Manuel du FORTH

Éditeur, Interpréteur et Compilateur

pour Atari ST(e)/TT/Falcon

version 0.8.3

16 mai 2026

Programmé en assembleur 68k
d'abord avec l'assembleur MadMac puis
à l'aide du logiciel Assemble (Brainstorm)

Sommaire

Vue d'ensemble

Le FORTH se présente comme un programme monobloc (pas de fichier supplémentaire) contenant un éditeur et un interpréteur. Il est (normalement) entièrement compatible GEM, se lance dans n'importe quelle résolution et supporte même MultiTOS.

Au lancement une fenêtre s'ouvre et l'interpréteur de commandes est prêt à recevoir vos instructions. Le FORTH fait une distinction entre majuscules et minuscules, les fonctions devront donc être employées telles qu'elles sont décrites. Toute instruction est séparée des autres par un ou plusieurs espaces. Voici quelques mots utiles pour commencer:

system

retour au bureau.

edit

passe dans l'éditeur, touche ESC pour revenir à l'interpréteur.

desk

redonne le contrôle de la souris à l'AES. La fenêtre FORTH peut être redimensionnée ou déplacée, les accessoires (ou d'autre applications sous MultiTOS) peuvent être appelés. Pour revenir au FORTH on choisit l'option "retour" du menu.

.

le point affiche le contenu du sommet de la pile (ce nombre est alors perdu). Ce mot servira à contrôler les résultats de vos premiers pas en FORTH.

```
2   3   +   .       \ affiche le résultat de l'addition 2+3
```

..

même opération que le mot précédent sauf que le nombre n'est pas perdu, il reste disponible sur la pile.

```
2   3   +   ..      \ affiche le résultat de 2+3, et 5 reste sur la pile
6   *   .           \ calcule ensuite 6*5 et affiche donc 30.
```

ver

Affiche la **version** de l'interpréteur (STE ou TT, puis numéro, puis date), donne la **place libre** pour stocker les nouveaux noms de mots ainsi que leurs adresses.

Touche Help (ou Shift F1)

rappelle les principales commandes.

Shift Ins (ou Shift F10)

donne accès à la mémoire des quatre dernières commandes entrées. Tant que Shift reste appuyé, les quatre lignes vous sont proposées pendant une seconde chacune et, dès que la ligne désirée apparaît, relachez la touche Shift. Votre ligne peut être réutilisée et éventuellement modifiée.

Les possibilités du FORTH:

- arithmétique complète sur 4 octets, calculs en 35 bases différentes.
- calculs réels au format DOUBLE du MC68882 (aussi sur 68000).
- variables et tableaux de nombres ou de chaines.
- types structures (struct du C) ou ensembles (setof du pascal).
- nombreuses boucles (avec compteur, while, until ...)
- appel des fonctions AES, VDI, TOS par leur nom (le même qu'en C).
- gestion (sommaire) du son DMA et de la carte ST REPLAY16.
- gestion du Supercharger (carte Nec V30 + 1Mo RAM)
- gestion des modules M&E de Parx (import/export images)
- appels possibles à un assembleur extérieur

Le fichier FORTH.INF

Il doit se trouver dans le répertoire courant lors du lancement du FORTH, sinon ce sont les valeurs par défaut qui sont choisies.

Ce fichier peut être généré et modifié par le programme **FORTHINS.PRG** qui facilite grandement la gestion des fichiers d'information de FORTH et du compilateur. Il s'accompagne de deux fichiers ressource : FORTHINS.RSC (en **français**) et FORTHINS.ENG (en **anglais**, à renommer en RSC pour pouvoir l'utiliser).

Il comporte 5 ou 6 lignes et est éditable avec un traitement de texte rudimentaire. Les lignes doivent être terminées par CR+LF et les informations doivent commencer dès la première colonne:

Ligne facultative

Permet de régler certains flags et doit commencer par le signe # et se trouver au tout début du fichier. Les flags sont contigus.

Wnnn : règle la taille de la fenêtre Forth à nnn% de l'écran. Par défaut, c'est plein écran.

M : permet d'imposer la souris en permanence. Par défaut, le curseur est éteint si l'utilisateur ne le force pas.

B : règle la gestion des erreurs de bombes dans le FORTH, par défaut, l'interpréteur intercepte ces vecteurs et affiche une erreur. Si B est présent, on laisse le système gérer les bombes (dans ce cas l'interpréteur plante et retour au bureau).

F : mode fast activé au lancement. Par défaut c'est le mode slow qui est actif.

Lnnn : force le code de la langue à la valeur indiquée. Par défaut, c'est celui de la ROM qui est utilisé (0=US, 2=FR...). Pour l'instant, cette valeur est simplement utilisée par les fonctions datetostr et strtodate pour choisir entre JJ/MM/AA et MM/JJ/AA. Voir la variable **langage**.

Annn : décide si certains avertissements doivent apparaître (alertes) ou non. La valeur nnn est un masque de bits:

bit0 à 1 : avertit si on écrase un fichier avec saveb/append.
bit1 à 1 : avertit si un bloc est marqué pour saveb/append
bit2 à 1 : avertit en quittant si le source a changé dans l'éditeur.
bit3 à 1 : affiche les avertissements de compilation.

Par défaut, toutes les alertes sont affichées.

S : ignore les standards FORTH (impacte **.s** sur le sens d'affichage de la pile).

exemple:

```
#W80MFA2    ; 80% de l'écran et souris présente, mode Fast, alerte bloc
```

Première ligne

Permet de calculer la taille maximale d''un texte sous l'éditeur, son format est:
%xxx+yyyyy: c'est à dire xxx% de la mémoire libre plus yyyyy Ko
%xxx-yyyyy: c'est à dire xxx% de la mémoire libre moins yyyyyKo
Par exemple, %0+100 réservera 100Ko quel que soit la mémoire libre.
Par défaut, c'est %17+0 qui sera pris en compte (1/6 de la mémoire).
Ligne en liaison avec 'edit', voir l'éditeur.

Deuxième ligne

Permet de calculer la taille maximale du dictionnaire sous l'interpréteur, son format est le même que pour le texte édité:

Par exemple, %50+100 reservera la moitié de la mémoire plus 100Ko (après que l'éditeur soit réservé).

Par défaut c'est %75+0 qui sera pris en compte (3/4 de la mémoire).

Ligne en liaison avec 'free', 'full', voir le dictionnaire et le pré-processeur (focntion >exec).

Troisième ligne

Définit le chemin d'accès aux fichiers sources *.FOR.

Ligne liée à 'saveb', 'loadb'.

Quatrième ligne

Définit le chemin d'accès aux fichiers *.TXT (lors de l'exportation ou importation ASCII).

Ligne liée à 'export' et 'import'.

Cinquième ligne

Définit le chemin d'accès aux fichiers FORTH.HLP et FORTH.HYP, il se termine par un slash. Elle peut être complétée, après une virgule, par le chemin complet d'accès à ST_-GUIDE ou HYPVIEW en mode multitâche. Si vous travaillez sous TOS, cette application est normalement déjà chargée en accessoire.

Ligne liée à l'aide en ligne.

Un fichier possible est:

```
%0+100
%50+100
F:\FORTH\SOURCES\*.FOR
F:\FORTH\TEXTES\*.TXT
F:\FORTH\HELP\,F:\OUTILS\HYP105\HYPVIEW.APP
```

Le fichier AUTOEXEC.FOR

C'est un fichier qui sera automatiquement chargé et exécuté au lancement de l'interpréteur sous la condition que FORTH.INF existe afin de fournir le chemin par défaut des fichiers FOR.

Si on ne désire pas exécuter ce fichier, il faut maintenir l'une des touches Shift lors du lancement (par exemple si le fichier déclanche une erreur).

Exemple de fichier AUTOEXEC.FOR:

```
v_hide_c
." Version     : " ver cr
." RAM FORTH   : " free . cr
." RAM Système : " -1 malloc . cr
cr
." Bienvenue dans le FORTH !"
0 v_show_c
```

Tutoriel rapide

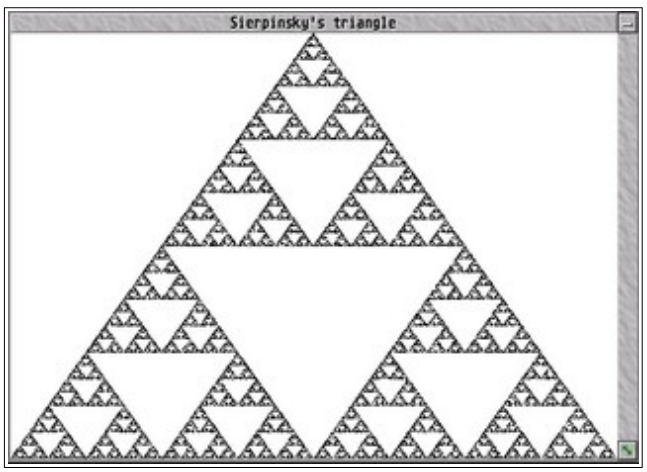
Voici une prise en main rapide de l'ensemble des fonctions principales du FORTH, nous verrons comment **utiliser l'éditeur**, **savegarder** le programme, l'**exécuter** puis créer un **programme indépendant** avec le compilateur.

Nous allons dessiner le **triangle de Sierpinski** d'une manière... aléatoire.

Imaginons trois personnes disposées en triangle, appelant un chien très indécis en lui proposant à manger.

Chaque fois que le chien semble faire un choix, il se dirige vers l'une des personnes, mais ne parcourt que la moitié de la distance et s'assoit. Il se laisse alors à nouveau tenter et se redirige aléatoirement vers l'une des trois personnes, mais s'arrête à mi-distance.

Si on marque d'un point chaque endroit où le chien s'est assis, le triangle de Sierpinski apparaît !



Editer

Lancer le FORTH (FORTH.PRG ou FORTHSTE.PRG) et tapez:

```
edit [enter]
```

Vous entrez sous l'**éditeur**.

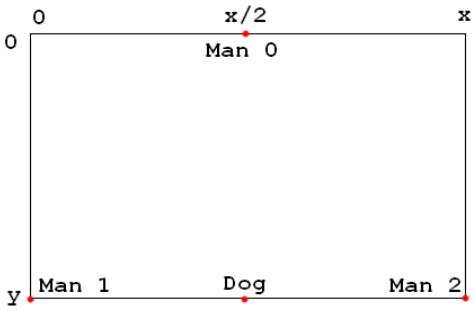
Pour commencer, définissons nos variables, les trois personnes auront leurs coordonnées rangées dans deux tableaux pour les abscisses X et les ordonnées Y. Le chien aura deux variables simples pour ses coordonnées.

```
3 array MenX
3 array MenY

variable DogX
variable DogY

>comp
```

Note: >comp *vous sera expliqué ultérieurement, mettez-le!*

Ensuite il nous faut initialiser ces coordonnées selon la taille de la fenêtre de l'interpréteur.

La fonction AES **wind_get** associée à **get_xyxy** nous donnera les coordonnées des points extrêmes de la fenêtre en mode relatif (par défaut), le point du haut (0,0), celui à l'opposé (x,y).



L'homme 0 sera en haut au centre: (x/2 ; 0)
L'homme 1 sera en bas à gauche: (0 ; y)
L'homme 2 sera en bas à droite: (x ; y)
Le chien en bas au milieu: (x/2 ; y)

```
: init
   fastopen 4 wind_get \ fastopen donne le handle
   intout 2+ get_xyxy  \ sur la pile 0 0 x y
   dup    1 MenY !     \ MenY(1) = y
   dup    2 MenY !     \ MenY(2) = y
            DogY !     \ DogY = y
   dup    2 MenX !     \ MenX(2) = x
   2/ dup 0 MenX !     \ MenX(0) = x/2
            DogX !     \ DogX = x/2
          1 MenX !     \ MenX(1) = 0
          0 MenY !     \ MenY(0) = 0
;
```

Vous n'êtes pas obligés de taper les commentaires qui suivent le "\", si vous le faites, le slash doit être isolé (entouré d'espaces).

A noter:

- le mot **"!"** sert à stocker une valeur dans une variable
- **dup** duplique la valeur du haut de la pile pour l'utiliser sans la perdre
- **2+** et **2/** sont bien en un seul mot, des versions optimisées de 2 + ou 2 / séparés.

Exécuter

Il est temps de tester tout ceci ! Sortez de l'interpréteur avec **Esc**, le FORTH vous dit qu'aucun bloc n'est marqué. Tapez:

```
compilb [enter]
```

et vos mots sont prêts. Pour le vérifier, tapez:

```
vlist [enter]
```

Vous devriez voir la liste suivante:

```
ManX ManY DogX DogY init
```

Essayons le programme:

```
init [enter]
```

A ce stade, vos variables doivent être correctement remplies, par exemple, visualisons les coordonnées du chien:

```
DogX @ . [enter]
```

Affichera l'abscisse du chien au centre et:

```
DogY @ . [enter]
```

Affichera son ordonnée en bas.

```
2 MenX @ . [enter]
```

Affichera l'abscisse de l'homme 2 qui doit être le double de celle du chien. On peut éditer les coordonnées des hommes avec cette boucle:

```
3 0 do i MenX @ . space i MenY @ . cr loop [enter]
```

Retenez:

- le **"@"** dépile la variable et la remplace par son contenu.
- le **"."** dépile le sommet de la pile et l'affiche.
- les variables tableau doivent toujours être précédées de l'indice désiré.
- **space** affiche un espace et **cr** passe à la ligne suivante

Sauvegarder

Si tout fonctionne, on sauvegarde notre source. Tapez:

```
saveb [enter]
```

Et donnez le nom TUTO.FOR au programme. En cas de plantage à un moment, vous pourrez toujours relancer le FORTH et récupérer votre programme avec **loadb**.

Il est temps de retourner sous l'éditeur pour compléter le programme:

```
edit [enter]
```

Nous allons créer la procédure principale **main** qui attendra sur la pile le nombre de points à dessiner. Ce nombre de points sera justement l'argument d'une boucle **ndo..nloop**.

Le XBIOS fournit la fonction **random** qui renvoie un nombre aléatoire sur 24 bits. En prenant le reste de la division par 3 avec **mod**, nous obtenons une valeur 0 ou 1 ou 2. Exactement l'indice nécessaire pour le choix d'un des trois hommes.

Ensuite, comment calculer les coordonnées du point où s'arrêtera le chien à mi-chemin? C'est simplement la moyenne entre ses coordonnées et celles de l'homme, donc:

$$Dogx = \frac{MenX(i) + DogX}{2} \; et \; DogY = \frac{MenY(i) + DogY}{2}$$

```
: main
   cls
   init
   ndo
      random 3 mod
      >r
      r@ MenX @ DogX @ + 2/ DogX !
      r> MenY @ DogY @ + 2/ DogY !
      DogX @ DogY @ 1 v_pmarker
   nloop
;

>comp
```

cls efface l'écran
init appelle notre procédure qui remplit les coordonnées
ndo prend sur la pile l'argument passé à main pour boucler
random 3 mod renvoie un nombre au hasard entre 0 et 2, on l'appellera i
>r envoie i sur la pile des retours (astuce pour éviter une variable)
r@ rappelle i (sans l'effacer de la pile des retours) comme indice pour MenX et calcule une moyenne qu'on stocke à nouveau dans DogX avec "!"
r> rappelle i (en l'effaçant de la pile des retours) comme indice pour MenY et calcule une moyenne qu'on stocke à nouveau dans DogY avec "!"
Cette nouvelle position sert de coordonnées pour la fonction **v_pmarker** qui place un point.

On sort de l'éditeur avec **Esc** et vous pouvez encore sauver votre travail avec:

```
saveb [enter]
```

Notez que le nom du programme est conservé dans le sélecteur de fichiers.
On recompile avec:

```
compilb [enter]
```

Et là, une erreur vous est signalée:

```
Redéfinition d'un mot existant: ManX
```

Le FORTH n'a pas oublié notre travail précédent. Si nous voulons repartir à zéro, il faut lui demander d'oublier avec:

```
forget ManX [enter]
```

Cette fois-ci, compilb fonctionne.
Il est temps de tester le programme:

```
1000 main [enter]
```

Une silhouette du triangle apparaît. Testez d'autres valeurs plus grandes pour le visualiser de manière plus précise.

Le compilateur

Dernière étape, réaliser un programme indépendant. Nous voyons bien qu'il faut compléter ce source car pour l'instant, nous lançons la procédure **main** à la main avec différents arguments.

Il faut automatiser ce système. Nous allons demander à l'utilisateur le nombre de points. Tant que celui-ci n'est pas nul, on trace un nouveau triangle, sinon on quitte.

Voici la boucle à ajouter sous l'éditeur après le dernier **>comp**:

```
fastopen " Sierpinsky's triangle" 2 wind_set
begin
   ." Points : " input
   ?dup
while
   main
repeat
```

La première ligne donne un titre à la fenêtre.
Notez l'**espace après "** pour le titre, c'est un mot indépendant !
De même, un **espace après ."** pour le prompt!
input lit une valeur "n" au clavier.
?dup la duplique si n<>0
while dépile le sommet et poursuit tant que n<>0
n est passé en argument à **main**.

Note: pensez à bien valider la dernière ligne afin que le curseur soit en-dessous de **repeat**.

Sauvons un dernière fois le programme avec:

```
saveb [enter]
```

Et quittons l'interpréteur pour le compilateur avec:

```
system [enter]
```

Lancez le compilateur COMP.PRG ou COMPSTE.PRG. Dans le même dossier doit se trouver la librairie FORTH.LIB ou FORTHSTE.LIB. Pensez à régler vos fichiers COMP.INF ou COMPSTE.INF (*voir en fin de manuel le compilateur*).

Dans le sélecteur de fichiers, cherchez TUTO.FOR et validez.
En fin de compilation, on vous demande où sauvegarder TUTO.PRG, validez.

Il ne vous reste plus qu'à exécuter depuis le bureau TUTO.PRG pour dessiner votre triangle, rappelez-vous de mettre zéro points pour quitter le programme.

Une brève histoire

Au début des années 1980, le langage FORTH m'intriguait déjà, fourni avec le Jupiter Ace il me restait encore inaccessible de par son prix à l'époque.

Une fois indépendant, avec un peu d'argent de côté, j'ai pu m'offrir un Atari 520 STf avec lecteur simple face. C'est par hasard que je découvre dans la bibliothèque de mon université le manuel du Forth sur TO7, l'un des ordinateurs fabriqués par Thomson.

Rapidement conquis, je me mets au travail et, dès 1989, j'écris en GFA Basic un premier interpréteur. J'en vois vite les limites en termes de rapidité et d'occupation mémoire et l'envie de passer par l'assembleur s'impose.

Sauf que... Je ne connaissais pas l'assembleur du 68000, jusqu'ici je n'avais travaillé que sur le TMS9900 du TI-99.

J'ai trouvé un logiciel gratuit : **MadMac Assembler**. Un très bon produit de Motorola qui nécessitait l'utilisation d'un éditeur séparé. Mon choix s'est porté sur **EDIMAX**, un éditeur de texte hyper rapide en monochrome.

Dans un premier temps, je n'avais pas encore l'argent pour m'offrir le moniteur SM124. Je travaillais donc sur un téléviseur en 640×200 avec **SEBRA**, un émulateur monochrome. J'ai dû me fusiller les yeux pendant de longs mois.

Avec les livres sur l'assembleur de l'Atari, celui du FORTH du TO7 me voilà lancé dans la programmation de ce langage original. Lorsque je relis les blocs programmés à cette époque, ils sont très maladroits, peu optimisés, et montrent que je ne maîtrisais pas toutes les subtilités du MC68000. J'ai appris conjointement l'assembleur et le FORTH.

Si au départ c'était une programmation "à l'ancienne", plein écran avec simplement les fonctions GEMDOS pour les entrées sorties, il a fallu passer en fenêtre, utiliser la VDI et l'AES. L'interpréteur s'est étoffé d'un éditeur, d'une compilation en RAM amenant plus de puissance, d'un assembleur intégré (qui fut remplacé ensuite par des appels à un assembleur externe), et d'un compilateur externe permettant de générer un programme indépendant.

Le passage à l'Atari TT fut important avec les nouvelles possibilités du processeur, une mémoire étendue et la présence du FPU. J'avais beaucoup travaillé pour créer des routines de calcul en virgule flottante sur le 68000, et là, tout devenait simple. Mais ce travail n'était pas perdu puisqu'il perdure dans la version STE.

J'abandonnais également MadMac pour **ASSEMBLE** programmé par Brainstorm. L'environnement intégré est d'un confort inégalé et la rapidité de l'assemblage permet de tester rapidement ce qu'on programme.

Même aujourd'hui, avec plus de 400Ko de source, il ne faut que 5 secondes au TT pour générer l'éxécutable.

Le pack s'accompagnait d'un debugger qui devint rapidement indispensable. Tracer un programme pas à pas permet de lever les erreurs.

Plus tard, j'ai dû remplacer ADEBUG par **PureDebugger** qui fonctionnait sur carte graphique.

Ce qui est agréable quand on code un langage de programmation c'est que toute nouvelle connaissance sur la machine se transforme en instructions !

La découverte de SpeedoGDOS fut l'occasion de gérer les fontes vectorielles, la découverte de la carte NOVA l'occasion de gérer les modes TrueColor, le Supercharger fut l'occasion de travailler sur le parallélisme et tout dernièrement, la gestion des modules M&E de PARX a amené son lot d'instructions pour gérer les images.

Nous sommes en 2022 et je continue ma petite aventure... Toujours avec mon duo ASSEMBLE et PureDebugger sur le TT principalement, mais également sur l'Apollo Vampire V4.

Le développement de ce système continuera certainement aussi longtemps que j'aurais un Atari sous la main.

L'aide en ligne

Vous avez plusieurs manières d'obtenir une aide en ligne:

✔ l'instruction **help** affiche l'aide dans la fenêtre FORTH
✔ l'instruction **guide** affiche l'aide dans l'accessoire ST-Guide ou HypView.
✔ sous l'éditeur **Shift+F9** affiche l'aide du mot sous le curseur

Pour cela, vous devez régler le chemin d'accès dans FORTH.INF et copier FORTH.HLP et FORTH.HYP à cet endroit.

Si aucune aide n'est disponible pour un mot, cela vous est signalé.

help

'mot help : affiche l'aide en ligne sur le mot spécifié. Pour cela, le fichier FORTH.HLP doit être accessible. L'article affiché peut contenir des liens, exemple :

« -47 help » Autres modules

Dans ce cas, tapez -47 help pour aller vers le lien depuis la ligne de commande.

➢ `'>fom help` : donne accès à l'aide sur les **Forth Modules**.

guide

'mot guide : affiche l'aide dans l'accessoire ST-Guide ou HypView à la page contenant le mot spécifié. Vous pouvez ensuite naviguer dans tout le fichier hypertexte. Pour cela, il vous faut à la fois FORTH.HLP et FORTH.HYP.

Pendant la navigation dans l'accessoire, le FORTH est passé en mode bureau avec l'instruction **desk**. Vous pouvez revenir vers la ligne de commande en ramenant la fenêtre FORTH au premier plan ou en cliquant sur "Retour" dans le menu.

Il est conseillé de fermer la fenêtre de l'accessoire juste avant, le FORTH ayant une gestion sommaire des fenêtres.

Si les deux sont disponibles, c'est HypView qui est choisi en priorité.

Si aucun des deux accessoires n'est chargé, **guide** agit comme **help** sous TOS. Par contre, en multitâche, le programme utilisera (si disponible) le chemin d'accès complet fourni dans FORTH.INF afin de lancer ST-Guide ou HypView.

Utiliser **help** ou **guide** sur **help** ou **guide** vous renvoie des informations sur la présence des visualiseurs et sur le nombre d'instructions référencées.

Shift+F9

Sous l'éditeur, placez le curseur sur un mot et tapez Shift+F9, cela vous ouvre l'article d'aide correspondant. Si le curseur est sur un espace ou un nombre, rien ne se passe.

Lorsque l'article est affiché, appuyez sur Esc (*en fait, toute autre touche que le « - » produite le même effet*) pour en sortir.

Si un lien s'affiche dans l'article, exemple :

« -47 help » Autres modules

Il vous suffit de taper -47 [Enter] pour accéder à la rubrique voulue.

Les instructions arithmétiques sur la pile

Le FORTH repose sur l'utilisation d'une pile, pour chaque fonction exécutée on doit au préalable empiler les paramètres d'entrée et on récupérera sur cette même pile les paramètres de sortie (si il y en a).

1 Les mouvements de la pile

Chaque nombre rencontré par l'interpréteur est simplement déposé sur la pile, les fonctions utilisent toujours comme paramètres les dernières valeurs déposées (le sommet de la pile). Afin de ramener au sommet la valeur voulue, il existe plusieurs mots permettant de modifier l'ordre des valeurs sur la pile:

dup

duplique la valeur au sommet de la pile.

?dup

duplique le sommet de la pile si il est non nul.

drop

jette le sommet de la pile.

swap

permute les deux valeurs au sommet de la pile.

dropm

n dropm : jette n valeurs du sommet de la pile.

dupm

n dupm : duplique les n valeurs du sommet de la pile.

rot

a b c rot : effectue une permutation circulaire sur 3 valeurs, on obtient c a b.

roll

n roll : effectue un permutation circulaire sur n valeurs (la première devient la dernière).

over

duplique la deuxième valeur par dessus la première.

pick

n pick : duplique la valeur de rang n par dessus le sommet de la pile.

sp!

vide entièrement la pile.

.s

affiche le contenu de la pile (le sommet indiqué par "top" selon le flag S de FORTH.INI). L'affichage de "*" signale un débordement inférieur de pile (trop de valeurs dépilées). Il faut immédiatement utiliser sp!.

2 Les opérations de base

Chacune de ces opérations dépile d'abord ses paramètres, seul le résultat reste sur la pile.

+

a b + : fournit a+b sur la pile.

-

a b - : fournit a-b sur la pile.

*****

a b * : fournit a*b sur la pile.

/

a b / : fournit a/b sur la pile.

mod

a b mod : fournit le reste de la division de a par b.

/mod

a b /mod : fournit d'abord le quotient, puis au sommet le reste de la division de a par b.

w/

a b w/ : divise a par b avec les limites suivantes : b et le quotient doivent être dans l'intervalle [-32768,32767], c'est à dire tenir sur 2 octets. L'opération est plus rapide que /.

w*

a b w* : fournit a*b plus rapidement que * si a et b sont dans l'intervalle [-32768,32767].

w/mod

a b w/mod : avec les limitations de **w/**, fournit le quotient et le reste de a/b.

isqr

x isqr : renvoie une approximation rapide de la racine carrée d'un nombre. Il peut y avoir un décalage de 1 ou 2 sur le résultat.

1+ 1-

augmente ou diminue la valeur au sommet de 1.

2+ 2-

augmente ou diminue la valeur au sommet de 2.

2* 2/

multiplie ou divise par 2 la valeur au sommet.

0<

renvoie 1 (vrai) si la valeur au sommet est négative, 0 (faux) sinon.

0>

renvoie 1 (vrai) si la valeur au sommet est positive, 0 (faux) sinon.

0=

renvoie 1 si la valeur au sommet est nulle, 0 sinon.

> >=

a b > : renvoie 1 si a>b, 0 sinon. (de même si a>=b)

< <=

a b < : renvoie 1 si a<b, 0 sinon. (de même si a<=b)

=

a b = : renvoie 1 si a=b, 0 sinon.

<>

a b <> : renvoie 1 si a est différent de b, 0 sinon.

min

a b min : renvoie le plus petit de a et b.

max

a b max : renvoie le plus grand de a et b.

and

a b and : effectue un ET logique entre a et b.

or

a b or : effectue un OU logique entre a et b.

xor

a b xor : effectue un OU EXCLUSIF logique entre a et b.

not

effectue un NON logique sur le sommet de la pile.

negate

fournit l'opposé du nombre au sommet de la pile.

+-

a b +- : applique la règle des signes de b sur a:
si b<0, a est changé en son opposé
si b>0, a reste tel quel.
b est dépilé et seul reste a éventuellement modifié.

sign

renvoie le signe du sommet de la pile. 1 s'il est positif, -1 s'il est négatif, 0 s'il est nul.

abs

fournit la valeur absolue du sommet de la pile.

<seg>

x a b <seg> : renvoie 1 si x est entre a et b (a et b inclus), 0 sinon.

>seg<

x a b >seg< : renvoie 1 si x est entre a et b (a et b exclus), 0 sinon.

irandom

renvoie un nombre aléatoire sur 24 bits. Beaucoup plus rapide que la fonction **random** du Xbios (×10 sur un MegaSTE et ×30 sur un TT).

irandomize

n irandomize : réinitialise le générateur de nombres aléatoires avec la valeur n. Pour la même valeur n donnée, la série obtenue est exactement la même, ce qui n'est pas possible avec la fonction **random** du Xbios.

Pour un départ plus aléatoire, on peut utiliser: `timer irandomize`

3 Travail sur plusieurs piles

Il existe deux piles en FORTH, la pile des données (celle utilisée pour les calculs courants) et la pile des retours. Cette deuxième pile est gérée de manière interne et conserve les adresse de retour d'un sous-programme, les données d'une boucle, etc. On peut accéder à cette seconde pile pour peu qu'elle retrouve son état initial avant un mot l'utilisant (fin de boucle, fin de mot...)

r0

dépose sur la pile des données l'adresse de base de la pile des retours. Cette pile est descendante (on empile à l'aide de -(a4) et on dépile avec (a4)+), c'est-à-dire que la base est au-dessus de la pile elle-même.

rp@

dépose sur la pile des données l'adresse du pointeur de la pile des retours. Cette adresse est donc inférieure à r0.

>r

déplace le sommet de la pile des données sur le sommet de la pile des retours. Utile pour écarter momentanément une valeur.

r@

copie le sommet de la pile des retours sur la pile des données (la valeurs reste donc encore sur la pile des retours).

r>

déplace le sommet de la pile des retours sur la pile des données.

rp!

vide la pile des retours (à ne pas faire en plein milieu d'un programme, son utilisation est réservée à un nettoyage si un programme s'est interrompu prématurément lors d'une erreur).

