $\left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]$  for vertical grouping, if occurring with the same level, whereas for  we could have:



$$\ominus \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(2)} \quad \triangleleft \quad \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(1)} \quad \text{---} \quad \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(1)} \quad \text{---} \quad \left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]_{(0)} \quad \text{---}$$

where we assume that the infix operator  $\left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]$  for horizontal grouping has higher precedence than  $\left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]$ , if occurring with the same level. Note that  $\left[ \begin{array}{c} \text{---} \\ \text{---} \end{array} \right]$  must be able to occur in at least three different levels. If we extend this to the whole system, it means we need at least three times as many control characters as there are primitives. So it is possible to have only infix operators instead of brackets (apart from cartouches, which have inherently bracket-like behaviour), but the costs of swapping brackets for infix operators should be understood. Furthermore, if we underestimate the needed number of levels, and add too few to Unicode, we will run into difficulties in the future. Note that if a font handles too few levels of nested brackets, one can refine the font. If there are too few levels of operator precedence, new control characters need to be added to Unicode.

Further note that the encoding is already a lot more complicated than that in L2/16-177, which used prefix operators. We were told that these were not acceptable, and we should use infix operators. As a result, ad hoc restrictions had to be introduced to avoid ambiguity; see further Appendix A.

<sup>4</sup>BM EA 581.

Table 8: Enclosures.

Appearance	Unicode	PLOTTEXT	RES
 <sub>a</sub>		%Z1 N5 L1 D28 %Z2	cartouche(N5-L1-D28)



<sup>a</sup>BM EA 586 [2, p. 25]





## 10 Secondary issues


The following are on our wish list, but should not distract from the main issues.

### 10.1 Cartouches and other enclosures

It may be noted that the existing hieroglyphs in Unicode include individual symbols for starting and ending parts of several kinds of ‘enclosures’, but [3] writes “Plain text and general purpose software should likewise treat these signs as characters and not render the fully enclosed form.” As this is the only authoritative source we have been able to find that tells us anything at all about intended purpose and use of hieroglyphs in Unicode, we infer we cannot reuse these characters for representing actual, full-form cartouches, serekhs, etc.

Furthermore, cartouches in hieratic are written as two isolated starting and ending parts. It seems natural to reserve  and  for representing these, and to introduce additional primitives to denote full-form cartouches. These primitives may differ between one protocol and another, so it is a pair < and > in JSesh, a pair %Z1 and %Z2 in PLOTTEXT, and `cartouche( )` in RES,

For Unicode we propose an open cartouche sign  (or open serekh or open ‘castle walls’), followed by groups separated by  or , and closed by . An example is given in Table 8.

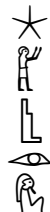


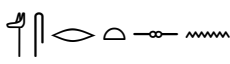






An alternative notation would be a special use of the **INSERT** , consisting of a sign for a cartouche (there is such a sign in the existing Unicode repertoire already, but not for serekh or ‘castle walls’), followed by **INSERT**, followed by a horizontal group (in horizontal text) or a vertical group (in vertical text). The disadvantage is that automatic conversion from vertical text to horizontal text or vice versa becomes more difficult, as in vertical text a cartouche can be unboundedly high, whereas in horizontal text a cartouche can be unboundedly wide. A restructuring between vertical and horizontal grouping would therefore be required for the conversion to give satisfactory results. The question of how to best encode cartouches is thereby connected to Section 10.2.

### 10.2 Changes in text direction

In the simplest case, hieroglyphic text consists of a sequence of signs that are reasonably wide and high. The signs can then be placed next to one another for horizontal text directions and underneath one another for vertical text directions. However, two tall narrow signs would typically be put next to one another and two wide thin signs would typically be put above one another. Note however that top-level horizontal groups are strictly speaking redundant in our encoding of horizontal text, and so are top-level vertical groups in vertical text, and as a result we may miss appropriate groupings if we convert between text directions.

