

Title: Notes on the control characters for Ancient Egyptian in Unicode 15
From: Mark-Jan Nederhof (University of St Andrews)
To: UTC
Date: 2024-02-17

1 Introduction





Unicode 12 introduced 9 control characters for Ancient Egyptian. Notes on their syntax and semantics appeared as L2/18-236 and L2/19-331. With the additional 29 control characters from L2/21-248, Unicode 15 now includes 38 controls for Ancient Egyptian, listed in Table 1. This document provides updated notes on syntax and semantics, and introduces an implementation in JavaScript for use in web pages, called **HieroJax**.

The controls for Ancient Egyptian involve difficulties not seen in other writing systems in Unicode. In particular, the problem of matching pairs of parentheses is beyond the power of regular languages. It is important to formally document which sequences of characters are or are not valid encodings of hieroglyphic text. Only then can interoperability be guaranteed between different software implementing the controls.

It should be stressed however that there is no expectation that OpenType fonts can render each valid encoding. One reason is that the depth of nesting of vertical and horizontal subgroups is unbounded, which is beyond the finite-state power of OpenType.

2 Global syntax

L2/21-248 gave syntax rules for the new controls as additions to the rules for the first 9 controls. A complete grammar is given in Table 2, with a slightly relaxed syntax. At the time of submission of L2/21-248, it seemed justified not to allow enclosures as inserted groups, as this has never been attested as far as we are aware. However, by the specified syntax, an inserted group can be a horizontal or vertical group, and an enclosure can indirectly be part of a horizontal or vertical group. Hence, the syntax in L2/21-248 failed to disallow enclosures that are nested *within* inserted groups, which made it pointless to disallow enclosures as inserted groups directly. Moreover, the restriction was found to cause significant complications for graphical editors (cf. Section 6). Hence, we have lifted the restriction. Moreover, for technical reasons, it can be convenient to allow the empty sequence of characters as ‘fragment’ in our formal specification.

The meanings of five of the terminals in the grammar are specified in Table 3. A singleton differs from an (ordinary) sign in that its orientation depends on the text direction; if the text direction is vertical, then singletons rotate by 90°. As already noted in L2/21-248, the grammar is ambiguous, as a plain/walled opening or closing can either be attached to a pair  or a pair  as part of an enclosure, or be an isolated singleton. We assume the lexical analyzer resolves this ambiguity by the principle of ‘longest match’, preferentially treating openings and closings as parts of enclosures. Further note that here we have constrained the grammar such that a pair  can only be combined with ‘plain’ delimiters and a pair  can only be combined with ‘walled’ delimiters. Some implementations may however fail to enforce this constraint; the consequence of incorrect use of delimiters is then typically that the encoding is rendered in an unsatisfactory manner.














































































The opening and closing brackets that may appear for  and , respectively, are listed in Table 4. The apparent complexity of the rules for ‘hor_group’ in Table 2 requires further justification. The intuition is that, syntactically at least, a philological bracket behaves like a ‘hor_subgroup’, except that an explicit operator  between that bracket and the neighbouring ‘hor_subgroup’ to which it attaches is omitted. For

Table 1: The 38 controls (abbreviated, to omit 13 of the 15 ‘damaged’ characters).

