

Title: A comprehensive system of control characters for Ancient Egyptian hieroglyphic text (preliminary version)

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: 2016-06-30

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”

2 Introduction

Few would disagree with the following observation:

An encoding scheme for a script only makes the least bit of sense if there is reasonable hope one can encode a text not seen before.

Because of the peculiarities of Ancient Egyptian writing, it may not be obvious to non-specialists why this is a problem. To explain, let us imagine an encoding scheme for English that would require a special provision, either in the encoding scheme itself or in the font, for every individual compound noun, so for ‘ice-cream’, for ‘ice-cream truck’, for ‘ice-cream truck driver’, etc. It would be foolish to assume we could simply make a long list of all compound nouns, extracted from say 1000 English texts, and then hope that text number 1001 can be encoded as well. This is because it is almost certain any new text will have a compound noun not seen before.

Ancient Egyptian hieroglyphic writing was highly variable, not only in the sign repertoire and in the choices of hieroglyphs for writing words, but also in the spatial layout of hieroglyphs, which is inherently 2-dimensional. This means, among other things, that there is no such thing as ‘the’ graphical order, as at first sight a group may be read top-to-bottom or left-to-right or even a mixture of the two (not to mention transposition, where the reading order is inverted). One can only disambiguate using context and knowledge of the writing system and language. Another implication is that it makes no sense whatsoever to try to collect ‘the’ list of groups of hieroglyphs with particular layouts, just as it makes no sense to collect ‘the’ compound nouns of English. If one has created provisions for the spatial arrangements of groups in the first 1000 Egyptian texts, then this is certainly inadequate to handle text number 1001. One may do the same for 1,000,000 texts, but this is still inadequate to handle text number 1,000,001. The only reasonable solution of course is to devise an encoding that includes primitives powerful enough to approximately describe spatial layout of hieroglyphic text, which is what we will turn to starting in Section 4.

One may object that unfettered use of ideographic description characters for CJK ideographs has been found to be problematic, as it allows formation of arbitrary nonsense characters and complicates data

exchange and search. This concern is entirely irrelevant to Ancient Egyptian however. A typical CJK text is unlikely to include unknown characters, while a typical Ancient Egyptian text is unlikely to have spatial arrangements of signs that are exactly like in some other Ancient Egyptian text. Furthermore, any esthetically pleasing spatial arrangement is in principle possible.

3 Background

Our document was written as a response to L2/16-018R, which proposed three control characters for Ancient Egyptian hieroglyphic text. One of these was called LIGATURE JOINER. What is here referred to as a ‘ligature’ is a combination of signs (i.e. hieroglyphs) with a relative positioning that is not purely horizontal or purely vertical.

By its very nature the LIGATURE JOINER cannot be defined within Unicode, but its purpose would become clear through a vague notion described as “guidance [to be] evolved within the Egyptology community” (L2/16-104, p. 2), assuming a near-exhaustive collection of “attested forms” (L2/16-018R, p. 2).

The fatal flaw of the approach is that existence of a near-exhaustive collection of ligatures is and will remain a chimera. This is because formation of ligatures is highly productive. There is no reason to believe any combination of two, three, four or five signs could not in principle occur together in a ligature in any esthetically pleasing spatial arrangement. This is supported by empirical evidence that on average more than 2 out of 3 ligatures that occur in the comprehensive corpus of the Ramses project [8] occur only once. This means that it is almost certain that any new text to be encoded contains at least one hitherto unseen ligature, and a rendering engine built on these principles would then be unable to handle this ligature until it is appropriately reconfigured to do so. Such an encoding scheme would obviously be useless for any serious application. See also L2/16-090 and L2/16-156.

L2/16-104 claims that the LIGATURE JOINER was not intended for ‘monograms’, i.e. superimposed signs, and that monograms should be listed in a font. Yet there is an example of a monogram specified using the LIGATURE JOINER in L2/16-018R, viz. no. 4 on p. 4. Such internal inconsistencies do not bode well for the prospect that “guidance [to be] evolved within the Egyptology community” might spontaneously lead to rigorous consistency in the use of the LIGATURE JOINER. Moreover, because monograms are also productive to a large extent, compilation of an exhaustive list of monograms is as impossible as compilation of an exhaustive list of ligatures.

The signatories of the present document include representatives from *Thesaurus Linguae Aegyptiae* (TLA) and from the Ramses project. The TLA is with 1.4 million tokens the largest electronic corpus of Egyptian full text data. The Ramses Project is working on a richly annotated corpus of Late Egyptian texts, with 530,000 lemmatized tokens so far, annotated for parts of speech, inflection, and crucially in this context, the hieroglyphic spelling (68,000 different spellings). These projects provide us with unique experiences with encodings of hieroglyphic texts. These experiences lead us to unequivocally reject the LIGATURE JOINER from L2/16-018R. Not only would an encoding based on such a control character be useless for serious applications, but the LIGATURE JOINER would also create novel problems that did not exist before, due to the absence of a proper definition, which will cause confusion, lead to mistakes, and make it very hard if not impossible to use the encoding as input or output format for Egyptological databases.