L'utilisation de cette deuxième pile est limitée et peut être fatale pour un programme. On peut être amené à créer d'autres piles de données qui n'auront plus les contraintes de la pile des retours. Une deuxième pile peut aussi s'avérer utile lors de la mise au point d'un programme pour conserver des valeurs prises en cours d'exécution sans perturber son déroulement.

&stack

n &stack XXX : crée un mot XXX qui sera une pile contenant au maximum n entiers de 4 octets (un réel compte pour 2 entiers=8 octets). A l'exécution XXX laisse sur la pile courante l'adresse de sa propre base.

current

XXX current : la pile XXX devient la pile courante, toutes les opérations se font dans cette nouvelle pile maintenant.

stack0

pile du FORTH, par exemple : stacko current , permet de revenir à la pile normale. Cette pile de 4096 octets contient jusqu'a 1024 entiers ou 512 réels.

>st

x XXX >st : déplace la valeur x de la pile courante sur la pile XXX.

st>

XXX st> : déplace le sommet de la pile XXX sur la pile courante.

st@

XXX st@ : copie le sommet de la pile XXX sur la pile courante.

s0

dépose sur la pile la base de la pile courante. Les piles de données sont descendantes comme la pile des retours (on empile avec -(a6) et on dépile avec (a6)+).

sp@

dépose sur la pile courante le pointeur de la pile courante, cette valeur est inférieure à so.

depth

dépose sur la pile courante le nombre d'entiers présents sur cette pile.

4 Les nombres réels

Ils sont au format IEEE double (celui du MC68882). Tous les calculs seront effectués avec cette précision. Pour spécifier au FORTH qu'on utilise un nombre réel il faut le faire précéder de %f. Un nombre réel utilise 8 octets sur la pile, la place de deux entiers.

f+ f- f* f/ fneg

remplacent +, -, *, / et negate pour les réels.

f= <> f> f>= f< f<=

remplacent =, <>, >, >=, < et <= pour les réels.

fdup fdrop fswap fover frot fpick f>r r>f

remplacent dup, drop, swap, over, rot, pick >r, r> pour les réels.

itof ftoi

transforment l'entier (int) en réel (fl) ou le contraire. La conversion vers un entier est faite par troncature.

sin cos tan

calculent le sinus, le cosinus ou la tangente d'un angle réel. Les angles sont en radians.

sincos

calcule simultanément le sinus et le cosinus d'un angle. Le cosinus est au sommet de la pile.

asin acos atan

calculent l'arcsinus, l'arccosinus ou l'arctangente d'un réel. La valeur retournée est en radians.

pi

dépose la valeur de pi sur la pile.

cosh sinh tanh

calculent le sinus hyperbolique, le cosinus hyperbolique ou la tangente hyperbolique d'un réel.

asinh acosh atanh

calculent l'arcsinus hyperbolique, l'arccosinus hyperbolique ou l'arctangente hyperbolique d'un réel.

exp log

calcule l'exponentielle ou le logarithme népérien d'un réel.

1/x x^2 sqr

calcule l'inverse, le carré ou la racine carrée d'un nombre.

x^y

x y x^y : calcule x à la puissance y.

fabs int frac round

calcule la valeur absolue, la partie entière, la partie fractionnaire ou l'arrondi d'un réel (le résultat reste au format réel).

f*2^n

x n f*2^n : multiplie le réel x par 2^n (n étant au format entier). Pour multiplier par 4 on utlise n=2, par 32 c'est n=5, pour diviser par 2 on utilise n=-1 etc... Cette opération est extrèmement rapide par rapport à f* puisqu'il suffit de réaliser une addition sur l'exposant du réel.

fsign

renvoie le signe d'un nombre réel. 1 si il est positif, -1 si il est négatif, 0 si il est nul. La valeur renvoyée est un entier.

setdigits

n setdigits : fixe, pour les réels, le nombre de chiffres à afficher. Ceci n'influe pas sur la précision des calculs.

fval

chaine fval : transforme une chaine de caractères en un nombre réel (les caractères %f devant le réel ne sont pas nécessaires). Fonction utile en liaison avec input$ pour l'entrée de paramètres réels.

>rad >deg

%fx >rad transforme x degrés en radians.
%fx >deg transforme x radians en degrés.

_fpu

variable sur 1 octet pour la version STE
_fpu c@ : 0 si il n'y a pas de coprocesseur
: %hFF si un coprocesseur est présent en mode périphérique. Dans ce cas, on peut encore mettre la variable à zéro pour revenir aux routines 68000:
0 _fpu c!

Variables et accès à la mémoire

1 Organisation de la mémoire en FORTH

On peut distinguer trois parties essentielles de la mémoire lorsqu'on se trouve sous l'interpréteur du FORTH:

1 : le buffer contenant le texte saisi sous l'éditeur (1/6 de la mémoire limité à 512 ko, ou valeur fixée dans FORTH.INF)

2 : le dictionnaire (3/4 de la mémoire ou valeur fixée dans FORTH.INF)

3 : la mémoire laissée au système (1/4 de la mémoire ou ce qui reste selon FORTH.INF).

C'est le dictionnaire qui nous intéresse ici plus particulièrement. Dans cette zone sont rangées (à partir du bas) les variables (numériques ou chaines) et les définitions des nouveaux mots (procédures ou fonctions comme on voudra). Y sont rangées également (à partir du haut) les variables allouées dynamiquement (créées ou détruites en cours de programme et dont le nombre ne peut pas être déterminé au lancement).

Son fonctionnement est simple, une variable est créée? La place nécessaire lui est allouée et le pointeur de fin de dictionnaire est augmenté d'autant. C'est le même principe pour une nouvelle fonction, une chaine de caractères, etc...

here

dépose sur la pile la valeur du pointeur de fin de dictionnaire.

top

variable contenant la valeur de here. Elle se manipule comme une variable FORTH:

top @ : est équivalent à here.

100 top +! : avance de 100 octets dans le dictionnaire.

, w, c, f,

x , : dépose en fin de dictionnaire la valeur x sur 4 octets. **w,** réalise la même opération mais sur 2 octets et **c,** sur un octet et **f,** sur 8 octets pour un réel. Attention, avec **c,** le pointeur n'augmentant que de 1, il risque de ne plus être pair (ce qui est indispensable si en suivant se trouve la définition assemblée d'une fonction).

even

assure la parité de top lui ajoutant 1 si il est impair.

align

n align: assure l'alignement de top sur un multiple de n, puissance de 2.

allot

n allot : alloue n octets dans le dictionnaire et fournit en retour l'adresse de cette zone (l'ancienne valeur here en fait). On pourrait définir allot ainsi:

```
: allot  here swap top +! ;
```

allot0

n allot0 : équivalent à allot, sauf que la zone est initialisée à 0.

free

full

renvoient la taille en octets de la zone libre (free) du dictionnaire ou de la zone déjà occupée (full).

remember

restore

remember garde en mémoire l'état actuel du dictionnaire, restore permet de lui redonner cet état après un certain nombre de modifications. Restore exécuté sans aucun remember préalable redonne au dictionnaire l'état initial. Attention, restore n'efface pas les variables créées après remember, si il en existe, elles pointeraient donc vers des zones considérées comme libres et risqueraient d'être effacées.

Ces appels peuvent être imbriqués jusqu'à 49 fois. Chaque restore revenant à l'état du dernier remember.

top>

adr top> : échange la valeur de top avec celle sur la pile. Ceci permet de déplacer momentanément la position "logique" du dictionnaire en mémoire. Permet alors l'utilisation de "**w,**" de "**c,**" de "**,**" ou de "**f,**" sur d'autres zones que dans le dictionnaire.

>top

reprend la valeur sur la pile et la remet dans top.

2 Lire et écrire en mémoire

@

adr @ : ramène sur la pile l'entier long (4 octets) pointé par adr.

!

x adr ! : range l'entier long x à l'adresse adr.

?

adr ? : affiche la valeur sur 4 octets contenue à l'adresse adr.

w@ c@

même principe que @ mais pour des mots (W=2octets) ou des octets (C=1octet). Le signe des valeurs n'est pas étendu à 4 octets, pour l'étendre il faudra utiliser extwl pour un mot ou extbl pour un octet.

w! c!

même principe que ! mais avec des mots ou des octets. Les valeurs correspondent au mot ou à l'octet de poids faible de l'entier sur 4 octets présent sur la pile.

extbl

étend le signe d'un octet à 4 octets. Par exemple 255 devient -1.

extwl

étend le signe d'un mot à un mot long, par exemple 65535 devient -1.

extbw

étend le signe d'un octet à un mot. Par exemple 255 devient 65535.

highw

renvoie le mot haut, par exemple%h12345678 donne%h1234.

loww

renvoie le mot bas, par exemple%h12345678 donne%h5678.

highb

renvoie l'octet haut du mot bas, par exemple%h12345678 donne%h56.

lowb

renvoie l'octet bas du mot bas, par exemple%h12345678 donne%h78.

+!

x adr +! : ajoute x au mot long pointé par adr.

-!

x adr -! : soustrait x au mot long pointé par adr.

1+!

adr 1+! : ajoute 1 au mot long pointé par adr.

1-!

adr 1-! : soustait 1 au mot long pointé par adr.

2*!

adr 2*! : multiplie par 2 le mot long pointé par adr.

2/!

adr 2/! : divise par 2 le mot long pointé par adr.

w*!

x adr w*! : multiplie par x le mot long pointé par adr (avec les limitations de w*).

w/!

x adr w/! : divise par x le mot long pointé par adr (avec les limitations de w/).

3 Les variables et les tableaux

Ne pouvant pas tout le temps travailler avec la pile, les mouvements deviendraient trop complexes, on est amené à créer des variables pour conserver certaines valeurs.

variable

variable XXX : crée un mot XXX qui sera une variable sur 4 octets. A l'exécution XXX laisse simplement l'adresse d'une zone de 4 octets en mémoire. Puisqu'il se comporte comme une adresse, tous les mots précédents sont valables pour XXX :

```
variable   p       \ crée la variable
50 p !                   \ maintenant p=50
21 p +!            \ p=50+21=71
p 2/!              \ p=71/2=35 (en nombre entier)
p ?                \ affiche la valeur 35.
```

&wvar

&wvar XXX : équivalent à variable, sauf que la valeur est sur 2 octets. On y accède donc avec w@ et w!.

array

n array XXX : crée un mot XXX qui sera un tableau de n entiers longs. Le mot XXX a besoin d'un indice sur la pile, il renvoie l'adresse de l'élément ainsi pointé:

```
10 array TABLEAU  \ crée le tableau
50 3 TABLEAU !    \ TABLEAU(3)=50
0 TABLEAU ?       \ affiche la valeur de TABLEAU(0)
84 5 TABLEAU +!   \ ajoute 84 à TABLEAU(5)
```

Les indices commencent à 0 et vont jusqu'à n-1, donc dans un tableau de 10 valeurs les indices vont de 0 à 9.

&warray

n &warray XXX : équivalent à array, sauf que les entiers sont sur 2 octets, on y accède avec w@ et w!.

constant

x constant XXX : crée un mot XXX qui aura systématiquement pour valeur x. A l'éxécution, XXX laisse x sur la pile.

table

table XXX : crée un tableau de constantes. On remarque qu'aucune dimension

n'est indiquée, en fait, il faut remplir ce tableau immédiatement après sa définition:

```
table TEST
7 , 9 , 15 , -6 , 3 ,          \ dépose 5 valeurs
0 TEST                              \ ramène TEST(0), c'est à dire 7.
3 TEST                              \ ramène TEST(3), c'est à dire -6.
2 TEST 4 TEST +                \ calcule TEST(2)+TEST(4), donc 18.
```

Pour plus de détails sur le mot "," se référer à la partie traitant du dictionnaire.

Les variables peuvent servir à pointer des zones mémoires, voici une série de mots utiles pour balayer un bloc pointé par une variable:

.b .w .l .f

size

indique la taille (.b=1 octet, .w=2 octets, .l=4 octets, .f 8 octets) des nombres à traiter avec les mots qui suivent (ainsi qu'avec **uh.** et **ub.**). Par défaut, la taille est .l.

size?

renvoie la taille actuelle (0, 1, 2 ou 3 pour byte, word, long ou float).

)@

p)@ : ramène la valeur pointée par p, p reste inchangé.

-(@

p -(@ : décrémente d'abord la variable p de la taille spécifiée avec size, puis ramène sur la pile la valeur (1, 2 ou 4 octets) pointée par p.

+(@

p +(@ : incrémente d'abord p puis ramène la valeur pointée par p.

)-@

p)-@ : ramène la valeur pointée par p puis décrémente p.

)+@

p)+@ : ramène la valeur pointée par p puis incrémente p.

)!

x p)! : range x à l'adresse pointée par p, p reste inchangé.

-(!

x p -(! : décrémente p puis range x à l'adresse pointée par p.

+(!

x p +(! : incrémente p puis range x à l'adresse pointée par p.

)-!

x p)-! : range x à l'adresse pointée par p puis décrémente p.

)+!

x p)+! : range x à l'adresse pointée par p puis incrémente p.

Exemple, on veut copier 100 mots de zone1 vers zone2:

```
variable p     zone1 p !
variable q     zone2 q !
.w size
100 ndo p )+@ q )+! nloop
```

cmove

adr adr' n cmove : transvase n octets, ou mots ou mots longs ou réels (selon la valeur passée à size) de adr vers adr'. La routine tient compte du fait que les zones peuvent se recouvrir en partie et réalise la copie soit par la fin, soit par le début pour éviter les erreurs. La copie est plus lente pour des octets, et si on est certain d'avoir des adresses paires et un nombre pair d'octet à déplacer, autant déplacer deux fois moins de mots (.w).

Le **fichier ARRAYS.INC** contient en plus une définition générale pour des tableaux de 2 ou 3 dimensions dont la taille des éléments est fixée au préalable par **size**. On doit le charger avec par exemple le chemin par défaut:

```
>include "%\INC\ARRAYS.INC"
```

&ARRAY2

co li &ARRAY2 XXX
Crée un tableau à deux dimensions de co colonnes et li lignes.

&ARRAY3

co li pl &ARRAY3 XXX
Crée un tableau à 3 dimensions de co colonnes, li lignes et pl plans.

Exemple:

```
.b size                          \ les éléments sont des octets
10 10 &ARRAY2 MUL                \ table de 10×10 octets
10 0 do
     10 0 do
          i j w* i j MUL c!      \ l'élément i,j vaut i×j
     loop
loop
5 4 MUL c@ .                     \ affiche 20, 5×4
```

4 les chaines de caractères

Un problème se pose pour la gestion des chaines par rapport à celle des nombres. Le résultat de n'importe quel calcul est de toutes façons un entier de 4 octets (taille fixe). De ce fait le déposer sur la pile n'est pas un problème. Par contre, le résultat d'une opération de chaine peut être une chaine de 10000 octets, la pile serait alors vite débordée. La solution est de ne déposer sur la pile que l'adresse de la chaine (4 octets), les caractères eux-mêmes étant rangés dans une chaine propre au FORTH (pad) ou dans des variables de chaines déclarées par l'utilisateur.

Les chaines FORTH peuvent contenir jusqu'à 32763 octets, elles seront toujours terminées par un 0 final assurant la compatibilité avec les fonctions de la ROM.

string

l string XXX : crée un mot XXX qui sera une chaine de l octets maximum. A l'exécution, XXX laisse l'adresse du premier caractère de la chaine.

array$

l n array$ XXX : crée un mot XXX qui sera un tableau de n chaines de l octets chacune. XXX a besion d'un indice sur la pile, il renvoie l'adresse du premier octet de la chaine ainsi pointée:

```
14   8   array$   TEST    \ 8 chaines de 14 octets
5 TEST                        \ adresse de la sixième chaine
0 TEST 1 TEST $=         \ compare les deux premières chaines.
```

" "

" xxxxx" : dépose sur la pile l'adresse de la chaine encadrée par " et ". Remarquez l'espace nécessaire entre le premier guillemet et le début du texte. Il faudra accéder à cette chaine uniquement en lecture car elle est temporaire. Si sa valeur doit être conservée il faut la transférer dans une chaine créée avec string.

pad

dépose sur la pile l'adresse de la chaine servant de mémoire tampon pour les résultats de certaines opérations.

len

chaine len : renvoie la taille actuelle de la chaine.

mlen

chaine mlen : renvoie la taille maximale de la chaine (celle passée à string par exemple).

$!

chaine1 chaine2 $! : copie la chaine1 dans la chaine2.

$+

chaine1 chaine2 $+ : ajoute la chaine1 à la suite de la chaine2.

$=

chaine1 chaine2 $= : renvoie 1 si les chaines sont égales, 0 sinon.

$<

chaine1 chaine2 $< : renvoie 1 si la chaine1 est avant la chaine2 (selon l'ordre ASCII), 0 sinon.

$>

chaine1 chaine2 $> : renvoie 1 si la chaine1 est après la chaine2 (selon l'ordre ASCII), 0 sinon.

asc

chaine asc : renvoie le code ASCII du premier caractère de la chaine.

chr$

c chr$: crée dans pad une chaine de 1 caractère de code ASCII c, pad est laissé sur la pile.

str$

x str$: crée dans pad une chaine représentant le nombre x dans la base courante, pad est laissé sur la pile.

val

chaine val : renvoie sur la pile le nombre représenté par la chaine dans la base courante.

instr

chaine1 chaine2 n instr : cherche si chaine2 est une sous chaine de chaine1 à partir du nième caractère (la numérotation commençant à 1). Si la recherche est positive renvoie la position de chaine2 dans chaine1, 0 sinon.

left$

chaine n left$: envoie dans pad les n premiers caractères de chaine et laisse pad sur la pile.

mid$

chaine k n mid$: envoie dans pad n caractères de chaine à partir du kième et laisse pad sur la pile.

right$

chaine n right$: envoie dans pad les n derniers caractères de la chaine et laisse pad sur la pile.

puts

chaine adr puts : envoie le contenu de la chaine FORTH à partir de l'adresse adr (avec un 0 final).

gets

adr chaine gets : récupère dans la chaine FORTH les octets présents à partir de adr, jusqu'à trouver un 0.

bstr

chaine bstr : permet d'interpréter la chaîne d'entrée comme une description contenant éventuellement des codes de caractères non accessibles au clavier ou non supportés par l'éditeur du FORTH (comme ceux en dessous de 32). Voici les codes possibles :

:nn : ajoute le caractère de code hexadécimal nn (2 chiffres exactement)
:: : ajoute le caractère « : »
.nnn : ajoute le caractère de code décimal nnn (3 chiffres maxi).
.. : ajoute le caractère « . »
autre : ajoute les autres caractères.

Renvoie dans pad, sur la pile, la chaîne ainsi interprétée et au-dessus la taille de cette nouvelle chaine.

```
" 4:E3R.253" bsrt . cr type
```

Affichera 4 (taille de la chaine) puis $4 \pi R^2$.
Le caractère %hE3 étant Pi et le caractère 253 étant le carré.

5 Les ensembles

Un ensemble est un objet dont le nombre maximal et la valeur des éléments sont prédéfinis par l'utilisateur. L'objet ensemble, lui, contient une suite d'indicateurs attestant la présence ou l'absence de chaque élément.

Il y a deux sortes d'ensembles en FORTH, les énumérés (on donnera explicitement la liste des éléments) ou les segments (tous les entiers compris entre deux valeurs). Les segments sont créés directement, les ensembles énumérés sont créés à partir d'un modèle de référence.

Comme pour les chaînes de caractères, les ensembles ne se déposent pas sur la pile, seule leur adresse y est. On a donc encore une fois recours à un ensemble "pads" prédéfini dans lequel iront les résultats de certaines opérations.

&segment

sup inf &segment XXX : crée un mot XXX qui sera un ensemble pouvant contenir tous les entiers de l'intervalle [inf,sup]. Il faut absolument que -32769<inf<sup<32768. Au départ l'ensemble est vide, à l'exécution XXX laisse sur la pile l'adresse de l'ensemble:

adr : inf
adr+2 : n = nombre d'éléments (sup-inf+1)
adr+4 : suite de n bits (0=élément absent, 1=présent).

&(... & ... &)

&(elem1 & elem2 & ..... & elemn &) XXX : crée n mots (elem1 à elemn) qui seront les noms des éléments de l'ensemble. Le modèle lui même s'appelant XXX. A l'exécution XXX laisse sur la pile l'adresse d'un en-tête d'ensemble:

adr : code de elem1
adr+2 : n, nombre d'éléments.

A l'exécution, chaque elemi laisse sur la pile son code FORTH. Puisque les mots ont été créés les un à la suite des autres, leurs codes se suivent, donc cet ensemble se comporte comme un segment.

&setof

modèle &setof XXX : crée un ensemble dont le type est calqué sur le modèle, celui créé avec &(&). Au départ, cet ensemble est vide et laisse à l'exécution l'adresse du bloc suivant:

adr : code de elem1
adr+2 : n=nombre d'éléments
adr+4 : suite de n bits.

pads

ensemble prédéfini recevant les résultats de certaines opérations. Son type (valeur inf et nombre d'éléments) est automatiquement calqué sur celui des opérandes.

initset

e initset : donne à pads le type de l'ensemble e (e peut aussi être un modèle), pads ne contient alors aucun élément.

elmt+ elmt-

x e elmt+ : ajoute à l'ensemble e l'élément x.
x e elmt- : retire l'élément x de l'ensemble e.

elmtin

x e elmtin : renvoie 1 si x appartient à l'ensemble e, 0 sinon.

elmt@

n e elmt@ : renvoie sur la pile la valeur du nième élément présent dans l'ensemble e, renvoie inf-1 sinon.

card

e card : renvoie le nombre d'éléments actuellement présents dans l'ensemble e, donc 0 s'il est vide.

set!

e e' set! : donne à e' la valeur (et le type) de l'ensemble e. Si e et e' n'étaient pas du même type (allez savoir pourquoi...), ils le deviennent. Par contre si l'ensemble e' utilisait moins de mémoire que e il se produit un débordement dangereux.

set=

e e' set= : renvoie 1 si les ensembles e et e' sont égaux, 0 sinon.

setin

e e' setin : renvoie 1 si l'ensemble e est contenu dans l'ensemble e', 0 sinon.

set+ set* set- -set

e e' set+ : calcule l'union des deux ensembles.
e e' set* : calcule l'intersection des deux ensembles.
e e' set- : calcule leur différence (éléments de e pas dans e').
e -set : calcule le complémentaire d'un ensemble.

Pour chacune de ces quatre instructions, le résultat est envoyé dans pads dont l'adresse est laissée sur la pile.

Exemple pour les segments:

```
255 0 &segment OCTET          \ OCTET est inclus dans [0,255]
256 0 do
            i OCTET elmt+           \ tous les nombres pairs
2 +loop
256 0 do
            i OCTET elmt-           \ enlève les multiples de 3
3 +loop
```

Maintenant OCTET contient tous les nombres pairs non multiples de 3 entre 0 et 255 (c'est à dire 2, 4, 8, 10, ...)

```
256 0 do
            i OCTET elmtin          \ élément i dans OCTET ?
            if
                  i . space         \ si oui, on l'affiche
            then
loop
```

Cette boucle affiche tous les éléments d'OCTET.

Exemple pour les ensembles énumérés:

```
&( lun & mar & mer & jeu & ven & sam & dim &) SEMAINE
                                        \ crée le modèle semaine
SEMAINE &setof MOI
SEMAINE &setof ELLE           \ deux ensembles semaines vides


\ ce mot liste les éléments de l'ensemble dont l'adresse est sur la pile

variable p

: voir
      p !                             \ adresse dans p
      dim 1+ lun do                   \ boucle de lun à dim
            i   p @ elmtin            \ élément présent?
            if i word$ type space     \ si oui l'affiche
            then
      loop
;


list
            lun mar jeu ven sam
dolist
            i MOI elmt+
lloop MOI voir                \ ma semaine de travail


list
            mar mer jeu ven sam
dolist
            i ELLE elmt+
lloop ELLE voir               \ la sienne


MOI ELLE set* voir            \ intersection, jours ou les 2 travaillent
MOI ELLE set- voir            \ je travaille mais pas elle
MOI ELLE set+                 \ jours ou l'un travaille
-set voir                     \ complémentaire, jours ou tous 2 libres
```

6 Les nombres réels

&float

&float XXX : crée un mot XXX qui sera une variable de type réel (8 octets au format double IEEE). A l'exécution XXX laisse sur la pile l'adresse d'une zone de 8 octets.

f@

adr f@ : ramène sur la pile le réel pointé par l'adresse adr. Peut aussi servir à ramener plus rapidement deux entiers longs.

f!

x adr f! : range le réel x à l'adresse adr.

```
Par exemple:
&float REEL                     \ crée une variable réelle
%f1.23   REEL f!                \ REEL vaut 1,23
REEL f@                         \ ramène 1,23 sur la pile
frac   f.                       \ partie fractionnaire 0.23 qu'on affiche
```

7 Les structures

Permettent de grouper sous un seul nom plusieurs objets de types différents qui dans l'application sont logiquement liés. Ce qui permet de les créer, les détruire ou les transférer en bloc. Contrairement aux autres variables qui sont crées dans la zone basse du dictionnaire, les stuctures sont crées dynamiquement en zone haute et peuvent de ce fait être effacées et libérer la place mémoire précédemment utilisée.

Pour illustrer les instructions nous allons créer une structure correspondante aux informations sur une personne (nom, prénom, age).

struct .... &name

prépare le modèle d'une structure. Ce modèle se trouvera en mémoire basse, les structures crées par lui seront en mémoire haute. Entre struct et &name se trouvent la définitions des composant de la structure:

```
struct
            40 string NOM           \ nom sur 40 caractères
            20 string PRENOM        \ prénom sur 20 caractères
            variable AGE            \ l'age est une variable numérique
&name   HUMAIN                  \ nom global du modèle.
```

Pour accéder à une composante de la structure il faut d'abord déposer l'adresse de la structure suivie du nom de la composante.

screate

modèle screate : crée une structure en utilisant le modèle (ce modèle peut être remplacé par une autre structure correspondant au même modèle). renvoie sur la pile l'adresse de cette structure.

```
variable p
HUMAIN screate   p !          \ p=adresse d'un nouvel humain

" DUPONT" p @ NOM $!          \ son nom
" PIERRE" p @ PRENOM $!       \ son prénom
34 p @ AGE !                  \ son age

\ ce mot affiche les composantes de la structure dont l'adresse est
\ passée sur la pile.

: voir
            dup NOM type cr
            dup PRENOM type cr
            AGE ? cr
;

p @ voir                      \ affiche DUPONT, PIERRE, 34.

variable q
HUMAIN screate q !            \ encore un humain
```

s!

s1 s2 s! : transfère le contenu de la structure s1 vers s2.

```
p @ q @ s!        \ maintenant Pierre dupont est en double
q @ voir          \ affiche les mêmes données
15 q @ AGE +!     \ modifie l'age en y ajoutant 15 ans.
p @ voir          \ le jeunot de 34 ans
q @ voir          \ le même à 49 ans.
```

sdelete

adr sdelete : libère la mémoire utilisée par la structure se trouvant à l'adresse adr. Cet emplacement pourra être réalloué plus tard.

```
p @ sdelete       \ Pierre Dupont n'existe plus.
q @ sdelete       \ son vieux double non plus
```

sizeof

s sizeof : fournit la taille mémoire d'une structure.

On voit que ces structures utilisées ainsi ne sont pas forcément simples, en effet, il faut une variable différente pour garder l'adresse de chaque structure. Or, leur intérêt est d'être créées ou détruites à volonté, donc sans limitation préalable de nombre, ce qui est en contradiction avec l'utilisation de variables qui sont en nombre connu (puisque déclarées nommément une à une).

On a alors recours aux chaines de structures. Chaque élément contenant l'adresse d'un autre élément. Il suffit de garder dans une variable l'adresse du premier élément et on peut parcourir ainsi une chaine de taille indéfinie en n'utilisant qu'une variable!

```
\ ce mot servira à remplir les trois champs d'un humain

: remplir
            >r                \ adresse structure sur pile des retours
            r@ AGE !
            r @ PRENOM $!
            r> NOM $!
;

\ cette variable contiendra l'adresse du début de la chaine

variable p
0 p !                   \ p=0, chaine vide.
```

pchain

p modèle pchain : crée et insère une structure copie du modèle en tête de chaine, cette chaine étant pointée par la variable p. Après cette fonction, p pointe sur le nouvel élément et cet élément pointe sur l'ancien début de chaine.

```
p HUMAIN pchain         \ p pointe sur un nouvel élément, celui-ci
                        \ pointant sur 0 (avant p contenait 0!)
" Dupont" " Pierre" 34 p @ remplir        \ initialise ce bonhomme

p HUMAIN pchain         \ p pointe sur un nouveau, celui-ci pointe sur
                              \ pierre dupont et dupont sur 0.

" Tom"   " Oncle"   95 p @ remplir         \ initialise le nouveau
```

next

s next : fournit l'adresse de la structure qui suit la structure s dans la chaine. renvoie 0 si c'est la fin.

```
p @ voir                \ montre le premier élément (Oncle Tom)
p @ next voir           \ montre le second (Pierre Dupont)
p @ next next .         \ affiche 0, c'est la fin.

\ ce mot affiche tous les éléments de la chaine pointée par p

: tous
            p @                 \ le début
            begin
                  ?dup          \ duplique si non nul
            while
```

```
                dup voir cr \ si non nul, affiche et saut de ligne
                next        \ adresse suivante
            repeat
;
```

punchain

p adr punchain : enlève de la chaine pointée par p l'élément dont l'adresse est précisée. Comme avec sdelete, la mémoire est alors libérée.

pinchain

p adr pinchain : ajoute en tête de la chaine pointée par p l'élément déjà existant en mémoire pointé par l'adresse adr. Celui-ci pointera sur l'ancienne tête de chaine. (Cet élément aura été créé par screate par exemple).

pempty

p pempty : vide totalement la chaine pointée par p, tous les éléments sont détruits et p=0. Si on désire simplement créer une nouvelle chaine p=0 suffit (d'ailleurs pempty serait une erreur, car p aurait au départ une valeur quelconque et non un pointeur sur une structure).

