JSesh Demonstration File

This file (allmdc.gly) demonstrates Jsesh capabilities
as far as the Manuel de Codage is concerned
It contains tests for the usual constructs
Its also a tutorial about the *Manuel de Codage*

Jsesh Is a work in progress.

Encoding hieroglyphic texts, tips and trics

Before describing the *manuel de codage* itself, I'd like to make a point or two about how one should encode a hieroglyphic text.

Typically, you have a source, which might be a) a printed source, with typeset hieroglyphs, b) a photograph or an acurate facsimile of the original text, or c) handwritten hieroglyph from an egyptological publication (for instance, the *Urkunden).*

You must decide how faithfull to the original you must be, and it's not a simple question. The first point is that you can be sure that your *Manuel de Codage* text can't be a facsimile, so, one way or another, it will betray the original. If the text is ultimately a hieratic text, you are already creating something completely different anyway.

Now, when you have a sign, should you look for the most precise variant for it, or use a somehow standardised one? To answer this, you should wonder if this variant is relevant, in the text, and in the use you intend to make of it. For instance, in the Kana√Øs texts by Sethi the Ist, the first person pronoun uses many variants of A40 ( ). If you intend the text to be for grammatical study, it's not very relevant. If you wonder whether it's a free play to add diversity to the text, then you may take the time to use code the variants. As a rule, if you are a beginner in Egyptian, I'd advise you to encode the "standard" sign, as it will force you to read the text, not to copy signs. The problem you have when you want to be very accurate is that you will find many cases of variants which have no encoding anyway.

When your source is an handwritten version of a text, like in the Urkunden, beware of one point: unless a sign is very particular, an egyptologist will usually draw his glyphs in the easiest way. That means, for instance, he will use V23A ( ) instead of (V22) which is the normal hieroglyphic sign. You must understand that, and when a sign selection is simply caused by the individual hand of the author, render it with the "standard" sign. On the other hand, when the author has drawn a very specific sign form, this demands more work.

Encoding hieratic texts

The above rule is even stronger for hieratic texts. There is no point in making a distinction between and , as both are rendered the same in hieratic!

Now, there are a number of peculiarities of hieratic which deserve more care. Those have been pinpointed by sir Alan Gardiner in his article "The Transcription of New Kingdom Hieratic", *JEA* 15 (1929), p. 48-55. The rendering of hieratic texts should hint at the way the original signs are placed, in order to allow the reader to understand the reason for the rendering, and to go back to the original when necessary.

JSesh includes a number of signs which are usefull for rendering hieratic texts:

(Ff1)	a sign particular to Ramesside texts, which serves as a kind of "wildcard". It is **not** the same as Z5 ( ).

(Ff100) a non-standard sign, "dot space filler". Use it to render the various dots a scribe may use, when they are not a ponctuation.

(Ff101) a non standard sign, horizontal space filler. Use it to render various meaningless horizontal sign the scribe may use to fill voids.

JSesh includes also specific signs for numbers in hieratic texts. In those texts, the "determinative" is often smaller than the "digit" . Careless rendering might cause to be read *hrw 3,* whereas the original has , which is clearly *hrw 2.*

Description of the manuel de codage

The *Manuel de codage* is a system for describing hieroglyphic texts in ASCII characters

It's the result of the work of an international working group, and it's described in :

J. Buurman, N. Grimal, M. Hainsworth, J. Hallof, D. van der Plas, *Inventaire des signes hi√©roglyphiques en vue de leur saisie informatique,* Paris, 1988 (first ed. 1984)

The *manuel* was inspired by a software called GLYPH, by J. Buurman ; the first versions of glyph can be traced back to the early 70s

Most hieroglyphic editors use a more or less modified version of the *manuel,* notable modifications have been proposed in 1994 by H. van den Berg for winglyph

In the Pisa 2002 *Table ronde Informatique et √âgyptologie,* M.-J. Nederhoff has proposed a very interesting *Revised Encoding System,* which addresses some of the *manuel*' shortcomings.

We plan to take it into account later on.

Simple manuel de codage codes

The basic principle of the *manuel* is that a sign is described by its code in Gardiner's fonts.

For example, is "A1" and is "G5"

Variant signs (those with a "*" in Gardiner's list) are indicated by uppercase letters after the code

For instance, is "A14A", vs. which is "A14"

A number of signs can also be described by their phonetic value.

For instance, can be coded "b".

A complete list of these will be given in JSesh's help.