U+13430		EGYPTIAN HIEROGLYPH VERTICAL JOINER
U+13431		EGYPTIAN HIEROGLYPH HORIZONTAL JOINER
U+13432		EGYPTIAN HIEROGLYPH INSERT AT TOP START
U+13433		EGYPTIAN HIEROGLYPH INSERT AT BOTTOM START
U+13434		EGYPTIAN HIEROGLYPH INSERT AT TOP END
U+13435		EGYPTIAN HIEROGLYPH INSERT AT BOTTOM END
U+13436		EGYPTIAN HIEROGLYPH OVERLAY MIDDLE
U+13437		EGYPTIAN HIEROGLYPH BEGIN SEGMENT
U+13438		EGYPTIAN HIEROGLYPH END SEGMENT
U+13439		EGYPTIAN HIEROGLYPH INSERT AT MIDDLE
U+1343A		EGYPTIAN HIEROGLYPH INSERT AT TOP
U+1343B		EGYPTIAN HIEROGLYPH INSERT AT BOTTOM
U+1343C		EGYPTIAN HIEROGLYPH BEGIN ENCLOSURE
U+1343D		EGYPTIAN HIEROGLYPH END ENCLOSURE
U+1343E		EGYPTIAN HIEROGLYPH BEGIN WALLED ENCLOSURE
U+1343F		EGYPTIAN HIEROGLYPH END WALLED ENCLOSURE
U+13440		EGYPTIAN HIEROGLYPH MIRROR HORIZONTALLY
U+13441		EGYPTIAN HIEROGLYPH FULL BLANK
U+13442		EGYPTIAN HIEROGLYPH HALF BLANK
U+13443		EGYPTIAN HIEROGLYPH LOST SIGN
U+13444		EGYPTIAN HIEROGLYPH HALF LOST SIGN
U+13445		EGYPTIAN HIEROGLYPH TALL LOST SIGN
U+13446		EGYPTIAN HIEROGLYPH WIDE LOST SIGN
U+13447		EGYPTIAN HIEROGLYPH MODIFIER DAMAGED AT TOP START
⋮	⋮	⋮
U+13455		EGYPTIAN HIEROGLYPH MODIFIER DAMAGED

Table 2: The complete syntax.

```

fragment ::= ( top_group ) *
top_group ::= group | singleton
group ::= ver_group | hor_group | basic_group
ver_group ::= ver_subgroup (  ver_subgroup ) +
ver_subgroup ::= hor_group | basic_group
hor_group ::=  hor_subgroup  (   hor_subgroup  ) * |
    hor_subgroup  (   hor_subgroup  ) * |
    hor_subgroup (   hor_subgroup  ) +
hor_subgroup ::= ( ver_group ) | basic_group
basic_group ::= core_group | insert_group | placeholder | enclosure
insert_group ::= core_group insertion
insertion ::=  in_group  in_group  in_group  in_group  in_group
     in_group  in_group |
     in_group  in_group  in_group  in_group  in_group
     in_group |
     in_group  in_group  in_group  in_group  in_group |
     in_group  in_group  in_group  in_group |
     in_group  in_group |
     in_group
in_group ::= ( ver_group ) | ( hor_group ) | ( insert_group ) |
    core_group | placeholder | enclosure
core_group ::= flat_hor_group  flat_ver_group | literal
flat_hor_group ::= ( literal (  literal ) + ) | literal
flat_ver_group ::= ( literal (  literal ) + ) | literal
literal ::= sign [  |  |  |  |  |  |  ] [  ] [ damaged ]
placeholder ::=  |  | (  |  |  |  ) [  ]
enclosure ::= [ plain_opening ]  ( group ) *  [ plain_closing ] |
    [ walled_opening ]  ( group ) *  [ walled_closing ]
singleton ::= plain_opening | plain_closing | walled_opening | walled_closing
plain_opening ::= plain_opening_delimiter [ damaged ]
plain_closing ::= plain_closing_delimiter [ damaged ]
walled_opening ::= walled_opening_delimiter [ damaged ]
walled_closing ::= walled_closing_delimiter [ damaged ]
damaged ::=  |  |  |  |  |  |  |  |  |  |  |  |  | 

```

Table 3: The 1072 hieroglyphic signs in the range U+13000 – U+1342F are divided into 1057 (ordinary) signs, and 15 delimiters.

sign	U+13000 – U+13257, U+1325E – U+13281, U+13283 – U+13285, U+1328A – U+13378, U+1337C – U+1342E
plain_opening_delimiter	U+13258 – U+1325A, U+13379, U+1342F
plain_closing_delimiter	U+1325B – U+1325D, U+13282, U+1337A – U+1337B
walled_opening_delimiter	U+13286, U+13288
walled_closing_delimiter	U+13287, U+13289

Table 4: Philological brackets.