Toutes les opérations ont été jusqu'ici réalisées en tête de chaine. Ceci ne sera pas toujours souhaitable, voici donc les instructions nécessaires pour obtenir les mêmes résultats en milieu de chaine:

chain

adr chain : adr est l'adresse d'une structure insérée dans une chaine. Chain crée une nouvelle structure semblable à la précédente et l'insère dans la chaine juste après.

Cette fonction renvoie l'adresse du nouvel élément (pour l'initialiser par exemple).

unchain

adr1 adr2 unchain : supprime l'élément adr2 de la chaine, libère également la mémoire qu'il occupait, à cet effet adr1 doit être un élément le précédant dand la chaine (pas forcément celui juste avant) afin de réaliser le déchainage correctement.

inchain

adr1 adr2 inchain : insère l'élément déjà existant pointé par adr2 juste à la suite de celui pointé par adr1. Ce dernier doit faire partie d'une chaine.

empty

adr empty : vide la chaine à partir de l'élément pointé par adr, celui-ci non compris, il devient alors le dernier de la chaine (son pointeur à 0) et la place mémoire occupée par ses suivants est à nouveau disponible.

islast

adr islast : définit l'élément pointé par adr comme étant le dernier en annulant son pointeur. Contrairement à empty aucun élément n'est effacé. Ceci peut servir à commencer une chaine dont (adr) serait le premier élément (sans utiliser de variable).

previous

adr1 adr2 previous : recherche dans la chaine l'élément précédant adr2 (donc celui qui pointe sur adr2). A cet effet adr1 doit être l'adresse d'un élément précédant adr2 (la tête de liste le plus souvent). Si le début de chaine est rangé dans une variable p, on utilise:

p @ adr2 previous \ p @ donnant l'adresse du premier élément.

Pour réaliser ceci, le programme parcourt toute la chaine jusqu'à trouver l'élément dont le pointeur est adr2. Si cet élément existe, son adresse est laissée sur la pile, sinon on obtient 0 (c'est le cas si adr2 est lui même en tête de chaine, cette valeur 0 correspond donc bien à une "fin" de chaine parcourue à l'envers).

settable

-1 settable : initialise la table d'allocation en conservant sa taille. La partie haute du dictionnaire est considérée comme libre à nouveau.

n settable : (n>0), crée une nouvelle table de n octets et l'initialise, la partie haute du dictionnaire est considérée comme libre à nouveau.

Cette table sert à gérer les allocations et désallocations de structures en haut du dictionnaire. Pour illustrer son principe, supposons qu'on crée à la suite 1000 structures, elles forment un bloc compact. La table contiendra donc un seul pointeur (4 octets) sur un bloc occupé, un seul (4 octets) sur un bloc libre (le reste de la mémoire) et un indicateur de fin de table (-1 sur 4 octets). La table ne demandera alors que 12 octets de gestion.

Supposons maintenant qu'on efface une structure au milieu du bloc. La table sera alors:

pointeur sur un bloc occupé	: ceux avant la structure effacée
pointeur sur un bloc libre	: celui qu'on vient d'effacer
pointeur sur un bloc occupé	: ceux après la structure
pointeur sur un bloc libre	: le reste de la mémoire
fin de table	: -1

La table demandera alors 20 octets avec une structure de moins. Si pour finir j'efface exactement une structure sur deux pour avoir un maximum de trous, j'aurai besoin de 500 pointeurs sur des blocs occupés, 500 sur des blocs libres et l'indicateur de fin de table:

500*4 + 500*4 + 4, c'est à dire 4004 octets.

Tout ceci dépend donc de la manière dont votre programme va agir. S'il libère les structures dans l'ordre inverse (ou le même ordre) ou il les crée, 20 octets seront largement suffisants, si par contre le jeu des allocations et libérations de mémoire est plus sauvage il faudra prévoir au moins 4 fois le nombre maximal de structures que vous pensez créer, plus 4 octets pour l'indicateur de fin.

Par défaut, la table est longue de 1028 octets, ce qui permet de gérer jusqu'a 256 éléments et d'en arriver au pire (la libération d'un sur deux). De toutes façons, lorsqu'une structure est créée on tente de la rattacher à un bloc déjà existant, ceci n'augmente donc pas la taille de la table (puisque le nombre de blocs est le même!), parfois ce nouvel élément permet de raccrocher deux blocs, ainsi la table est même réduite!

*Les mots précédents sont adaptés au stuctures crées par le FORTH lui même (il s'occupe des pointeurs, des allocations et désallocations de mémoire, etc... Mais il se peut que les données soient déjà en mémoire et qu'on ait besoin d'y accéder de manière claire à l'aide des structures. Pensons en particulier aux fichiers *.RSC dont chaque objet est défini par une structure.*

Puisque dans ce cas les données existent déjà, il nous suffit d'un en-tête de stucture dont l'adresse des données sera fixée par l'utilisateur, c'est justement ce que fait &pointer.

&pointer

modèle &pointer XXX : crée le mot XXX qui sera un simple pointeur sur une sructure conforme au modèle. Aucune place mémoire n'est allouée pour les données. Pour fixer l'adresse des données, XXX se comporte comme une variable simple:

```
adr   XXX !          \ fixe l'adresse
```

A l'exécution, XXX laisse l'adresse d'un en-tête.

Reprenons donc l'exemple des arbres d'objets:

```
struct
            &wvar       ob_next
            &wvar       ob_head
            &wvar       ob_tail
            &wvar       ob_type
            &wvar       ob_flags
            &wvar       ob_state
            variable    ob_spec
            &wvar       ob_x
            &wvar       ob_y
            &wvar       ob_width
            &wvar       ob_height
&name OBJECT
```

Supposons maintenant qu'à la suite du chargement du fichier on obtienne l'adresse de l'arbre avec rsrc_gaddr qu'on a rangé dans la constante TREE.

```
OBJECT &pointer objet          \ pointeur sans données

TREE objet !                        \ il pointe sur le premier objet!
objet ob_flags w@ b.           \ affiche ses flags
0 objet ob_state w!                 \ annule state (apparence normale)

objet sizeof 5 *               \ taille de 5 objets
objet +!                       \ qu'on ajoute à l'adresse des
                                    \ données, il pointe sur le 5°!
objet ob_x   get_xylh          \ ramène ses 4 coordonnées
```

En pratique, si les structures sont collées les unes aux autres, toutes de même taille et commençant à l'adresse ADR, la formule est celle-ci pour accéder aux données de l'objet d'indice i (indices commençant à 0):

```
ADR objet sizeof i * + objet !
```

Pour les objets GEM la taille est toujours 24 octets, on pourra donc remplacer objet sizeof par 24.

8 Créer un type de données

On a parfois besoin de données particulières qui ne peuvent pas s'exprimer à l'aide des types de base. Il faut donc créer un type. Ce type sera un mot équivalent à variable ou &float ou array etc... qui lui-même créera un autre mot lui donnant un certain comportement.

Par exemple, le mot variable a trois actions successives:

1- le mot qui suit est nouveau, il faut le créer

2- réserver une zone de 4 octets

3- donner comme comportement à la variable celui de déposer l'adresse des 4 octets.

La syntaxe générale d'un type sera:

```
: &TYPE   <actions2>   n :does>   <actions3>   ;end   <autres> ;
```

Le fait que le nom commence par le caractère & assure l'action n°1, c'est à dire l'inclusion du mot suivant dans le vocabulaire (comme &float, ou &wvar, &name...). Les <actions2> sont donc les initialisations (réservation de mémoire par exemple, la partie entre :does> et ;end correspondra au comportement du nouveau mot, le reste, <autres> contiendra des actions supplémentaires, c'est à dire rien le plus souvent.

:does> ... ;end

n :does> : la valeur n indique combien de paramètres sont renvoyés par <actions2> et doivent être inclus au début de la définiton du nouveau mot.

... : tout ce qui est compris entre les deux mots sera ajouté à la définition du nouveau mot.

;end : termine cette nouvelle définition.

Créons (ou recréons) le type variable:

```
: &VAR
            4 allot     \ réserve de la place et renvoie l'adresse
            1 :does>    \ l'adresse est placée dans la définition
            ;end        \ la variable ne fait rien d'autre qu'empiler
                        \ l'adresse!
;

&VAR a                  \ crée la variable a
&VAR b                  \ même chose pour b
```

Créons un tableau de nombres réels

(genre array mais avec 8 octets par nombre), le nombre d'éléments sera passé sur la pile:

```
: &farray
      8 * allot0        \ réserve 8*nombre d'éléments
      1 :does>          \ adresse passée dans la définition
            swap 8 * +  \ caculera adr+8*i
      ;end
;

5 &farray TABLEAU       \ tableau de 5 réels
%f5.36                  \ dépose 5.36
2   TABLEAU             \ adresse du 3° élément (adr+8*2)
f!                      \ range 5.36 à l'adresse indiquée

5 0 do                          \ boucle de 0 à 4
      i TABLEAU f@ f. cr        \ affiche l'élément i
loop
```

Créons la constante

```
: &CONST
      1 :does> ;end             \ valeur dans définition
;
20   &CONST vingt       \ vingt sera une constante égale à 20.
0 &CONST zéro           \ zéro sera une constante égale à 0.
```

Créons une matrice de variables entières

Elle s'utilisera ainsi X Y &MAT xxx pour créer le tableau et i j xxx pour accéder à un élément :

```
: &MAT
      over w* 4 w*            \ X  4*X*Y=taille à allouer
      allot0                  \ X   adr
      2 :does>                \ ajoutera X adr à i,j -> i j X adr
            >r w* +           \ (écarte adr) i+j*X
            4 w*              \ 4*(i+j*X)
            r> +              \ adr+4*(i+j*X) = adresse élément
      ;end
;

10 10 &MAT pythagore
10 0 do
      10 0 do
            i j w*
            i j pythagore !   \ crée une table de multiplication
      loop
loop

5 7 pythagore @               \ renvoie l'élément 5,7 c'est à dire 35.
```

En remplaçant 4 par 8, on crée une matrice de réels.

Note: *Plusieurs :does> ;end sont possibles dans le même type pourvu qu'une seule paire soit exécutée (par un choix conditionnel).*

::does> ;end

Même principe que :does> ;end sauf qu'a chaque nouvelle création de mot, sa définition sera assemblée et non interprétée. Ce mot sera utilisable dans un mot assemblé (ou compilé).

La création de nouveaux mots

1 Les mots et le dictionnaire

Un programme FORTH peut être décomposé en plusieurs sous-tâches, chacune recevant un nom. Lorsque ce nouveau mot est créé il est utilisé comme un mot standard, il prend ses paramètres sur la pile et y renvoie les résultats. Il peut aussi se servir de variables, c'est au choix, ou n'avoir aucun paramètre.

La définition de chaque mot est rangée dans le dictionnaire, comme une variable. Le dictionnaire est une pile de type LIFO (dernier arrivé, premier sorti) au fonctionnement primaire. C'est à dire qu'un mot créé ne peut utiliser que ceux déjà créés avant lui (ou lui même, le FORTH est récursif). Deuxième effet, lorsqu'un mot est effacé, le FORTH est obligé d'effacer tous les mots créés après lui également.

Il existe néanmoins une manière (peu jolie, mais à quoi bon?) de faire des appels "en avant" vers des mots pas encore créés.

Lorsqu'il est créé, un mot reçoit un code FORTH (c'est son numéro d'ordre dans la suite des mots créés), ce code sera utile à certaines fonctions mais est surtout indipensable à l'interpréteur.

: ... ;

: XXX ... ; ceci créé un nouveau mot XXX, sa définition correspond à tout ce qui se trouve entre XXX et ;. Par exemple, voici un mot qui additionne 65 au sommet de la pile:

```
: plus65
            65 +
;
```

Donc:

```
15 plus65 .       \ affiche 80 (65+15)
```

:: .... ;

:: XXX ... ; Même principe que la paire précédente, sauf qu'à l'exécution ce mot sera plus rapide car il devient une routine assembleur au lieu de codes FORTH.

Lorsque le texte d'une définition est envoyé à l'interpréteur, celui-ci remplace chaque mot FORTH par son code (sur 2 octets). Lors de l'exécution, l'interpréteur lit un à un les codes et saute à la routine assembleur (pour les mots standards) ou à une autre définition (pour les mots de l'utilisateur). Pour accélérer ce processus, on peut tout simplement remplacer la suite des codes par les

routines assembleur elles-mêmes évitant ainsi le cycle lecture-code/recherche de l'adresse/saut au sous-programme. C'est ce que fait :: ; améliorant ainsi la paire : ;. Mais alors pourquoi utiliser encore : ;? Premièrement parce que tous les mots ne sont pas utilisables dans l'autre mode, ensuite pour des questions de taille mémoire. Avec : ; (mode interprété) chaque mot n'utilise que 2 octets, avec :: ; (mode compilé ou assemblé) chaque mot est une routine assembleur qui excède bien souvent 2 octets, et parfois la vitesse d'un mot est sans influence. Imaginons par exemple un mot qui attend une frappe au clavier.

Voici les règles à retenir:

Un mot interprété peut faire appel à n'importe quel autre mot.

Un mot compilé (ou assemblé) ne peut faire appel qu'à des mots standards ou d'autres mots compilés (les variables sont considérées comme compilées).

Pour pourvoir assembler des variables/tableaux, il faut absolument qu'elles soient définies et donc que le code précédant :: ; soit déjà exécuté. Vous devez donc insérer >comp *dans le source juste avant un assemblage (à moins que votre routine n'utilise aucune variable).*

find ou apostrophe

find XXX : ramène sur la pile le code FORTH du mot XXX.

Raccourci : ‘XXX (nom précédé de l’apostrophe), équivalent à find.

adress

est un tableau de type array. Chaque élément contient l'adresse d'un mot FORTH (rangés dans l'ordre de leurs codes). Exemples:

```
find system adress ?    \ affiche l'adresse du mot system
find cls adress @       \ prend l'adresse de cls...
find system adress !    \ ... et la met à system.
```

Au prochain appel de system pour quitter l'interpréteur, on aura simplement un cls...!

Pour les mots standards, l'adresse pointe directement sur une routine assembleur. Pour un mot utilisateur, le premier code sur 2 octets peut indiquer son type:

140 : variable (variable, &float, &setof, &pointer, &name, &stack)
141 : constant
142 : array
143 : string
144 : array$
145 : table
437 : mot compilé (:: ... ;), le reste est la routine assembleur
628 : élément d'un ensemble énuméré (&)
706 : warray
Autre : c'est le premier code de la définition d'un mot interprété.

&f

&f XXX crée un mot XXX qui, à l'exécution, se comporte comme "forget tout ce qui me suit". Utile en phase de développement pour repartir à zéro avant compilation.

forget

forget XXX : efface du dictionnaire le mot XXX et tous ceux créés après lui. Libère ainsi la place précédemment occupée. Utilisé sur un mot standard, la stabilité de l'interpréteur n'est plus assurée.

Par exemple forget system, ne permet plus de quitter proprement l'interpréteur.

vlist

affiche la liste de tous les mots créés par l'utilisateur.

word$

c word$: envoie dans pad la chaine de caractères correspondant au mot de code c, l'adresse pad est laissée sur la pile. Ceci s'avère utile en liaison avec les éléments d'un ensemble énuméré.

Exemple:

```
500 0 do i word$ type space loop
```

Affiche le nom de tous les mots FORTH jusqu'au code 499.

fast slow

Dans le mode **slow**, on peut arrêter l'exécution de mots interprétés à l'aide de la combinaison Control+Alternate+les deux shift. Le dépassement inférieur de pile est aussi réalisé (interdiction de dépiler plus qu'on avait empilé). Si un arrêt est demandé, ou si un dépassement se produit, l'interpréteur dépose sur la pile l'adresse de son pointeur d'exécution (qui pointe sur le code de l'instruction qu'il allait exécuter).

La boîte d'alerte qui s'ouvre vous permet également de lancer une analyse de la pile des retours afin de situer le point d'arrêt. Une ligne de ce type :

```
test ⇩ [qt]
```

signifie que le mot test a été appelé et qu'on est entré dans sa définition. Au retour c'est [qt] qui sera exécuté (quit, retour à la ligne de commande). Une ligne de ce type :

```
begin ⇨ dup
```

signifie qu'une boucle commençant par « begin dup » est en cours. Attention, car cette analyse peut planter l'ordinateur sur des mots assemblés. À vous de voir. Vous devrez également nettoyer les deux piles avec **rp!** et **sp!**.

Dans le mode **fast**, ces possibilités n'existent pas, mais on y gagne en rapidité.

Le mode slow est le mode par défaut sous l'interpréteur *(depuis la version 0.2.2)* : réglage possible dans le fichier FORTH.INF.

Le mode fast est celui par défaut pour un programme indépendant créé par le compilateur.

exec

c exec : exécute le mot de code c. Deux utilisations possibles pour ce mot, d'abord l'équivalent d'un ON .... GOSUB du Basic:

```
table CODES
find sub0 ,
find sub1 ,
find sub2 , \ crée une table avec les codes de 3 routines

input       \ l'utilisateur tape 0, 1 ou 2
CODES       \ ramène le code correspondant
exec        \ exécute sub0, 1 ou 2.
```

Deuxième utilisation, l'appel en avant d'un mot pas encore créé:

```
variable CODE           \ contiendra le code
: essai CODE @ exec ;   \ exécutera le mot pointé par CODE
: coucou ." COUCOU!" ;  \ le deuxième mot
find coucou CODE !            \ CODE pointera sur coucou
essai                   \ exécutera le mot coucou créé après lui.
```

exe+

n exe+ : exécute le mot dont le code est égal à celui du mot courant plus n. Par exemple si n=1, exécutera le mot suivant.

```
: essai 1 exe+ 2 exe+ ; \ exécutera les deux mots qui suivent.
: coucou ." Cou" ;
: coucou2 ." cou!" ;
essai                   \ affiche Coucou!
```

2 Utilisation de l'éditeur

Pour faciliter la mise au point d'un programme un éditeur est intégré dans l'interpréteur (on peut néanmoins taper toutes ses instructions en mode direct à la suite de Ok>). La taille du texte éditable est fixée par le fichier FORTH.INF, ou, à défaut, au 1/6 de la mémoire libre.

edit

passe sous l'éditeur. La fenêtre est coupée en deux parties. En haut une ligne indique la taille mémoire restant pour le texte du programme ainsi que le numéro de la ligne sous le curseur (commence à 0). En bas, c'est le texte du programme qui est affiché.

Chaque ligne peut contenir jusqu'à 80 caractères. Si la largeur de la fenêtre est insuffisante, un scrolling horizontal se fera pour accéder à toute la ligne.

Les lignes commençant par une définition (avec : ou::) sont en gras. Celles commen-

çant par un commentaire (avec \) sont en clair. La ligne éditée est en style normal et reprend son marquage une fois quittée.

Touches actives sous l'éditeur:

F1 : début du texte
F2 : demi écran vers le haut
F3 : marque le début du bloc — **Shift F3** : aller au début du bloc
F4 : marque la fin du bloc — **Shift F4** : aller à la fin du bloc
F5 : annule le marquage du bloc — **Shift F5** : effacer bloc
F6 : ou **Ctrl+L** aller à la ligne... (utilise la ligne d'informations)
F7 : ou **Ctrl+F** chercher une chaine (utilise la ligne d'informations)
F8 : ou **Ctrl+G** chercher encore
F9 : demi-écran vers le bas
F10 : fin du texte
Shift F6 : déplacer bloc avant la ligne sous le curseur.
Shift F7 : copier bloc avant la ligne sous le curseur.
Shift TAB : si un bloc est défini, le décale à droite ou à gauche selon la touche Shift choisie. Avec les deux Shift, cela reformate les tabulations sur des multiples de 3.

La fonction de recherche de texte ne détecte qu'une occurence de la chaine par ligne.
Le marquage du bloc est matérialisé par l'ascenseur vertical de la fenêtre. S'il est en position haute face à la ligne d'information, c'est que le bloc n'est pas dans la fenêtre. On peut marquer un bloc à l'envers (le début plus loin que la fin), l'éditeur remet tout en ordre. Les fonctions de bloc sont irrémédiables, pas de UNDO prévu.

Help ou Shift F1 : affiche un rappel des principales touches
Shift F9 : affiche l'aide du mot sous le curseur
flèches : haut, bas, droite, gauche.
Shift Gauche : fin de ligne
Shift Droit : début de ligne
Enter : fin de ligne et création d'une nouvelle (coupure de l'ancienne ligne au niveau du curseur)
Undo ou Shift F2 : annule les modifications de la ligne courante.
Backspace : efface en reculant et si assez de place, recolle la ligne avec la précédente.
Delete : efface sous le curseur
Contrl Del : efface la ligne courante
Tab : avance vers la droite sur le prochain bloc de 3 caractères.
Shift Haut : demi-écran vers le haut, équivalent de F2.
Shift Bas : demi-écran vers le bas, équivalent de F9.
L'identation est automatique.

Esc : quitte l'éditeur.
S'affiche alors un message indiquant s'il reste un bloc marqué ou non, car les fonctions **saveb**, **writeb** et **compilb** sont limitées au bloc s'il existe.

Si un **Clipboard GEM** est défini (normalement un dossier CLIPBRD dans le lecteur de boot, par exemple C:\CLIPBRD\ ), alors l'éditeur peut y accéder :

Ctrl+C	: copie le contenu du bloc vers un nouveau fichier SCRAP.TXT
Shift+Ctrl+C	: ajoute le contenu du bloc au fichier SCRAP.TXT
Ctrl+V	: insère le contenu du fichier SCRAP.TXT au niveau du curseur. La partie insérée devient le nouveau bloc.

writeb

imprime le bloc sélectionné (s'il existe) ou tout le texte.

saveb

sauve le bloc sélectionné (s'il existe) ou tout le texte. Le sélecteur de fichier est ouvert, il suffit d'entrer le nom désiré. Si un fichier du même nom existe déjà, son contenu est remplacé par celui de l'éditeur.

Selon les réglages dans FORTH.INF deux boîtes d'alerte sont prévues:

- une si le fichier existe déjà
- une si vous ne sauvez que le bloc au lieu du fichier entier.

export

comme saveb, sauf qu'il est envoyé au format ASCII simple (pourra être retraité par un autre traitement de textes). Efface **pads** (voir ensembles).

append

même principe que saveb, sauf que si le fichier existe déjà, le contenu de l'éditeur est ajouté à l'ancien contenu.

loadb

charge un fichier dans l'éditeur par l'intermédiaire du sélecteur de fichier. Si un texte est déjà présent, alors le fichier est ajouté à la suite de ce texte. (en mode select+) Si le fichier n'est pas trouvé, une seconde tentative est faite en forçant l'extension à ".FOR".

import

comme loadb, sauf que le fichier est au format ASCII simple et sera recodé avant sa copie dans l'éditeur. Efface **pads** (voir ensembles). (en mode select+) Si le fichier n'est pas trouvé, une seconde tentative est faite en forçant l'extension à ".TXT".

select+

select-

déterminent le mode de fonctionnement des mots **loadb**, **saveb, import, export rsc2for** et **append**. En mode select+ (par défaut), le selecteur de fichier est ouvert. En mode select-, une chaine contenant le chemin complet d'accès est attendue sur la pile, par exemple:

```
" D:\FORTH\TEST.FOR" saveb
```

compilb

envoie le bloc sélectionné (s'il existe) ou tout le texte à l'interpréteur. Les parties exécutables sont exécutées, les définitions de mots sont rangées dans le dictionnaire. Si une erreur est détectée elle est signalée et l'exécution de **edit** amènera le curseur sur la ligne fautive.

Si vous utilisez des définitions assemblées avec :: xxx ; des statistiques de compilation sont disponibles à l'adresse global-2 et global-4:

global 4 - w@ . affiche le nombre d'optimisations liées à l'utilisation d'une constante comme argument d'une fonction

global 2- w@ . affiche le nombre d'optimisations liées à l'enchaînement de mots dont l'un récupère le résultat de l'autre sans passer par la pile.

>comp

l'utilisation de ce mot est réservée à l'intérieur de l'éditeur. Comme expliqué précédemment, le FORTH lit d'abord un texte, le code, puis l'exécute. De ce fait, il utilise un buffer d'exécution dans lequel il entrepose en première lecture les codes des instructions. Ce buffer est limité et il arrive qu'il se remplisse. Si le FORTH signale cette erreur, il suffit de couper la partie exécutable par un >comp qui enverra en plusieurs parties les données à l'interpréteur.

>comp ne peut pas se trouver au milieu d'une boucle ou de toute autre structure de contrôle (if, case, etc...). De ce fait, les parties exécutables inscrites dans une boucle (ou autre) sont limitées en taille. Attention, ceci n'affecte pas les définitions des instructions (entre : et ;, ou :: et ;) puisque le buffer d'exécution n'est pas utilisé, tout est directement envoyé dans le dictionnaire.

>comp est également utilisé en liaison avec **>export** (voir le pré-processeur).

eval

ce mot se comporte comme compilb sauf qu'au lieu d'aller chercher le texte dans l'éditeur, il le prend dans une chaine.

" chaine" eval : interprète la chaine comme une suite de commandes FORTH, exécute ce qui doit l'être et range les définitions rencontrées dans le dictionnaire. On peut ainsi réaliser un interpréteur FORTH en FORTH:

```
begin
      ." Ok>" input$ eval
again
```

Cette boucle réalise le même travail que le mode direct de l'interpréteur. La chaine peut elle même contenir la fonction eval. A chaque niveau une nouvelle couche de l'interpréteur est créé.

lmax

variable sur 2 octets contenant le numéro de la dernière ligne de l'éditeur. Modifier cette variable est un peu dangereux si on ne conait pas le codage des lignes de l'éditeur.

bstart bsize

Ces deux variables sur deux octets chacune contiennent respectivement le numéro de la première ligne du bloc et le nombre-1 de lignes du bloc. Si bstart=-1 (ou 65535) aucun bloc n'est défini. Si le bloc est défini et que bsize=-1 (ou 65535), le bloc s'étend jusqu'à la fin du texte.

```
      10 bstart w!            \ marque manuellement un bloc commençant à la
ligne 10
      5  bsize w!            \ et long de 6 lignes.
      writeb                 \ imprime ces 6 lignes.
```

buf0 buf

buf0 est une constante contenant l'adresse de départ du texte dans l'éditeur. Buf est une variable (4 octets) contenant l'adresse de la fin du texte dans l'éditeur.

```
      buf @ buf0 - .    \ affiche la taille du texte dans l'éditeur.
```

buf!

vide le texte de l'éditeur.

Dans l'éditeur les lignes sont codées ainsi:
2 octets : offset pour la prochaine ligne ou 0 si dernière ligne
1 octet : 1+nombre d'espaces en début de ligne
ensuite vient le texte de la ligne complété éventuellement par un espace pour que la ligne suivante commence à une adresse paire.

( )

Les parenthèses ont simplement un rôle de lisibilité. Elles sont complètement ignorées à la compilation et absentes lors de l'exécution.

Elles sont utilisées pour encadrer un groupe logique, par exemple les paramètres d'une fonction afin de mieux lire quelle valeur sera utilisée par quelle fonction.

Exemple:

```
( " D:\DOSSIER\FICHIER.EXT" 0  )
      fopen fhd !
( fhd @ 1000 here )
      fread drop
( fhd @ )
      fclose drop
```

La présentation montre les paramètres des appels fopen, fread et fclose.

\\

Uniquement sous l'éditeur, ce signe permet de faire suivre une chaîne de caractères sur la ligne suivante. Ce signe doit être le tout dernier de la ligne. Fonctionne avec **."** et **"** uniquement.

La ligne suivante est ajoutée à la chaîne à partir de son premier caractère non vide, et donc ne tient pas compte des tabulations. Si on désire mettre un espace, il faut qu'il soit placé avant le signe \\

Exemple:

```
." Cette chaîne se \\
      pousuit ici !"
```

Affichera:

```
Cette chaîne se poursuit ici!
```

Les structures de contrôle

1 Les boucles

do ... loop

x y do ... loop : exécute la boucle avec y comme indice de départ, l'indice est incrémenté de 1 à chaque loop et la boucle se poursuit tant que l'indice est inférieur à x. Par exemple: 10 2 do ... loop, l'indice prendra les valeurs de 2 à 9.

Note : *si x=y, la boucle est ignorée.*

do ... +loop

x y do ... +loop : exécute la boucle avec y comme indice de départ, l'indice est incrémenté de la valeur trouvée sur la pile par +loop et se poursuit tant que l'indice est inférieur à x (si l'incrément est positif) ou tant que l'indice est supérieur ou égal à x (si l'incrément est négatif). Par exemple: 16 2 do ... 4 +loop indice=2, 6, 10, 14

6 18 do ... -3 +loop indice=18, 15, 12, 9, 6.

Note : *si x=y, la boucle est ignorée.*

do ... ifloop

x y do ... ifloop : éxécute la boucle avec y comme indice de départ. Si ifloop rencontre une valeur fausse sur la pile (zéro), la boucle s'arrête sans incrémentation et la valeur du dernier indice est laissée sur la pile. Si au contraire la valeur rencontrée est vraie (autre que 0), l'incrémentation de 1 se fait et la boucle se poursuit tant que l'indice est inférieur à x. La valeur renvoyée est alors x si la condition a toujours été vraie.

Note : *si x=y, la boucle est ignorée, et x est renvoyé sur la pile.*

list ... dolist ... lloop

Il y a **deux syntaxes** possibles:

1) list ... (valeurs déposées) ... dolist ... (corps de la boucle) ... lloop : Le corps de la boucle est éxécuté, l'indice prenant successivement les valeurs déposées sur la pile entre list et dolist. Par exemple:

list 12 14 -1 -3 dolist ... lloop : indice=12, 14, -1, -3.