The codes for the egyptian consonnants are the following :

ꜣ	A	*ỉ*	i	*y*	y	*ꜥ*	a
w	w	*b*	b	*p*	p	*f*	f
m	m	*n*	n	*r*	r	*l*	l
h	h	*ḥ*	H	*ḫ*	x	*ẖ*	X
z	z	*s*	s	*š*	S	*ḳ*	q
k	k	*g*	g	*t*	t	*ṯ*	T
d	d	*ḏ*	D				

For uniliteral signs, there are two special codes for some frequent variants :

is coded "w", and is coded "W"

is coded "m", and is coded "M"

is coded "n", and is coded "N"

To these codes, the manuel adds also two groups :

"nTrw" for

"nn" for

Grouping of signs

Grouping hieroglyphs is done with a few simple operators :

"-" separates cadrats. For instance, is coded "G41-A"

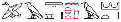
":" stack groups. For instance, is coded "p:n"

"*" group signs on the same level in a cadrat.

For instance, is coded "p*t:pt"

More complex groups can be built using parenthesis :

pA-A-N16:N23*Z1- **n:(x:t)*U30** -xAst:t*Z1

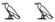


Special signs

a cadrat-long space is coded ".." :

A-..-A-

a half-cadrat-long space is coded "." :

 A-.-A-

black punctuation point ● is coded "O" : ● is O:.

red punctuation point ● is coded "o" : ● is o:.

Unknown codes

When JSesh doesn't know a sign, it will replace it in display by its code e.g. :

A9991-n:A9991*A1 gives

A9991
A9991

Shading

When nothing is visible under a shading, you can use the following codes :

"//" for cadrat-size shading :

"/" for quarter shading :

"v/" for vertical shading :

"h/" for horizontal shading :

Those are "real" signs, which can be combined with others: p:h/ gives .

They are seldom of use, however, as the method below is much more practical:

For shading parts of a cadrat, use "#", followed by the figures 1,2,3,4

'1' fills the start-top corner of the cadrat: p*t:pt#1 gives

'2' fills the end-top corner of the cadrat : p*t:pt#2 gives

'3' fills the start-bottom corner of the cadrat: p*t:pt#3 gives

'4' fills the end-bottom corner of the cadrat: p*t:pt#4 gives

These codes can be combined :

p*t:pt#13 gives

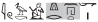
p*t:pt#14 gives

p*t:pt#1234 gives

For shading large parts of texts, use "#b" (shade begin) and "#e" (shade end) :

zone shading

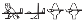
i-W-bA:k-A1-#b-Xr:r-n:h:t-#e-xt:Z1



Sign overlapping

It's possible to combine signs using the operator "##". The result is not always very good, though.

Exempli gratia:



m##a-a##b-M3##M1-M1##M3

Note that the "official" operator for this is "#", but I suggest you use "##" instead,

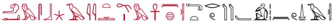
for"#" has been much over abused (it means too many things).

In any case, JSesh will transform all "#" into "##" when saving !

Red zones

A zone of text can be written in red with "$r" (begin red) and $b (begin black) operators

$r-HAt:a_-m_-s-b-sbA-A-i-i-t:Y1-arq:Z1_-m_-anx-n:x-Y1v_-$b-mt:t-r:W-Dba-Dba-Hw-A2-Z3_-n_-W-DA-A-Y1v_-



Individual signs can be written in red by writing "\red" after them

This is a JSesh specificity.

p*p:p\red*p-

□□
□□

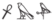
Sign Modifications

Inversion, rotation

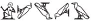
A sign can be reversed by writing "\" after it : "A1\" is

In JSesh, you can turn signs by any angle, using \R and the angle in degrees

anx\R30-G5-G7

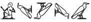


A1-A\R90-A\R180-A\R270-A\R360-



These two features can be combined :

A1-A\R90-A\R180-A\R270-A\R360-



Note that the "\R" operator is non standard.

It was proposed for Winglyph in 1994, but apparently not used.

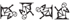
The original manuel did not know about rotations at all

and most implementation use only 90¬∞ rotations, with two operators:

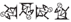
\r + 1, 2, 3, 4: rotate 90, 180, 270 or 360 ¬∞

\t + 1, 2, 3, 4: same, plus reverse.

A1\r1-A1\r2-A1\r3-A1\r4-



A1\t1-A1\t2-A1\t3-A1\t4-



Scaling

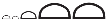
It's possible to scale a sign by writing "\" and a percentage.

The sign's base size will be modified accordingly :

A1-A1\80-A1\50-A1\20

The percentage can even be greater than 100:

t-t\80-t\200-t\400-t\1000-



note that the sign is limited by the cadrat maximal size.

This operator wasn't in the 1988 format, but is widely used.

This can be used to customize the effect of scaling in cadrats :

Let's consider the group , written "zA:zA*zA".

Notice that the upper is a bit larger than the others.