[]	
[=U+005B]	=U+005D
{	=U+007B	}	=U+007D
<	=U+27E8	>	=U+27E9
⌈	=U+2E22	⌋	=U+2E23
⌈	=U+27E6	⌋	=U+27E7

example, a pattern of the form:

[hor_subgroup]

is analyzed as a horizontal arrangement of three elements, similarly to a pattern of the form:

hor_subgroup [*] hor_subgroup [*] hor_subgroup

The horizontal joiner has higher operator precedence than the vertical joiner, even if the horizontal joiner is implicit as in the case of philological brackets. For example, a pattern of the form:

ver_subgroup [:] [hor_subgroup]

is analyzed as a vertical arrangement of two vertical subgroups, the second of which is a horizontal arrangement with two philological brackets.

As to rendering of these brackets, we make two assumptions:

1. Brackets ideally do not take up space for themselves, but are positioned in the ‘natural’ spaces between hieroglyphs.
2. Brackets attached to subgroups are ignored for the purpose of scaling and positioning of signs.

Consider for example:

([(= : =) : =])

If we ignore the two square brackets, then this is a simple vertical arrangement of three times the sign N17. Therefore, the three signs are ideally scaled by the same factor, and ideally there is a uniform distance

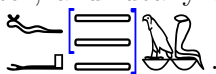
between the signs. In context, this may result in for example: 

Table 5: Variation selectors for rotation of signs. The indicated angles of rotation are clockwise (assuming left-to-right text). The expression $\pm \delta$ represents a relatively small angle to be added or subtracted, depending on the sign.

$\overline{\text{VS}}_1$	=U+FE00	90°
$\overline{\text{VS}}_2$	=U+FE01	180°
$\overline{\text{VS}}_3$	=U+FE02	270°
$\overline{\text{VS}}_4$	=U+FE03	45° $\pm \delta$
$\overline{\text{VS}}_5$	=U+FE04	135° $\pm \delta$
$\overline{\text{VS}}_6$	=U+FE05	225° $\pm \delta$
$\overline{\text{VS}}_7$	=U+FE06	315° $\pm \delta$

3 Platform-independent representation of data

Some combinations of specific signs and control characters and variation selectors may need to be interpreted in a special way. In the case of rotations, relevant information could be partly expressed in terms of StandardizedVariants.txt on the Unicode website. However, other information falls within the realm of font design, including fine-tuning of the exact way in which a fragment of hieroglyphic text is rendered. Here we discuss the JSON formats in which we encode such information.

3.1 Rotations

In addition to rotation by multiples of 90°, achieved with variation selectors $\overline{\text{VS}}_1 - \overline{\text{VS}}_3$, L2/21-248 also reserved $\overline{\text{VS}}_4 - \overline{\text{VS}}_7$ for rotations by remaining multiples of 45°, but in practice, the angle of rotation may need to be somewhat smaller or greater than an exact multiple of 45°, depending on the sign, as indicated in Table 5.

It is not the intention that all rotations are available for all signs. We use a JSON file to compactly encode:


- which rotations are available for which signs, and
- which adjustments should be made to the angle of rotation to be different from an exact multiple of 45°.

For example, one line from the JSON file reads:

```
"\ud80c\udd39": { "90": 0, "180": 0 },
```

This means that for sign U+13139 (= \ud80c\udd39 in UTF-16), the available rotations are by 90° and 180°, which are multiples of 45° as required. The two occurrences of 0 here indicate that the rotations are by exactly 90° and 180°, respectively. A value other than 0, which might be negative or positive, indicates that the rotation is slightly smaller or slightly larger than a multiple of 45°. For example, one might have:


```
"\ud80c\udd26": { "45": -15 },
```


This specifies that if U+1312A is used with insertion at the bottom, then an alternative glyph is used, which in this particular font is stored in the PUA at U+E487. The rendering of U+1312A without and with insertion may then be something like .


Multiple forms may be associated with a sign. The first form that comes with all the required places is chosen by the rendering algorithm. An example is:

```
"\ud80c\udea3": [{ "ts": { }, "te": { } },
  { "glyph": "\ue496", "ts": { }, "bs": { }, "te": { }, "be": { } }],
```

Here the default glyph is chosen if the occurrence is without insertions, or with TS or TE insertion or both. The alternative glyph at U+E496 is chosen if the occurrence is with BS or BE insertion or both, and possibly


with TS or TE insertions. Examples of rendering are: .