Notes : *si la liste est vide, la boucle est ignorée.*

On ne peut pas avoir une seconde boucle du même type entre list et dolist.

Si dolist n'est pas précédé d'un list, un avertissement s'affiche.

2) adr n dolist ... lloop : les valeurs de l'indice sont rangées à l'adresse adr sur 4 octets chacune (c'est le format d'un tableau d'entiers créé avec array), n est le nombre de valeurs à prendre (c'est à dire le nombre de bouclages).

Note : *si n=0, la boucle est ignorée.*

On peut obtenir l'index de la valeur en cours avec **r@** (de 0 à n-1), en restant au niveau de la boucle sans qu'une autre structure soit ouverte à l'intérieur (boucle, case...).

i j k

Ce sont les noms des indices de boucles. Lorsqu'une boucle est ouverte elle se voit attribuer l'indice i. Si une boucle est ouverte à l'intérieur d'une autre, elle prend l'indice i, celle immédiatement avant prend l'indice j. Si une troisième boucle est ouverte dans les deux autres, elle prend l'indice i, la précédente l'indice j, et la plus extérieure l'indice k. Exemple:

```
102 100 do
            i . cr                          \ i centaines
                  12 10 do
                  i . space                 \ ici i devient dizaine
                  j . cr                    \ j est la centaine
                  2   0   do
                        i . space           \ ici i devient unité
                        j . space           \ j devient dizaine
                        k . cr              \ k devient centaine
                  loop
                  i . space                 \ i est redevenu dizaine
                  j . cr                    \ j redevenu centaine
            loop
            i . cr                          \ i est redevenu centaine
loop
```

Ces indices sont applicables aux boucles présentées précédement.

ndo ... nloop

n ndo ... nloop : exécute la boucle n fois sans qu'on puisse utiliser d'indice dans le corps de la boucle. Cette boucle est alors plus rapide que do/loop.

Note : *si n=0, la boucle est ignorée.*

leave

permet de quitter prématurément une boucle, la sortie se fera dès le prochain loop, +loop ou nloop.

begin ... again

Le corps de la boucle sera exécuté indéfiniment (sauf sortie avec exit ou quit).

begin ... until

Le corps de la boucle sera exécuté jusqu'a ce que la valeur prise sur la pile par until soit vraie (autre que 0). Exemple: begin mousek until, attend une pression sur l'un des boutons de la souris.

begin ... while ... repeat

La boucle contient une condition de sortie en son milieu. On sort dès que la condition prise par while est fausse (zéro).

```
adr p !           \ p pointe sur une adresse quelconque
.b size           \ on manipule des octets
begin
            p )+@       \ ramène la valeur pointée par p, p incrémenté
            ?dup        \ doublée si non nulle
while               \ tant qu'elle est différente de zéro
            emit        \ sort le caractère correspondant
repeat                  \ au suivant
```

Cette boucle a donc affiché une chaine se trouvant à l'adresse adr et se terminant par un octet nul.

2 Les tests

if ... then

Si la condition trouvée par if sur la pile est vraie (non nulle), alors la partie entre if et then est exécutée. Si non, on saute directement àprès then (en FORTH then signifie ensuite, et non alors).

if ... else ... then

Si la condition trouvée par if sur la pile est vraie (non nulle) la partie entre if et else est exécutée, sinon, c'est la partie entre else et then qui l'est.

```
x 0>
if
            ." Positif"
else
            ." Négatif ou nul"
then
```

Affiche en toutes lettres le signe de x.

case of endof endcase

x case
    x1 of ... endof
    x2 of ... endof
    ...
    xn of ... endof
    (partie par défaut)
endcase

La valeur x trouvée par case est successivement comparée à x1, x2, ... xn. Dès qu'il y a égalité, la partie comprise entre les of et endof suivants est exécutée ensuite on saute après le endcase. Si aucune valeur xi n'a convenu, c'est la partie par défaut qui est exécutée. C'est uniquement dans la partie par défaut que x est laissé sur la pile (a vous de le dépiler ou de l'utiliser).

```
key                                     \ lit une touche
%hff and                                \ et conserve le code ASCII
case
      65  of   ." C'est le A" endof     \ le A a pour code ASCII 65
      66  of   ." C'est le B" endof     \ le B c'est 66
      drop                              \ autre touche jetée
      ." Touche incorrecte"             \ message d'erreur
endcase
```

>of <of <>of

Permettent de remplacer of (dont le seul test est l'égalité) par d'autres tests >of : x est il supérieur à xi?, <of : x est-il inférieur à xi?, <>of : x est-il différent de xi?

```
x case                                  \ retient x
            0  >of   ." Positif" endof  \ x>0?
            0  <of   ." Négatif" endof  \ x<0?
            drop ." Nul"                \ seul cas restant, x=0
endcase
```

-> and-> or-> ->.

Cette structure permet de réaliser des tests complexes en évitant les calculs inutiles. Par exemple je veux exécuter un sous programme seulement si deux conditions sont simultanément vraies. Si la première est fausse, la seconde est inutile à vérifier. De même, je veux exécuter un sous programme si l'une ou l'autre de deux conditions est vraie. Si la première est déjà vraie, la seconde est inutile.

```
->    calcul1
and-> calcul2
and-> calcul3
etc..
->.
```

renvoie -1 (vrai) si toutes les conditions sont vraies, 0 dès que l'une est fausse. Chaque calcul ne doit laisser sur la pile qu'une valeur (0=faux, autre=vrai). Le plus intéressant est de placer en premier les conditions ayant le moins de chances de se vérifier (selon le programme) et d'éviter ainsi les tests inutiles.

```
->    calcul1
or->  calcul2
or->  calcul3
etc...
->.
```

renvoie -1 (vrai) si l'une des conditions est vraie, 0 si toutes sont fausses. Chaque calcul ne doit laisser sur la pile qu'une valeur (0=faux, autre=vrai). Le plus intéressant est de placer en premier les conditions ayant le plus de chances de se réaliser (selon le programme) afin de quitter la structure au plus vite et d'éviter les tests inutiles.

Ces deux structures sont imbricables:

```
->
      ->   A   and->   B   ->.
or->
      C   not
or->
      -> D   and-> E not ->.
->.
```

Ceci teste la condition (A et B) ou (non C) ou (D et non E).

3 Autres commandes

exit

permet de sortir prématurément d'un mot. **exit** ne doit pas se trouver à l'intérieur d'une structure utilisant la pile des retours (boucle avec do...).

quit

arrête toute exécution et revient à l'interpréteur.

system

quitte l'interpréteur.

Si vous avez modifié le programme source depuis la dernière sauvegarde (avec **saveb** ou **append**), une boîte d'alerte vous le signale et permet de rester sous l'interpréteur.

Cette alerte peut être évitée avec les réglages dans FORTH.INF.

Clavier et écran

1 Représentation des nombres

Par défaut les nombres sont écrits en base dix, mais le FORTH permet de modifier la base, on peut ainsi utiliser la base 16 (hexadécimale) ou 2 (binaire) ou d'autres plus exotiques (comme la base 13...).

%d %b %o %h

Ce ne sont pas à proprement parlé des mots FORTH, mais des préfixes signalant que le nombre à suivre est en base 10 (%d), 2 (%b), 8, (%o) ou 16 (%h). Ceci ne modifie pas la base courante, seul un nombre est concerné:

```
%d15        \ 15
%h15        \ 21 (1*16 + 5)
%o15        \ 13 (1*8 + 5)
%b1011      \ 11 = 8 + 2 + 1
```

base

variable sur 4 octets contenant la base courante. La séquence:

```
base @ .
```

affichera toujours 10, puisque quelle que soit la base, elle s'écrit 10 dans sa propre base... En effet 2 s'écrit 10 en base 2, 23 s'écrit 10 en base 23. Si on désire avoir la valeur en base 10 on doit spécifier:

```
base @ d.
```

De même, si je suis en base 2, la séquence 10 base ! me laissera en base 2 car ici 10 vaut 2. Il faudra donc utiliser:

```
%d10 base !
```

ou l'un des mots suivants:

bin hex decimal

fixe la base à 2, 16 ou 10.

Pour que la base devienne effective en entrée, il faut que la séquence d'instructions modifiant la base et le nombre soient séparés par un Enter (en mode direct) ou un >comp sous l'éditeur. En cours de programme, le changement de base est immédiat (avec input notamment).

%f

préfixe indiquant que le nombre à suivre est un réel. Le format est:
préfixe: %f
signe: - ou + (celui ci est optionnel)
mantisse: un nombre à virgule ou non
exposant: cette partie est optionnelle, elle se compose de:
préfixe: e ou E
signe: - ou + (celui-ci est optionnel)
valeur: un entier en base 10.

Facilité: si un nombre contient un point décimal, il est considéré comme réel même sans le préfixe.

Par exemple:

```
%f-2.36e5   =     -236000
%f72e-6     =     0.000072
%f.5        =     0.5
8.          =     8
```

2 Les entrées au clavier

input

attend la frappe au clavier d'un nombre à la position courante du curseur. Enter met fin à la saisie. Le nombre exprimé dans la base courante est converti en entier et déposé sur la pile.

Pour la saisie de **nombres réels**, on doit utiliser input$ fval.

input$

attend la frappe au clavier d'une chaine de caractères, Enter met fin à la saisie. La chaine est envoyée dans **pad** dont l'adresse est laissée sur la pile.

expect

chaine expect : affiche le contenu courant de la chaine et permet de la modifier. La chaine résultat est rangée à la même adresse (l'adresse est dépilée). Utiliser une constante de chaine avec expect est limité puisque sa taille maximale correspond à sa taille initiale, exemple:

" D:*.*" expect ne permettra l'introduction que de 6 caractères!, juste de quoi changer le nom du lecteur. Il faut alors utiliser une chaine normale:

```
80 string CHEMIN
" D:\*.*" CHEMIN $!
CHEMIN expect           \ on aura jusqu'à 80 caractères.
```

key

attend la frappe d'une touche et renvoie un mot contenant dans l'octet inférieur le code ASCII et dans l'octet supérieur le SCAN CODE.

```
key %hff and            \ isole le code ASCII
key 256 w/        \ isole le SCANCODE
key 256 w/mod     \ sépare les deux (ASCII au sommet).
```

inkey

lit le clavier "au vol" et renvoie le code ASCII et le SCANCODE de la touche appuyée ou 0 si aucune touche n'est détectée. Ici le code ASCII est dans le mot faible et le SCAN-CODE dans le mot fort:

```
inkey %hff and    \ isole le code ASCII
inkey 65536 /     \ isole le scancode
inkey 65536 /mod  \ sépare les deux (ASCII au sommet)
```

?terminal

renvoie -1 si un caractère est présent sur le port clavier ou 0 sinon.

3 Les sorties à l'écran

. f.

dépile l'entier au sommet de la pile et l'affiche dans la base courante (pour .) ou le nombre réel (pour f.).

.. f..

affiche l'entier (..) ou le réel (f..) du sommet de la pile (il reste sur la pile).

d. b. o. h.

forcent l'affichage en base 10, 2, 8 ou 16. La base courante n'est pas modifiée, seul cet affichage est affecté.

uh. ub.

Comme h. et b. mais non signé, exemple -1 serait affiché FFFFFFFF. et exactement sur 2, 4, 8 ou 16 chiffres hexa (ou 8, 16, 32, 64 binaires) selon le réglage de **size**.

type

chaine type : affiche la chaine.

." ....."

affiche la chaine comprise entre ." et ", ceci est plus commode que type si le message est une constante de chaine, en fait:

```
" xxx" type
```

est équivalent à

```
." xxx"
```

Notez l'espace nécessaire entre ." et le premier caractère de la chaine.

space

affiche un espace à la position courante du curseur.

spaces

n spaces : affiche n espaces à la position courante du curseur.

cr

saute une ligne et scrolle l'écran si nécessaire.

emit

c emit : affiche le caractère c (les caractères de contrôle sont simplement affichés eux aussi).

cls

vide la page FORTH actuelle et ramène le curseur en haut de l'écran.

4 Les sorties formatées

start

vide la chaine temporaire pour en préparer une nouvelle. Cette chaine est comme pad une zone prédéterminée de la mémoire.

pushs

chaine pushs : ajoute la chaine à droite de la chaine temporaire.

pushv

x pushv : ajoute le nombre x à droite de la chaine temporaire (il sera exprimé dans la base courante).

pushc

c pushc : ajoute le caractère c à droite de la chaine temporaire.

display

affiche à l'écran la chaine temporaire (celle-ci n'est pas vidée, elle peut encore être réaffichée ou même complétée).

getfmt

copie la chaine temporaire dans **pad** et laisse **pad** sur la pile. Ceci est utile lorsque cette chaine est destinée à un autre usage que l'affichage à l'écran.

Utilisons cette stucture pour sortir la valeurs d'un tableau (array) sur l'imprimante:

```
5 array TABLEAU
5 0 do
      i dup *
      i TABLEAU !
loop                          \ remplit le tableau avec les carrés
                              \ des nombres de 0 à 4 (0, 1, 4, 9, 16).

5 0 do
            start                     \ nouvelle ligne
            " TABLEAU[" pushs         \ TABLEAU[
            i pushv                   \ TABLEAU[i
            " ]=" pushs               \ TABLEAU[i]=
            i TABLEAU @ pushv         \ TABLEAU[i]=i*i
            13 pushc 10 pushc         \ retour chariot et saut de ligne
            getfmt ltype              \ envoie à l'imprimante
loop
```

Il existe une autre série de mots pour formatter des données numériques (elle permettent l'affichage avec un nombre fixe de chiffres ou même l'affichage en virgule fixe):

<#

x <# : vide la chaine temporaire et prépare la valeur x à être formattée.

#

effectue la division euclidienne de x par la base courante. Le reste est converti en un caractère ajouté à gauche de la chaine temporaire. Le quotient vient remplacer x sur la pile.

hold

c hold : ajoute le caractère c à gauche de la chaine temporaire (une virgule par exemple).

#s

convertit x en chaine et l'ajoute à la chaine temporaire à gauche, 0 remplace x sur la pile.

#>

termine le formatage, dépile x, copie dans pad la chaine temporaire et laisse pad sur la pile.

#nul_char

modifie le comportement de # lorsque la valeur sur la pile est nulle, par défaut # ajouterait un zéro, on peut fixer un autre caractère pour 'combler' à gauche d'un nombre:

```
<# 48 #nulchar    \ le zéro
#                 \ extirpe un chiffre au moins
32 #nul_char            \ un espace pour la suite
# # # # #         \ sort 5 autres chiffres avec un blanc dès que le
                        \ nombre devient nul
#>
```

Evitera l'écriture 000231 pour le nombre 231 tout en gardant 6 caractères (facilite la mise en page).

Supposons qu'on cacule en centimes et qu'on aimerait afficher en euros (avec donc deux chiffres après la virgule), voici la marche à suivre:

```
: euros
      <#                \ démarrage
            #           \ extirpe le chiffre des unités de centimes
            #           \ celui des dizaines de centimes
            44 hold     \ ajoute une virgule
            #s          \ le reste est affiché devant la virgule
      #> type           \ récupère la chaine et l'affiche
;

4590  euros  affichera     45,90
12    euros  affichera       0,12
```

Supponsons qu'on désire afficher un nombre binaire sur 16 bits avec tous ses chiffres (éventuellement des zéros devant):

```
: 16bits
            <#                  \ prépare
            16 ndo # nloop      \ extirpe 16 chiffres
            #> type             \ fin et affichage
;

21    16bits            affichera   0000000000010101
-1    16bits            affichera   1111111111111111
```

Le TOS

1 Le GEMDOS

Toutes les fonctions du GEMDOS renvoient une valeur, si l'opération s'est mal passée cette valeur est négative et correspond à l'un codes d'erreurs listés en dessous:

codes d'erreur

0	tout va bien	>0	paramètre renvoyé
-32	mauvaise fonction	-49	plus de fichiers
-33	fichier non trouvé	-58	déjà verrouillé
-34	chemin non trouvé	-59	mauvaise requete déverrouillage
-35	plus de handles	-64	limites dépassées
-36	accès refusé	-65	erreur interne
-37	mauvais handle	-66	mauvais format programme
-39	mémoire insuffisante	-67	erreur redimensionnement bloc
-40	mauvaise adresse bloc	-80	trop de liens symboliques
-46	mauvais lecteur	-200	mount point crossed
-48	cross device rename		

fcreate

" fichier" attribut fcreate : crée un fichier et renvoie le handle.

fopen

" fichier" mode fopen : ouvre un fichier (0=read,1=write,2=r+w), renvoie le handle.

fopensize

" fichier" mode fopensize : comme le précédent, mais si l'ouverture s'est bien passée, on obtient sous le handle la taille totale du fichier.

fclose

handle fclose : ferme un fichier.

fread

handle n adr fread : lit n octets sur le fichier et les place à l'adresse adr.
renvoie le nombre d'octets lus.

fwrite

handle n adr fwrite : écrit n octets sur le fichier se trouvant à l'adresse adr.
renvoie le nombre d'octets écrits.

int>

x handle int> : écrit l'entier x sur le fichier (4 octets).

str>

" chaine" handle str> : écrit la chaine sur le fichier au format FORTH.

line>

" chaine" handle line> : écrit la chaîne dans un fichier texte en ajoutant CR, LF comme séparateur.

flt>

%fx handle flt> : écrit le réel x sur le fichier (8 octets).

set>

adr handle set> : écrit l'ensemble pointé par adr sur le fichier.

struct>

adr handle struc> : écrit la structure pointée par adr sur le fichier.

>int

handle >int : lit un entier sur le fichier et l'envoie sur la pile, puis dépose 1 (ok) ou 0.

>str

chaine handle >str : lit une chaine au format FORTH sur le fichier et la place dans la chaine spécifiée, renvoie 1 (ok) ou 0.

>line

chaine handle >line : lit une ligne complète dans un fichier texte (avec CR, LF ou zéro comme séparateur). Renvoie 1 (ok) ou 0. Si la ligne ne tient pas dans la chaîne, elle est tronquée et une lecture suivante permet d'accéder au reste de la ligne.

>flt

handle >flt : lit un réel sur le fichier et l'envoie sur la pile, puis dépose 1 (ok) ou 0.

>set

adr handle >set : lit un ensemble sur le fichier et le place en adr (adresse d'un ensemble de même type) et renvoie 1 (ok) ou 0.

>struc

adr handle >struc : lit une structure sur le fichier et la place en adr (adresse d'une structure de même type) et renvoie 1 (ok) ou 0.

fseek

n handle fseek : déplace le pointeur fichier de n octets (relatifs) et renvoie la nouvelle position par rapport au début du fichier.

fseek>

n handle fseek> : place le pointeur à n octets (>0) par rapport au début du fichier, renvoie la position.

fseek<

n handle fseek< : idem mais à partir de la fin du fichier.

fdelete

" fichier" fdelete : efface le fichier et renvoie 0 si ok.

fgetdta

fgetdta : renvoie l'adresse de la DTA actuelle:
- adr : 21 octets réservés
- adr+21 : attribut
- adr+22 : heure
- adr+24 : date
- adr+26 : taille
- adr+30 : nom (14 octets terminés par un nul).

fsetdta

adr fsetdta : fixe la DTA à l'adresse indiquée (ne renvoie rien).

dta

dta : renvoie l'adresse d'une zone de 44 octets servant de DTA et déjà fixée par le FORTH au lancement du programme.

ffirst

" chemin" attribut ffirst : cherche le 1° fichier correspondant au masque donné. renvoie 0 si trouvé et remplit la DTA, code d'erreur sinon.

fnext

fnext : cherche le fichier suivant, renvoie 0 et remplit la DTA.

dir

" masque" dir : affiche les fichiers correspondants au masque. Par défaut seul le nom et la taille sont listés. Si on désire la date et l'heure, on ajoute **d** ou/et **t** entre parenthèses au début du masque:

```
" (dt)*.PRG" dir  \ affiche nom, taille, date et heure des fichiers PRG
```

dsetdrv

d dsetdrv : fixe le lecteur d, renvoie un masque des lecteurs présents.

dgetdrv

dgetdrv : renvoie le numéro du lecteur actuel (A=0).

dsetpath

" chemin" dsetpath : fixe le répertoire courant et renvoie 0.

dgetpath

chaine n dgetpath : renvoie dans la chaine le chemin courant du lecteur n et renvoie 0.

dfree

adr n dfree : renvoie les informations sur le lecteur n:

adr	: clusters libres	adr+8	: octets par secteur
adr+4	: total clusters	adr+12	: secteurs par cluster

dcreate

" dossier" dcreate : créé un dossier et renvoie 0.

ddelete

" dossier" ddelete : efface un dossier et renvoie 0.

frename

0 " ancien" " nouveau" frename : renommne un fichier et renvoie 0.

fattrib

" fichier" f attribut fattrib : modifie (f=1) ou lit (f=0) l'attribut du fichier, l'attribut est renvoyé.

fduplic

n fduplic : donne au périphérique n un handle normal, renvoie le handle.

fforce

n handle fforce : redirige le périphérique n sur un fichier.

malloc

n malloc : réserve n octets et renvoie l'adresse du bloc. Si n=-1 renvoie la taille maximale disponible.

mfree

adr mfree : libère le bloc alloué à l'adresse adr, renvoie 0.

mshrink

0 adr n mshrink : réduit à n octets le bloc déjà réservé situé à l'adresse adr.

cauxis

cauxis : renvoie 0 si buffer série vide, %hFFFF sinon.

cauxin

cauxin : attend un caractère sur le port série et le renvoie sur la pile.

cauxos

cauxos : renvoie 0 si port série prêt, %hFFFF sinon.

cauxout

c cauxout : écrit le caractère c sur le port série.

cprnos

cprnos : renvoie 0 si imprimante non prête, %hFFFF sinon.

cprnout

c cprnout : envoie le caractère c sur le port imprimante et renvoie -1 si ok ou 0 si erreur.

cconis

cconis : renvoie 0 si buffer clavier vide, %hFFFF sinon.

crawio

c crawio : lit (c=255) un caractère au clavier et renvoie son code clavier+ASCII sur la pile ou (c<255) affiche le cractère c et renvoie une valeur quelconque.

cconws

" chaine" cconws : affiche la chaine.

cconrs

adr cconrs : saisit une chaine au clavier de n octets maxi et renvoie les caractères en adr+2:

adr	: n (entrée)
adr+1	: n'=nbr de caractères lus (sortie)
adr+2	: n' octets.

pterm0

ptermo : termine un programme, fonction équivalente à **system**.

pterm

n pterm : appelle **system** et renvoie n au processus père.

ptermres

p n ptermres : appelle **system**, renvoie n au père et conserve p octets en mémoire à partir de la page de base.

pexec

mode " nom" " commande" " envir" pexec : charge et/ou exécute un programme selon le mode.

super

super : passe en mode superviseur.

user

user : revient au mode utilisateur.

tgettime

tgettime : renvoie l'heure courante au format GEMDOS sur 16 bits:
hhhh hmmm mmms ssss, les secondes vont de 0 à 29 et doivent être doublées de 0 à 58.

tgetdate

tgettime : renvoie la date courante au format GEMDOS sur 16 bits:
jjjj jmmm maaa aaaa, l'année de 0 à 63 doit être augmentée de 1980.

tsettime

x tsettime : fixe l'heure courante, x est l'heure au format GEMDOS sur 16 bits.

tsetdate

x tsetdate : fixe la date courante, x est la date au format GEMDOS sur 16 bits.

timetostr

x timetostr : convertit une heure au format GEMDOS en une chaine stockée dans **pad** (qui est laissé sur la pile) selon le format HH:MM:SS.
la valeur x peut provenir de **tgettime** ou de la valeur time d'un fichier dans la DTA lors d'une recherche avec **ffirst** et **fnext**.

datetostr

x datetostr : convertit une date au format GEMDOS en une chaine stockée dans **pad** (qui est laissé sur la pile) selon le format JJ/MM/AAAA ou MM/JJ/AAAA dans le cas ou le code langue est 0=US.
la valeur x peut provenir de **tgetdate** ou de la valeur date d'un fichier dans la DTA lors d'une recherche avec **ffirst** et **fnext**.
Par défaut, l'année est sur quatre chiffres. En mettant à 1 le bit 31 de x, on obtient une année sur 2 chiffres (comme le fait **dir**)
Exemple :

```
tgetdate %h80000000 or datetostr type
```

strtotime

" hh:mm:ss" strtotime : convertit une chaine vers le format time GEMDOS.

strtodate

" jj/mm/aa" strtodate : convertit une chaine vers le format date GEMDOS.
Si le code langue est 0 (US), il faut fournir mm/jj/aa.
Si l'année est sur 2 chiffres, elle est considérée comme étant dans l'intervalle [1980;2079], mais on peut préciser l'année sur 4 chiffres.

Exemples:

```
tgettime timetostr type            \ affiche l'heure système

DTA fsetdta
" TEST.PRG" 0 ffirst 0=            \ cherche le fichier TEST.PRG
if                                 \ si trouvé
      DTA 22 + w@ timetostr type cr \ affiche son heure de création
      DTA 24 + w@ datetostr type cr \ et sa date
then
```

langage

variable sur deux octets contenant le code de langue détectée dans la ROM ou fixé par le fichier FORTH.INF.

ddate

" jj/mm/aa" " jj/mm/aa" ddate : renvoie le nombre de jours entre deux dates (positif ou négatif dans le sens date1-date2). Si les dates tiennent dans le format GEMDOS [1980 ;2079], le siècle n'est pas nécessaire. Sinon, on doit indiquer l'année sur 4 chiffres.

dofw

" jj/mm/aa" dofw : renvoie le numéro du jour de la semaine d'une date donnée. 0=samedi et 6=vendredi. Pad est en plus rempli du nom du jour (anglais ou français selon le code langage) mais pas empilé.

date+

" jj/mm/aa" n date+ : renvoie dans pad la date décalée de n jours par rapport à celle indiquée. La valeur n pouvant être positive ou négative.

Exemples:

```
tgettime timetostr " 4/7/1969" ddate .
      \ affiche le nombre de jours écoulés depuis le 4 juillet 1969.
tgetdate datetostr 1000 date+ type
      \ affiche la date qu'il sera dans 1000 jours.
" 8/5/1945" dowf . Space pad type
      \ affiche le code et le nom du jour de la semaine du 8 mai 45.
```

unpackdate

" jj/mm/aa" unpackdate : renvoie le jour, le mois et l'année sur la pile (l'année au sommet). Le format de l'année (2 ou 4 chiffres) est respecté.

packdate

jj mm aa packdate : renvoie dans pad (déposé sur la pile) la chaîne correspondant à la date spécifiée. Le format de l'année, 2 ou 4 chiffres, est res pecté.

ARGC

renvoie le nombre d'arguments sur la ligne de commande. Si seul le nom de programme est disponible, ARGC = 0. Cette valeur est limitée à 63.

ARGV

tableau de chaînes contenant les différents arguments de la ligne de commande. En particulier:

0 ARGV renvoie l'adresse du chemin+nom de l'application en cours (sauf pour un accessoire ou un programme du dossier AUTO)

Exemple, si un programme FORTH est appelé ainsi:

```
C:\TEST\MYPROG.PRG -lFILE.TXT -t3
      On aura: ARGC = 2
      0 ARGV      pointe sur "C:\TEST\MYPROG.PRG"
      1 ARGV      pointe sur "-lFILE.TXT"
      2 ARGV      pointe sur "-t3"
```

Le FORTH et les programmes compilés par lui supportent le protocole par ligne de commande (dans ce cas **pads** est utilisé pour ARGV[0]) ainsi que le protocole étendu ARGV=. (dans ce cas **pads** est libre).

2 Le BIOS

Comme pour le GEMDOS, la valeur renvoyée par ces fonctions est soit paramètre (positif) ou un code d'erreur (négatif):

codes d'erreur

0	Ok	-9	plus de papier
-1	erreur générale	-10	erreur d'écriture
-2	lecteur non prêt	-11	erreur de lecture
-3	commande inconnue	-12	périphérique protégé en écriture
-4	erreur CRC	-14	media-change détecté
-5	mauvaise requête	-15	périphérique inconnu
-6	piste non trouvée	-16	secteurs défectueux détectés
-7	support inconnu	-17	insérer un autre dique
-8	secteur non trouvé		

Les périphériques d'entrée sortie standards ont chacun un handle prédéterminé:

0	PRN: port parallèle
1	AUX: port série
2	CON: clavier, écran
3	MIDI
4	IKBD intelligent keyboard
5	écran (sans codes de contrôle

bconstat

n bconstat : renvoie 0 si aucun caractère n'est disponible sur le périphérique n (en entrée), -1 sinon.

bcostat

n bcostat : renvoie 0 si le périphérique n n'est pas prêt à emettre, -1 sinon (en sortie).

bconin

n bconin : lit un caractère sur le périphérique n.

bconout

n c bconout : envoie le caractère c sur le périphérique n.

rwabs

f adr n s d rwabs : selon f, lit ou écrit n secteurs de l'unité d à partir du secteur s, les échanges se faisant à l'adresse adr.

f bit0 : 0=lire, 1=écrire
bit1 : 0=tenir compte de media-change, 1=ingnorer

Cas particulier adr=0, alors simule (n=0) ou efface (n=1) le changement de disque.

mediach

d mediach : indique le changement de disque sur l'unité d, renvoie 0 (pas changé), 1 (peut-être), 2 (surement changé).

drvmap

drvmap : renvoie un masque de bits indiquant les lecteurs connectés.

getmpb

adr getmpb : copie le Memory Parameter Bloc à l'adresse adr (12 octets).

getbpb

d getbpb : renvoie l'adresse d'un bloc concernant l'unité d ou 0 si erreur.

adr	: octets par secteur	adr+10:	1° sect de 2° FAT
adr+2	: secteurs par cluster	adr+12:	1° sect du 1° cluster libre.
adr+4	: octets par cluster	adr+14:	clusters de données
adr+6	: secteurs dans repertoire	adr+16:	FAT 12 bits(0), 16bits(1)
adr+8	: secteurs par FAT		

kbshift

m kbshift : lit (m=-1) ou fixe (autre valeur) l'état des touches de controle:

bit0	: shift droit	bit2	: control
bit1	: shift gauche	bit3	: alternate

setexec

n vect setexec : donne la valeur vect au vecteur d'exeption n.