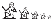
We can correct this with scaling :

zA\70:zA\70*zA\70-

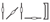
In traditionnal typography, the sizes of fonts are ruled by a decrease of a factor sqrt(2).

This gives a nice visual effect, and you might try using these factors :

`A1-A1\70-A1\49-A1\35-A1\24`



A stress test for effect combination : rotated scaled sign (any use ?)

rotation test with non-square sign : 

Ignored signs

(JSesh extension)

As cadrats don't respect words boundaries, you might find yourself quoting a word, or a text,

which start in the middle of a cadrat. In JSesh, the "\i" operator means that a sign is irrelevant.

it will be drawn in very light gray, simply for space filling.

`M\i:p*t:pt`

Wide signs and groups (non standard extension)

In Late Egyptian texts, in particular hieratic texts, some groups can be very wide.

Often, they contain a hieroglyph which is very elongated, an lots of smaller signs below it.

The normal scaling rules would reduce the cadrat's width, and give both a difficult to read and unfaithful rendering.

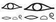
Thus, JSesh provides the operator "\l". It says that the current sign can have any width it likes.

without it, : "m-xt:x\80*t\80*D54\80" gives

with it, : "m-xt\l:x\80*t\80*D54\80" gives

The best results are obtained by combining this and scaling :

`ir\200:r*t*W-ir\200\l:r*t*W`



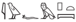
Grammatical codes

The *manuel* states that " " (space) after a sign means that this sign ends a word, and that " " (two spaces) mark a sentence end

JSesh stores this information along with the signs, but currently doesn't do anything with it.

As an alternative, after the *Winglyph* practice, you can use underscores instead of spaces.

`i-w-=f_-m_-p*t:pt_`



Absolute sign positionning

JSesh borrows the syntax from macscribe; the interpretation is made up from scratch, though.

A group made with the operator ** will use *absolute positioning.*

each sign in such a group should be followed by a position of the form "{{x,y,s}}"

where x is the distance of the sign to the *left* of the group

y is the distance to the *top* of the group

s is the scale of the group (as a percentage)

If nothing is specified, the default values are 0,0,100.

x and y are expressed in 1/1000 of the height of an (A1) sign

`stp{{0,0,100}}**n{{0,800,100}}**ra{{700,0,70}}`

more fun with positioning :

`A1{{200,20,100}}**T31\R50{{120,0,40}}`

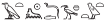
Important note: We previously used &&, which was in fact a mistake, as its meaning in Macscribe is indeed quite different. We intend to correct this in a future release (with a mecanism for compatibility). In future versions of JSesh, the operator for explicit placement will be **.

Ligatures

A ligature system was introduced quite soon, in the form of special group. This is a "standard" extension to the manuel de codage.

JSesh knows about a number of special groups. The operator "&" means that signs are to be grouped together in a special way.

G1&X1-stp&n&ra-D&d-Ax&x-t&w&t- gives :

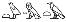


Complex ligatures

JSesh (from version 2alpha20 and later) has a mecanism inspired from macscribe's syntax for ligaturing a sign and a complex group.

To tell it fast, each sign has up to two rectangular zone in which a group can be inserted. One zone is "before" the sign, the other is "after" the sign. Example : in and , the groups and are inserted in front (first area) of . This is written :

(Z1:Z1*Z1)^^^nTr and X2^^^nTr

It works well for most birds : 

The other zone is "after sign":

is written : Ax&&&x-..-A&&&t-..-w&&&t-..-ns&&&(mSa*Z3)-..-D&&&(d:d:t)

Both operators can be combined : , t^^^w&&&t

When JSesh has no explicit data about a sign, it tries to guess. Note that in future versions of JSesh, the &&& and ^^^ operator will be replaced by && and ^^.

User Modifiers

To allow extensions of the manuel, JSesh can parse all modifiers after a sign, if they respect this rule :

* a modifier is introduced by "\"

* it contains a sequence of letters, which is his name, and a sequence of digits, which is his value.

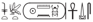
A1\IdontMeanAthing\number1000

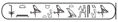
Cartouches, Serekh, etc..

Cartouches are written between "-<-" and "->-".

cartouche

sw:t-bit:t-<-ra-mn:n-xpr->-anx-DA-s





<-sw-t:n-G7-N41:t*B1-G7-t:A-A51-s-y:B1-G7->

serekh

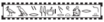
<S-E1:D40-mAat-mr->

Hut-sign

<H-Z3-A19->

Enclosure (not yet drawn, but correctly read)

<F-bA-A-y:r-i-i-Aa18-Z1-mr:r-W:t-A2-r-D:t:N17->



Some parts of the cartouche may be absent, or may be drawn in a different place.