If a sign is used with a combination of insertions that is not covered by any of the listed forms in the JSON file, then an implementation may either fail to render the sign combination altogether, or output an error message while attempting to render the sign combination in a reasonable way.


The choice which insertion primitives to use can seem somewhat arbitrary. An example is , where we opt for encoding with BS insertion, rather than with B or M insertion. It is important that encoding be consistent, following the JSON file, or else search for certain groups will not be effective.

We used the following guidelines:


- If the inserted group would appear immediately above the feet of a bird (or other animal or human), then this is generally seen as BS or BE insertion, provided the bird or animal is not placed on top of another object (in which case this is seen as TS or TE insertion instead, reserving BS or BE for


potential groups to be inserted below that object). An example is , where TS inserts a group above the feet of the falcon, while BS inserts a group in the bottom-left corner, below the standard.

- Consistency between related signs takes precedence over absolute position. For example, in the case of , BS insertion would be used for an insertion immediately above the legs of the giraffe, following the general rule above, even though the inserted group might be placed about midway between upper left

and lower left corner. A similar example is , where TE insertion would be used for an insertion above the back of the lion animal, consistent with the use of TE insertion for many other mammals, although geometrically this might look closer to a T insertion, as the inserted group would appear closer to the middle of the top edge of the bounding box due to the tail extending into the top-right corner.

- M insertion generally encodes that the inserted group appears entirely within the bounding box of the base sign, whereas in the case of T and B insertions, the inserted group might (but is not guaranteed to) protrude outside the bounding box. Variant glyphs may serve to accentuate the distinction between

M insertion on the one hand, and T or B insertion on the other. For example,  might appear in its usual form if used with B insertion, but might appear as a taller, more voluminous shape if used with M insertion, to allow the inserted group to be placed entirely inside.

A sign that is mirrored using the  control requires any allowed places to be swapped with their mirrored equivalents. For example, if an unmirrored sign is allowed to take a TS insertion, then its mirrored form can take a TE insertion. An overlay is always allowed to take TS, BS, TE, BE insertions.

However, rotation cannot be handled in the same way as mirroring. The main reason is that T and B do not translate straightforwardly to equivalents after rotation by 90° clockwise or counterclockwise. Rotations by 45° or 135° would cause further difficulties. Hence, for angles of rotation of a sign that could be expected to occur with one or more insertions, separate forms need to be specified in the JSON file. An example is:

```
"\ud80c\udf0f": [{ "bs": { } }, { "rot": 270, "be": { } } ],
```




This means that in its unrotated form, the sign can take a BS insertion and if it is rotated a quarter turn counterclockwise, then it can take a BE insertion. See [3] for more information.

3.3 Ligatures

For some combinations of signs joined by controls, it may be desirable to render a ‘ligature’, by which we mean a single composed glyph stored in the font. We maintain another JSON file to list such sign combinations, mainly for the ‘overlay’ control. If one of the constituent signs is marked as being damaged (see Section 4) then in the eventual rendering of the ligature, the corresponding part of the area spanned by the ligature needs to be shaded. For this reason, the JSON file encodes those parts of the area as well.


An example is:

```
"\ue505": { "type": "overlay",
  "horizontal": [
    { "ch": "\ud80c\udf4", "mirror": true, "x": 0.4, "y": 0, "w": 0.3, "h": 1 } ],
  "vertical": [
    { "ch": "\ud80c\udd93", "x": 0.1, "y": 0, "w": 0.9, "h": 0.8 },
    { "ch": "\ud80c\udd93", "x": 0, "y": 0.2, "w": 0.9, "h": 0.8 } ] }
```

This expresses that ligature , which is U+E505 in the PUA, would be used in place of a mirrored occurrence of  overlaid with a vertical arrangement of twice . The parts of the area corresponding to the three constituent signs are indicated by x , y , w , h . Here x and y are as in Section 3.2, and w and h are the widths and heights of the constituent signs as portions of the width and height of the ligature.