4 The new encoding

In this document we present a proposal very different from the above, on the basis of proven concepts taken from various frameworks for encoding hieroglyphic text, most notably PLOTTEXT [11, 12] and RES [5, 6], which shares some primitives with JSesh [9]. The first was used, among other things, to prepare a grammar

[3], the second was used, among other things, for the St Andrews corpus [7], and the third in the Ramses Project [8].

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.

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.


It should be understood that Ancient Egyptian writing poses an exceptional challenge to computer processing. It is therefore unrealistic to expect that any satisfactory encoding of hieroglyphic text in Unicode would be simple.

5 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

^aBM 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.


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

Table 1: The proposed control characters.


































default glyph	code point	short name	character name
	13440	HLR	EGYPTIAN HIEROGLYPH HLR
	13441	HRL	EGYPTIAN HIEROGLYPH HRL
	13442	VLR	EGYPTIAN HIEROGLYPH VLR
	13443	VRL	EGYPTIAN HIEROGLYPH VRL
	13444	JOIN	EGYPTIAN HIEROGLYPH JOIN
	13445	EMPTY	EGYPTIAN HIEROGLYPH EMPTY
	13446	HOR	EGYPTIAN HIEROGLYPH HORIZONTAL
	13447	VERT	EGYPTIAN HIEROGLYPH VERTICAL
	13448	CARTOUCHE	EGYPTIAN HIEROGLYPH CARTOUCHE
	13449	OVAL	EGYPTIAN HIEROGLYPH OVAL
	1344A	SEREKH	EGYPTIAN HIEROGLYPH SEREKH
	1344B	INB	EGYPTIAN HIEROGLYPH INB
	1344C	RECTANGLE	EGYPTIAN HIEROGLYPH RECTANGLE
	1344D	HWT	EGYPTIAN HIEROGLYPH HWT
	1344E	END	EGYPTIAN HIEROGLYPH END
	1344F	INSERT	EGYPTIAN HIEROGLYPH INSERT
	13450	INSERT_T	EGYPTIAN HIEROGLYPH INSERT TOP
	13451	INSERT_B	EGYPTIAN HIEROGLYPH INSERT BOTTOM
	13452	INSERT_S	EGYPTIAN HIEROGLYPH INSERT START
	13453	INSERT_E	EGYPTIAN HIEROGLYPH INSERT END
	13454	INSERT_T_S	EGYPTIAN HIEROGLYPH INSERT TOP START
	13455	INSERT_T_E	EGYPTIAN HIEROGLYPH INSERT TOP END
	13456	INSERT_B_S	EGYPTIAN HIEROGLYPH INSERT BOTTOM START
	13457	INSERT_B_E	EGYPTIAN HIEROGLYPH INSERT BOTTOM END
	13458	STACK	EGYPTIAN HIEROGLYPH STACK

Table 2: Linear sequences of signs.

Appearance	Unicode	PLOTTEXT	RES
 ^a		%DHR R8/R8/R8/V30/G43	[hrl]R8-R8-R8-V30-G43
		%DVR N14/A30/Q1/D4/A40	[vrl]N14-A30-Q1-D4-A40
		%DVL N14/A30/F13/N31/X1	[vrl]N14-A30-F13-N31-X1

^aBM EA 584 [2, p. 122]^bBM EA 101 [2, p. 58]^cBM EA 101 [2, p. 58]

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. Even if many rendering engines can only realize (hieroglyphic) text from left to right, a plausible encoding should have the possibility to express the text direction, in terms of a control character at the beginning of a sequence of hieroglyphic signs. Such a control character may be omitted however; in many situations we do not know, or do not care about, the text direction in the original manuscript.

Examples are listed in Table 2. Note that in vertical text the signs are centered horizontally.

We foresee the UTC will raise objections against the control characters that indicate text direction, on the basis that it is the application that should determine the actual printed text direction. We do not disagree with this principle, which is in fact fully consistent with past and present practices in Egyptology. For example, interlinear text showing a comparison of different versions of the same text is commonly normalized to be horizontal from left to right, regardless of the text direction of the various manuscripts. However, there are two important reasons why it is essential to be able to encode the text direction of the original manuscript:

- Knowing the original text direction could explain the particular formation of groups of signs, and help disambiguate the reading order.
- When the application chooses a different text direction from the original, in particular vertical versus horizontal, the **JOIN** character between top-level groups should be ignored, and the rendering engine should group individual signs together differently to improve the appearance. This will be discussed in Section 10.4.

6 Groups and boxes

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 $\overset{\sim}{\lambda} \triangle$, which is a vertical arrangement of two groups, of which the bottom one, $\lambda \triangle$, is a horizontal arrangement of two groups, of which the second, \triangle , is a vertical arrangement of two signs.

Another reason why grouping is necessary is because of cartouches, which are an essential element in the writing of kings' names. A cartouche is a shape that encloses more signs, as for example $\textcircled{\text{ⓂⓂ}}$. We will use the general term *box* for a sign that encloses more signs. Next to cartouches, boxes also include serekhs, fortress walls ('inb'), and several more.

