

**Title:** A note on the syntax of Ancient Egyptian hieroglyphic control characters  
**From:** Mark-Jan Nederhof (University of St Andrews)  
**To:** UTC  
**Date:** 2018-06-13

## 1 Introduction

Ancient Egyptian is an exceptional writing system in that arrangements of signs can be deeply nested. For example, two signs can be arranged next to one another, and both signs can be underneath a third sign, and these three signs can together be next to a fourth sign. An encoding of the spatial layout must therefore be capable of specifying such nesting.

The control characters for Ancient Egyptian hieroglyphic text, as described in document L2/17-112, are listed in Table 2. There were two main differences with the syntax in the initial proposal L2/16-177. First, the collection of primitives was originally much larger, with some being clearly essential, while others seemed merely desirable. Starting from L2/16-210, the proposal was scaled down to only include the most essential primitives.

Second, the initial proposal had a very simple syntax, motivated by font technology such as OpenType, which is not able to ‘parse’ in the usual sense of the word. Our syntax had a combination of bracketing and Polish notation, avoiding any notion of operator precedence. Proof of concept was established that OpenType would be able to process this form of bracketing.

The downside was that a large number of control characters were needed to describe relatively simple groups of hieroglyphs. It was argued by some that this would hinder manual input of hieroglyphic texts, under the assumption that dedicated hieroglyphic editors with graphical user interfaces could not be developed. Others argued that bracketing was a concept alien to Unicode, and therefore to be avoided altogether.

This made us change direction and L2/16-233 and ultimately L2/16-210R explored syntax without brackets, which unavoidably resulted in a complicated system of control characters, in which each primitive would exist on several levels of operator precedence. It also meant that there would be a hard limit on the depth of nesting.

Eventually, resistance against the concept of bracketing waned, which paved the way towards reintroduction of brackets, albeit in combination with operator precedence, to require relatively few control characters for simple groups. It was shown in Appendix C of L2/16-210R how OpenType could deal with operator precedence.

However, the eventually accepted proposal L2/17-112 went one step further than we did in Appendix A.1

Table 1: The control characters from L2/17-112 in the left column, with in the right column the abbreviated names used in this document.

13430 EGYPTIAN HIEROGLYPH VERTICAL JOINER	<b>VERT</b>
13431 EGYPTIAN HIEROGLYPH HORIZONTAL JOINER	<b>HOR</b>
13432 EGYPTIAN HIEROGLYPH START AT TOP	<b>ST</b>
13433 EGYPTIAN HIEROGLYPH START AT BOTTOM	<b>SB</b>
13434 EGYPTIAN HIEROGLYPH END AT TOP	<b>ET</b>
13435 EGYPTIAN HIEROGLYPH END AT BOTTOM	<b>EB</b>
13436 EGYPTIAN HIEROGLYPH OVERLAY MIDDLE	<b>OVERLAY</b>
13437 EGYPTIAN HIEROGLYPH BEGIN SEGMENT	<b>BEGIN</b>
13438 EGYPTIAN HIEROGLYPH END SEGMENT	<b>END</b>

of L2/16-210R, and this causes a complication for standard parser generators. Fortunately, a workaround was recently found. The aim of this document is:

- to ensure that the UTC is aware of the issue,
- to ask confirmation that the syntax as formally described in this document is what was agreed with the approval of L2/17-112,
- to inform developers of the workaround,
- to give the UTC the opportunity to initiate a revision of the syntax if the proposed workaround is deemed impractical.

## 2 Syntax

### 2.1 Accepted syntax

Table 2 presents the syntax as formally specified in Appendix A.1 of L2/16-210R, with modifications implicit in L2/17-112. In this Backus-Naur specification, lower-case non-bold names are classes. Bold-face names represent characters, with upper-case boldface names representing the particular characters specified in Table 1, 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 first few lines of Table 2 specify that a fragment of hieroglyphic text consists of one or more groups, and that a group may be a vertical group, a horizontal group, or a basic group. A vertical or horizontal group may have subgroups, separated by the appropriate control character. A subgroup of a vertical group may not be a vertical group itself however, nor may a subgroup of a horizontal group be a horizontal group itself. In order to avoid ambiguity, if a vertical group is a subgroup of a horizontal group, then it must be enclosed in a pair of control characters **BEGIN** and **END**, which act as ‘brackets’.

The next few lines specify that a basic group is a core group optionally followed by inserted groups. In order to avoid ambiguity, an inserted group (in\_group) must be enclosed in brackets, unless it is a core group by itself, i.e. without inserted groups. Specifying an insertion with at least one out of **ST**, **SB**, **ET**, **EB** requires more rules than one may expect.

A core group can be a single sign, or an overlay of a flat horizontal group and a flat vertical group. If such a flat horizontal or vertical group has more than a single sign, it must be enclosed in brackets, once more to avoid ambiguity.