There may be more than one ligature for a combination of signs joined by the ‘overlay’ control, depending on possible insertions. In this case, there is a default ligature if there are no insertions, and there may be ‘alternative’ ligatures, which are selected as explained in Section 3.2. An example is:

```
"\ue506": { "type": "overlay",
  "horizontal": [
    { "ch": "\ud80c\udcc0", "x": 0, "y": 0, "w": 0.45, "h": 1 },
    { "ch": "\ud80c\udcc0", "x": 0.45, "y": 0, "w": 0.45, "h": 1 } ],
  "vertical": [ { "ch": "\ud80c\ude83", "x": 0.2, "y": 0.35, "w": 0.8, "h": 0.2 } ] },
"\ue507": { "type": "overlay", "alt": true,
  "horizontal": [
    { "ch": "\ud80c\udcc0", "x": 0, "y": 0, "w": 0.45, "h": 1 },
    { "ch": "\ud80c\udcc0", "x": 0.45, "y": 0, "w": 0.45, "h": 1 } ],
  "vertical": [ { "ch": "\ud80c\ude83", "x": 0.2, "y": 0.05, "w": 0.8, "h": 0.2 } ] }
```


The first of these, U+E506, is the default ligature , while U+E507 is an ‘alternative’ ligature, as indicated by “alt”: true. The JSON file for insertions (cf. Section 3.2) might contain:









```
"\ue506": [
  { "ts": {}, "bs": {"y": 0.7}, "te": {}, "be": {}, "t": {"x": 0.6} },
  { "glyph": "\ue507",
    "ts": {}, "bs": {"y": 0.7}, "te": {}, "be": {}, "m": {"x": 0.6, "y": 0.6} } ]
```

This implies that the alternative ligature U+E507 would be used only if there is a M insertion, and U+E506 in all other (allowable) cases. See [3] for more information.

4 Shading

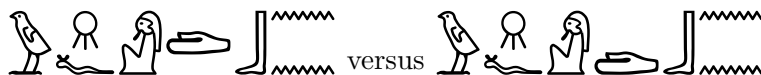
A sign can be combined with one of 15 controls that indicate damage of an inscription. With these 15 controls, the available granularity is down to the level of individual quarters of signs. Damage is typically rendered by shading or hatching part of the background of the sign. Some clarification is in order about what the exact area is that is shaded or hatched.

First, we must reject the option of only shading the bounding box of a sign, as it would then be impossible for readers to discern the shading of for example  (Z1). The best option seems to be to divide the entire surface of a group into areas occupied by the signs in that group, in such a way that if all signs are damaged, then the entire surface of that group appears shaded. The result is then close to typical published hieroglyphic transcriptions exemplified in Table 24 of L2/21-248.

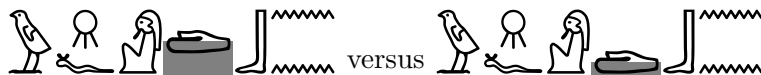
For the special case of overlays, we assume that either of the two overlaid groups covers the entire surface of the overlay. For example, in , we can mark the top or bottom occurrence of  as damaged to give  or , respectively. If both occurrences of  are marked as damaged, or if  is marked as damaged, then in both cases this results in .

If the base sign of an insertion is marked as entirely damaged, then the entire surface of the group is shaded, even the corners where there are insertions that are not marked as damaged.

We also need to decide where the four quarters of a sign meet. This is most appropriately the center point of the bounding box. To motivate this, consider two legitimate ways to render an inscription with a low sign appearing on its own in a top-level group. In the first rendering, the sign is vertically centered, and in the second rendering, inspired by a similar feature in JSesh [4], the sign rests on the baseline:









If only the bottom half of the sign is marked as damaged, then the second rendering should shade a smaller area than the first rendering, to avoid the suggestion that the entire sign is damaged:



5 LALR(1) grammar

Typical parser generators such as Yacc and PLY require a grammar to be LALR(1). However, the grammar in Table 2 has a great number of LALR(1) conflicts. Similarly to what we did earlier in L2/18-236, we have transformed the latest syntax to LALR(1), resulting in Tables 6 and 7. We assume here that **literal**, **singleton**, **placeholder**, **enclosure_open** and **enclosure_close** are specified by regular expressions and

handled by the lexical analyzer. By the principle of ‘longest match’, an opening delimiter followed by  or  should be analyzed as a single token of type **enclosure_open**, rather than as a **singleton** followed by a separate **enclosure_open** consisting only of the  or . Similarly,  or  and an immediately following closing delimiter are together analyzed as an **enclosure_close**.