The above observations imply that for any plausible encoding of hieroglyphic text, we need, at the very least, to be able to indicate:

- where a horizontal or vertical group begins, and where it ends, and
- where a box 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 $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$ and $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, and for a box. For the end of a group or box, we use a common marker **END**, with default glyph $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$. Examples are listed in Table 3.

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. The same holds for a vertical group in vertical text. Note that a cartouche has a different orientation for vertical text.

It is convenient to have an **EMPTY** sign $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$ 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 4.

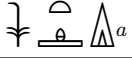
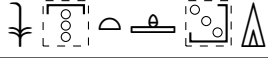

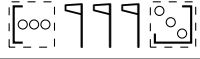

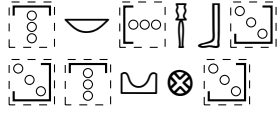

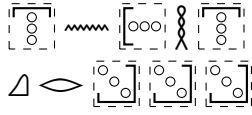

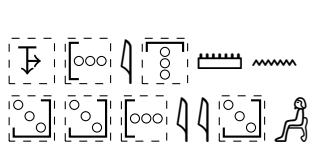



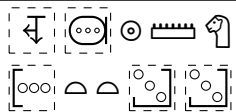

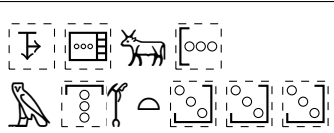


If the UTC advises us to do so, we can replace the proposed **EMPTY** by an existing empty character in Unicode. Note however that our **EMPTY** has a particular behaviour as placeholder for a sign within a group, with both zero width *and* zero height, while it preserves the default inter-sign distance between it and neighbouring signs. If our **EMPTY** is replaced by an existing character from Unicode, then it should have, or should be allowed to take these properties if used within groups of hieroglyphs.

7 Insertion

A fair number of signs have empty space on one of the sides or 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.

First, we have the primitive **INSERT**, with default glyph $\textcircled{\text{---}}$. It is followed by two groups, the second is to be inserted within the first. This only makes sense if there is empty space at the center of the first group. Then, we have eight more primitives **INSERT_T** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_B** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_S** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_E** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_T_S** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_T_E** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_B_S** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$, **INSERT_B_E** $\begin{array}{|c|} \hline \text{---} \\ \hline \text{---} \\ \hline \end{array}$. Here **T** and **B**

Table 3: Groups and boxes.


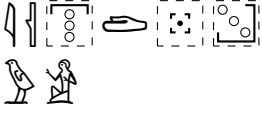

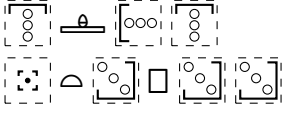
Appearance	Unicode	PLOTTEXT	RES
 ^a		M23/X1,R4/X8	M23-X1:R4-X8
 ^b		"R8 R8 R8"	R8*R8*R8
 ^c		V30,U23 D58/ N26,O49	V30:U23*D58-N26:O49
 ^d		N35,V28 "N29,D21"	N35:V28*(N29:D21)
 ^e		%DVL M17 Y5,N35/ M17 M17/A50	[vlr]M17*(Y5:N35)- M17*M17-A50
 ^f		%Z1 N5 L1 D28 %Z2	cartouche(N5-L1-D28)
 ^g		%DVR %Z1v N5 Y5 F9 "X1 X1" %Z2v	[vrl]cartouche(N5-Y5- F9-X1*X1)
 ^h		%DVL %Z3v E1/G17 "R19,X1" %O33	[vlr]serekh(E1- G17*(R19:X1))
 ⁱ		%Z3 S12 %O6	Hwtcloseunder(S12)

^aBM EA 571 [2, p. 77]^bBM EA 585 [2, p. 48]^cBM EA 587 [2, p. 46]^dBM EA 1783 [2, p. 74]^eBM EA 162 [2, p. 125]^fBM EA 586 [2, p. 25]^gBM EA 117 [2, p. 31]^hCairo Museum CG 34002 [10, p. 26]ⁱ[10, p. 53]

stand for ‘top’ and ‘bottom’, and **S** and **E** stand for ‘start’ and ‘end’; we deliberately avoid using ‘left’ and ‘right’ because that would be confusing for right-to-left text, in which signs and groups are mirrored. In the typical use of these nine primitives, the first group is an individual sign, while the second group can consist of several signs.

Examples are listed in Table 5. In these cases, we assume that the smaller sign fits entirely within the bounding box, possibly after scaling it down. There are cases however where the visual appearance is better described by extending the bounding box with extra whitespace. That is realized by forming a horizontal or vertical group with **EMPTY**. The amount of extra whitespace is then determined by the default inter-sign distance. An example is:




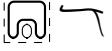

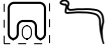



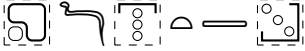






Table 4: Empty signs.

Appearance	Unicode	PLOTTEXT	RES
 ^a		M17/Aa28/"D46,"/G43/A1	M17-Aa28-D46:.- G43-A1
 ^b		R4," ,X1" Q3	R4:(.X1)*Q3

^aBM EA 584 [2, p. 122]

^bBM EA 581 [2, p. 59]

Table 5: Insertions.