These problems are partially avoided in PLOTTEXT by allowing the user to omit groupings of signs, leaving it to the application to find suitable groupings based on the dimensions of the individual signs. In this proposal, we have assumed that we do need to specify groupings in the encoding, relieving the font

Table 9: Change of text direction from vertical to horizontal imposed by the application (all examples from BM EA 101 [2, p. 58])

Original text	Allowable rendering	Better rendering
		
		
		
		

and rendering engines from this difficult task. However, we wish to keep open the possibility that adequate rearrangements of groups are made automatically by the application in case it imposes a change of text direction relative to the original manuscript. This however requires that the original text direction is or can be encoded.

Some examples are given in Table 9. In the first, the vertical text is best changed to horizontal text by simply stringing signs together horizontally. In the second example however, a more pleasing appearance is achieved by introducing some vertical groups. In the third example, there is a **JOIN** between the two signs that was meant to apply to the vertical direction only, but there is no reason to believe the **JOIN** would be appropriate for horizontal direction as well so there it should be ignored. In the fourth example, the group is exceptionally high for horizontal text, and becomes quite small if scaled down to fit within a row of height 1, and a thorough restructuring would be desirable.

It would be advantageous therefore to be able to encode text direction. This could be done by, optionally, putting one from the set of four newly introduced characters  $\begin{bmatrix} \rightarrow \\ \leftarrow \\ \downarrow \\ \uparrow \end{bmatrix}$  at the beginning of a fragment of hieroglyphic. This does *not* dictate the direction of rendering, but it informs the application for which direction the encoding is meant.

## 11 Rendering

Here we discuss the ideal scaling and positioning of signs within groups. Practical implementations may deviate from this ideal due to technical limitations; see Appendix B.

### 11.1 Horizontal and vertical groups

What is described here is consistent with both JSesh and RES. Formatting of groups is done in two steps. First, we determine how much signs need to be scaled down (signs are never scaled up) to fit two main

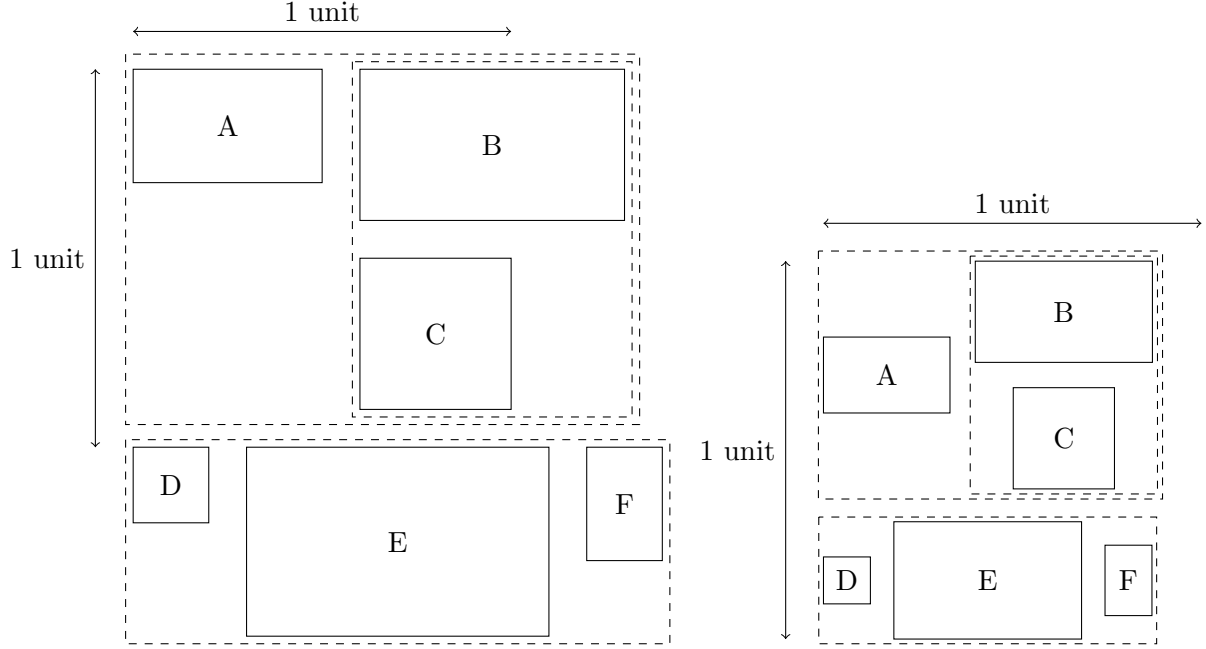


Figure 1: A nested group, before scaling and positioning (left) and after (right).

constraints. Second, we insert additional whitespace to center and align signs and groups.