3 Le XBIOS

physbase

physbase : renvoie l'adresse de l'écran physique.

logbase

logbase : renvoie l'adresse de l'écran logique.

getrez

getrez : résolution actuelle:

0 : maxi 320x240 1: 640x200
2 : 640x400 ou 640x480 (sur Falcon)
4 : 640x480 (sur TT) 6: 1280x960
7 : 320x480

setscreen

l p r setscreen : fixe la base logique (l), physique (p) et la résolution (r) de l'écran, mettre -1 si un paramètre est inutilisé.

setpalette

adr setpalette : fixe la palette de couleurs (adr pointe sur 16 mots).

setcolor

n c setcolor : fixe la couleur (c) du registre n.

floprd

adr o d s p f n floprd : lit n secteurs de la piste p sur la face f du lecteur d à partir du secteur s, les octets lus sont entreposés à l'adresse adr. renvoie o si Ok.

flopwr

adr o d s p f n flopwr : comme floprd mais en écriture.

flopver

adr o d s p f n flopver : mêmes paramètres que floprd, mais compare le contenu de la mémoire à celui sur disque.
renvoie 0 si Ok, ou un code d'erreur et met en adr la liste des secteurs défectueux.

format

d f p s format : formate l'unité d (0=A,1=B) avec f faces (1 ou 2), p pistes (80 par exemple) et s secteurs par piste (9,10 ou 11). Avec s=-9 on obtient un formatage accélérant les accès disques.

mfpint

n adr mfpint : donne au vecteur n du MFP l'adresse adr.

jenabint

n jenabint : active l'interruption n du MFP.

jdisint

n jdisint : désactive l'interruption n du MFP.

xbtimer

tim ctrl dat adr xbtimer : lance un timer du MFP.

kbdvbase

kbdvbase : renvoie l'adresse d'un tableau d'adresses des routines clavier:

adr : entrée MIDI	adr+20 : routines heures
adr+4 : erreur clavier	adr+24 : routine joystick
adr+8 : erreur MIDI	adr+28 : vect. système MIDI
adr+12: état IKBD	adr+32 : vect. système IKBD
adr+16: routines souris	

midiws

n adr midiws : envoie au port MIDI n+1 octets trouvés à l'adresse adr.

giaccess

val reg giaccess : lit (bit7 de reg à 0) ou écrit (bit7 de reg à 1) une valeur dans le registre reg du circuit son. En cas de lecture val est sans signification, en cas d'écriture la valeur renvoyée est sans signification.

dosound

adr dosound : envoie une chaine de commandes au circuit son. Renvoie l'adresse précédente.
Et -1 dosound renvoie l'adresse en cours ou 0 si le son est fini.

offgibit

masque offgibit : annule certains bits du port A (ceux à 0 dans masque).

ongibit

masque ongibit : fixe certains bits du port A (ceux à 1 dans masque).

setptr

masque setprt : lit (masque=-1) ou fixe les caractèristiques de l'imprimante, renvoie les caractéristiques.

rsconf

baud ctrl ucr rsr tsr scr rsconf : configure le port série.

iorec

n iorec : renvoie l'adresse du buffer de réception du périphérique n:
n=0 (série), 1 (clavier) ou 2 (MIDI):

adr	: adresse du buffer	adr+8	: dernière position
adr+4	: taille du buffer	adr+10	: bas du buffer
adr+6	: première position à lire	adr+12	: haut du buffer

initmous

type param vect initmous : initialise la gestion de la souris.

keytbl

unshift shift capslock keytbl : redéfinit les correspondances entre codes claviers et code ASCII.

bioskeys

bioskeys : restaure la définition standard du clavier.

kbrate

delai repet kbrate : fixe le temps avant répétition (delai<256) et celui entre deux répétitions (repet<256), renvoie les valeurs antérieures:
bit0-7 : delai bits8-15 : repet.
Si les deux paramètres valent -1, seul l'état est renvoyé.

random

random : renvoie un nombre aléatoire sur 24 bits.

supexec

adr supexec : exécute une routine en mode superviseur (routine pointée par adr, et se terminant par RTS, renvoie la valeur de D0).

vsync

vsync : attend la fin du balayage d'un écran.

nvmaccess

op start n buf nvmaccess : lit (op=0) ou écrit (op=1) n octets à partir de l'octet start (50 octets maxi) depuis ou vers la mémoire (adresse buf). Avec op=2 la Non Volatile Ram est remise à zéro.
renvoie 0 si Ok ou un code d'erreur négatif.

4 Autres fonctions

Les fonctions non implantées peuvent être appelées "à la main" à l'aide des trois mots suivants:

gemdos

on l'utilise en empilant les paramètres dans le dictionnaire comme suit:

```
here              \ conserve le début de la zone
opcode w,         \ n° de fonction toujours sur 2 octets
param1 w, (si 2 octets) ou , (si 4 octets)
param2 w, ....
valeur de D0 ,    \ mettre 0 si inutilisée
gemdos
```

renvoie alors la valeur de D0 et restaure le dictionnaire.

bios

xbios

même ordre de paramètres que gemdos.

Les fonctions VDI

1 Présentation

Pour tous les appels VDI il y a différents tableaux à remplir, en réponse, outre les sorties graphiques, cetains paramètres seront renvoyés dans les tableaux de sortie.

Tableaux d'entrée:

control

dépose sur la pile l'adresse du champ control:

control : code de la fonction
control+2 : nombre de couples de coordonnées dans ptsin
control+4 : nombre de couples de coordonnées dans ptsout
control+6 : nombre de valeurs dans intin
control+8 : nombre de valeurs dans intout
control+10 : code de sous-fonction
control+12 : handle de la station de travail
control+14 à +24 : selon la fonction

intin

dépose sur la pile l'adresse du champ intin ou seront passés les paramètres d'entrée.

ptsin

dépose sur la pile l'adresse du champ ptsin ou seront passées les coordonnées d'entrée.

Tableaux de sortie:

intout

dépose sur la pile l'adresse du champ intout dans lequel est renvoyé le résultat de l'opération.

ptsout

dépose sur la pile l'adresse du champ ptsout dans lequel sont renvoyées des coordonnées de sortie.

*Dans le FORTH en **utilisation normale**, control, intin et ptsin sont remplis pour vous. Il ne reste plus qu'à lire les données éventuelles dans intout et ptsout. Explications sur le handle: c'est le numéro attribué à la station graphique (l'écran) déjà ouvert par le FORTH. Là encore en utilisation normale il ne sera d'aucune utilité.*

Coordonnées des points:

L'affichage se fait dans une fenêtre dont l'utilisateur ne peut pas prévoir la position. Aussi, au lieu d'obliger l'utilisateur à effectuer les calculs lui même, il existe un mode relatif dans lequel toutes les coordonnées passées au VDI seront décalées pour n'être affichées que dans la fenêtre FORTH actuelle.

relative

le coin supérieur gauche de la fenêtre ou page actuelle devient le point 0,0. Toutes les sorties graphiques sont donc décalées. C'est le mode par défaut.

absolute

c'est le coin supérieur gauche de l'écran qui est le point 0,0. Les sorties graphiques pourraient s'effectuer hors de la page actuelle.

Fonctions manuelles:

Même si le FORTH s'occupe de tout, il peut être utile de lancer une fonction "à la main" (si par exemple elle n'est pas implantée), en **utilisation normale** ces mots sont inutiles:

handle

variable sur 2 octets contenant le handle.

vctrl

f p i sf vctrl : remplit le champ control avec f: code de la fonction, p: nombre de couples de ptsin, i: nombre de valeurs de intint, sf:code de sous fonction. Handle est mis automatiquement en control+12.

vintin!

remplit le champ intin en tenant compte de la valeur en control+6. Les valeurs sont prises sur la pile.

ptsin!

remplit le champ ptsin en tenant compte de la valeur en control+2. Les valeurs sont prises sur la pile mais ne subissent aucun ajustement même en mode relatf!

16b$

" chaine" adr 16b$: copie la chaine à partir de adr en passant les caractères de 8 à 16 bits, renvoie la longueur de la chaine.

vdi

lance la fonction en supposant que control, intin et ptsin sont correctement alimentés.

vdi!

remplit d'abord le champ ptsin (avec les décalages), puis le champ intin et enfin lance l'exécution de la fonction. Control doit être rempli d'abord. C'est le mix de ptsin! vintin! et vdi.

2 Les fonctions générales:

Pour l'écran, vous n'aurez normalement pas à utiliser les fonctions d'ouverture et de fermeture de stations. Sous l'interpréteur, une station est déjà ouverte. Lorsque vous créez un programme indépendant, un simple appel à **fastopen** réalise l'ouverture.

Si votre programme est lancé depuis le dossier AUTO, un appel à **fastopen** ouvre une station physique (**v_opnwk**) et donne accès aux autres appels VDI.

v_opnvwk

ouvre une station virtuelle de travail. Le FORTH n'est prévu pour travailler qu'avec une seule station ouverte. Pour utiliser correctement cette fonction, il faut donc sauvegarder la valeur de handle car elle sera modifiée, puis il faudra la restaurer après l'appel de v_clsvwk. Si on désire écrire dans deux stations à la fois, il faudra jongler avec les deux handles en alimentant correctement la variable handle avant chaque appel VDI.

id l cl m cm f ct sr mr cr c v_opnvwk

id	identificateur (celui de ASSIGN.SYS si station externe: imprimante=21, metafile=31...) ou getrez+2 pour l'écran.
l	type de ligne
cl	couleur de ligne
m	type de marques
cm	couleur des marques
f	fonte de caractères
ct	couleur de texte
sr	style de remplissage
mr	motif de remplissage
cr	couleur de remplissage
c	système de coordonnées : 2 pour l'écran, 0 pour les externes (de 0 à 32767).

en retour, handle contient le numéro de la station ou 0 si l'ouverture n'a pas été possible.

En fait, pour une station écran, le plus simple est de mettre id=0, dans ce cas le FORTH le remplace par getrez+2 et alimente correctement handle avant l'appel avec la valeur retournée par graf_handle.

work_out

dépose sur la pile l'adresse du tableau intout contenant les informations renvoyées par v_opnvwk lors de l'ouverture par le FORTH.

vq_extnd

m vq_extnd : renvoie des informations concernant la station de travail, si m=0 ce sont les mêmes informations que celles contenues dans work_out, si m=1 ce sont des informations supplémentaires.

screen_info / graphic_card

screen_info renvoie un mot long correspondant au mode écran en cours. Ceci n'est pas un appel VDI mais une fonctionnalité du FORTH.

BPP: mot de poids fort:
- contient le nombre de bits par pixel sur 16 bits
- si le bit 15 est mis, il y a eu une erreur

SUB: mot de poids faible:
- contient le code d'encodage des bits sur 16 bits
- si le bit 15 est mis, une erreur VDI est survenue mais SUB a été déterminé par des accès directs à la mémoire écran.

graphic_card renvoie 0 pour un mode compatible Atari, 1 sinon.

BPP		SUB	
0001	monochrome	0000	Un plan
0002	4 couleurs	0000	Mode Atari, deux plans entrelacés
		0001	deux plans séparés
		0002	2 bits/pixel, 4 pixel/octet
0004	16 couleurs	0000	mode Atari, plans entrelacés
		0001	mode PC, plans séparés
		0002	4 bits/pixel, 2 pixels/octet
0008	256 couleurs	0000	mode Atari, plans entrelacés
		0001	mode PC, plans séparés
		0002	un octet par pixel, NOVA
		0003	un octet par pixel, Matrix
0010	High Color	0000	mode Atari rrrrrvvv vvvbbbbb
		0001	mode PC vvvbbbbb rrrrrvvv
		0002	mode PC xrrrrrvv vvvbbbbb
		0003	mode PC vvvbbbbb xrrrrrvv
0018	True Color 24 bits	0000	RVB, 3 octets
		0001	BVR, 3 octets
0020	True Color 32 bits	0000	xRVB, 4 octets
		0001	xBVR, 4 octets
		0002	BVRx, 4 octets
		0003	RVBx, 4 octets

Si le programme est lancé depuis le dossier AUTO, screen_info renvoie zéro tant que fastopen n'a pas été exécuté.

vqt_devinfo

id vqt_devinfo : renvoie 0 si le driver correspondant n'est pas disponible et 1 dans le cas contraire. id est le numéro qui précède le nom du driver dans le fichier ASSIGN.SYS.

ptsout : 0 ou 1
intout : chaine correspondant au nom du driver

v_clsvwk

ferme une station virtuelle de travail. Il faut redonner à handle la valeur précédente.

savevdipal

adr savevdipal : sauve la palette actuelle au format VDI (6 octets par couleur) à partir de l'adresse spécifiée. (max 256 couleurs même en TC, donc il suffit de 1536 octets à l'adresse indiquée.)

setvdipal

adr setvdipal : fixe la palette avec les données en adr au format VDI. Mêmes limitations qu'avec savevdipal.

pal2xbios

PALv PALx n pal2xbios
: transforme une palette VDI en une palette XBIOS en tenant compte des indexs VDI/XBIOS qui ne sont pas les mêmes.

PALv adresse de la palette VDI (6 octets par couleur, rouge, vert, bleu de 0 à 1000)

PALx adresse de la palette XBIOS (2 octets par couleur)

n nombre de couleurs (2, 4, 16 ou 256)

- si n est positif, c'est le format STE/TT qui est utilisé avec des composantes allant de 0 à 15 (donc couleurs de %h0000 à %h0FFF, pour rester compatible avec le ST, les quatre bits 3210 de chaque composante sont ordonnés ainsi: 0321)
- si n est négatif, c'est le format ST qui est utilisé avec des composantes allant de 0 à 7 (donc couleurs %h0000 à %h0777).

v_opnwk

Mêmes paramètres que v_opnvwk. La station ouverte est ici la station physique, pour l'écran cet appel est déjà réalisé par le bureau, donc on n'ouvre plus que des stations virtuelles. Par contre pour l'imprimante (id=21) ou un metafile (id=31) il faut ouvrir une station physique dans laquelle on pourra travailler soit directement, soit en y ouvrant plusieurs stations virtuelles (avec des paramètres différents). Pour v_opnwk, placer 0 dans id n'est pas reconnu par le FORTH contrairement à v_opnvwk.

En retour, la variable handle contient 0 si erreur ou le handle de la station ouverte si tout s'est bien passé.

v_clswk

ferme une station physique. Il faut alors restaurer la variable handle.

v_clrwk

efface une station de travail. Si elle est virtuelle, elle efface aussi toutes les autres dépendant de la même station physique. Si elle est de type imprimante, elle réalise une éjection de page.
A l'ouverture, une station est automatiquement effacée.

vq_gdos

renvoie une valeur indiquant la présence de GDOS ou non:

```
-2      : non installé
autre   : installé et   %h5f464e54  : FONTGDOS
                        %h5f46534d  : FSMGDOS ou SpeedoGDOS
                        autre       : GDOS 1, 1.1 ou 1.2
```

vswr_mode

m vswr_mode : fixe le mode d'écriture pour les sorties graphiques. Le paramètre m peut prendre les valeurs suivantes:

```
m=1   remplacement
m=2   transparent
m=3   XOR
m=4   transparent inverse
```

vs_color

i r v b vs_color : indique les taux de rouge, vert et bleu pour la couleur d'index i. Les taux sont des valeurs de 0 à 1000, l'index dépend du nombre de couleurs supporté.

vq_color

i f vq_color : demande la composition de la couleur d'index i. Le paramètre f influe sur le comportement de la fonction:

f=0 renvoie les valeurs demandées par vs_color.
f=1 les valeurs sont celles effectivement prises en compte.

Paramètres renvoyés dans intout:

intout : index i (ou -1 si la couleur n'existe pas)
intout+2 : taux de rouge (0 à 1000)
intout+4 : taux de vert
intout+6 : taux de bleu

v_get_pixel

x y v_get_pixel : renvoie la couleur (en mode 16bits ou plus) ou l'index de couleur du point aux coordonnées x,y.

intout : hardware index ou 0
intout+2 : VDI index ou RRRRRVVVVVVBBBBB.

vq_key_s

Permet de connaître l'état des touches spéciales du clavier. La valeur renvoyée est un masque de bits:

intout : bit 0 shift droit
: bit 1 shift gauche
: bit 2 Control
: bit 3 Alternate

vs_clip

Active ou désactive le clipping:

x1 y1 x2 y2 1 vs_clip active le clipping, les coordonnées passées sont celles des deux coins opposés du rectangle.
0 vs_clip désactive le clipping, tout l'écran est à nouveau accessible.

vex_timv

a vex_timv installe une routine à l'adresse "a" qui sera exécutée 50 fois par seconde. L'adresse de l'ancienne routine est retournée dans control+10. La routine que l'on installe devrait se terminer en sautant vers l'ancienne, sans modifier les registres et en n'utilisant que des appels BIOS et XBIOS.

v_updwk

sort la page courante sur la station actuelle (celle pointée par handle).

v_output_window

x y x' y' v_output_window : même effet que v_updwk, sauf que seule une portion est concernée.

v_form_adv

même effet que v_updwk mais réalise en plus un éjection de page. La page actuelle n'est pas effacée.

v_pgcount

n v_pgcount : réalise n+1 copies de la même page sur une station Laser.

v_clear_disp_list

efface la page actuelle sans la sortir sur la station ouverte.

3 Les sorties de texte

vst_load_fonts

charge les fontes listées dans ASSIGN.SYS si GDOS est présent.
intout: nombre de fontes chargées.
La fonte 1 étant toujours diponible, la première fonte chargée a pour index 2, la dernière n+1. Il faut charger les fontes pour chaque station ouverte.

vst_unload_fonts

libère la place occupée par les fontes chargées à l'aide de vst_unload_fonts.

vst_charmap

m vst_charmap : sélectionne l'utilisation de chaines standard ATARI (m=1) en 8 bits ou de chaines Bitstream (m=0) en 16 bits par caractère. Cet appel doit précéder l'utilisation de v_ftext16, v_ftext_offset16 et vqt_f_extent16.

vqt_get_table

renvoie l'adresse d'un tableau de 7 adresses pointant sur des tables de correspondances entre la fonte ATARI et les fontes Bitstream:
intout : adresse (mot long). Chaque table contient 224 mots indiquant les indexes 16 bits Bitstream des caractères ASCII 32 à 255.

v_gtext

x y " chaine" v_gtext : sort la chaine à la position x,y précisée.

v_ftext

x y " chaine" v_ftext : sort la chaine à la position x,y précisée en utilisant une fonte vectorielle (SpeedoGDOS).

v_ftext_offset

adr_off x y " chaine" v_ftext_offset : même principe que v_ftext, sauf que l'adresse adr_off pointe sur un tableau d'offsets pour chaque caractère à sortir. Un couple d'offsets est une paire de coordonnées relatives sur 2 octets chacune.

v_ftext16

x y adr v_ftext16 : même principe que v_ftext sauf que l'adresse adr pointe sur une chaine dont chaque caractère utilise 16 bits au lieu de 8 (Voir vst_charmap).

v_ftext_offset16

adr_off x y adr v_ftext_offset16 : même principe que v_ftext_offset16, sauf que les caractères sont sur 16 bits chacun (Voir vst_charmap).

v_justified

x y " chaine" l m c v_justified : sort la chaine à la position x,y précisée. Les 3 autres paramètres influent sur la longueur du texte à l'écran:

l : longueur totale imposée (en points écran)
m : jouer sur l'espace entre les mots (1:oui, 0:non)
c : jouer sur l'espace entre les lettres (1:oui, 0:non)

vst_height

h vst_height : sélectionne la taille des caractères en pixels. On obtient en retour les valeurs effectivement prises en compte:

ptsout : largeur d'un caractère
ptsout+2 : hauteur d'un caractère
ptsout+4 : largeur d'une cellule
ptsout+6 : hauteur d'une cellule.

vst_point

h vst_point : sélectionne la taille des caractères en points (1/72 de pouce). Si la taille demandée n'est pas listée dans EXTEND.SYS, c'est celle immédiatement en dessous qui est choisie. Les paramètres renvoyés sont les mêmes que pour vst_height.

vst_arbpt

h vst_arbpt : sélectionne la taille en points d'une foncte vectorielle (sous Speedo-GDOS). La valeur de h n'est pas limitée aux valeurs présentes dans ASSIGN.SYS. renvoie les mêmes paramètres que vst_height.

vst_arbpt32

hi hf vst_arbpt32 : même effet que vst_arbpt sauf que la taille est au format fix31 séparé en partie entière (hi) et fractionnaire (hf).

vst_setsize

l vst_setsize : fixe la largeur en points d'une fonte vectorielle indépendemment de sa hauteur (vst_arbpt). renvoie les mêmes valeurs que vst_height.

vst_setsize32

li lf vst_setsize32 : même effet que vst_setsize sauf que la largeur est au format fix31 séparé en partie entière (li) et partie fractionnaire (lf).

vst_rotation

a vst_rotation : sélectionne l'angle d'écriture en 1/10° de degré. On trouve dans intout la valeur sélectionnée:

intout : angle effectivement pris en compte.

vst_skew

a vst_skew : fixe l'angle d'inclinaison en 1/10° de degré pour une fonte vectorielle. On peut obtenir ainsi de l'italique vers la gauche (valeurs -900 à 0) ou de l'italique vers la droite (0 à +900).

intout : inclinaison prise en compte.

vst_font

id vst_font : sélectionne une fonte. La numéro 1 est toujours disponible (fonte système). L'identificateur est un numéro personnel de la fonte qui est renvoyé par vqt_name.

intout : identificateur de la fonte sélectionnée.

vqt_fontheader

adr vqt_fontheader : renvoie à l'adresse adr une copie du header de la fonte actuelle sous Speedo (minimun 421 octets), renvoie en plus dans **pad** le nom complet du fichier correspondant à cette fonte. **Pad** est déposé sur la pile.

vst_color

i vst_color : fixe la couleur pour les sorties de texte.

intout index de couleur sélectionné.

vst_effects

m vst_effects : sélectionne différents effets de texte selon le masque de bits m:

m	bit 0:	gras	bit 3:	souligné
	bit 1:	léger	bit 4:	contour
	bit 2:	italique	bit 5:	ombre

intout : masque effectivement pris en compte.

vst_alignment

h v vst_alignment : spécifie à quel point du texte correspondent les coordonnées x,y des fontions v_gtext ou ses dérivées. Par défaut x pointe sur la gauche du texte et y sur le bas.

h=0	: x à gauche	v=0	: y pointe sur la ligne d'écriture
h=1	: x au centre	v=1	: y pointe sur le sommet des minuscules
h=2	: x à droite	v=2	: y pointe sur le sommet des majuscules
		v=3	: y pointe sur le bas de la cellule
		v=4	: y pointe sur le bas des lettres
		v=5	: y pointe sur le haut de la cellule

intout : h sélectionné
intout+2 : v sélectionné.

vqt_attributes

renvoie l'ensemble des paramètres pour les sorties de texte:

intout	: id de la fonte	ptsout	: largeur de caractère
intout+2	: index de couleur	ptsout+2	: hauteur
intout+4	: angle	ptsout+4	: largeur d'une cellule
intout+6	: pos. horizontale	ptsout+6	: hauteur
intout+8	: pos. verticale		
intout+10	: mode d'écriture		

vqt_extent

" chaine" vqt_extent : renvoie les coordonnées du cadre contenant la chaine en prenant en compte tous les effets de texte:

ptsout	: x haut gauche	ptsout+8	: x bas droite
ptsout+2	: y haut gauche	ptsout+10	: y bas droite
ptsout+4	: x haut droite	ptsout+12	: x bas gauche
ptsout+6	: y haut droite	ptsout+14	: y bas gauche

vqt_f_extent

" chaine" vqt_f_extent : même action que vqt_extent, sauf que la fonction s'applique à des fontes gérées par SpeedoGDOS et tient compte de leurs spécificités.

vqt_f_extent16

adr vqt_f_extent16 : même action que vqt_f_extent mais l'adresse pointe sur des caractères Speedo 16 bits au lieu de 8 bits. (Voir vst_charmap).

vqt_width

c vqt_width : renvoie des informations sur le caractère c:

ptsout	: largeur de la cellule
ptsout+4	: espace disponible à gauche
ptsout+8	: espace disponible à droite
intout	: -1 si le caractère n'est pas dans la fonte.

vqt_name

n vqt_name : renvoie dans **pad** le nom de la fonte de numéro n:
n : numéro de la fonte (selon le nombre renvoyé par vst_load_fonts)
intout : identificateur de la fonte (à conserver pour le passer à vst_font).
pad est déposé sur la pile.

vqt_fontinfo

renvoie des informations (en pixels) sur la fonte actuelle:

intout	: ASCII de départ
intout+2	: ASCII de fin
ptsout	: largeur cellule
ptsout+2	: écart base/bottom line
ptsout+4	: elargissement lors d'effets
ptsout+6	: écart descent/base line
ptsout+8	: offset gauche pour l'italique
ptsout+10	: écart half/base line
ptsout+12	: offset droit pour l'italique
ptsout+14	: écart ascent/base line
ptsout+18	: écart top/base line

v_flushcache

vide le cache (cette zone sert à stocker les images Bitmap des fontes proportionnelles lorsqu'une taille a été choisie, ceci évite de recalculer l'image écran d'un caractère à chaque appel).

v_savecache

" fichier" v_savecache : sauvegarde le cache sur disque, la chaine contient le nom complet du fichier à créer.

v_loadcache

" fichier" mode v_loadcache : charge dans le cache un fichier précédement sauvé. Si mode=0, le fichier est ajouté au cache existant, si mode=1, le cache est préalablement vidé.

vqt_cachesize

n vqt_cachesize : retourne dans intin la taille d'un des deux caches. Si n=0, c'est la cache Bitmap, si n=1 c'est l'autre.

intint : sur 4 octets, taille du cache.

v_getbitmap_info

c v_getbitmap_info : renvoie des informations sur l'image bitmap du caractère c en tenant compte des attributs actuels de texte:

intout	: largeur	intout+12	: xoff
intout+2	: hauteur	intout+16	: yoff
intout+4	: advx	intout+20	: adresse du bitmap
intout+8	: advy		

advx,advy ainsi que xoff,yoff sont au format fix31.

v_getoutline

c n adr1 adr2 v_getoutline : renvoie les informations nécessaires pour reconstruire le caractère c à l'aide de courbes de Bézier. Les adresses adr1 et adr2 pointent sur deux buffers qui recevront les coordonnées des points et leurs flags (voir v_bez), n indique le nombre maximum de points que ces buffers peuvent contenir.

intout : nombre de points renvoyés.

vqt_advance

c vqt_advance : renvoie les vecteurs de déplacement pour le caractère c. Le déplacement indique où dessiner le caractère suivant à partir de la position du caractère c (attributs de texte compris).

ptsout et ptsout+2 : dx et dy en pixels.

ptsout+4 et ptsout+6 : rx et ry. Ce vecteur est le reste (modulo 16384). Pour un rendu plus précis, il suffit d'additionner ces restes caractères par caractères et d'effectuer un décalage d'un pixel supplémentaire dès que 16384 est atteint.

vqt_advance32

c vqt_advance32 : renvoie le vecteur de déplacement pour le caractère c. Par rapport à vqt_advance, le vecteur est au format fix31 (donc pas de reste) sur deux mots longs:

ptsout+8 : dx en fix31
ptsout+12 : dy en fix31.

vst_error

adr m vst_error : si m=1 les erreurs GDOS seront affichées à l'écran sous forme de textes, si m=0 les messages sont remplacés par un code d'erreur sur 2 octets rangé à l'adresse adr. L'application doit alors consulter cette valeur après un appel VDI, un zéro indique que tout va bien, les autres valeurs sont des codes d'erreur, là encore l'application doit remettre cette valeur à zéro après l'avoir lue.

vst_scratch

m vst_scratch : fixe le mode d'allocation de buffers pour SpeedoGDOS selon les valeurs de m:

m=0 : taille suffisante pour effets sur fontes vectorielles et bitmap
m=1 : taille suffisante pour effets sur fontes bitmap uniquement
m=2 : taille réduite, pas d'effets sur les fontes.

vst_kern

tmode pmode vst_kern

vqt_pairkern

c1 c2 vqt_pairkern

vqt_trackkern

renvoie au format fix31 les coordonnées x et y:

ptsout : x
ptsout+4 : y

4 Les fonctions de lignes

v_pline

x1 y1 .... xn yn n v_pline : trace une ligne brisée liant les n points.

vsl_type

t vsl_type : fixe le type de ligne selon les valeurs de t:

t=1	----------	t=5	--------
t=2	----------	t=6	---- -- --
t=3	-- --	t=7	défini par vsl_udsty
t=4	-------- --		

vsl_udsty

m vsl_udsty : définit le motif de ligne lorsque le type 7 est choisi avec vsl_type. A cet effet m est un masque de 16 bits (1=le point est mis, 0=le point n'est pas mis) qui sera indéfiniment répété pour former les lignes.

vsl_width

e vsl_width : fixe l'épaisseur des lignes en pixels. Une valeur paire est arrondie à l'impair inférieur.

ptsout : épaisseur sélectionnée

vsl_color

i vsl_color : fixe l'index de couleur pour les lignes.

intout : index sélectionné.

vsl_ends

d f vsl_ends : fixe les styles pour les débuts et fin de ligne selon les valeurs de d et f:

0 : extrémité rectangulaire
1 : extrémité en flèche
2 : extrémité arrondie.

vql_attributes

renvoie les attributs de ligne actuels:

intout	: type	intout+6	: extrémité de départ
intout+2	: index couleur	intout+8	: extrémité de fin
intout+4	: mode d'écriture	ptsout	: épaisseur

5 Les fonctions de marques

v_pmarker

x1 y1 .... xn yn n v_pmarker : trace une marque en chacun des n points.

vsm_type

t vsm_type : fixe le type de marque à utiliser

t=1	point	t=4	carré
t=2	croix (plus)	t=5	croix (multiplié)
t=3	étoile	t=6	losange

Toute autre valeur sera remplacée par 3.

intout : marque sélectionnée.

vsm_height

h vsm_height : fixe la taille des marques (aucun effet sur le point). Toutes les hauteurs ne sont pas toujours possibles, aussi on obtient en retour:

ptsout : largeur effectivement prise en compte

ptsout+2 : idem pour la hauteur.

vsm_color

i vsm_color : fixe l'index de couleur pour les marques.

intout : index sélectionné

vqm_attributes

renvoie les attributs de marque actuels:

intout	: type	ptsout	: largeur
intout+2	: couleur	ptsout+2	: hauteur
intout+4	: mode d'écriture		

6 Les fonctions de remplissage

v_contourfill

i x y v_contourfill : commence le replissage en x,y avec la couleur de remplissage courante et s'arrête:

si i=-1 : dès qu'un point a une autre couleur que celle en x,y
si i>0 : dès qu'un point a la couleur i.

v_recfl

x1 y1 x2 y2 v_recfl : remplit une surface rectangulaire selon les attributs courants, les coordonnées sont celles de deux points opposés du rectangle.

v_fillarea

x1 y1 .... xn yn n v_fillarea : remplit le polygone délimité par les n points. Le premier et le dernier sont automatiquement reliés.

vsf_interior

t vsf_interior : fixe le type de remplissage à utiliser selon t :

t=0 utiliser la couleur de fond
t=1 utiliser la couleur passée par vsf_color
t=2 utiliser des motifs (voir vsf_style)
t=3 utiliser des hachures (voir vsf_style)
t=4 utiliser le motif défini par vsf_udpat.
intout : type sélectionné

vsf_style

i vsf_style : sélectionne le type de hachures (1 à 12) ou de motifs (1 à 24) selon le type choisi par vsf_interior.

intout : index sélectionné

vsf_color

i vsf_color : choisit la couleur de remplissage.

intout : couleur sélectionnée

vsf_perimeter

f vsf_perimeter : active (f=1) ou désactive (f=0) le tracé du contour des surfaces remplies. Seule v_recfl n'en tient pas compte, il faut donc utiliser v_bar à la place.

vsf_udpat

adr n vsf_udpat : définit un motif de remplissage. Le paramètre n définit le nombre de plans (n=1 en monochrome), adr est l'adresse ou sont rangés successivement les n plans. Chanque plan est une suite de 16 fois 16 bits correspondant à une grille carrée de 16 sur 16 (32 octets par plan). Le premier plan contient les bits de poids fort, le dernier les bits de poids faible.

vqf_attributes

renvoie les attributs de remplissage actuels:

intout	: type (vsf_interior)	intout+6	: mode d'écriture
intout+2	: couleur	intout+8	: flag vsf_perimeter
intout+4	: index du motif (vsf_style)		

7 Les objets de base

v_bar

x1 y1 x2 y2 v_bar : remplit un rectangle.

v_pieslice

x y r a0 a1 v_pieslice : remplit une portion de cercle de centre x,y de rayon r et d'angles de début et de fin a0,a1 en 1/10° de degré.

v_circle

x y r v_circle : remplit un cercle de centre x,y et de rayon r.

v_arc

x y r a0 a1 v_arc : trace le contour d'un arc de cercle de centre x,y de rayon r et limité par les angles a0,a1 en 1/10° de degré.

v_ellpie

x y rx ry a0 a1 v_ellpie : remplit une portion d'ellipse de centre x,y de rayons rx,ry et limitée aux angles a0,a1 en 1/10° de degré.

v_ellipse

x y rx ry v_ellipse : remplit une ellipse de centre x,y et de rayons rx,ry.

v_ellarc

x y rx ry a0 a1 v_ellarc : trace le contour d'une ellipse de centre x,y de rayons rx,ry et limitée par les angles a0,a1.

v_rfbox

x1 y1 x2 y2 v_rfbox : remplit un rectangle aux coins arrondis.

v_rbox

x1 y1 x2 y2 v_rbox : trace le contour d'un rectangle aux coins arrondis.