6 HieroJax

An early JavaScript implementation of the first 9 controls was mentioned in L2/18-236. This implementation was built on top of an existing implementation of RES [2]. RES shares a number of primitives with Unicode, in particular the insertion primitives, but the rendering of these is slightly different. In RES, insertion remains strictly within the bounding box of the base sign (unless whitespace is explicitly added to that bounding box). The insertion primitives in Unicode however are more appropriately implemented to allow an inserted group to protrude from the bounding box. Moreover, the RES implementation rendered hieroglyphic text using HTML canvas, which suffers from pixelization.

An entirely new, open-source implementation of Unicode encoding of Ancient Egyptian was released in May 2023, named **HieroJax** [3]. It is comparable to MathJax [1], in that it renders complex expressions in web pages using JavaScript. It offers a choice of three technologies:

- SVG,
- DOM elements (CSS and HTML), and
- HTML canvas.

For most applications, SVG seems to be the best choice.

Apart from supporting hieroglyphic text in web pages, with the full set of 38 Unicode controls and 1072 hieroglyphs currently in Unicode, HieroJax also offers:

- a graphical editor,
- a means to download a fragment of hieroglyphic text as PNG or SVG image,
- automatic conversion from Manuel de Codage to Unicode,
- automatic conversion from RES to Unicode, and
- the three JSON files mentioned in Sections 3, in combination with graphical representations to make them easier to interpret.

The graphical editor displays the hierarchical structure of groups. By placing the focus on a subgroup, individual elements can be changed, added or removed. This avoids the need to re-type an existing encoding if only small changes are required. The editor guarantees that a created encoding is syntactically correct. The user interface also indicates which rotations and insertions are allowed by the two relevant JSON files. This can be overridden if a new rotation or new insertion is found, and HieroJax aims to produce an acceptable rendering regardless of what is allowed by the JSON files.

Acknowledgements

Many thanks go to Andrew Glass for continued collaboration and to Serge Rosmorduc, Daniel Werning, and Deborah Anderson for advice on philological brackets.

Table 6: LALR(1) grammar (Part 1/2)

```

fragment ::= top_groups
top_groups ::=  $\varepsilon$  | top_groups group | top_groups singleton
groups ::=  $\varepsilon$  | groups group
group ::= ver_group | hor_group | basic_group | literal
ver_group ::= ver_subgroup rest_ver_group
rest_ver_group ::= [ : ] ver_subgroup | [ : ] ver_subgroup rest_ver_group
br_ver_group ::= [ ( ] ver_subgroup rest_br_ver_group
rest_br_ver_group ::= [ : ] ver_subgroup [ ) ] | [ : ] ver_subgroup rest_br_ver_group
br_flat_ver_group ::= [ ( ] literal rest_br_flat_ver_group
rest_br_flat_ver_group ::= [ : ] literal [ ) ] | [ : ] literal rest_br_flat_ver_group
ver_subgroup ::= hor_group | basic_group | literal
hor_group ::= hor_subgroup rest_hor_group |
             literal rest_hor_group |
             [ [ ] hor_subgroup opt_bracket_close opt_rest_hor_group |
             [ [ ] literal opt_bracket_close opt_rest_hor_group |
             hor_subgroup [ ] opt_rest_hor_group |
             literal [ ] opt_rest_hor_group
opt_rest_hor_group ::=  $\varepsilon$  | rest_hor_group
rest_hor_group ::= [ * ] hor_subgroup opt_rest_hor_group |
                  [ * ] literal opt_rest_hor_group |
                  [ * ] [ [ ] hor_subgroup opt_bracket_close opt_rest_hor_group |
                  [ * ] [ [ ] literal opt_bracket_close opt_rest_hor_group |
                  [ * ] hor_subgroup [ ] opt_rest_hor_group |
                  [ * ] literal [ ] opt_rest_hor_group
br_hor_group ::= [ ( ] hor_subgroup rest_br_hor_group |
                 [ ( ] literal rest_br_hor_group |
                 [ ( ] [ [ ] hor_subgroup opt_bracket_close opt_rest_br_hor_group |
                 [ ( ] [ [ ] literal opt_bracket_close opt_rest_br_hor_group |
                 [ ( ] hor_subgroup [ ] opt_rest_br_hor_group |
                 [ ( ] literal [ ] opt_rest_br_hor_group
opt_rest_br_hor_group ::= [ ) ] | rest_br_hor_group