For the first step, that of scaling down, we consider inner-most groups before considering enclosing groups. A first constraint is that a vertical or horizontal subgroup within an enclosing group, together with the default inter-sign distance, should not be higher or wider, respectively, than 1 (in terms of the unit size). We illustrate this using Figure 1. Here, the natural size of the signs B and C plus the default inter-sign distance add up to a height smaller than 1. Therefore B and C by themselves need not be scaled down. However, they form a horizontal group with A (which is enclosed in another vertical group), of which the natural width exceeds 1. Therefore, A, B and C are all scaled down uniformly, to make that width exactly 1. Similarly, D, E and F need to be scaled down to make their added width exactly 1. A second constraint is that a group within a line of horizontal text does not exceed the height of that line, which is normally 1. This may require further uniform scaling down of all signs and their inter-sign distances.

If we have a group with a similar structure but with signs of different sizes, the following would happen, with  $w$  for the natural width,  $h$  for the natural height, and  $sep$  for the default inter-sign distance.

- If  $h(B) + sep + h(C) > 1$ , then determine scaling  $f_1$  such that  $f_1 \cdot (h(B) + sep + h(C)) = 1$ ; otherwise let  $f_1 = 1$ .
- If  $w(A) + sep + \max(f_1 \cdot w(B), f_1 \cdot w(C)) > 1$ , then determine  $f_2$  such that  $f_2 \cdot (w(A) + sep + \max(f_1 \cdot w(B), f_1 \cdot w(C))) = 1$ ; otherwise let  $f_2 = 1$ .
- If  $w(D) + sep + w(E) + sep + w(F) > 1$ , then determine scaling  $f_3$  such that  $f_3 \cdot (w(D) + sep + w(E) + sep + w(F)) = 1$ ; otherwise let  $f_3 = 1$ .
- If the text is written horizontally in rows, with line height 1, then in the same vein we compute  $f_4$  to make the whole group fit within the line.

For the second step, we distribute ‘excess whitespace’ equally over subgroups. We need to distinguish between two cases, namely subgroups consisting of a single sign, and subgroups consisting of several recursive

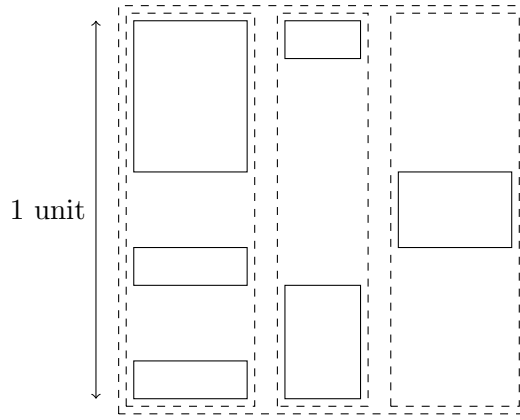


Figure 2: In this horizontal group, there is excess whitespace in all three vertical subgroups. In the rightmost subgroup, there is only a single sign, which is centered. In the leftmost two subgroups, the excess whitespace is divided equally between the (recursive) subgroups (which here happen to be three and two single signs, respectively; they could have been nested horizontal groups as well).

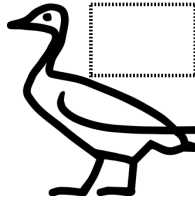





Figure 3: The rectangle that can be designated for the second group after an occurrence of **INSERT\_T\_R**, if the main sign is the duck.

subgroups. In the first case the single sign is centered within the available space, and in the second case, the excess whitespace is divided equally between the subgroups. This is illustrated in Figure 2.




## 11.2 Insertion

For the implementation of   , the sun sign is placed in the upper right corner of the bounding box of the duck, with the top-most and right-most points of the sun flushed against the bounding box. The sun is ideally as large as possible (but not bigger than the natural size), while keeping some distance (ideally the default inter-sign distance) away from the duck.