Appearance	Unicode	PLOTTEXT	RES
 ^a		D60;/X1/;	insert(D60,X1)
 ^b		-	insert[b](F20,Z1)
 ^c		-	insert[b](I10,S29)
 ^d		I10;D46/;	insert[s](I10,D46)
 ^e		I10;X1,N17;	insert[bs](I10,X1:N17)
 ^f		G39;/N5;	insert[te](G39,N5)
 ^g		A17;/X1;	insert[be](A17,X1)
		G39;X1/;/N21;	insert[te](insert[s](G39,X1),N21)

^aBM EA 143 [2, p. 110]

^bBM EA 581 [2, p. 59]



^cBM EA 1783 [2, p. 74]


^dBM EA 581 [2, p. 59]

^eBM EA 101 [2, p. 58]

^fBM EA 117 [2, p. 31]

^gP. Turin Cat. 2070

Appearance	Unicode	PLOTTEXT	RES
		G17;/%B4+D36;	insert[te](G17*.,D36)









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 in another sign.¹





¹Stela Cairo, JE 60539, l. 8




JSesh has two primitives for insertion, each corresponding to one of up to two rectangular *zones* per sign. These primitives are used as $G^{^^}S$ 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 $G1^{^^}S\&\&G2$.

The granularity of our choice of the nine insertion primitives, rather than say the two insertion primitives of JSesh, is motivated by the following considerations:

- In all cases that we are aware of, the primitives are precise enough to ensure that the inserted, second group is placed in a satisfactory position within the first group. Because of the 2-dimensional nature of hieroglyphic writing, it would be highly problematic if, for example, a sign would be placed in the right upper or left lower corner if it is supposed to be in the left upper corner. In some cases, this may change the suggested reading.
- The primitives are precise enough to be characterized in procedural terms, allowing implementations to render them automatically; see further Section 10.2.

Nonetheless, for some choices of signs, more than one insertion primitive may be used without changing the meaning and without greatly affecting the appearance. For example, the appearances  and , encoded using  or  respectively, could be confused without causing significant problems for typical users. Moreover, if the inserted, second group has the same shape as the available space within the first group, more than one insertion primitive may lead to the same appearance. For example,  could best be achieved by using , but  and  lead to a very similar result.

We foresee that the UTC may raise objections against existence of several ways to achieve the same appearance. We see this however as an unavoidable consequence of the required granularity of insertion primitives, which was motivated earlier. The scope of the issue could be reduced by enforcing that the most appropriate primitive be used in case several are possible. For example, one could demand that  be encoded using  rather than  or .

We also foresee an objection may be raised against existence of invalid uses of insertions, which is another unavoidable consequence of the required granularity. For example, the sequence    would make little sense, as there is no space for the hand to the right of the cobra, that is, without the hand being scaled down to virtually disappear. In practice there should be a bound on the scaling factor, and the used software should notify encoders when they have used insertions in the wrong way.

8 Stacking







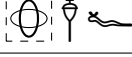

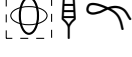
Sometimes two signs or groups are superimposed. Table 6 presents examples. The first two happen to also exist as individual Unicode characters, while the third is not part of any established sign list as far as we know. There are also many examples of whole groups being stacked, such as , which is the stacking of horizontal group  and vertical group . In JSesh, stacking of two signs is expressed using binary operator `##`.

Table 6: Stacking.

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




^aBM EA 581 [2, p. 59]

^bBM EA 584 [2, p. 122]

^c[10, p. 758]

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 7. 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 which 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 *cb*, where the non-stacked alternative  may be used, and in words starting with *bc*, where the non-stacked alternative  may be used.²

We foresee that the UTC will raise objections against generic stacking, as some existing signs among the 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.
- We reiterate the principle stated in Section 2 that we need to be able to 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.

²See WB I p. 173-178 and p. 446-450, respectively.

Table 7: Stacking as compositional operation.

Stacking	Attested alternatives	Transliteration
 <i>a</i>		h^c
 <i>b</i>		h^{cc}
 <i>c</i>		h^c
 <i>d</i>	 <i>e</i>	cbb
 <i>f</i>	 <i>g</i>	bcb^c
 <i>h</i>		$b\check{s}$
 <i>i</i>	 ,  <i>j</i>	bt

^aWB III p. 40

^bDendera VII, 148.4

^cStacked form and non-stacked alternative: WB III p. 40

^dASAE 43, p. 254

^eWB I p. 178

^fBIFAO 43, p. 118 and WB I p. 447

^gWB I p. 446

^hStacked form and non-stacked alternative: WB I p. 477

ⁱMIFAO 16, p. 49

^jBoth non-stacked alternatives: WB I p. 485

9 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 put between two groups to indicate that they may, but need not, move towards each other, to have their bounding boxes overlap. The signs should preferably not touch each other however.

Table 8 presents examples.

10 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 C.