8 La souris

vsc_form

a vsc_form : définit l'apparence de la souris, a cet effet a est l'adresse d'un bloc de 37 mots ainsi définis:

a	: hot spot x de la souris	a+8	: couleur de masque
a+2	: hot spot y de la souris	a+10	: 16 mots pour le masque
a+4	: nombre de plans (1)	a+42	: 16 mots pour le devant
a+6	: couleur		

v_show_c

f v_show_c : affiche le curseur de la souris selon les valeurs de f

- f=0 afficher
- f=1 tenir compte du nombre d'appels de v_hide_c

v_hide_c

cache le curseur de la souris.

vq_mouse

renvoie l'état de la souris:
intout : mouse button (bit0=gauche, bit1=droite)
intout+2 : mouse x
intout+4 : mouse y

vex_butv

adr vex_butv : installe une routine appelée chaque fois qu'un bouton de souris est appuyé. Cette routine reçoit dans D0 l'état des boutons, elle doit restaurer tous les registres qu'elle utilise et ne faire appel qu'au BIOS ou XBIOS. On reçoit dans control+18 l'adresse de l'ancienne routine.

vex_curv

adr vex_curv : installe une routine appelée chaque fois que le curseur de la souris doit être dessiné. La routine reçoit dans D0 et D1 les coordonnées x,y de la souris. Pour le reste, voir vex_butv.

vex_motv

adr vex_motv : installe une routine appelée chaque fois que la souris a bougé. La routine reçoit dans D0 et D1 les coordonnées x,y de la souris, elle peut les modifier (pour ralentir, accélérer ou limiter les déplacements). Pour le reste, voir vex_butv.

9 Le curseur en mode texte

En mode "relative" toutes ces opérations sont détournées (pas de réel appel VDI) et gèrent le curseur de texte du FORTH dans la page actuelle. En mode "absolute" c'est l'appel VDI qui est réalisé:

vq_chcells

indique le nombre de lignes et de colonnes disponibles en mode texte
intout : lignes
intout+2 : colonnes

v_exit_cur

sort du mode texte et réaffiche la souris.

v_enter_cur

efface la souris et revient en mode texte.

v_curup

une ligne vers le haut. Si on est déjà en haut, rien ne se passe.

v_curdown

une ligne vers le bas. Si on se trouve déjà en bas, rien ne se passe.

v_curright

une position vers la droite, si on se trouve déjà tout à droite, rien ne se passe.

v_curleft

une position vers la gauche. Si on se trouve déjà tout à gauche, rien ne se passe.

v_curhome

ramène le curseur en haut à gauche de la fenêtre sans effacer l'écran.

v_eeos

efface depuis la position du curseur jusqu'au bas de la fenêtre.

v_eeol

efface depuis la position du curseur jusqu'à la fin de la ligne.

vs_curaddress

l c vs_curaddress : fixe la ligne et la colonne du curseur de texte. Les coordonnées comlmencent à 1 et sont ramenées à leur maximum en cas de dépassement.

v_curtext

" chaine" v_curtext : affiche une chaine à la position courante du curseur, affiche éventuellement sur la ligne suivante et scrolle si besoin.

v_rvon

passe en vidéo inverse.

v_rvoff

revient en video normale.

vq_curaddress

renvoie les coordonnées du curseur de texte.

intout : ligne
intout+2 : colonne.

vq_tabstatus

indique la présence d'une tablette graphique ou d'une souris:

intout : 0=absente, 1=présente.

v_hardcopy

lance une copie d'écran.

v_fontinit

adr v_fontinit : permet de remplacer la fonte système, à cet effet adr est l'adresse du header de la nouvelle fonte (au format LineA).

10 Les fonctions de bloc

Pour ces fonctions, la gestion des MFDB (définition des blocs) a été séparée. De ce fait, on définit d'abord les deux blocs (source et destination) puis on appelle la fonction voulue:

mfdbs mfdbd

renvoient l'adresse du mfdb source ou destination, chacun est défini ainsi:

adr	adresse des données	
adr+4	l	largeur en pixels
adr+6	h	hauteur en lignes
adr+8	L en mots = (l+15)/16	
adr+10	f	format 0 = shifter, 1 = VDI
adr+12	np	nombre de plans
adr+14,+16,+18	réservés.	

fillmfdb

remplit un mfdb selon trois syntaxes:

* adr l h np f m fillmfdb : remplit le mfdb m avec
  adr l'adresse du bloc en mémoire
  l h longueur et hauteur totale du bloc, L en mots est calculé pour vous.
  np nombre de plans
  f format

* -1 mfdb fillmfdb : c'est l'écran, tout est rempli pour vous, l'adresse est physbase.

* adr m fillmfdb : copie dans m les données d'un autre mfdb en adr.

Note: pour les fonctions de bloc VDI utilisant l'écran, on ne passera pas par le remplissage -1 mfdb fillmfdb. Cette fonctionnalité sera par contre utile pour la gestion des modules M&E.

Ici, pour l'écran il suffit d'annuler l'adresse du bloc avec par exemple:

```
0 mfdbs !
```

Même si la largeur des zones à traiter n'est pas multiple de 16, il faut que l soit un multiple de 16. Il faut donc prendre le multiple de 16 immédiatement au dessus de la largeur voulue pour que les opérations se passent bien. Par exemple un sprite de 30x30 se verra allouer une zone de 32x30. En 16 couleurs, chaque point demande un demi-octet, donc 32x30/2=480 octets nécessaires, par contre dans les fonctions de copie suivantes on pourra préciser l=30.

imagesize

mfdb imagesize : renvoie la taille en octets d'un buffer pouvant stocker l'image décrite dans le mfdb.

vr_trnfm

transforme le bloc source (avec son format) en un bloc destination (avec un format différent). Normalement les deux blocs ont les mêmes caractéristiques, le format mis à part.

vrt_cpyfm

copie un bloc monochrome vers un bloc monochrome ou couleur:
x y x' y' l h m c c' vrt_cpyfm
- x,y coin supérieur gauche dans la source
- x',y' coin supérieur gauche dans la destination
- l,h taille du bloc à déplacer (peut être inférieure à la taille totale de chaque bloc)
- m mode (1=replace, 2=trans, 3=XOR, 4=inverse trans)
- c,c' indexs de couleur pour le premier plan et le fond.

Pour un masque d'icone ou de souris m=2, c=0 et c'=1.

vro_cpyfm

copie un bloc couleur sur un bloc couleur:
x y x' y' l h m vro_cpyfm
- x,y coin supérieur gauche dans la source
- x',y' coin supérieur gauche dans la destination
- l,h taille du bloc à déplacer
- m mode (0=efface dest, 3=replace, 6=XOR, 7=OR)

Pour des données d'icone (après avoir dessiné le masque) m=7.

Par exemple, pour copier une zone de l'écran vers l'écran:

```
0 mfdbs !
0 mfdbd !
0 0 100 100 16 16 3 vro_cpyfm    \ copie un carré 16x16 de la
                        position 0,0 vers la position 100,100
```

savebitmap

" fichier" mfdb pal savebitmap : sauvegarde le bloc pointé par le mfdb sur disque.
" fichier" : le nom complet du fichier.
mfdb : adresse d'un mfdb de 20 octets (cela peut être mfdbs ou mfdbd).
pal : adresse d'une palette

La palette est ignorée si le nombre de plans est supérieur à 8. En effet, la description des pixels se suffit à elle-même.
Par contre, de 1 à 8 plans, la palette doit contenir les couleurs au format VDI (qu'on peut obtenir avec savevdipal).

loadbitmap

" fichier" mfdb pal loadbitmap : charge le bloc depuis le disque et remplit le mfdb ainsi que l'éventuelle palette.
" fichier" : le nom complet du fichier.
mfdb : adresse d'un mfdb de 20 octets (cela peut être mfdbs ou mfdbd).
pal : adresse d'une palette

La palette est ignorée si le nombre de plans est supérieur à 8. En effet, la description des pixels se suffit à elle-même.
Par contre, de 1 à 8 plans, la palette doit contiendra les couleurs au format VDI (qu'on pourra utiliser avec setvdipal).
Le bloc de données lui-même est alloué dans le dictionnaire avec allot et son adresse est correctement renseignée dans le mfdb.

11 Les courbes de bézier

Ces fonctions ne sont utilisables qu'avec la présence de SpeedoGDOS.

v_bez

f1 f2 ... fn x1 y1 x2 y2 .... xn yn n v_bez : sort une courbe de Bézier s'appuyant sur n points dont les coordonnées sont passées sur la pile. Pour chaque point on doit préciser un flag (f1 à fn):

fi : bit0=0 : polyline (pas de courbures)
: bit0=1 : courbe
: bit1=0 : point traité normalement
: bit1=1 : saut à ce point sans tracer. (Coupure dans la courbe)

v_bez_fill

f1 f2 ... fn x1 y1 x2 y2 ... xn yn n v_bez_fill : même comportement que la fonction précédente, sauf que l'intérieur de la courbe est remplit avec les atrributs de remplissage courants.

v_bez_on

permet l'accès aux courbes de Bézier.

v_bez_off

interdit l'accès aux courbes de Bézier.

v_bez_qual

x v_bez_qual : fixe la qualité des courbes, x va de 0 (qualité faible, rapide) à 100 (qualité maxi, lent). Intout contient la valeur sélectionnée.

v_set_app_buff

adr n v_set_app_buff : force la VDI à utiliser un buffer précis pour la génération des courbes. Le buffer est défini par son adresse adr et le nombre de paragraphes n (16 octets chacun). Avant de quitter, une application doit désallouer ce buffer en appelant : 0 0 v_set_app_buff.

Par défaut, le buffer alloué de manière interne est de 8Ko.

12 Les Metafiles

Pour ouvrir un métafile on utilise le handle 31 avec v_opnwk. Un fichier est automatiquement créé et ouvert avec le nom GEMFILE.GEM dans le répertoire courant.

vm_filename

" fichier" vm_filename : permet d'affecter un nom différent de GEMFILE.GEM au métafile. Il faudra néanmoins effacer (fdelete) le fichier par défaut GEMFILE.GEM qui aura été créé par v_opnwk.

vm_pagesize

l h vm_pagesize : indique les dimensions de la page du metafile en dixièmes de millimètres.

vm_coords

xmin ymin xmax ymax vm_coords : fixe les coordonnées limites à utiliser sur la page du métafile. xmin,ymin sont les coordonnées en haut à gauche et xmax,ymax sont celles en bas à droite. Par défaut, on a:

```
0 32767 32767 0 vm_coords
```

v_meta_extents

xmin ymin xmax ymax v_meta_extents : fixe, dans le système de coordonnées choisi avec vm_coords, le rectangle de clipping dans le metafile.

v_write_meta

éléments_intin éléments_ptsin n_int n_pts v_write_meta : écrit dans un métafile une fonction ayant un sous code créé par l'utilisateur. Le premier des éléments intin doit être ce sous code (ceux de 0 à 100 sont réservés).

13 Autres instructions

v_alpha_text

" chaine" v_alpha_text : envoie la chaine à la station actuellement ouverte. Cette station doit être l'imprimante ou un métafile, il aura donc fallu l'ouvrir explicitement (v_opnwk) afin que la variable handle soit correctement alimentée.

v_bit_image

r x y h v "fichier" X Y X' Y' v_bit_image : charge un fichier image *.IMG du disque et l'affiche dans la station actuelle.

X Y X' Y' sont les coordonnées du rectangle entourant l'image.

" fichier" est le nom du fichier (avec chemin).

r : ingnorer (0) ou tenir compte (1) des proportions de l'image

x y : échelle fractionnaire (0) ou entière (1) sur chaque axe

h v : avec l'échelle entière, l'image sera cadrée du mieux possible avec les alignements suivants (h:horizontal, v:vertical):

h : droite(0), centre (1), gauche (2)

v : haut (0), centre (1), bas (2).

vq_scan

renvoie des informations sur la station imprimante ouverte (handle doit être correct):

intout : grh. (grh/div=nombre de lignes graphiques par passe)

intout+2 : nombre de passes par page imprimée

intout+4 : alh. (alh/div=nbr de lignes par ligne de texte)

intout+6 : nombre de lignes de texte par page

intout+8 : div.

xv_opnwk

variante de v_opnwk dans laquelle on peut préciser les coordonnées maximales de la station:

id l cl m cm f ct sr mr cr c xmax ymax xv_opnwk

Valeurs renvoyées:

handle : handle de la station

intout : xmax pris en compte (.w)

intout+2 : ymax pris en compte (.w)

control : adresse de la station Mémoire (si id=61)

xv_updwk

variante de v_updwk dans laquelle on impose l'adresse du buffer de la station:
adr xv_updwk.
Ces deux dernières instructions conjuguées permettent par exemple d'ouvrir une station imprimante avec les dimensions d'une image chargée en mémoire, puis d'appeler xv_updwk avec l'adresse de cette image, elle sera automatiquement imprimée.

Les instructions suivantes sont implantées dans le nouveaux drivers d'imprimantes couleurs sous SPEEDO GDOS. Après être passé par v_opnwk (21), voici les fonctions dont on dispose:

vq_driver_info

renvoie pad sur la pile avec le nom du driver et dans intout:

intout:	0, nouvelles fonctions non disponibles
intout+2:	n° de version de la librairie
intout+4:	n° de version du driver
intout+6:	1=mono, 3=CYM, 4=CYMK
intout+8:	masque des attributs supportés: 0: quadri 1: négatif 2: mirroir 3: copies multiples HARD 4: copies multiples SOFT 5: paysage

vq_bit_image

renvoie les infos suivantes:

intout:	0, appel inexistant
intout+2:	n° de version de v_bit_image
intout+4:	nombre maximum d'images sur une page
intout+6:	masque de formats disponibles: bits0 et 1: 00=IMG monochrome bits2 et 3: 00=TGA non supporté 01=TGA 2 non compressé Autres valeurs réservées.

vq_image_type

" fichier" adr vq_image_type: renvoie des infos sur le fichier image spécifié:

intout: 0, appel inexistant
intout+2: 0 fichier inconnu, 1=IMG, 2=TGA
adr: nombre de plans de couleurs de l'image
adr+2: largeur
adr+4: hauteur

vq_margin

renvoie les informations suivantes:

intout:	0, appel inexistant		
intout+2:	marge haute	intout+4:	marge basse
intout+6:	marge gauche	intout+8:	marge droite
intout+10:	DPI horizontal	intout+12:	DPI vertical

vs_crop

X Y X' Y' L P vs_crop

positionne des repères pour la découpe lors d'impression sur plusieurs pages:

X Y X' Y': rectangle d'encadrement
L: longueur des traits de découpe
P: position des traits par rapport aux coordonnées

(mettre tout à 0 pour supprimer ces marques)

intout: 0, appel inexistant

vs_page_info

n " chaine" vs_page_info: transmet des informations sur le document selon la valeur de n:

n=0 Nom de l'application
n=1 Titre du document
n=2 Nom du créateur du document
n=3 Remarques.

intout: 0, appel inexistant

Les fonctions AES

1 Présentation

Le principe d'appel des fonctions AES repose sur le remplissage de tableaux d'entrée et de récupération éventuelle de paramètres dans les tableaux de sortie. Certaines fonctions présentées ici sont propres au FORTH et ne servent qu'a faciliter l'utilisation des primitives AES.

Les tableaux d'entrée;

control

laisse sur la pile l'adresse du champ control contenant les informations suivantes:

control	code de la fonction
control+2	nombre de mots dans intin
control+4	nombre de mots dans intout
control+6	nombre d'adresses dans addrin
control+8	nombre d'adresses dans addrout

intin

laisse sur la pile l'adresse du champ intin contenant les paramètres d'entrée sur 2 octets chacun.

addrin

laisse sur la pile l'adresse du champ addrin contenant les adresses d'entrée sur 4 octets chacune.

Les tableaux de sortie:

intout

laisse sur la pile l'adresse du champ intout contenant en retour les informations sur 2 octets renvoyées par la fonction.

addrout

laisse sur la pile l'adresse du champ addrout contenant en retour les adresses sur 4 octets fournies par la fonction.

global

laisse sur la pile l'adresse du champ global rempli par appl_init et rsrc_load.

global	: version de l'AES
global+2	: nombre d'applications simultanées (-1 sous MultiTOS)
global+4	: ID renvoyée par appl_init
global+6	: long mot utilisateur (FORTH l'utilise pour les menus!)
global+10	: pointeur sur le fichier RSC chargé avec rsrc_load.
global+14	: 12 octets réservés
global+26	: _AESmaxchar (version 4.00 ou >)
global+28	: _AESminchar (version 4.00 ou >)

Fonctions manuelles:

En **utilisation normale** les tableaux d'entrée sont remplis pour vous, il ne reste qu'à lire éventuellement les informations dans les champs de sortie. Pourtant, pour certaines fonctions qui ne seraient pas encore implantées un appel "à la main" sera nécessaire:

actrl

fn ii io ad actrl : remplit le champ control avec f=code de la fonction, ii=nombre mots intin, io=nombre de mots intout, ad=nombre de mots addrin. La partie addrout n'est pas prise en compte et initialisée à zéro tant il y a peu de fonctions l'utilisant. Au pire il faudrait ajouter "n control 8 + w!" après cet appel.

aintin!

remplit le champ intin selon le nombre de valeurs indiqué en control+2. Les valeurs sont prises sur la pile.

addrin!

remplit le champ addrin selon le nombre de valeurs indiqué en control+6. Les valeurs sont prises sur la pile.

aes

appelle la fonction aes en supposant que les trois tableaux d'entrée sont correctement remplis.

aes!

suppose que le tableau control est rempli, ensuite remplit le tableau addrin, puis le tableau intin, puis appelle l'AES.

Voici maintenant la liste des fonctions AES, pour la syntaxe on a respecté cet ordre:

valeurs intin valeurs addrin nom de la fonction

Ceci est aussi valable pour la fonction aes!. Attention, ceci n'est pas toujours l'odre de la syntaxe en C, parfois les adresses précèdent les autres données (allez savoir pourquoi...).

2 Les fonctions générales

appl_init

déclare une application au GEM. La fonction est inutile, même sous le compilateur puisque le FORTH s'en charge tout seul. Néanmoins, la valeur retournée par app_init est elle importante (dans la gestion de certains messages ou évènements), donc c'est une simulation de la fonction qui est réalisée et qui renvoie le paramètre id:

intout : id de l'application en cours ou -1 si erreur.

appl_read

id n adr appl_read : lit n octets dans le registre des évènements et les dépose à partir de l'adresse adr. Le paramètre id est votre identificateur. La fonction est inutile pour tous les évènements standards du GEM.

intout 0 si erreur

appl_write

id n adr appl_write : écrit n octets dans le registre des évènements, ces octets sont pris à partir de l'adresse adr. Le paramètre id est votre identificateur.

intout 0 si erreur

appl_find

" nom" appl_find : renvoie l'appl_id de l'application dont le nom est précisé. Le nom doit contenir 8 caractères exactement (comblé avec des blancs et sans l'extension du fichier).

intout appl_id ou -1 si non trouvé.

appl_search

mode nom type ap_id appl_search

appl_trecord

n adr appl_trecord : l'AES enregistre n évènements se trouvant décrits à l'adresse adr. Chaque évènement utilise 6 octets:

mot : code de l'évènement
long mot : paramètres

intout : nombre d'évènements enrégistrés.

appl_tplay

n sc adr appl_tplay : l'AES rejoue n évènements à l'adresse adr. Le paramètre sc indique la vitesse voulue (100=normal, 1000=x10, 50=moitié, etc..)

appl_getinfo

type appl_getinfo : renvoie des informations sur l'AES.

scrp_read

adr scrp_read : renvoie à partir de l'adresse adr le nom du chemin d'accès au clipboard (presse papier GEM). Si intout contient 0, il suffit de créer un répertoire \clipbrd et de le spécifier à l'aide de scrp_write.

scrp_write

adr scrp_write : indique au système le chemin d'accès au clipboard. adr pointe sur une chaîne terminée par 0, par exemple : "C:\CLIPBRD\". Fonction à n'utiliser que si scrp_read renvoie 0 (donc que me clipboard n'existe pas encore).

shel_get

n adr shel_get : copie n octets du buffer AES (le desktop.inf) vers l'adresse adr.

shel_put

n adr shel_put : copie n octets de l'adresse adr vers le buffer AES.

shel_envrn

v " var=" shel_envrn : recherche la valeur d'une variable d'environnement.

v est une variable qui contient en sortie l'adresse du premier octet de la chaîne trouvée, ou contient zéro si la recherche a échoué.

shel_write

mode wisgr wiscr cmd tail shel_write

shel_read

name tail shel_read

shel_find

buffer shel_find

form_alert

b chaîne form_alert : ouvre une boîte d'alerte, b correspond au numéro (1, 2 ou 3) du bouton lié à la touche RETURN (0 si aucun). La chaîne a cette structure:

" [n][ligne1|ligne2|....][bouton1|bouton2|...]", il peut y avoir jusqu'à 5 lignes de 30 caractères et trois boutons de 10 caractères, n est le numéro de l'icône (0 aucune, 1=!, 2=?, 3=stop).

intout : numéro du bouton sélectionné.

form_error

n form_error : affiche une boîte d'alerte correspondant à l'erreur TOS n. Certaines valeurs de n correspondent à un message bien précis, d'autres à une simple boîte "Erreur TOS #n". Pour passer de l'erreur GEMDOS (-32 à -49) au code n correspondant on soustrait 31 à sa valeur absolue. Exemple, le code -36 (accès refusé) correspond à la boîte 36-31=5.

n=2, 3 ou 18 : dossier ou fichier non trouvé.
n=4 : mémoire insuffisante pour ouvrir un autre document.
n=5 : document déjà existant ou protégé en écriture.
n=8, 10 ou 11 : mémoire insuffisante pour lancer l'application.
n=15 : lecteur spécifié inexistant.

fsel_intput

chemin fichier fsel_input : ouvre le sélecteur de fichiers. A cet effet chemin et fichier sont deux chaînes contenant le chemin d'accès et le nom du fichier par défaut proposé à l'utilisateur. Il est dangereur d'utiliser des constantes de chaînes car toute modification du chemin et du fichier modifie la chaîne de départ (et donc éventuellement sa longueur).

intout : 0 si erreur
intout+2 : 0=annuler, autre=confirmer
fichier et chemin modifiés selon les actions de l'utilisateur.

fsel_exinput

chemin fichier titre fsel_exinput : même principe que fsel_input, sauf qu'on ajoute un texte s'affichant comme titre dans le sélecteur indiquant ainsi l'opération réalisée (30 caractères maxi).

Note: *sur un AES < 1.40 l'appel est redirigé sur **fsel_input** et "titre" est ignoré.*

change.ext

" ext" " chemin+fichier" change.ext : renvoie dans **pad** le même chemin et nom de fichier en changeant simplement l'extension par celle spécifiée. Exemple :

" FOR" " C:\FORTH\MYPROG.PRG" change.ext type
affichera: C:\FORTH\MYPROG.FOR

path

chemin fichier path : à utiliser en liaison avec les sélecteurs. Les deux chaînes sont dépilées et fusionnées dans **pad** dont l'adresse est laissée sur la pile. Par exemple:

chemin="C:\FORTH*.FOR" et fichier="TEST.FOR"

On trouvera alors dans pad="C:\FORTH\TEST.FOR". Notons que le masque éventuel dans chemin est oté.

getpath

chemin getpath : renvoie dans **pad** le chemin extrait de la chaîne en ôtant l'éventuel masque ou nom de fichier, pad est déposé sur la pile.

getfile

chemin getfile : renvoie dans **pad** le nom du fichier ou du masque en enlevant le reste du chemin, pad est déposé sur la pile.

3 Les fichiers ressource RSC

Ce sont eux qui contiennent traditionnellement les boites de dialogue, les menus et les messages d'alerte d'une application. Dans les paragraphes consacrés à ces différents éléments on verra comment créer de tels objets sans passer par un fichier indépendant.

rsrc_load

chaîne rsrc_load : charge un fichier ressource et l'adapte à la résolution actuelle, reloge les adresses.

intout : 0 si erreur

rsrc_free

libère la mémoire utilisée par un fichier ressource.

rsrc_gaddr

t i rsrc_gaddr : recheche l'adresse du iéme objet de type t dans le fichier ressource chargé en mémoire.

t=0	arbre	t=8	tedinfo-text
t=1	objet	t=9	tedinfo-masque
t=2	tedinfo	t=10	tedinfo-valid
t=3	ICONBLK	t=11	sous pointeur ICONBLK
t=4	BITBLK	t=12	idem
t=5	chaîne	t=13	idem
t=6	pointeur sur BITIMAGE	t=14	sous pointeur BITBLK
t=7	ob_spec	t=15	idem
		t=16	idem

intout : 0 si erreur

addrout : l'adresse recherchée.

Dans un fichier à un seul arbre, pour obtenir le pointeur indispensable aux fonctions objc_draw et form_do on utilisera : 0 0 rsrc_gaddr

rsrc_saddr

t i adr rsrc_saddr : modifie l'adresse du ième objet de type t, adr est l'adresse que l'on impose.

intout : 0 si erreur.

rsrc_obfix

o adr rsrc_obfix : convertit les coordonnées de l'objet o de l'arbre pointé par adr en pixels à partir de valeurs en caractères selon la fonte système en cours.

rsrc_rcfix

h rsrc_rcfix : fixe toutes les coordonnées et les pointeurs du ressource dont l'adresse du header est h. Si un fichier a déjà été chargé par rsrc_load, on doit utiliser rsrc_free avant cet appel. Cet appel n'existe que pour un AES 4.xx, il a été étendu par le FORTH pour tout AES.

Cas particulier pour un AES < 4.00 : l'adresse h est normalement paire, mais si on transmet h+1 cela transformera toutes les icones couleurs en icones monochromes. Ainsi, un fichier RSC étendu sera affichable avec un AES ancien.