Regrettably, this specification is not amenable to standard parser generators. To see this, consider two encodings starting with:

$s_1$  **ST BEGIN**  $s_2$  **HOR**  $s_3$  **END OVERLAY** *etc.*  
 $s_1$  **ST BEGIN**  $s_2$  **HOR**  $s_3$  **END SB** *etc.*

where  $s_1$ ,  $s_2$ ,  $s_3$  are signs. In the first case, a reduction to flat\_hor\_group would take place when **OVERLAY** is seen as lookahead, but in the second case, a reduction of  $s_2$  **HOR**  $s_3$  to hor\_group would take place earlier, upon seeing **END** as lookahead. In technical terms, the grammar would need to be LR( $k$ ) for  $k \geq 2$  in order to choose the correct parser action at the right time, while standard parser generators such as Yacc, Bison or Jison require grammars to be at least LR(1). Note that this problem did not exist in Appendix A of L2/16-210R, which assumed dedicated control characters in place of **HOR** and **VERT** in overlays.

Table 2: The agreed syntax.

```

fragment ::= ( group )+
group ::= vert_group | hor_group | basic_group

vert_group ::= vert_subgroup ( VERT vert_subgroup )+
vert_subgroup ::= hor_group | basic_group
hor_group ::= hor_subgroup ( HOR hor_subgroup )+
hor_subgroup ::= BEGIN vert_group END | basic_group

basic_group ::= core_group | insert_group
insert_group ::= core_group insertion
insertion ::= ST in_group [ SB in_group ] [ ET in_group ] [ EB in_group ] |
             SB in_group [ ET in_group ] [ EB in_group ] |
             ET in_group [ EB in_group ] |
             EB in_group
in_group ::= BEGIN vert_group END |
            BEGIN hor_group END |
            BEGIN insert_group END |
            core_group
core_group ::= flat_hor_group OVERLAY flat_vert_group | sign
flat_hor_group ::= BEGIN sign ( HOR sign )+ END | sign
flat_vert_group ::= BEGIN sign ( VERT sign )+ END | sign

```

## 2.2 LALR(1) grammar

By transforming the grammar, the equivalent LALR(1) grammar in Table 3 results, where  $\varepsilon$  denotes the empty string. Intuitively, the difference is that **sign** is taken out of core groups, basic groups and horizontal subgroups. Further, vertical and horizontal groups are specified using tail recursion. This means in particular that the choice between a bracketed flat horizontal group and a general bracketed horizontal group can be postponed long enough to allow lookahead to distinguish the two cases.

Note however that the price we pay for this workaround is a much larger number of rules, which moreover are more difficult to understand. Fortunately, details of the grammar can be hidden from end users, so that the issue is mainly of concern to developers.

## 3 Implementation

The Jison parser (JavaScript) in <https://mjn.host.cs.st-andrews.ac.uk/egyptian/res/js/> uses the grammar in Table 3. As far as we know, this is also the first full implementation of the control characters for Ancient Egyptian.

It should be understood that implementation of the control characters in font technology such as OpenType and Graphite is not directly affected by the issue described in this document, as such implementation would likely not make use of LR parsing.

Table 3: LALR(1) grammar for the same syntax.

```

fragment ::= groups
groups ::= group | group groups
group ::= vert_group | hor_group | basic_group | sign

vert_group ::= vert_subgroup rest_vert_group
rest_vert_group ::= VERT vert_subgroup rest_vert_group | VERT vert_subgroup

br_vert_group ::= BEGIN vert_subgroup rest_br_vert_group
rest_br_vert_group ::= VERT vert_subgroup rest_br_vert_group | VERT vert_subgroup END

br_flat_vert_group ::= BEGIN sign rest_br_flat_vert_group
rest_br_flat_vert_group ::= VERT sign rest_br_flat_vert_group | VERT sign END

vert_subgroup ::= hor_group | basic_group | sign

hor_group ::= hor_subgroup rest_hor_group | sign rest_hor_group
rest_hor_group ::= HOR hor_subgroup rest_hor_group | HOR sign rest_hor_group |
HOR hor_subgroup | HOR sign

br_hor_group ::= BEGIN hor_subgroup rest_br_hor_group | BEGIN sign rest_br_hor_group
rest_br_hor_group ::= HOR hor_subgroup rest_br_hor_group | HOR sign rest_br_hor_group |
HOR hor_subgroup END | HOR sign END

br_flat_hor_group ::= BEGIN sign rest_br_flat_hor_group
rest_br_flat_hor_group ::= HOR sign rest_br_flat_hor_group | HOR sign END

hor_subgroup ::= br_vert_group | basic_group

basic_group ::= core_group | insert_group
insert_group ::= core_group insertion | sign insertion
br_insert_group ::= BEGIN core_group insertion END | BEGIN sign insertion END

insertion ::= ST in_group opt_sb_insertion opt_et_insertion opt_eb_insertion |
SB in_group opt_et_insertion opt_eb_insertion |
ET in_group opt_eb_insertion |
EB in_group

opt_sb_insertion ::= SB in_group |  $\epsilon$ 
opt_et_insertion ::= ET in_group |  $\epsilon$ 
opt_eb_insertion ::= EB in_group |  $\epsilon$ 

in_group ::= br_vert_group | br_hor_group | br_insert_group | core_group | sign

core_group ::= flat_hor_group OVERLAY flat_vert_group
flat_hor_group ::= br_flat_hor_group | sign
flat_vert_group ::= br_flat_vert_group | sign

```