```

Table 7: LALR(1) grammar (Part 2/2)

$\text{rest_br_hor_group} ::=$ $\boxed{*}$ hor_subgroup $\text{opt_rest_br_hor_group}$ $|$
 $\boxed{*}$ **literal** $\text{opt_rest_br_hor_group}$ $|$
 $\boxed{*}$ $\boxed{[}$ hor_subgroup opt_bracket_close $\text{opt_rest_br_hor_group}$ $|$
 $\boxed{*}$ $\boxed{[}$ **literal** opt_bracket_close $\text{opt_rest_br_hor_group}$ $|$
 $\boxed{*}$ hor_subgroup $\boxed{]}$ $\text{opt_rest_br_hor_group}$ $|$
 $\boxed{*}$ **literal** $\boxed{]}$ $\text{opt_rest_br_hor_group}$
 $\text{opt_bracket_close} ::= \varepsilon$ $|$ $\boxed{]}$
 $\text{br_flat_hor_group} ::= \boxed{(}$ **literal** $\text{rest_br_flat_hor_group}$
 $\text{rest_br_flat_hor_group} ::= \boxed{*}$ **literal** $\boxed{)}$ $|$ $\boxed{*}$ **literal** $\text{rest_br_flat_hor_group}$
 $\text{hor_subgroup} ::= \text{br_ver_group}$ $|$ basic_group
 $\text{basic_group} ::= \text{core_group}$ $|$ insert_group $|$ **placeholder** $|$ enclosure
 $\text{insert_group} ::= \text{core_group}$ insertion $|$ **literal** insertion
 $\text{br_insert_group} ::= \boxed{(}$ core_group insertion $\boxed{)}$ $|$ $\boxed{(}$ **literal** insertion $\boxed{)}$
 $\text{insertion} ::=$ $\boxed{\text{in_group}}$ opt_bs_ins opt_te_ins opt_be_ins opt_m_ins opt_t_ins opt_b_ins $|$
 $\boxed{\text{in_group}}$ opt_te_ins opt_be_ins opt_m_ins opt_t_ins opt_b_ins $|$
 $\boxed{\text{in_group}}$ opt_be_ins opt_m_ins opt_t_ins opt_b_ins $|$
 $\boxed{\text{in_group}}$ opt_m_ins opt_t_ins opt_b_ins $|$
 $\boxed{\text{in_group}}$ opt_t_ins opt_b_ins $|$
 $\boxed{\text{in_group}}$ opt_b_ins $|$
 $\boxed{\text{in_group}}$
 $\text{opt_bs_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{opt_te_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{opt_be_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{opt_m_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{opt_t_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{opt_b_ins} ::= \varepsilon$ $|$ $\boxed{\text{in_group}}$
 $\text{in_group} ::= \text{br_ver_group}$ $|$ br_hor_group $|$ br_insert_group $|$
 core_group $|$ **literal** $|$ **placeholder** $|$ enclosure
 $\text{core_group} ::= \text{flat_hor_group}$ $\boxed{+}$ flat_ver_group
 $\text{flat_hor_group} ::= \text{br_flat_hor_group}$ $|$ **literal**
 $\text{flat_ver_group} ::= \text{br_flat_ver_group}$ $|$ **literal**
 $\text{enclosure} ::= \text{enclosure_open}$ groups enclosure_close

References

- [1] NumFOCUS Foundation. MathJax. <https://www.mathjax.org/>, 2019.
- [2] M.-J. Nederhof. RES (Revised Encoding Scheme). <http://mjn.host.cs.st-andrews.ac.uk/egyptian/res/>, 2019.
- [3] M.-J. Nederhof. HieroJax. <https://nederhof.github.io/hierojax/>, 2023.
- [4] S. Rosmorduc. JSesh. <http://jseshdoc.qenherkhopeshef.org>, 2023.