A less ideal, but still acceptable, rendering results if we precompile tables indicating for each sign where inserted groups are to be placed. For example, the table might indicate the rectangle as in Figure 3, to define how the duck is to be combined with another group if we use **INSERT\_T\_R**. This is similar to how insertions in JSesh are implemented. Note that this does not place any restrictions on the groups that can be inserted.

## 11.3 Stacking

The rendering of **STACK** followed by two groups lets (roughly) the centers of the two groups coincide. The rendering is simply the addition of the curves of the two constituent groups.



For some sign combinations, a more satisfactory realization may result if not the exact centers of the groups, but points a little distance away from the centers are chosen to coincide. For example, in  the center of  coincides with a point a little to the right of the center of . This can be realized by letting a font assign an *anchor* to a sign, which defines a ‘conceptual’ center, different from the center of the bounding box. (Cf. the notion of *anchor* in OpenType.) It may also be realized in the font through substitutions of entire stacked groups by optimized glyphs.

## 12 Outside Unicode

Because Unicode has its limitations, extended formats with additional functionality are needed for advanced purposes. It is highly desirable that Unicode and the extended formats can be converted seamlessly one to the other. In particular, converting Unicode to extended formats should be a matter of converting syntax only, and converting extended formats to Unicode should be a matter of systematically removing functionality that is unavailable in Unicode, while retaining an acceptable rendering. We mention RES in particular as a format whose functionality has the proposed Unicode system as a strict subset. We also address absolute positioning and scaling in JSesh.



### 12.1 Scaling







RES allows tweaking of the natural size of a sign, before the processing of group structure. This can be helpful to improve the rendering, but it is not essential to obtain an acceptable rendering.

Scaling in RES may also be applied on only the height or only the width of a sign. For example, a sign such as  may be manually flattened to become , which gives a more satisfactory result if it occurs in a vertical group together with more signs, and a similar distortion of the shape may be witnessed in original inscriptions.<sup>5</sup> The two shapes shown here in fact exist as separate codepoints in Unicode, which should definitely be avoided for future extensions of the sign list. A font may well include flattened graphical variants, which it may prefer over the original shape depending on context, but there should be only one codepoint per sign.

The **EMPTY** of this proposal is a special case, with zero width and height, of the ‘empty’ symbol in RES, which can be parameterized with arbitrary width and height. It is a useful auxiliary symbol to influence placement of neighboring signs.

### 12.2 Rotation

Some rotation of signs has semantic significance. For example,  is a logogram for the object depicted, namely a mace, while the tilted form , suggesting a mace being applied, is a determinative in words such as “smite”. These two signs have two distinct codepoints in the existing Unicode set, and this is entirely justifiable.



There are other cases however, such as , , , , that are pure graphical variants. The choice between any particular inclination and orientation was probably made by a scribe on a whim, partially depending on how well it would fit within the graphical context of other signs. The same holds for a number of thin signs that are used in lying or upright form, such as  and .

<sup>5</sup>Cf. pp. 4-5 of [5].

The fact that all these are currently separate codepoints in Unicode is difficult to justify other than by pragmatic considerations: rotation is difficult to realize using off-the-shelf font technology. For this reason we abstain from proposing generic rotation in Unicode at this time. We would make a note however that once font technology has evolved further, introduction of rotation as a primitive in the encoding of hieroglyphic text should definitely be considered.



Of course, RES and JSesh possess primitives for rotation by arbitrary angles.



### 12.3 Mirroring

Some mirroring of signs has semantic significance. For example, the sign depicting forward walking legs  is used as determinative in words involving (forward) movement, while the mirrored sign  is used as determinative in words involving backward movement. Having separate codepoints for a sign and its mirrored counterpart is fully justifiable in such cases.

However, some occurrences of mirroring were motivated by other considerations, as for example, symmetry of signs within groups. In such cases, a mirrored sign should be seen as a different graphical realization of the same sign, and does not deserve a separate codepoint. Moreover, such mirroring is not very common and for many applications the mirroring can be ignored. We therefore will not argue at this time in favour of including generic mirroring in Unicode. Extended formats, such as RES, do include a primitive for mirroring.