10.1 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 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

Table 8: 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

^aBM EA 581 [2, p. 59]^bBM EA 584 [2, p. 122]^cBM EA 585 [2, p. 48]^dBM EA 143 [2, p. 110]

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 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.

10.2 Insertion

For the ideal rendering of say **INSERT** followed by two groups, both groups are first recursively scaled as in the previous section. Then the second group is inserted in the center of the first group, but if necessary is scaled down, and/or shifted, to leave a minimum distance between any two curves in the two groups. One

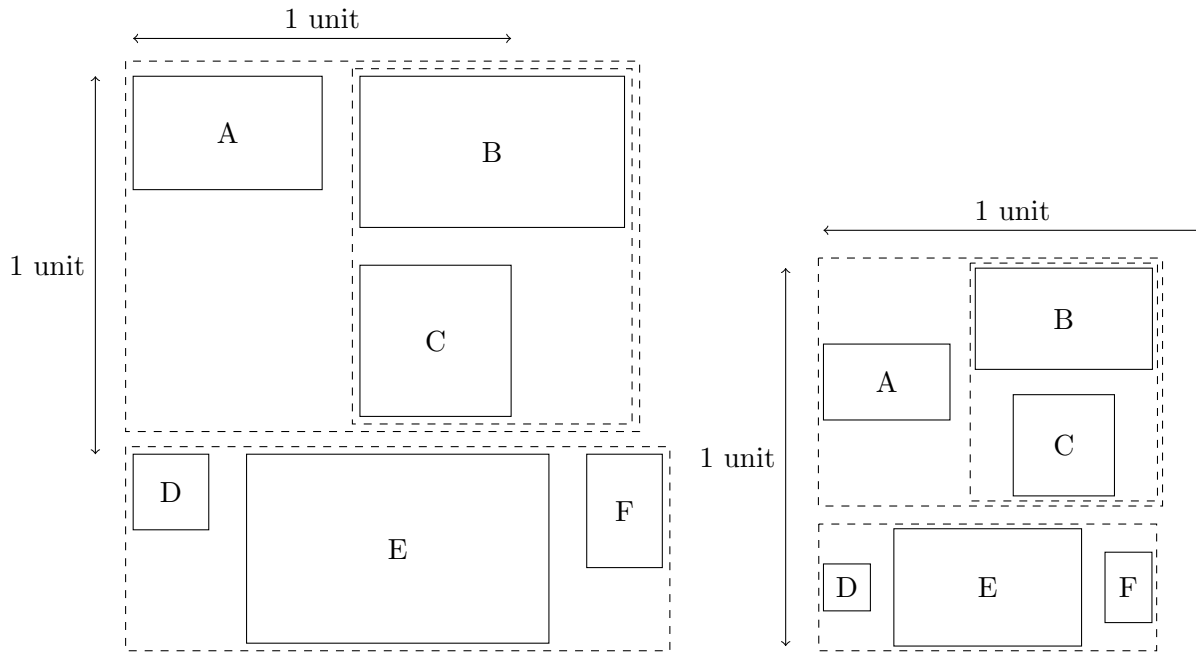


Figure 1: A nested group, before scaling and positioning (left) and after (right).

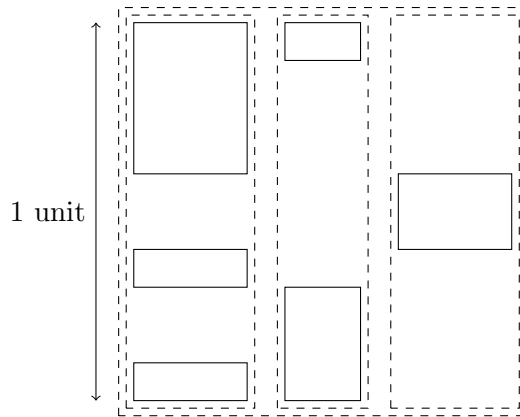


Figure 2: In this horizontal group, there is excess whitespace in all three vertical subgroups. In the rightmost subgroup, there is only a single sign, which is centered. In the leftmost two subgroups, the excess whitespace is divided equally between the (recursive) subgroups (which here happen to be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

should strive to have as little scaling down as possible. That is, if the second group can be rendered bigger after shifting it up/down or left/right, then it should be shifted.

The other insertion operations are similar, but there is no shifting of the position of the second group in the case of **INSERT_T_S**, **INSERT_T_E**, **INSERT_B_S**, and **INSERT_B_E**; the second group is flushed against two sides, in one of the four corners. For **INSERT_T** and **INSERT_B** only horizontal shifting is allowed and for **INSERT_S** and **INSERT_E** only vertical shifting of the second group is allowed. See further Appendix C.1.

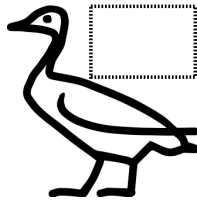








Figure 3: The rectangle that can be designated for the second group after an occurrence of **INSERT_T_E**, if the first group is the duck.

For example, in the case 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_E**. 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.

10.3 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.