Si intin= 'FORT', alors les routines FORTH sont utilisées quel que soit l'AES et global n'est pas mis à jour (utile pour un arbre temporaire non chargé par rsrc_load).

rsrc_mono

flag trame adr rsrc_mono : adapte un arbre/rsc pour un écran monochrome.

flag : 0 si adr est l'adresse de l'entête du RSC (se trouve en global+14 si il est chargé par rsrc_load)
: 1 si adr est l'adresse d'un seul arbre (obtenue avec rsrc_gaddr par exemple)

trame : de 0 à 7 force tous les objets ayant un fond coloré vers la trame indiquée en noir sur blanc
: -1 calcule une trame adaptée pour rendre compte de la luminosité de l'objet avec sa couleur initiale. De même pour les icones, la couleur la plus sombre (entre premier plan et fond) sera le noir et la plus claire le blanc.

form_center

adr form_center : calcule les coordonnées d'un arbre pour qu'il apparaisse centré sur l'écran.

intout+2 à +8 : X,Y,L,H

form_dial

f x y l h X Y L H form_dial : prépare l'écran au dessin d'un arbre d'objets, sans cet appel, les messages redraws ne seront pas correctement gérés par l'AES.

f=0 : réservation de l'écran (X,Y,L,H), (x,y,l,h sont indéfinis, 0 par exemple)
f=1 : dessin d'un rectangle qui grandit de (x,y,l,h) vers (X,Y,L,H)
f=2 : dessin d'un rectangle qui rétrécit de (X,Y,L,H) vers (x,y,l,h)
f=3 : libération de l'écran (X,Y,L,H), (x,y,l,h aussi indéfinis)

Les étapes 1 et 2 sont optionnelles.

form_do

i adr form_do : passe le contrôle à l'AES pour la gestion du formulaire pointé par adr, i est l'index du champ éditable recevant le curseur (0 si il n'y en a pas).

intout : index de l'objet ayant mis fin au dialogue, le bit 15 est mis en cas de double clic. Pour avoir l'index quoi qu'il arrive :

```
intout %h7fff and
```

form_button

o c adr form_button

form_keybd

o n kc adr form_keybd

gem_input

" V|Prompt : ____ " chaine gem_input

Fabrique un arbre temporaire d'objets avec le prompt, un bouton Ok et permet l'entrée d'une donnée qui sera renvoyée dans la chaîne spécifiée.

Si au départ la chaîne n'est pas vide, son contenu apparaît dans le champ éditable.

Le caractère V est optionnel et signale le type de donnée attendu :

X : tout caractère

9 : uniquement des chiffres

Par défaut, c'est X qui est utilisé.

Exemple :

```
3 string AGE
" 9|Âge : ___ "  AGE  gem_input
AGE type
```

4 Les menus

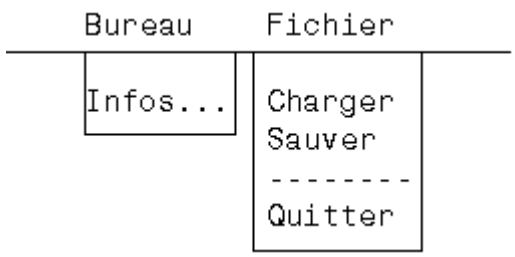
Soit le menu est chargé dans un fichier RSC, soit il est créé en FORTH. C'est seulement dans ce second cas que la première partie de ce paragraphe est à utiliser.

menu

adr menu : crée un menu. A cet effet, adr doit être l'adresse de la première chaîne d'un tableau de chaînes (type array$) organisé comme suit:

- d'abord tous les titres de la barre de menu
- une chaîne vide
- l'option du titre 1 (en général Info) + une chaîne vide
- les options du titre 2 + une chaine vide

Ainsi de suite pour chaque titre. Par exemple, pour créer le menu suivant :



Il faudra 10 chaînes de 12 octets on declarera:

```
12 10 array$ MENU
```

Ensuite il faudra remplir le tableau de chaines dans l'ordre. La partie concernant les accessoires est gérée par le FORTH.

```
"   Bureau "      0 MENU $!
"   Fichier "     1 MENU $!
" "               2 MENU $!
"   Infos... "    3 MENU $!
" "               4 MENU $!
"   Charger "     5 MENU $!
"   Sauver "      6 MENU $!
" -----------"    7 MENU $!
"   Quitter "     8 MENU $!
" "               9 MENU $!
```

```
Toujours prévoir un blanc à droite
et à gauche pour les check_marks
et l'esthétique.
```

Puis pour finir :

```
0 MENU  menu
```

renvoie sur la pile l'adresse de l'arbre (comme celle obtenue par rsrc_gaddr) qu'il faut conserver pour tous les appels ultérieurs. Par exemple :

```
0 MENU menu constant ARBRE
```

gemindex

n gemindex : fournit l'index GEM correspondant à la chaîne n du menu. Dans l'exemple précedent, la chaîne n°6 (Sauver) n'est pas forcément le 6° objet de l'arbre en mémoire. Or c'est cet index interne qu'il nous faut connaître pour utiliser correctement les fonctions AES. Par exmple, pour désactiver l'option 7 (un trait de séparation):

```
7 gemindex   0   ARBRE   menu_ienable
```

strindex

n strindex : fournit le numéro de chaîne d'une option dont on connait l'index interne.

Par exemple, evnt_mesag renvoie cet index lorsqu'une entrée du menu a été selectionnée:

```
1 ARBRE menu_bar        \ dessine le menu
begin
            here evnt_mesag
            here w@ 10 =      \ attend l'évènement "menu selected"
until
here 8 + w@             \ index interne de l'option choisie
strindex                \ son numéro de chaîne
case
            3 of ...          \ il ne peut être que 3, 5, 6 ou 8
            5 of ...          \ les autres chaînes n'étant pas des
            6 of ...          \ options.
            8 of ...
endcase
```

setmenu

adr setmenu : adr est l'adresse de l'arbre d'un menu (celle renvoyée par menu), le menu ainsi pointé devient le menu courant (celui utilisé pour gemindex, strindex et autres opérations de menu). Cette fonction n'est utile que lors de l'utilisation de plusieurs menus.

Les instructions qui suivent sont les appels AES proprement dits, ils sont donc applicables à tout menu (chargé avec rsrc_load ou créé en FORTH).

menu_bar

f adr menu_bar : dessine (f=1) ou annule (f=0) la barre de menu dont l'arbre est pointé par adr.

intout : 0 si erreur

menu_icheck

i f adr menu_icheck : met une marque (f=1) ou l'efface (f=0) devant l'entrée d'index GEM i du menu pointé par adr.

intout : 0 si erreur

menu_ienable

i f adr menu_ienable : active (f=1) ou désactive (f=0) l'entrée d'index GEM i du menu pointé par adr. Les options désactivées apparaissent en clair.

intout : 0 si erreur

menu_tnormal

i f adr menu_tnormal : passe en vidéo inverse (f=0) ou en vidéo normale (f=1) le titre d'index GEM i du menu pointé par adr. Dès qu'une entrée du menu est choisie, le titre reste en vidéo inverse tant que l'utilisateur ne l'a pas resaturé.

intout : 0 si erreur

menu_register

id chaîne menu_register : intègre un accessoire ou une application (sous MultiTOS) à la barre de menu. Le paramètre id est celui renvoyé par appl_init, la chaîne contient le texte qui apparaîtra dans le premier menu déroulant.

intout : id de l'accessoire à conserver (pour event_mesag) ou -1 si erreur.

Voir également >***accname*** *!*

menu_text

i arbre chaîne menu_text : modifie le nom de l'otion d'index GEM i dans l'arbre de menu. La chaîne contient le nouveau nom. Afin de ne pas déborder du cadre, la longueur de la chaîne ne devrait pas exéder celle de l'ancienne.

intout : 0 si erreur

menu_attach

flag item adr mdata menu_attach

menu_istart

flag imenu item adr menu_istart

menu_popup

xpos ypos menu mdata menu_popup

menu_settings

flag set menu_settings

5 Les évènements

evnt_keybd

attend qu'une touche soit pressée et renvoie:
intout : code clavier/code ASCII (octet fort/faible)

evnt_button

c m s evnt_button : attend un évènement sur les buttons de la souris.
c : nombre de clics maximum à attendre
m : bouton à surveiller (1=droit,2=gauche,3=les deux)
s : retour si bouton appuyé (s=1), relaché (s=2),
peu importe (s=3)
On obtient en retour :
intout : nombre de clics observés
intout+2 : mouse x lors de l'évènement
intout+4 : mouse y
intout+6 : copie de s
intout+8 : état des touches spéciales
(bit0=sh_d,bit1=sh_g,bit2=ctrl,bit3=Alt).

evnt_mouse

f x y l h evnt_mouse : attend l'entrée (f=0) ou la sortie (f=1) du curseur de la souris d'un rectangle défini par x,y,l,h. On obtient en retour:

intout+2	: mouse x	intout+6	: mouse k
intout+4	: mouse y	intout+8	: touches spéciales

evnt_timer

t evnt_timer : attend que le temps t (en millisecondes sur 4 octets) s'écoule.

evnt_dclick

v f evnt_dclick : lit (f=0) ou fixe (f=1) la vitesse du double click. Si f=1, v doit être un entier de 0 à 4, si f=0, v est sans signification.
intout : vitesse prise en compte.

evnt_mesag

adr evnt_mesag : attend qu'un évènement se produise et remplit le buffer de 16 octets pointé par adr. On obtient en retour:

adr : numéro de l'évènement
adr+2 : id du programme qui l'a déclanché
adr+4 : 0 si buffer suffisant, ou nombre d'octets supplémentaires (appl_read)

Le numéro peut être l'un des suivants :

10: MN_SELECTED : adr+6=index du titre, adr+8=index de l'option

20: WM_REDRAW : adr+6=handle de la fenêtre à redessiner, adr+8 à adr+14=x,y,l,h

21: WM_TOPPED : adr+6=handle de la fenêtre à activer.

22: WM_CLOSED : adr+6=handle de la fenêtre à fermer.

23: WM_FULLED : adr+6=handle de la fenêtre que l'on veut en plein écran.

24: WM_ARROWED : adr+6=handle de la fenêtre dont on a actionné les flèches adr+8: 0=page_up, 1=page_down, 2=up, 3=down, 4=page_left, 5=page_right, 6=left, 7=right

25: WM_HSLID : adr+6=handle, adr+8=position du curseur horizontal (0 à 1000)

26: WM_VSLID : adr+6=handle, adr+8=position du curseur vertical (0 à 1000)

27: WM_SIZED : adr+6=handle, adr+8 à adr+14=x,y,l,h les nouvelles dimensions

28: WM_MOVED : adr+6=handle, adr+8 à adr+14=x,y,l,h la nouvelle position

40: AC_OPEN : adr+8=id de l'accessoire à activer (celui renvoyé par menu_register)

41: AC_CLOSE : adr+6=id de l'accessoire à refermer.

evnt_multi

choix c m s f1 x1 y1 l1 h1 f2 x2 y2 l2 h2 t adr evnt_multi : attend une combianison d'évènements selon le masque de bits dans choix:

choix bit 0 : evnt_keybd
bit 1 : evnt_button (c m s)
bit 2 : evnt_mouse (f1 x1 y1 l1 h1)
bit 3 : evnt_mouse (f2 x2 y2 l2 h2)
bit 4 : evnt_mesag (buffer en adr)
bit 5 : evnt_timer (temps t)

Les paramètres inutiles doivent quand même apparaitre (des 0 par exemple). renvoie les valeurs suivantes:

intout : évènement observé (codé comme "choix")

intout+2	: mouse x	intout+8	: état des touches spéciales
intout+4	: mouse y	intout+10	: code touche actionnée
intout+6	: mouse k	intout+12	: nombre de clicks observés

6 Les fenêtres GEM et les pages FORTH

L'écriture dans une fenêtre GEM permet une programmation propre car la partie correspondante de l'écran est réservée, les différents mouvements de fenêtres, ouvertures ou fermetures sont automatiquement accompagnés de messages GEM permettant de redessiner les parties éventuellement redécouvertes. Cependant, à l'intérieur d'une fenêtre, rien ne nous empêche d'y créer plusieurs pages FORTH (pour faire du multicolonnage par exemple, ou pour séparer dessin et texte). Cependant, si les pages ne sont pas logées à l'intérieur d'une fenêtre GEM il se peut que des parties d'écran soient ainsi salies sant qu'aucune des applications ne soit avertie.

Au lancement du FORTH, la fenêtre principale est déjà ouverte, avec un programme compilé c'est fastopen qui s'en chargera. La page FORTH correspondante, page0, sera automatiquement calquée sur la fenêtre. Elle permet les sorties de texte, de graphiques, les saisies au clavier. Elle sera amplement suffisante pour bon nombre d'applications.

Il est à noter que le FORTH ouvre une deuxième station VDI pour les sorties de texte standard ceci afin que les réglages de l'utilisateur ne viennent pas perturber le bon fonctionnement de l'éditeur par exemple. Dans cette station VDI, le clipping est automatiquement réalisé sur la page0 ainsi les sorties de textes même bugués (ça peut arriver) ne sortiront pas de la fenêtre.

&page

&page XXX : crée un nouveau mot XXX correspondant à une page FORTH. Il n'a pour l'instant aucune définition de taille ni de position. Lorsqu'il est éxécuté, XXX laisse sur la pile une adresse contenant:

adr:	WORDS x,y,l,h de la fenêtre
adr+8:	WORDS x,y curseur et l,h d'un caractère
adr+16:	WORDS nombre de colonnes et lignes (voir setsubpage)
adr+20:	FLOATS x0, largeur, y0, longueur, z0, hauteur (utilisés par gr_xxx)
adr+68:	

defpage

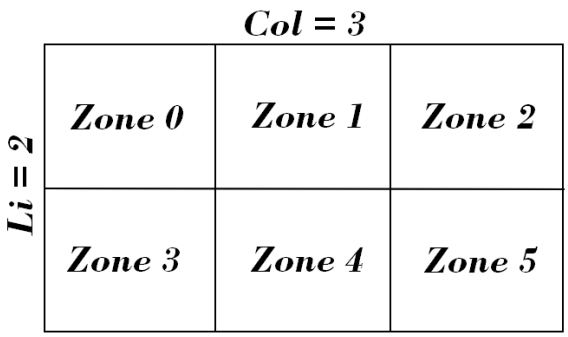
x y l h XXX defpage : donne des dimensions à la page XXX. Initialise la position du curseur de texte dans la page (0,0 en haut à gauche). Mais, la page n'est pas encore utilisable, il faut la déclarer comme page active.

setpage

XXX setpage : fixe XXX comme page active. Toutes les sorties de texte se feront dans cette page, si le mode relative est enclanché, toutes les sorties VDI seront relatives au coin supérieur gauche de cette page.

setsubpage

col li XXX setsubpage : fixe le nombre de colonnes et lignes pour le découpage de la page XXX.



getsubpage

i XXX getsubpage : renvoie les coordonnées X, Y, L, H de la zone i de la page XXX. Ci-contre un exemple avec:

```
&page ZONE4                          \ crée une page
3 2 page0 setsubpage                 \ découpe la page de base en 3×2
4 page0 getsubpage ZONE4 defpage     \ récupère coordonnées pour définir ZONE4
ZONE4 setpage                        \ et fixe cette page
```

Si i excède le nombre de zones disponible, c'est la zone 0 qui est renvoyée.

setmargin

m setmargin : fixe l'épaisseur des marges à l'intérieur des zones de découpage. Par défaut, c'est zéro.

Sinon, par rapport à la taille standard X, Y, L, H, on obtient:

X+m, Y+m, L-2m, H-2m en appelant **getsubpage**.

pageclip

réalise un clipping sur la page active afin de limiter les sorties VDI.

page0

page de base du FORTH se comportant comme créée par &page. C'est celle ouverte sous l'interpréteur ou avec fastopen dans un programme compilé. Ce mot laisse donc l'adresse d'une zone de 68 octets décrite sous **&page**.

page

page courante, même comportement que **page0**.

fastopen

dans un programme compilé ouvre une fenêtre plein écran avec les mêmes attributs que sous le FORTH et renvoie ensuite son handle sur la pile. On peut le conserver si on désire agir sur cette fenetre (changer le titre par exemple...). Si la fenêtre est déjà ouverte (comme sous l'interpréteur) fastopen se contente de renvoyer le handle.

fastclose

referme la fenêtre ouverte avec fastopen. Même dans un programme cette opération n'est pas nécessaire puisqu'elle est automatiquement réalisée avant la sortie du programme.

fasthdl

renvoie la handle de la fenêtre FORTH, même si celle-ci est fermée. Elle n'est pas détruite et l'on peut agir dessus.

relative

mode dans lequel les coordonnées des fonctions VDI sont relatives au coin supérieur gauche de la page active et ou les fonctions textes du VDI sont simulées dans cette page.

absolute

mode dans lequel les coordonnées des fonctions VDI sont absolues (relatives à l'écran total) et dans lequel les fonctions texte du VDI sont exécutées de manière normale.

tovdi

x y l h tovdi : transforme le rectangle AES en rectangle VDI x,y,x',y' en tenant compte du mode **relative** ou **absolute** (toujours absolu pour l'AES).

toaes

x y x' y' toaes : transforme le rectangle VDI en rectangle AES x,y,l,h en tenant compte du mode **relative** ou **absolute** (toujours absolu pour l'AES).

cls

efface la page courante et place le curseur de texte en haut à gauche.

wind_delete

hdl wind_delete : détruit la fenêtre hdl. Il faut d'abord être passé par wind_close.

intout : 0 si erreur

wind_new

efface toutes les fenetres de l'application et initialise les compteurs de v_hide_c ainsi que de wind_update.

wind_create

def x y l h wind_create : crée une fenêtre GEM (elle n'est pas encore affichée), xylh sont les dimensions maximales qu'elle pourra prendre, def est un masque de bits indiquant les éléménets de la fenêtre:

bit 0	: nom	bit 6	: flèche vers le haut
bit 1	: closer (ferme)	bit 7	: flèche vers le bas
bit 2	: fuller (maxi)	bit 8	: ascenseur vertical
bit 3	: mover (déplace)	bit 9	: flèche vers la gauche
bit 4	: info	bit 10	: flèche vers la droite
bit 5	: sizer (taille)	bit 11	: ascenseur horizontal

Il faut absolument conserver la valeur renvoyée, c'est le handle de la fenêtre, c'est à dire un identificateur qui servira dans toutes les fonctions suivantes:

intout : handle ou -1 si la fenêtre n'a pas pu être créée.

wind_delete

hdl wind_delete : détruit la fenêtre hdl. Il faut d'abord être passé par wind_close.

intout : 0 si erreur

wind_get

hdl f wind_get : renvoie des informations sur la fenêtre hdl selon la valeur de f:

- f=4 : dimensions de l'espace de travail
- f=5 : dimensions de l'espace total
- f=6 : dimensions de l'espace total précédant (avant fuller)
- f=7 : dimensions de l'espace total maximal
- f=8 : position du curseur horizontal (1 à 1000)
- f=9 : position du curseur vertical (1 à 1000)
- f=10 : renvoie le handle de la fenêtre active (hdl n'a pas de sens alors)
- f=11 : dimensions du premier catalogue REDRAW
- f=12 : le suivant
- f=15 : taille du curseur horizontal (1 à 1000)
- f=16 : taille du curseur vertical (1 à 1000)

Valeurs retournées:

intout	: 0 si erreur
intout+2	: x ou position (f=8,9) ou handle (f=10) ou taille (f=15,16)
intout+4	: y
intout+6	: l
intout+8	: h

wind_set

modifie une fenêtre, il y a plusieurs syntaxes possibles:

```
hdl  def  1  wind_set      :  modifie les éléments présents de la fenêtre hdl
                              (voir wind_create)
hdl  chaîne  2  wind_set   :  assigne une chaîne de titre à la fenêtre hdl
hdl  chaîne  3  wind_set   :  assigne une chaîne d'info à la fenêtre hdl
hdl  pos  8  wind_set      :  position curseur horizontal (pos=1 à 1000)
hdl  pos  9  wind_set      :  position du curseur vertical (pos=1 à 1000)
hdl  10  wind_set          :  active la fenêtre hdl
adr  obj  14  wind_set     :  installe un arbre d'objets (à partir de l'objet
                              obj) pointé par adr à la place du fond du bureau.
hdl  t  15 wind_set        :  fixe la taille du curseur horizontal (t=1 à 1000)
hdl  t  16 wind_set        :  fixe la taille du curseur vertical (t=1 à 1000)
```

Pour toute autre valeur du flag (autre que 1,2,3,8,9,10,14,15,16), il faut reprendre la syntaxe générale de wind_set:

```
hdl param1 param2 param3 param4   flag   wind_set
```

Exemple pour une nouvelle position de la fenêtre:

```
hdl   x   y   l   h   5   wind_set
```

wind_find

```
x  y  wind_find  :  renvoie le handle de la fenêtre sous les coordonnées x,y.
      intout        : handle (0 si c'est le fond du bureau)
```

wind_update

```
f  wind_update  :  gère le contrôle de l'écran et de la souris
      f=0    : l'écran est libre
      f=1    : vous réservez l'accès à l'écran pour entreprendre des
                    modifications
      f=2    : l'AES reprend le contrôle de la souris (menus,
                    formulaire)
      f=3    : vous prenez le contôle de la souris.
```

Lors de modifications il faut d'abord appeler wind_update avec f=1 avant d'effacer la souris.

wind_new

efface toutes les fenetres de l'application et initialise les compteurs de v_hide_c ainsi que de wind_update.

wind_calc

f def x y l h wind_calc : fournit l'espace total (f=0) à par tir de l'espace de travail ou l'espace de travail (f=1) à partir de l'espace total en tenant compte des éléments de la fenêtre dans le masque de bits def (voir wind_create).

intout : 0 si erreur
intout+2 à intout+8 : x, y, l, h.

En ajoutant 2 à f, on obtient en retour x,y centrés sur le bureau. Ceci est une fonctionnalité du FORTH non pas de l'AES!

get_xywh

adr get_xywh : ramène sur la pile les 4 valeurs de mots se trouvant à l'adresse adr.

get_xyxy

adr get_xyxy : ramène sur la pile les 4 valeurs de mots se trouvant à l'adresse adr puis transforme w et h en x' et y' pour le dialogue avec le VDI:

x'=x+w-1 y'=y+h-1

En effet, les rectangles VDI sont définis par deux points opposés et non par la largeur et hauteur à partir du coin supérieur gauche.

newin

p newin : redéfinit la taille de la fenêtre FORTH à p% de la surface du bureau. Attention, c'est une fermeture brutale (fenêtre détruite, recréée et rouverte, page0 est remis comme page courante et le handle de la fenêtre pourrait être modifié). Aucun test n'est réalisé sur la taille, vérifiez que votre paramètre p n'est ni trop petit ni supérieur à 100.

set_redraw

hdl x y l h set_redraw : prépare la boucle de redraw pour la fenêtre hdl. Les dimensions du rectangle sont données par evnt_mesag ou evnt_multi.

redraw

à l'aide des paramètres spécifiés dans set_redraw, renvoie un à un les rectangles à redessiner. Lors du premier appel, on réserve l'écran (1 wind_update) et on efface la souris (v_hide_c). Lorsqu'il n'y a plus de rectangles l'écran est libéré (0 wind_update) et la souris réaffichée (1 v_show_c).

renvoie 1 et x y x' y' si un rectangle est à redessiner
renvoie 0 si c'est terminé.

Petit exemple:

```
here evnt_mesag
here w@ 20 =                    \   c'est le message REDRAW
if
            here 6 + w@         \ le handle
            here 8 + get_xywh
            set_redraw          \ prépare la boucle
            begin
                  redraw
            while
                  ....          \ ici x y x' y' sont disponibles pour un
                                dessin
                  v_recfl       \ remplit un rectangle par exemple!
            repeat
then
```

7 La librairie Graf

graf_handle

renvoie les informations suivantes:

intout	: handle pour ouvrir une station VDI		
intout+2	: largeur caractère	intout+6	: largeur cellule
intout+4	: hauteur caractère	intout+8	: hauteur cellule

graf_mkstate

renvoie les informations suivantes:

intout+2	: mouse x	intout+6	: mousek
intout+4	: mouse y	intout+8	: touches spéciales

graf_mouse

f graf_mouse : fixe la forme de la souris:

f=0	flèche	f=4	main à plat
f=1	curseur	f=5	réticule fin
f=2	abeille	f=6	réticule épais
f=3	main qui pointe	f=7	contour de réticule

en plus: f=256 désactiver la souris
f=257 activer la souris
f>65535 considéré comme une adresse pointant sur un bloc de définition de la souris, la définition est la même que pour la fonction vsc_form du VDI.

graf_dragbox

l h x y X Y L H graf_dragbox : à l'intérieur d'un rectangle (XYLH), permet le déplacement d'un petit rectangle de dimensions (l,h) à partir des coordonnées (x,y). Le bouton de la souris doit être appuyé avant l'appel, la fonction s'arrête lorsqu'il est relâché.

intout	: 0 si erreur
intout+2	: x final du petit rectangle
intout+4	: y final

graf_growbox

x y l h X Y L H graf_growbox : dessine un rectangle qui s'agrandit.

graf_movebox

l h x y x' y' graf_movebox : déplace un rectangle (l,h) de x,y à x',y'.

graf_rubberbox

x y l h graf_rubberbox : dessine un rectangle dont un coin (x,y) est fixe et dont l'angle opposé suit le mouvement de la souris. (l,h) sont les dimensions minimales du rectangle. Il faut que le bouton de la souris soit appuyé avant l'appel, la fonction s'arrête lorsqu'il est relâché.

intout : 0 si erreur
intout+2 : largeur finale intout+4 : hauteur finale

graf_shrinkbox

X Y L H x y l h graf_shrinkbox : dessine un rectangle qui va en rétrécissant.

graf_slidebox

p f dir adr graf_slidebox : dans un arbre d'objets pointé par adr, permet le déplacement horizontal (dir=0) ou vertical (dir=1) d'un objet fils d'index f à l'intérieur d'un objet père d'index p. Il faut que le bouton de la souris soit appuyé avant l'appel, la fonction s'arrête lorsqu'il est relâché.

intout : position du fils de 0 (gauche ou haut) à 1000 (droite ou bas).

graf_watchbox

0 i s_in s_out adr graf_watchbox : surveille le passage de la souris dans l'objet d'index i de l'arbre pointé par adr. Si la souris est dedans son ob_state=s_in, si elle est dehors alors ob_state=s_out. La fonction s'arrête dès que le bouton est relâché.

Le premier paramètre 0 doit apparaître.

intout : souris en dehors (0) à la fin ou en dedans (1).

8 Les arbres d'objets

Ce paragraphe traite la création d'un arbre d'objets (formulaire) directement en FORTH, c'est à dire sans passer par un fichier RSC séparé. Cette manipulation demande quelques petits calculs et surement plusieurs essais pour les réglages. Le premier travail est la réservation des zones dans lesquelles seront rangées les définitions des objets.

dialogue

ob_zone ted_zone aut_zone dialogue : entame la création d'un arbre.
ob_zone : adresse du bloc objet, 24 octets par objet.
ted_zone : adresse du bloc TEDINFOS, 28 octets pour chaque objet de type TEXT, BOXTEXT, FTEXT ou FBOXTEXT.
aut_zone : autres structures, 14 octets pour chaque IMAGE, 34 par ICON et 36 par USERDEF.
Il est évident qu'au début ces zones sont vides.

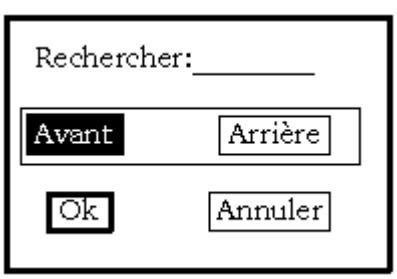
child<< >>

n child<< ... >> : déclare que tous les objets créés dans cette structure sont les enfants de l'objet n. La structure ne peut pas être imbriquée (ça ne veut pas dire qu'on ne pourra pas avoir d'enfants à l'intérieur d'enfants).

lastobj

marque le dernier objet comme étant le fin de l'arbre. Indispensable avec des FTEXT pour éviter toute erreur.

Illustrons ceci par un exemple, soit à créer le formulaire suivant :



L'objet 0 est un BOX contenant un FTEXT, deux BUTTONs et une IBOX qui contient deux radioBUTTONs.

```
24 7 * allot constant OB_ZONE              \ 7 objets en tout
28 allot constant TED_ZONE                 \ une seule TEDINFO
                                           \ (celle du FTEXT)

OB_ZONE    TED_ZONE   0 dialogue           \ le dernier
                                           \ paramètre est fictif

"création de la boite"                     \ objet 0


0   child<<                                \ les enfants du 0
            "création du FTEXT"                 \ objet 1
            "création de la IBOX"               \ objet 2
            "création du 1° bouton (Ok)"        \ objet 3
            "création du 2° bouton (Annuler)"   \ objet 4
      >>


2 child<<                                        \ les enfants de IBOX
            "création du radiobouton (Avant)"    \ objet 5
            "création du radiobouton (Arrière)" \ objet 6
      >>
lastobj                                          \ fini!
```

Chaque objet créé demande 24 octets ainsi utilisés (exemple pour l'objet d'index i):

```
      ob_zone+24*i      : pointeur sur le suivant
                  +2    : pointeur sur le premier enfant
                  +4    : pointeur sur le dernier enfant
                  +6    : type d'objet
                  +8    : flag
                  +10   : state
                  +12   : spec (mot long, 4 octets)
                  +16   : X
                  +18   : Y
                  +20   : L
                  +22   : H
```

flag est ainsi défini :
bit 0 : selectable
bit 1 : default
bit 2 : exit
bit 3 : editable (FTEXT)
bit 4 : radio button
bit 5 : last object
bit 6 : touch exit
bit 7 : hide tree
bit 8 : indirect

state est ainsi défini:
bit 0 : selected
bit 1 : crossed (type BOX)
bit 2 : checked
bit 3 : disabled
bit 4 : outlined
bit 5 : shadowed

Les coordonnées sont données en caractères sur deux octets:
octet faible : nombre de caractères
octet fort : nombre signé de points à ajouter.

```
Exemple :   5           -> 5 caractères
```

```
%h0205      -> 5 caractères plus 2 points.
%hff05      -> 5 caractères moins un point.
```

Pour la création des objets, voici les mots utiles:

BOX

IBOX

BOXCHAR

flag state spec x y l h BOX : même syntaxe pour les trois. spec est un masque de bits ainsi défini:

bits 31-24 : code ASCII du caractère (pour BOXCHAR)
bits 23-16 : largeur du bord (+ intérieur, - extérieur)
bits 15-12 : couleur de bord
bits 11-8 : couleur caractère
bit 7 : 0 transparent, 1 opaque
bits 6-4 : style de remplissage
bits 3-0 : couleur de remplissage.