### 12.4 Groups

Rendering of groups can be fine-tuned in several ways. For example, one may override the default inter-sign distance to obtain  rather than . In RES this is done by `R8-[sep=0]R8-[sep=0]R8`. It is definitely not justifiable to have a separate code point for a combination of signs with particular inter-sign distances, even when that distance is 0.

As explained in Section 11.1, the width of a horizontal subgroup is reduced to 1 and the height of a vertical subgroup is reduced to 1, before the scaling of the enclosing group is considered. In some cases,  this is undesirable. For example, in , the top-most two signs should appear with the same scaling as the bottom sign. This is achieved in RES by changing the above value 1 to something else, or by avoiding prior scaling down of the horizontal group altogether, by using `inf` in `D28*D28:[size=inf]D28`.




In some later periods of Ancient Egyptian history, lines tended to be much higher than 1 unit, and this greatly affects layout of signs, due to the interaction between the natural sizes of signs and the line height. In RES, the line height (and the column width) can be adjusted.

None of the above seem essential for Unicode and will not be proposed at this time.

### 12.5 Insertion

RES allows fine-tuning of the  $x$  and  $y$  positions of a group that is inserted into another, as well as fine-tuning of the minimum distance between the two groups. If the distance is chosen to be 0, then the second group may touch the first.

### 12.6 Stacking

In RES, the stacking primitive can be extended with functionality to let one group erase the underlying curves of the other group. For example, we may obtain `stack[x=0.4](S12, D58)` , `stack[x=0.4,under](S12, D58)` , or `stack[x=0.4,on](S12, D58)` . We know of no examples

where this difference in appearance has semantic significance, and would therefore not consider any corresponding primitive for inclusion in Unicode.

As also illustrated by the above examples, RES allows fine-tuning of the relative positions of stacked signs. For Unicode, we rely on the font to choose suitable positioning, as explained in Section 11.3.

## 12.7 Modify

RES includes a primitive that replaces the physical bounding box by a virtual bounding box. This is a powerful operation that can in rare cases be useful to give complex groups a more pleasing realization than would otherwise be possible. It can also be used to let a part of a sign be rendered outside a line of text. Additionally, one may erase parts of a sign. For many applications however the intricacies of the modify primitive do not outweigh its benefits, and consequently we are not considering adoption of an analogue for Unicode.

## 12.8 Absolute positioning and scaling

There is a strong demand from the Egyptological community for rendering exact appearances from original texts, mainly for publication purposes. JSesh therefore allows expression of absolute positioning and scaling. For example, `S34\R30{{0,357,51}}**G5{{194,0,97}}` expresses that sign S34 is to be rotated by 30 degrees, scaled by factor 0.51, and placed at  $(x, y)$  coordinate (0.0, 0.357), while G5 is scaled by factor 0.97 and placed at coordinate (0.194, 0.0); coordinates refer to the top-left corners of bounding boxes of signs. In this syntax, `**` connects a number of signs together that are formatted by absolute scaling and positioning relative to the same reference point (0.0, 0.0). If the triple is absent, it defaults to `{{0,0,100}}`.

The disadvantage of absolute scaling and positioning is that it makes the encoding dependent on the exact shapes and natural sizes of signs, in other words on the font, which complicates exchange of encodings between tools. Absolute scaling and positioning is therefore best avoided, unless it is essential to the application.

## 12.9 Shading

In our domain it is the rule rather than the exception that manuscripts are only partially legible, as the passing of several millennia has left few textual artefacts completely unscathed. It is important to let an encoding of a text express that identities of certain damaged signs are merely hypothesized, in order to avoid incorrect interpretations. JSesh, PLOTTEXT and RES all allow shading (also called hatching) of signs or parts of signs with varying precision, to indicate damage to the text. RES is the most flexible of the three, allowing the bounding box of a sign or inter-sign distance to be divided into smaller rectangles, through repeated partitioning into two equal parts. For each such smaller rectangle, shading can be individually set.

Our current proposal does not include shading, for two reasons:

- We are unsure about a satisfactory syntax to express shading as part of plain-text encoding.
- We anticipate that the UTC will want to delegate shading to a higher-level protocol.

However, because shading is very important to our domain, we intend to revisit this matter in the near future.