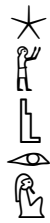


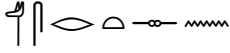





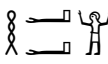
For some sign combinations, a more satisfactory realization may result if not the exact centers of the groups, but points a little distance away from the centers are chosen to coincide. For example, in  the center of  coincides with a point a little to the right of the center of . This can be realized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

10.4 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 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; we referred to this before in Section 5. The phenomena that we need to deal with here are particular to Ancient Egyptian.

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
		
		
		
		



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.

11 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.

11.1 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. 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.

11.2 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 .

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.

11.3 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.

11.4 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 10.1, 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 $\cup\cup$, 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.

11.5 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.

11.6 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 10.3.

11.7 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.

11.8 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 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 of the encoding using existing rendering engines difficult.

One may be tempted to cut corners and opt for a solution that looks impressive to a lay audience, while having no practical value whatsoever for those who were supposed to be the users, namely Egyptologists working with real hieroglyphic texts. As representative such users, we kindly request the UTC to give Ancient Egyptian the consideration that it deserves, and to work with us to find a better solution than those that were proposed in the recent past.

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] E. Graefe. *Mittelägyptische Grammatik für Anfänger*. Harrassowitz Verlag, Wiesbaden, 1994.
- [4] N. Grimal, J. Hallof, and D. van der Plas. *Hieroglyphica*. Publications Interuniversitaires de Recherches Égyptologiques Informatisées, Utrecht, Paris, 1993.
- [5] M.-J. Nederhof. A revised encoding scheme for hieroglyphic. In *Proceedings of the 14th Table Ronde Informatique et Égyptologie*, July 2002. On CD-ROM.
- [6] 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.
- [7] M.-J. Nederhof. ORC of handwritten transcriptions of Ancient Egyptian hieroglyphic text. In *Altertumswissenschaften in a Digital Age: Egyptology, Papyrology and beyond*, Leipzig, 2015.
- [8] 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.
- [9] S. Rosmorduc. JSesh Hieroglyphic Editor. <http://jseshdoc.qenherkhopeshef.org>, 2016.
- [10] K. Sethe. *Urkunden der 18. Dynastie, Volume I*. Hinrichs, Leipzig, 1927.
- [11] N. Stief. Hieroglyphen, Koptisch, Umschrift, u.a. – ein Textausgabesystem. *Göttinger Miscellen*, 86:37–44, 1985.
- [12] 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 ()^{*} indicate repetition zero or more times, and round brackets followed by a plus symbol ()⁺ indicate repetition one or more times

The following states that groups may, but need not, be preceded by a specification of the text direction.

```
fragment ::= [ direction ] [ groups ]
direction ::= HLR | HRL | VLR | VRL
```

The following states that two consecutive groups may, but need not, be connected by a joiner. A basic group may be an individual sign, an empty character, a box, an insertion, or a stacking. The two remaining kinds of groups are horizontal and vertical groups.

```
groups ::= group ( [ JOIN ] group )*
group ::= basic_group | horizontal_group | vertical_group
basic_group ::= sign | EMPTY | box | insert | stack
```

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 ( [ JOIN ] hor_subgroup )+ END
hor_subgroup ::= basic_group | vertical_group
vertical_group ::= VERT vert_subgroup ( [ JOIN ] vert_subgroup )+ END
vert_subgroup ::= basic_group | horizontal_group
```

The following states that a box, such as a cartouche, contains zero or more groups.

```
box ::= box_type [ groups ] END
box_type ::= CARTOUCHE | OVAL | SEREKH | INB | RECTANGLE | HWT
```

The following concerns an insertion of two groups, the second to be inserted into the first, at a location specified by the type. The second group is scaled down as required to fit in the specified location without touching the first group.

```
insert ::= insert_type group group
insert_type ::= INSERT |
               INSERT_T | INSERT_B |
               INSERT_S | INSERT_E |
               INSERT_T_S | INSERT_T_E |
               INSERT_B_S | INSERT_B_E
```

The following concerns stacking of two groups.

```
stack ::= STACK group group
```

B The characters

The following defines the individual characters used in the above description of the structure.

HLR, HRL, VLR, VRL: “indications of horizontal (rows) or vertical (columns) text, from left to right or from right to left”

JOIN: “control character put between two groups to indicate their bounding boxes may overlap”

EMPTY: “a zero-width and zero-height (empty) character”

HOR: “marker of the beginning of a horizontal group”

VERT: “marker of the beginning of a vertical group”

CARTOUCHE, ...: “markers of the beginning of some type of box”

END: “marker of the end of a horizontal or vertical group or box”

INSERT, ...: “insertion of a group into the bounding box of another, at the indicated position”

STACK: “stacking of two groups”

C Implementation

C.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 10. 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.

Omitting some optimizations, a concrete procedure to implement **INSERT_S** G_1 G_2 is roughly as in Algorithm 1. For the various insert operations and for **JOIN**, we need to check whether there is sufficient distance, say *sep*, between two groups. This can be realized in one of two ways:

- One may draw a circle of radius *sep* around each black pixel of the second group and test whether any of the drawn pixels coincide with a pixel of the first group. If so, the two groups are too close.
- One may draw the second group several times, say 8 times, a distance of *sep* away from its proposed position, in 8 different directions 45° apart. Analytical methods can then be used to test whether any of the resulting curves intersect with any of the curves of the first group.

The first method is used in the mentioned implementations of RES. We experimented with the second method for the JavaScript implementation using SVG, but found the computational costs were prohibitive

Algorithm 1 Simplified implementation of the equivalent of **INSERT_S** G_1 G_2 in RES