In this case, the letter-code (if any) should be lowercase.

a figure indicates what should be drawn ; the available constructs are :

cartouche : nothing = 0 ; no-knot : 1 ; knot : 2

<1-ra-mn:n-xpr-1>

<1-ra-mn:n-xpr-0>-!?800<1-ra-mn:n-xpr-0>!?800<2-ra-mn:n-xpr-1>

hut-sign : 0: nothing, 1: no square, 2: square in the bottom; 3: square in the top.

<h0-Z3-A19-h3>

<h3-nTr-h1>

<h2-nTr-h2>

Serekh

<S-E1:D40-mAat-mr->

<s2-E1:D40-mAat-mr-1>

Alphabetic Text

alphabetic text is introduced by "+" followed by :

l for latin script

i for *italic* script

b for **bold** script

t for translitteration *(ꜥnḫ wḏꜣ snb)*

c for coptic script (not yet)

h for hebrew script (not yet)

g for greek script (not yet)

r for cyrillic script (not yet)

+ for comments (comments are invisibles)

s to revert to hieroglyphs

Notes ** the manual says nothing about the encoding of coptic, greek, hebrew or cyrillic !*

** you should go back to the hieroglyphic mode (+s) before the end of the line*

** Translitteration is typed using the encoding explained above for hieroglyphs.*

** Uppercase translitteration letters are supposed to be coded like this :*

"^" + translitteration code. (JSesh doesn't support this yet).

"^pAnb" for "P ꜣnb".

Avoid at all the direct use of the codes supported by your particular font

(except probably when the unicode standard for translitteration appears).

Exempli gratia :

+l this is a text +iwith italics and +s nTr-md-Z3A++ and a comment+s-!

gives

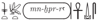
this is a text *with italics and*

In Jsesh, you can easily include a + sign in a latin text ;

simply write \+ (this is not in the *manuel*)

You can write text in cartouches :

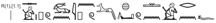
sw:t-bit:t-<-+tmn-xpr-ra+s->-anx-DA-s



Superscript (line numbers)

Superscript is done by writing "|*your text*-" . e.g

|R(1),[1.1]-z:A1*Z1-p-Z7-wn:n-i-n:p*Z7-E15-x:D43-Z7-Y1:n-A1-r:n-f-



Line skips

Line skips are indicated by "-!"

If a line skip is of the form by "-!=*number*%", a space of *number* percent times the normal skip is skipped

e.g. four time the normal skip : -!=400%

next line

New pages

new page are indicated by -!!

horizontal lines

A line can be drawn with the construct "{l*number,number*}".

The first number is the start position, the second is the ending position.

The unit is 1/200th of the width of a 𓈖 sign

The letter can be either 'l' or "L", for a thin or bold line

{l200,800}

{L200,800}

Tabulation

tabulation is indicated by "?*number*"; the number is expressed in 1/200th of the size of an 𓈖 sign.

Exempli gratia : to align our examples, we use "?800" :

?800+lan example

an example

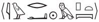
the 𓀁 sign should appear after the 𓀀 below :

?400-A1-?100-A2-

Ecdotic codes

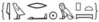
Added

-[&-i-w-r:a-ra-m-p*t:pt-&]



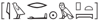
Superfluous

-[{-i-w-r:a-ra-m-p*t:pt-}]



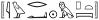
Erased

-[[-i-w-r:a-ra-m-p*t:pt-]]



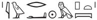
Previously readable

```
-["-i-w-r:a-ra-m-p*t:pt-"]
```



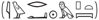
Added by scribe

```
-['-i-w-r:a-ra-m-p*t:pt-']
```



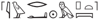
Minor editor addition **(Non standard. Initially in MacScribe)**

```
-[(-i-w-r:a-ra-m-p*t:pt-)]
```



Dubious reading **(Non standard. Initially in MacScribe)**

```
-[?-i-w-r:a-ra-m-p*t:pt-?]
```



The original *manuel* suggested that philological indications worked as parenthesis,

but this doesn't correspond to actual typographical practices.

Hence, winglyph(tm) considers these are simple signs.

which means that you don't need to match them.

JSesh supports both this interpretation and the original one (which was supported in tksesh).

Currently, it loads any file with a ".gly" extension using the winglyph rule,

and any file with a ".hie" extension using tksesh's rule

In cadrat :

```
p*[[*t*]]:pt_-p*[[*t:pt-]]
```

Support for deprecated *Manuel* practices

JSesh is able to read texts that don't follow the exact manuel de codage.

For instance, it's quite forgiving about spaces.

However, when saving a text, *JSesh* will write it in a very normalized form, to ensure a maximal readability.