## 13 Conclusions

The Ancient Egyptian writing system is not simple by any stretch of the imagination. A satisfactory encoding will therefore necessarily involve a few non-trivial elements. Moreover, Ancient Egyptian writing



is very different from other writing systems, which makes implementation using existing rendering engines challenging. Nonetheless, proof-of-concept exists for critical parts of the proposed encoding.

## Acknowledgements

We thank our kind host Nigel Strudwick for inviting Egyptologists and Unicode experts to Cambridge for a meeting on encoding issues. The present version of this proposal could not have been produced without it. We thank all those present at the meeting for their generous help and advice. The suggestion to reduce the number of insertion primitives and to move towards infix notation we owe to Michael Everson.

We further wish to thank Andrew Glass for fruitful correspondence about the Universal Shaping Engine, and his considerable efforts to explore the feasibility of the insertion primitives in OpenType.

## References

- [1] J. Buurman, N. Grimal, M. Hainsworth, J. Hallof, and D. van der Plas. *Inventaire des signes hiéroglyphiques en vue de leur saisie informatique*. Institut de France, Paris, 1988.
- [2] M. Collier and B. Manley. *How to read Egyptian hieroglyphs: A step-by-step guide to teach yourself*. British Museum Press, 1998.
- [3] M. Everson and B. Richmond. Proposal to encode Egyptian hieroglyphs in the SMP of the UCS. Working Group Document ISO/IEC JTC1/SC2/WG2 N3237, International Organization for Standardization, 2007.
- [4] E. Graefe. *Mittelägyptische Grammatik für Anfänger*. Harrassowitz Verlag, Wiesbaden, 1994.
- [5] J. Moje. *Untersuchungen zur Hieroglyphischen Paläographie und Klassifizierung der Privatstelen der 19. Dynastie*. Harrassowitz Verlag, 2007.
- [6] M.-J. Nederhof. A revised encoding scheme for hieroglyphic. In *Proceedings of the 14th Table Ronde Informatique et Égyptologie*, July 2002. On CD-ROM.
- [7] M.-J. Nederhof. The Manuel de Codage encoding of hieroglyphs impedes development of corpora. In S. Polis and J. Winand, editors, *Texts, Languages & Information Technology in Egyptology*, pages 103–110. Presses Universitaires de Liège, 2013.
- [8] M.-J. Nederhof. ORC of handwritten transcriptions of Ancient Egyptian hieroglyphic text. In *Altertumswissenschaften in a Digital Age: Egyptology, Papyrology and beyond*, Leipzig, 2015.
- [9] S. Polis, A.-C. Honnay, and J. Winand. Building an annotated corpus of Late Egyptian. In S. Polis and J. Winand, editors, *Texts, Languages & Information Technology in Egyptology*, pages 25–44. Presses Universitaires de Liège, 2013.
- [10] S. Rosmorduc. JSesh Hieroglyphic Editor. <http://jseshdoc.qenherkhopeshef.org>, 2016.
- [11] K. Sethe. *Urkunden der 18. Dynastie, Volume I*. Hinrichs, Leipzig, 1927.
- [12] N. Stief. Hieroglyphen, Koptisch, Umschrift, u.a. – ein Textausgabesystem. *Göttinger Miszellen*, 86:37–44, 1985.
- [13] N. Stief. PLOTTEXT. <https://www.hrz.uni-bonn.de/rechner-und-software/pc-anwendungen/textverarbeitung/plottext-1>, 2003.

## A Structure of hieroglyphic encoding

For a sequence of signs and control characters to have their intended meanings, it should comply with the following (Backus-Naur) specification. Lower-case non-bold names are classes. Bold-face names represent characters, with upper-case boldface names representing particular characters, and **sign** representing any hieroglyph. The pipe symbol | separates alternatives. Square brackets [ ] indicate optional elements, round brackets followed by an asterisk ( )<sup>\*</sup> indicate repetition zero or more times, and round brackets followed by a plus symbol ( )<sup>+</sup> indicate repetition one or more times

The following states that a fragment of hieroglyphic text consists of one or more groups, and that a general group may be a basic group, a horizontal group or a vertical group. We need to have two special kinds of groups to avoid ambiguity.

```
fragment ::= ( group )+
group ::= basic_group | horizontal_group | vertical_group
group1 ::= basic_group1 | horizontal_group | vertical_group
group2 ::= basic_group2 | horizontal_group | vertical_group
```