```
1:  $y_{best} \leftarrow 0.5$  ▷ Start in middle y-coordinate of  $G_1$ 
2: while true do ▷ Iterate until convergence
3:    $y_{up} \leftarrow y_{best} + 0.05$  ▷ Consider going up
4:    $y_{down} \leftarrow y_{best} - 0.05$  ▷ Consider going down
5:    $f_{best} \leftarrow \text{MAX\_SCALING}(y_{best})$  ▷ Biggest scaling with current y-coordinate
6:    $f_{up} \leftarrow \text{MAX\_SCALING}(y_{up})$  ▷ Biggest scaling if we go up
7:    $f_{down} \leftarrow \text{MAX\_SCALING}(y_{down})$  ▷ Biggest scaling if we go down
8:   if  $f_{down} \leq f_{best}$  and  $f_{up} \leq f_{best}$  then ▷ Scaling is not getting bigger
9:     render with  $y_{best}$  and  $f_{best}$  and halt ▷ So do rendering and we're done
10:  else if  $f_{up} < f_{down}$  then ▷ Best improvement by going down
11:     $y_{best} \leftarrow y_{down}$  ▷ Go down
12:  else
13:     $y_{best} \leftarrow y_{up}$  ▷ Go up
14: function  $\text{MAX\_SCALING}(y)$  ▷ Determine biggest possible scaling at given y-coordinate
15:    $f_{best} \leftarrow 0.05$  ▷ Start with smallest scaling
16:   while  $f_{best} < 1$  do ▷ We cannot scale up beyond natural size
17:      $f \leftarrow f_{best} + 0.05$  ▷ Consider larger scaling
18:     if  $\text{CAN\_RENDER}(y, f)$  then ▷ Is larger scaling ok?
19:        $f_{best} \leftarrow f$  ▷ Continue with larger scaling
20:     else return  $f_{best}$  ▷ We've gone too far, so stop now
21:   return  $f_{best}$  ▷ Return maximum scaling possible for  $y$ 
22: function  $\text{CAN\_RENDER}(y, f)$ 
23:   render  $G_2$  with scaling  $f$ , with its middle at y-coordinate  $y$  of  $G_1$ ,
     and its left edge at x-coordinate 0 of  $G_1$ 
24:   return  $G_2$  falls within the bounding box of  $G_1$  and
     there is sufficient distance between  $G_1$  and  $G_2$ 
```

for fast rendering of web pages; this may change with future browsers, faster hardware, and/or better support for SVG operations.

C.2 Realization in OpenType

The functionality of our horizontal and vertical grouping roughly corresponds to the horizontal joiner and the vertical joiner from L2/16-018R. That document further claims that “All of the samples can be implemented in OpenType using glyphs substitutions” (p. 4), but it stops short of explaining how this was done. One would hope that at least the horizontal joiner and the vertical joiner were, or could be, implemented in a reasonably general way. However the phrase “focus on attested forms [...] rather than [...] arbitrary quadrats” (p. 2) reveals they were not.

Even if one would abstract away from particular signs and implement grouping for particular choices of sign *dimensions*, it is clear we are running into severe problems. Recall the algorithm for rendering of horizontal and vertical groups in Section 10.1. In principle, the width and height of any sign within a group can influence the scaling and positioning of any other sign within the same group. Concretely, this means a rendering engine that scales and positions a sign after analyzing its context should simultaneously look at many aspects of that context.

Suppose we divide signs into classes, depending on their rough dimensions. We might distinguish between 4 different heights and 4 different widths, which gives us 16 classes of signs. If a group may contain up to 7







signs (this seems already fairly restrictive), then we would have to distinguish at least $16^7 \approx 268$ million kinds of groups. How many of these would be “attested” ? Probably a small portion of the 268 million, but every text would contain a few we have not seen before. We would reiterate a now familiar point, which is that an encoding scheme has no value unless we can encode a text we have not seen before, which makes reliance on “attested forms” a fatal flaw.


In this section, we aim to investigate to what extent OpenType is able to handle horizontal and vertical groups without cheating, that is, without storing an exceedingly large (while still wholly inadequate) table of “attested forms”. While OpenType is clearly not intended for hieroglyphic text, and any solution will be clumsy to the extreme, we feel forced to address this issue, because of the importance that is given, justly or unjustly, to OpenType and the Universal Shaping Engine.

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 10. 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).

Figure 4 depicts the records involved in the analysis of the group , which has a horizontal group within a vertical group. We assume here that the natural width/height of ,  and  are $1.0/0.7$, $0.5/0.6$ and $1.0/0.4$, respectively. In this example,  and  are eventually replaced by scaled-down versions, which gives them dimensions $0.6/0.5$ and $0.3/0.4$. Note that we inevitably need to round off, as only a finite number of values can be manipulated. Here we pragmatically round off to multiples of 0.1 times the unit size.

The right-to-left analysis of the outer, vertical group starts in record (12), where the maximum width (w), sum of heights (h) and number of subgroups (n) are all initially 0. Record (10) updates that information by considering the natural width and height of . This information is copied unchanged through the bottom halves of (8), (6) and (4). In the inner, horizontal group, a similar analysis is done starting with w (now the sum of widths), h (now the maximum height), and n all being 0 in the top half of (8). In (3), the width (w) is truncated to 1.0, as the natural width 1.6 exceeds 1.0, and here it is found that the horizontal group needs to be scaled down by factor 0.7 at least.

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).

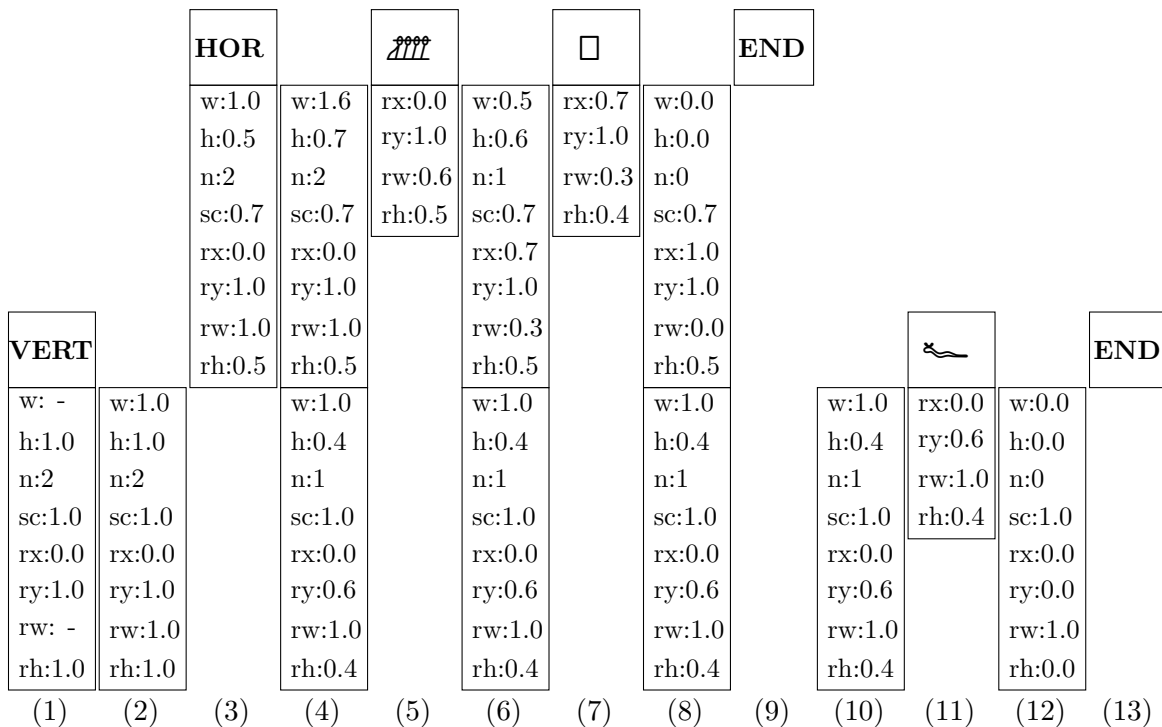


Figure 4: Record keeping in our partial OpenType implementation of formatting of groups.

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.
- There is no particular reason to exclude the possibility of, e.g., vertical groups within horizontal groups within vertical groups (as L2/16-018R did without giving adequate motivation; see also L2/16-90).
- Connected to the previous point, our syntax with start and end markers is more convenient than the syntax of L2/16-018R, which relied on inadequately motivated operator precedence.
- 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.

C.3 The fall-back option

Some may argue that the primitives of this proposal are too powerful to be included in Unicode, in the light of inadequacies of OpenType technology. We would contend however that our primitives are not per se more powerful than those from proposal L2/16-018R. The central difference is that the behaviour of, for example, our insertion primitives can be precisely defined, whereas the LIGATURE JOINER from L2/16-018R is and would remain inherently *undefined*, which is unacceptable from the perspective of reliable exchange of textual data between different parties.

If no suitable implementations of our primitives can be achieved in OpenType that would work for any combination of signs, then the obvious fall-back option is to make the same assumption as was made

by L2/16-018R, namely that only a fixed collection of special groups would be implemented, each by one dedicated substitution rule in the font. The crucial advantage of our proposal over the LIGATURE JOINER is however that we can then still unambiguously encode new groups not yet implemented in the font, even if these cannot be immediately rendered as such. One font may implement more special groups than another, but two fonts correctly implementing the primitives cannot render the same encoding in two essentially different ways, due to the primitives being well-defined.

Hence, if the UTC was of the opinion that the control characters in L2/16-018R were technically feasible, then we see no reason why evaluation of the present proposal would have a different outcome.