The following states that horizontal and vertical groups have a left marker **HOR** or **VERT**, respectively, and a right marker **END**. These enclose two or more subgroups. A subgroup of a horizontal group may not be another horizontal group and a subgroup of a vertical group may not be another vertical group.

```
horizontal_group ::= HOR hor_subgroup ( separator hor_subgroup )+ END
hor_subgroup ::= basic_group | vertical_group
vertical_group ::= VERT vert_subgroup ( separator vert_subgroup )+ END
vert_subgroup ::= basic_group | horizontal_group
separator ::= SEP | JOIN
```

A basic group is empty, or it is a core group, surrounded by optional insertions. We need to have two special kinds of basic groups to avoid ambiguity in the case of an occurrence of **INSERT\_T\_L** followed by one or more occurrences of **INSERT\_B\_L**; the **INSERT\_T\_L** then applies to the same core\_group as the last of the **INSERT\_B\_L**. The situation for **INSERT\_T\_R** and **INSERT\_B\_R** is symmetric. A core group is a sign by itself, or is a stacking of two or more signs or horizontal or vertical groups.

```
basic_group ::= EMPTY |
               [ group INSERT_T_L ] [ group1 INSERT_B_L ] core_group
               [ INSERT_T_R group2 ] [ INSERT_B_R group ] |
               core_group INSERT group
basic_group1 ::= EMPTY |
                [ group1 INSERT_B_L ] core_group
                [ INSERT_T_R group2 ] [ INSERT_B_R group ] |
                core_group INSERT group
basic_group2 ::= EMPTY |
                [ group INSERT_T_L ] [ group1 INSERT_B_L ] core_group
                [ INSERT_T_R group2 ] |
                core_group INSERT group
core_group ::= sign | core_subgroup ( STACK core_subgroup )+
core_subgroup ::= sign | horizontal_group | vertical_group
```

We have tried to keep this specification as simple as possible. Practical implementations may restrict themselves to a subset of the fragments that are possible in principle. In particular, if a core\_subgroup is a horizontal\_group or vertical\_group, then one may assume the subgroups therein are all individual signs. One may further restrict the total depth of nesting of groups to be no more than 3 or 4.

## B Implementation

### B.1 RES

The encoding that this document proposes is a functional subset of RES, which has been implemented in C, Java and JavaScript, with the ideal formatting described in Section 11. There is a graphical editor at:

<https://mjn.host.cs.st-andrews.ac.uk/egyptian/res/js/edit.html>

which allows experimentation with the JavaScript implementation. The existence of these implementations shows that the functionality of the proposed encoding can be realized. The differences in syntax are inessential.















### B.2 Realization in OpenType

In this section, we aim to investigate to what extent OpenType is able to handle horizontal and vertical groups without enumerating groups.

We have prepared a small font with a handful of hieroglyphs, by adding OpenType substitution rules, divided over a large number of ‘lookups’ of the ‘liga’ feature, to format a few sample groups of hieroglyphs, attempting to approximate the ideal rendering discussed in Section 11. As it soon became clear to us that writing these substitution rules by hand would be close to impossible, we wrote a Python script to generate these rules.

Our architecture uses three passes. In a first pass, substitution rules insert ‘records’ between signs and control characters. Each record consists of a sequence of auxiliary symbols, which are empty glyphs with zero width. The auxiliary symbols serve to help analyze a group. Most auxiliary symbols initially represent ‘null’ values, indicating that various counts, widths, and heights, etc., have not yet been determined. In the second and third passes, contextual substitution rules are then repeatedly applied to fill in the missing values, copied from neighbouring records to the left or to the right, possibly in combination with operations such as addition and maximization (which have to be realized in terms of precompiled substitution rules for finite sets of values).

More precisely, in the second pass, which is right-to-left, the natural width, natural height and number of subgroups in each group are computed. In the third pass, which is left-to-right, appropriate scaling factors and positions are propagated, and signs are scaled and positioned appropriately. Scaling is realized by having scaled versions of each sign in the font (again for a finite number of values); an appropriate ratio between scalings could be  $\sqrt{2}$ , to follow typographical tradition.

Figure 4 depicts the records involved in the analysis of the group                

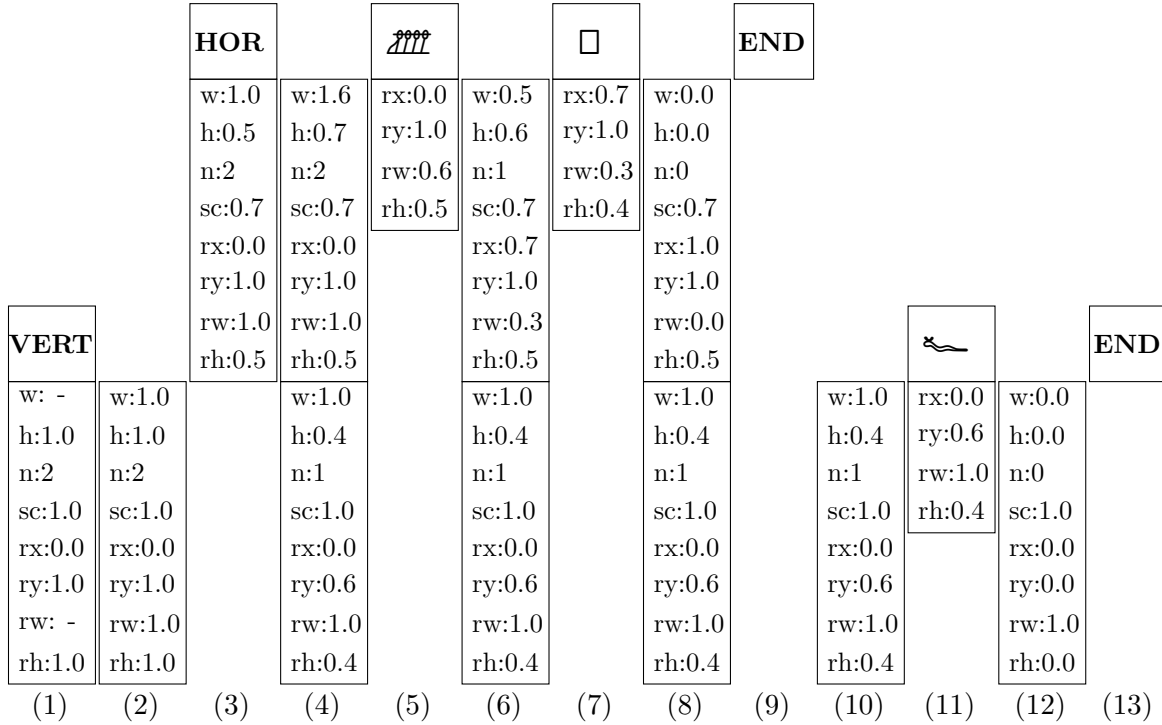


Figure 4: Record keeping in our partial OpenType implementation of formatting of groups.

The left-to-right analysis starts in (1), with the height of the line being 1.0, but without constraints on the width of the group. As the total height of the vertical group is exactly 1.0, no further scaling down is necessary. The four values rx, ry, rw and rh represent the position of the upper left corner of the rectangle in which the group needs to be rendered and its width and height. The values are passed on and updated from left to right. Once more, the values in the bottom halves in (4), (6) and (8) are copied unchanged, so they are available again in (10).

This partial OpenType implementation is extremely inelegant and moreover we run into the problem that many tools do not support lookup offsets exceeding 2 bytes, which stands in the way of scaling this up to a full implementation for the full sign list. Nonetheless, we can draw a few tentative conclusions:

- A general OpenType implementation seems in principle possible, in a way that avoids exponential complexity in the number of signs and control characters per group.
- We can extend this architecture to insertions (cf. the running text relevant to Figure 3 above), by assigning different values for rx, ry, rw and rh at the beginning of an inserted group.

### B.3 The fall-back option

Another option is to let a font contain precomposed groups. Note however that the number of such groups should be less than 64K in order to implement the font in OpenType. It is unclear at this time how many different groups exist in big corpora.

One may consider various intermediate implementations, where fonts know how to realize certain groupings with certain combinations of sign dimensions (rather than combinations of signs), making use of classes.