BUTTON

STRING

TITLE

flag state chaîne x y l h BUTTON : même syntaxe pour les trois.

TEXT

BOXTEXT

flag state x y l h chaîne fonte just couleur TEXT : même syntaxe pour les deux.

La fonte est petite (5) ou normale (3), la justification centrée (2), à gauche (0) ou à droite (1). couleur correspond à spec d'un BOX (bits 0 à 23).

FTEXT

FBOXTEXT

flag state x y l h chaine1 chaine2 chaine3 just couleur FTEXT : même syntaxe pour les deux. just et couleur sont identiques à TEXT. Les trois chaînes sont le texte, le masque de saisie et le masque de validation.

IMAGE

flag state x y l h adr couleur IMAGE : pour cet objet, l est la largeur en octets (pair) et h la hauteur en points. La couleur est de 1 à 16. Adr pointe sur les données, chaque mot constitue 16 points consécutifs.

ICON

flag state x y x' y' coul_car chaîne adr ICON : crée une icone de dimensions 32x32.

x' y'	: coordonnées du caractère dans l'image 32x32.
coul_car	: couleur du caractère : bits15-12=couleur 1°plan, bits11-8=couleur fond, bits7-0=code ASCII du caractère.
chaîne	: affichée sous l'icone en fonte 6x6 sur 13 caractères maximum.
adr	: adresse des données, 128 octets pour 32*32 points du masque, et 128 octets pour 32x32 points de l'icone.

USERDEF

flag state x y l h code param USERDEF : crée un objet dont l'aspect et le comportement est géré par une routine de l'utilisateur. La routine doit être un mot compilé ou assemblé, il reçoit sur une pile provisoire (n'oublions pas qu'on se trouve dans l'AES) l'adresse d'un bloc ainsi défini:

adr	: adresse de l'arbre
adr+4	: index de l'objet
adr+6	: state précédent
adr+8	: nouveau state
adr+10	: X, Y, L, H absolus de l'objet
adr+18	: x, y, x', y' du rectangle de clipping
adr+26	: param (sur 4 octets)

Si les deux valeurs de state sont égales, c'est qu'il faut dessiner l'objet dans cet état (un appel de ob_draw). Si elles sont différentes, il faut s'occuper de passer l'objet d'un état à l'autre (par exemple, l'objet a été sélectionné). La valeur de param est laissée libre à l'utilisateur, on peut par exemple concevoir une même routine pour plusieurs objets, le param permettant de les distinguer.

La routine doit renvoyer un state dont les bits restants sont ceux dont le traitement sera laissé à l'AES, on renvoie 0 si on s'occupe de tout. Les coordonnées sont en mode absolu par rapport à l'écran physique. Si une routine VDI doit être utilisée, il faut spécifier "absolute" avant.

objc_change

index 0 x y l h state f adr objc_change : modifie le state de l'objet d'index i dans l'arbre pointé par adr, x y l h sont les dimensions du cadre de clipping si l'objet doit être redessiné vaec son nouvel état (f=1), f=0 sinon et x y l h sont indéfinis.

intout : 0 si erreur

objc_draw

i prof X Y L H adr objc_draw : dessine un arbre d'objets pointé par adr, les coordonnées sont celles renvoyées par form_center, i est l'index du premier objet à dessiner (0 le plus souvent) et prof est le nombre de générations à dessiner en profondeur (avec 2, l'AES ira jusqu'aux enfants des enfants de l'objet de départ), on choisira une valeur dépassant 7 pour avoir tout l'arbre.

intout : 0 si erreur

objc_find

i prof x y adr objc_find : renvoie l'index de l'objet se trouvant sous les coordonnées x,y. i et prof définissent la fourchette de recherche (voir objc_draw) et adr est l'adresse de l'arbre.

intout : index de l'objet cherché ou -1 si aucun.

objc_ edit

obj kc idx mode tree objc_edit

objc_ offset

i adr objc_offset : calcule les coordonnées de l'objet i de l'arbre pointé par adr par rapport à l'origine de l'écran.

intout : 0 si erreur
intout+2 : x
intout+4 : y

objc_add

p e adr objc_add : lie l'enfant d'index e au parent d'index p dans l'arbre d'adresse adr.

intout : 0 si erreur

objc_delete

i adr objc_delete : défait les liens de l'objet d'index i dans l'arbre pointé par adr. Les objets ne sont pas effacés.

intout : 0 si erreur

objc_order

i p adr objc_order : donne à l'enfant d'index i la position p dans l'ordre des enfants du même parent, l'arbre est pointé par adr.

p=-1 : 0 de vient le dernier enfant
p=0 : 0 devient le premier enfant
p=1 : 0 devient le second enfant
p=2 : etc...

objc_sysvar

m a x1 x2 objc_sysvar : renvoie (m=0) ou fixe (m=1) les attributs des objets en 3D, a dertermine quels attributs et x1,x2 sont les valeurs (si m=1 et sans signification si m=0).

intout : 0 si erreur
intout+2 : x1 en lecture
intout+4 : x2 en lecture

getradio

i adr getradio : i est l'index d'une boite de l'arbre pointé par adr contenant des radio-boutons. La fonction envoie l'index du premier radio-bouton sélectionné. Elle renvoie zéro si aucun n'est sélectionné.

La recherche ne se fait que sur un seul niveau, les radio-boutons doivent être les enfants directs de la boîte d'index i.

9 Les formulaires dans les fenêtres

L'appel de form_do est bloquant, c'est à dire que l'utilisateur doit absolument y répondre et en sortir avant de pouvoir à nouveau utiliser le menu ou un autre formulaire. Pour pallier à ceci, on place les formulaires dans des fenêtres. Ainsi on pourra passer d'un formulaire à l'autre en sélectionnant la fenêtre appropriée, l'accès au menu sera également possible.

&wdial_create

tree &wdial_create XXXX : crée un mot XXXX correspondant à une structure qui permettra l'inclusion de l'arbre d'objets tree dans une fenêtre. A ce stade rien n'est encore affiché, seul un appel à form_center a été fait pour que la première ouverture se fasse au centre de l'écran.

A l'exécution XXXX laisse sur la pile l'adresse de la structure suivante:

adr : tree (long)
adr+4 : x,y,l,h (mots) de l'espace de travail (taille de l'arbre)
adr+12 : x,y,l,h (mots) de l'espace total.
adr+20 : handle fenetre ou 0 (pas encore ouvert ou refermé)
adr+22 : index de l'objet éditable contenant le curseur ou 0.
adr+24 : position (en caractères) du curseur

Le cadre de base de l'arbre est automatiquement simplifié: bordure à zéro, pas d'effet OUTLINE ni SHADOW.

wdial_open

i " titre" XXXX wdial_open : ouvre la fenêtre pour la stucture XXXX (elle contient les coordonnées, l'adresse de l'arbre). " titre" est le titre de la fenêtre, i l'index de l'objet éditable qui recevra le curseur en premier ou 0 si il n'y en a pas.

Si la fenêtre est déjà ouverte, elle est ramenée au premier plan.

La fonction renvoie le handle de la fenêtre ou un code d'erreur.

(l'arbre n'est pas dessiné, c'est par evnt_multi et wdial_formdo que le dessin se fera par l'intermédiaire des REDRAWS)

wdial_close

XXXX wdial_close : referme la fenêtre et le formulaire disparait.

wdial_evnt

buf intout wdial_evnt : il s'agit d'un appel standardisé à evnt_multi comprenant les évènements suivants:

evnt_mesag (dont le buffer est buf)
evnt_keybd
evnt_button (seul le bouton gauche est surveillé)

Il permet donc de surveiller les actions sur les fenêtres, le menu, les boutons des formulaires et les objets éditables (FTEXT ou FBOXTEXT). Il remplit buf (16 octets mini) et intout (14 octets mini) comme evnt_multi.

Le fait d'imposer le paramètre intout permet de choisir n'importe quelle autre zone (au lieu du intout standard) et ainsi d'effectuer d'autres appels AES ou VDI avant de traiter l'évènement (il serait sinon effacé par un autre appel).

wdial_formdo

buf intout wdial_formdo : gère les formulaires dans les fenêtres dès qu'un évènement a surgi. Buf et intout sont ceux de evnt_multi ou de wdial_evnt. (Si il s'agit du intout de evnt_multi, c'est le intout standard et il peut être effacé à chaque appel AES ou VDI!).

- pour un évènement fenêtre, vérifie qu'elle est dans la liste de celles créees avec wdial_open et prend en charge: REDRAW, MOVED, TOPPED et CLOSED.
- pour un évènement bouton vérifie que le clic est au dessus d'un objet de la fenetre au premier plan ouverte avec wdial_open et prend en charge les boutons SELECTABLEs, le passage du curseur d'un FTEXT à l'autre.
- pour un évènement clavier vérifie que la fenêtre au premier plan contient bien un objet EDITABLE (ou un bouton DEFAULT) et prend en charge l'édition complète du champ, le passage d'un FTEXT à l'autre (tab ou flèches) ainsi que la touche ENTER (bouton DEFAULT).

Valeurs renvoyées:

0 : l'évènement ne concerne pas une fenêtre ouverte avec wdial_open ou l'évènement a entièrement été géré par la fonction.

1 : l'évènement nécessite un complément de traitement par l'utilisateur, on trouve alors sur la pile XXXX (la stucture concernée) ainsi que l'index i du bouton EXIT ayant provoqué l'évènement:

i=-1 : bouton CLOSER de la fenêtre
i>=0 : autre bouton EXIT du formulaire.

Une fois ce complément d'action pris en charge, il faut rappeler wdial_formdo afin de s'assurer qu'aucun autre évènement n'était en attente. Ce n'est que lorsqu'on reçoit 0 que l'on peut rappeler wdial_evnt pour une autre série d'évènements. C'est pour ceci qu'il est important de définir une seconde zone 'intout' indépendante de la zone standard afin d'en conserver les valeurs sur plusieurs appels à wdial_formdo.

A chaque appel, le masque d'évènement (dans intout) est mis à jour pour oter ceux déjà traités.

wdial_change

i XXXX wdial_change : permet de désélectionner le bouton EXIT i ayant été sélectionné par l'utilisateur. Si la fenêtre de l'arbre d'objets est au premier plan le bouton sera redessiné, sinon seul son STATE est modifié et il sera correctement redessiné dès que la fenêtre repassera en avant.

Voici un squelette de programme utilisant ces mots:

création des arbres avec 'dialogue' ou chargement avec 'rsrc_load'

```
arbre1      &wdial_create  fen1           \ création des structures
arbre2      &wdial_create  fen2
....
16 allot constant BUF               \ buffer MESAG
14 allot constant OUT               \ nouvel intout
```

*affichage éventuel d'un menu et/ou d'un formulaire (**wdial_open**)*

```
begin
            BUF   OUT wdial_evnt     \ surveille les évènements
      réaction aux évènements ne concernant pas
      les formulaires (menu->ouverture de formulaires avec
            wdial_open, autres fenêtres...)
      begin
            BUF   OUT wdial_formdo         \ s'occupe du reste
      while                                \ si autre que 0
            case
            fen1 of                        \ traitement séparé par arbres
                  case
                  -1 of   fen1 wdial_close endof\ fermer
                  i1 of   réagir au bouton i1          \ bouton EXIT
                        i1   fen1   wdial_change \ état normal
                        endof
                  i2 of ....
                  endcase
            endof
            fen2   of                      \ idem pour les autres
                  ....
            endof

            endcase
      repeat                  \ continuer formdo pour finir évènements
again                         \ reprendre evnt, autres évènements
```

10 Pseudos fenêtres de texte

La gestion d'une fenêtre de texte demande du travail, en particulier celui d'enregistrer ce qui a été affiché afin de pouvoir le redessiner en cas de recouvrement partiel ou total de l'espace. On peut faciliter cette opération en créant une pseudo-fenêtre de texte qui sera un simple arbre d'objets GEM dont chaque ligne est associée à une chaîne d'un tableau.

Ceci est très pratique en particulier en liaison avec les instructions précédentes de formulaires dans une fenêtre, les déplacements, redessins, mise en avant sont tous gérés automatiquement.

gem_list

i T$ l n " options" gem_list : fabrique une pseudo-fenêtre de n lignes de texte, chacune ayant l caractères de long et associées au tableau de chaînes T$ à partir de son élément i.

Les options sont codées sur une lettre chacune :

A : (arrows) deux flèches sélectionnables haut et bas sont également présentes
S : les lignes sont sélectionnables à la souris
E : les lignes sont éditables (des FTEXT)
B : une bordure est dessinée

La fonction renvoie l'adresse de l'arbre créé dans le dictionnaire.

S'il y a des boutons flèche, ils sont toujours les objets 1 et 2 de l'arbre.

Si les lignes sont sélectionnables, la première ligne a le numéro d'objet 1 (sans les flèches) ou le numéro 4 (s'il y a les flèches).

assign_array

i T$ adr assign_array : permet de réassigner à la pseudo-fenêtre pointée par adr les éléments du tableau T$ à partir de l'élément i.

resize_list

n adr resize_list : redimensionne une gemlist dans la limite du nombre de chaînes indiquées lors de la création. Si des flèches sont présentes, n doit être au moins égal à 2.

Renvoie 1 si ok, et 0 si erreur (n en dehors des valeurs autorisées)

11 Générer un code source automatiquement

La gestion d'un menu et de ses boîtes de dialogue demande un certain travail de rédaction qui peut être automatisé. Le FORTH dispose d'une instruction créant un code source pour gérer un fichier RSC contenant un menu, des dialogues et des alertes.

rsc2for

ouvre un sélecteur de fichier dans lequel vous devez sélectionner le fichier DEF correspondant à votre ressource. Ces deux fichiers doivent se trouver dans le même répertoire et partager le même nom (sauf l'extension DEF ou RSC).

Cette instruction est sensible au réglage de **select+/select-**.

Au retour, vous trouverez alors sous l'éditeur le code source gérant cet arbre.

Limitations :

- le fichier ressource ne peut contenir qu'un seul menu.
- seuls les objets ayant été nommés (et donc apparaissant dans le DEF) sont gérés.

Les initialisations concernent le chargement du DEF (pour créer toutes les variables), celui du RSC, la création des fenêtres pour chaque arbre et la définition de quelques variables.

Chaque arbre de nom AAAA se verra ajouter une routine **do_AAAA** qui gère la fermeture de sa fenêtre et qui fait appel à des sous-programmes **do_XXXX** pour chaque objet de type EXIT.

Il ne vous reste qu'à gérer les actions de ces objets.

Le menu de nom MMMM se verra ajouter une routine **do_MMMM** qui appelle les **do_XXXX** correspondant aux noms des entrées du menu.

Une routine **main** lance le programme, elle affiche le menu (si disponible), elle contient en commentaire les lignes pour ouvrir les fenêtres des arbres :

```
\ index " titre" win_AAAA wdial_open drop
```

Il vous appartient de modifier le titre de la fenêtre et de mettre l'index du premier champ FTEXT recevant le curseur, ou zéro s'il n'y en a pas. Ces lignes sont placées en commentaires, car vous ne désirez pas nécessairement que l'ouverture des fenêtres s'effectue au démarrage.

Sa boucle principale gère les appels au menu, aux dialogues et se termine si la variable **exit_flag** est non nulle. À vous de décider lors de quelle procédure il faudra activer cette variable.

On peut demander l'inclusion d'une **pseudo-fenêtre (gem_list)** directement à partir du fichier ressource. Pour ce faire, vous devez ajouter une chaîne libre (Free String) dont le nom commence par le caractère _ (souligné) et donner comme valeur à cette chaîne :

G=l,n,options

G pour gem_list, l caractères par ligne, n lignes et les options de gem_list. Le caractère souligné sera enlevé pour définir le nom de cette fenêtre. Il est donc limité à 7 caractères puisque les noms dans le fichier DEF ne peuvent dépasser 8 caractères.

La ligne peut être complétée des valeurs initiales des chaînes, par exemple pour créer une gem_list affichant des langues à sélectionner :

G=8,3,S,Français,English,Italiano

Et vos chaînes seront initialisées, la fenêtre prête à être affichée.

Ainsi, la free string de nom _LISTE et de valeur G=10,5,S génèrera :

- la création d'un tableau de 5 chaîne de 10 caractères nommé **LISTE$**
- la création d'un arbre de 5 lignes associées à ce tableau d'adresse **tree_LISTE**, dont les éléments sont sélectionnables.
- la création des routines de gestion de la fenêtre **win_LISTE** avec une routine **do_LISTE** appelée à chaque sélection de ligne.

Vous pouvez également gérer des **drop_down** : des listes qui apparaissent lorsqu'on clique sur un champ en particulier et vous permettent de choisir l'une des valeurs proposées. Pour ce faire, vous devez signaler le champ comme « drop_down » avec un flag étendu et créer un dialogue contenant une liste de chaînes auquel il sera associé. Pour ce faire :

➔ l'objet source doit être « selectable » et « exit » avec un nom XXXX par exemple, on lui ajoute le flag étendu 8 du object_status.

➔ le dialogue doit s'appeler exactement XXXX0 (avec un zéro ajouté), il ne doit contenir qu'un cadre (BOX) et une liste de chaînes (STRING) « selectable » et « exit ». La première chaîne est un équivalent de « annuler » et ne modifie pas la sélection. Vous pouvez la nommer d'une manière qui suggère qu'on revient en arrière.

La routine associée à l'objet source, **do_XXXX**, ouvrira le drop_down et vous renvoie l'index (1 à n) de la chaîne sélectionnée. De plus, si l'objet source est lui-même du type STRING ou TEXT (et tous ses dérivés), alors son contenu est automatiquement mis à jour avec la sélection opérée. Attention à ce que la chaine source et celles du drop_down aient **exactement la même longueur** !

Pour finir, une routine **set_drop_down** vous permet d'initialiser le contenu de l'objet source avec l'une des lignes du drop_down.

Au lieu de créer un dialogue complet pour le drop_down, vous pouvez utiliser une **gem_list**. Dans ce cas, cette gem_list doit s'appeler _XXXX0 (attention, tout doit tenir

dans 8 caractères!) et avoir exactement ces options : S (pour lignes sélectables) et B (pour une bordure) :

G=l,n,SB

Là encore, l'initialisation des chaines trouve toute sa raison d'être et facilite la programmation. Sinon, pour un drop down dynamique, il vous appartiendra d'initialiser le tableau XXXX0$ avec le contenu qui doit apparaître.

Il est possible d'ajouter les **routines de gestion de la fenêtre Forth** de base ou bien d'une **fenêtre utilisateur** plus complète à partir du ressource toujours. Pour ce faire, ajouter une chaîne libre (Free string) dont le nom commence par le caractère _ (souligné) et donner comme valeur à cette chaîne :

F=

Pour gérer la fenêtre du Forth, c'est celle qui est mappée sur la page0, elle ne contient qu'un élément de titre, le mover (déplacement) et le sizer (redimensionnement).

Ou bien la valeur :

W=options

Pour gérer une fenêtre utilisateur avec les options que vous désirez par mi celles-ci :

N (name), I (info), S (sizer), F (fuller), C (closer)

U (up), D (down), V (vertical slider), L (left), R (right), H (horizontal slider).

En fin de source, vous trouvez en commentaire la **structure** détectée de votre fichier ressource. Voici un exemple :

```
\ menu  MYMENU -> do_MYMENU tree_MYMENU
\    MItem MINFO -> do_MINFO
\    MItem MCPU -> do_MCPU
\    MItem MGRAPHIC -> do_MGRAPHIC
\    MItem MQUIT -> do_MQUIT
\ dial  CPU -> do_CPU tree_CPU win_CPU
\    FText CPUTEXT
\    Buttn CPUTEST (Exit ,Selct) -> do_CPUTEST
\    Buttn FPUFLAG (Selct)
\ dial  GRAPHIC -> do_GRAPHIC tree_GRAPHIC win_GRAPHIC
\    FText GWIDTH
\    FText GHEIGHT (Edit )
\    FText GBITS
\    Buttn GTEST (Exit ,Selct) -> do_GTEST
\ alrt  ALQUIT
\ alrt  ALINFO
\ glst  LIST -> do_LIST tree_LIST win_LIST
\ fwin  WINDOW -> do_WINDOW
\ uwin  USER -> do_USER
```

On dispose d'**un menu** appelé MYMENU (c'est une constante donnant l'ordre du menu dans les arbres du RSC, s'il est en premier MYMENU=0) géré par la procédure do_MYMENU (complète, vous n'aurez pas à la modifier normalement) et dont l'adresse de l'arbre en mémoire est tree_MYMENU. Cette dernière constante peut servir dans la fonction menu_icheck par exemple pour mettre une marque ou l'enlever devant une entrée du menu.

On a ensuite **quatre entrées de menu** associées à leur procédure (elles sont toutes vides pour l'instant). L'entrée MINFO activera la procédure do_MINFO. Dans celle-ci on peut simplement afficher une alerte d'information :

```
: do_MINFO
     1 ALINFO do_alert
;
```

Viennent ensuite les **deux dialogues** qui seront affichés dans des fenêtres. Chacun a son nom, sa procédure, l'adresse de son arbre ainsi que la structure de sa fenêtre. Par exemple, le dialogue CPU est géré par la procédure do_CPU (à laquelle il n'y a rien à ajouter normalement), son adresse en mémoire est tree_CPU et sa structure de fenêtre utilisable avec tous les appels wdial_* est win_CPU.

Par exemple, l'entrée de menu CPU peut ouvrir la fenêtre du CPU :

```
: do _MCPU
     0 " Cpu" win_CPU wdial_open drop
;
```

0 en premier paramètre, car il n'y a pas de FTEXT editable. Et l'entrée du menu MGRAPHIC l'autre fenêtre :

```
: do _MGRAPHIC
     GHEIGHT " Graphic" win_GRAPHIC wdial_open drop
;
```

GHEIGHT en premier paramètre, car c'est l'index du champ éditable.

Vous remarquerez que les boutons de type EXIT comme GTEST et CPUTEST ont leur procédure associée, elle est vide, il vous appartient de décider de leurs actions.

Si vous devez **modifier les chaîne de texte** d'un objet, le plus simple est d'utiliser une chaîne FORTH que vous associez à l'objet.

Par exemple l'objet CPUTEXT peut être géré ainsi :

```
10 string CPUTEXT$
>comp
CPUTEXT$ CPUTEXT tree_CPU assign_string
```

Au démarrage, sa valeur sera celle trouvée dans le RSC. Mais si vous modifiez son contenu, il suffira d'un redraw pour qu'il change à l'écran :

```
" MC68060" CPUTEXT$ $!
win_CPU update_win
```

Le bouton CPUFLAG a simplement le caractère Selct, montrant qu'il peut être sélectionné ou non. Vous pouvez **le tester** ainsi :

```
CPUFLAG tree_CPU get_obstate 1 and
```

Ceci renvoie 1 s'il est sélectionné et 0 sinon.

Vous pouvez le **sélectionner** avec :

```
CPUFLAG tree_CPU get_obstate 1 or
CPUFLAG tree_CPU set_obstate
win_CPU update_win
```

Ou le **désélectionner** avec :

```
CPUFLAG tree_CPU get_obstate %hFE and
CPUFLAG tree_CPU set_obstate
win_CPU update_win
```

On trouve en suivant, la **liste des alertes** qui vous rappelle leurs noms. La procédure d'appel do_alert est détaillée ci-dessous.

Par une Free String on a défini une **pseudo-fenêtre (gem_list)** appelée LIST. Sa routine do_LIST pourra faire appel à :

up_LIST et down_LIST si les flèches sont présentes dans les options

lines_LIST au cas où les lignes soient sélectionnables.

Cette dernière procédure reçoit sur la pile le numéro de la ligne sélectionnée en commençant par zéro.

Pour terminer, on trouve la définition des **deux fenêtres** par l'intermédiaire d'une Free String. La première est la **fenêtre Forth (fwin)** de nom WINDOW qui s'ouvre et se ferme avec Fastopen/Fastclose et est liée directement à la page0 pour les sorties texte et graphiques. Vous n'aurez dans la pratique qu'à gérer le redessin du contenu dans la routine :

redraw_WINDOW

qui par défaut contient un v_recfl pour remplir de blanc la zone à recréer.

La seconde est une **fenêtre utilisateur (uwin)** de nom USER qui aura les options que vous avez voulues. Elle pourra donc, selon ses éléments, faire appel à :

arrows_USER pour gérer les clics sur les flèches

hslid_USER et vslid_USER pour les mouvements des ascenseurs

redraw_USER pour le redessin.

Quelques procédures intégrées au code source :

Si le RSC contient des alertes, la procédure **do_alert** est ajoutée et elle s'appelle avec :

```
n XXXX do_alert
```

n étant le numéro du bouton par défaut et XXX le nom donné dans le DEF.

assign_string

Elle permet d'assigner une chaîne FORTH à un objet de type STRING, BUTTON vous donnant une plus grande facilité de modifier ces objets.

```
str XXXX tree_AAAA assign_string
```

Dans le dialogue de nom AAAA, assigne la chaîne str à l'objet XXXX. La chaîne est initialisée avec la valeur trouvée dans le RSC.

assign_fstring

```
str XXXX tree_AAAA assign_fstring
```

Fonctionne comme la routine précédente mais pour les objets de type TEXT ou FTEXT.

get_obstate

Renvoie l'octet d'état d'un objet.

```
XXXX tree_AAAA get_obstate
```

Renvoie l'état de l'objet XXXX du dialogue AAAA.

set_obstate

Fixe l'octet d'état d'un objet.

```
s XXX tree_AAAA set_obstate
```

Met à s l'état de l'objet XXXX du dialogue AAAA.

update_win

```
win_AAAA update_win
```

Permet de redessiner l'aspect de la fenêtre contenant l'arbre AAAA si des éléments de celui-ci ont été modifiés (des chaînes, l'état d'un objet,...).

Si une fenêtre est définie dans le RSC, alors la procédure suivante est disponible :

max_xywh

Elle s'utilise sans paramètre et renvoie les coordonnées et dimensions maximales d'une fenêtre à l'écran. Elle peut s'utiliser en liaison avec wind_create ou wind_open.

Si un drop_down est utilisé, vous avez accès à :

set_drop_down

```
source tree_source n tree_dropdown set_drop_down
```

donne à l'objet source de l'arbre tree_source la valeur de la chaîne n de l'arbre

tree_dropdown.

Les inclassables

1 Le son DMA

setplay

deux syntaxes possibles:
1) -1 setplay : renvoie le système DMA détecté
0 : aucun
1 : DMA (STE/TT/Falcon)
2 : SAGA (Apollo Vampire)

2) r ms f setplay : fixe les paramètres du son DMA.
r : jouer une seule fois (0), répéter le son (1).
ms : bit #0: son mono (0) ou stéréo (1)
bit #1: 8 bits (0) ou 16 bits (1) Uniquement sur SAGA!
f : fréquence en kHz (0=6.25, 1=12.5, 2=25, 3=50).

play

trois syntaxes possibles:
1) a t play : joue le son se trouvant à l'adresse a et de taille t en octets. Le son est sur 8 bits signé, pour le son stéréo un octet sur deux correspond à la voie de droite, l'autre à la voie de gauche. Les réglages sont ceux passés à setplay.
2) 0 play : arrête le son en cours.
3) -1 play : rend la main lorsque le morceau en cours est terminé.

st_allot

tt_allot

ce sont des variantes de allot qui forcent l'allocation en ST RAM ou en TT RAM sur un TT.
n st_allot : alloue un bloc de n octets dans le dictionnaire si celui-ci est en ST RAM, sinon demande au GEMDOS un bloc en ST RAM. L'adresse est renvoyée sur la pile.
n tt_allot : alloue un bloc de n octets dans le dictionnaire si celui-ci est en TT RAM, sinon demande au GEMDOS un bloc en TT RAM. L'adresse est renvoyée sur la pile.
st_allot s'impose pour les fichiers sons, car le DMA ne peut pas accéder à la TT RAM. Sur SAGA, cela n'a pas d'importance.

loadbin

d t " fichier" adr loadbin :
si adr<>0 : charge un fichier à l'adresse indiquée, d est l'offset du premier octet par rapport au début du fichier, t est la taille en octets. renvoie la taille effectivement lue ou un code d'erreur GEMDOS.
si adr=0 : même travail mais FORTH alloue un bloc avec malloc de la taille nécessaire et renvoie soit un code d'erreur, soit la taille lue avec en dessous l'adresse du bloc. Toujours dans ce cas, si d<0, c'est le nombre d'octets libres demandés dans le bloc avant le fichier. Ainsi, si adr=0 et d=-100, le programme alloue un bloc de "t+100" octets et charge le fichier à partir de la position 100. La taille renvoyée est toujours la taille lue sur le disque.
Note: en utilisant t=-1, on charge jusqu'à la fin du fichier.

savebin

t " fichier" adr savebin : crée un fichier et y écrit t octets se trouvant à l'adresse adr. renvoie le nombre d'octets effectivement écrits ou un code d'erreur GEM-DOS.

2 La cartouche ST REPLAY 16 (abandonné)

setreplay

f init inter setreplay : fixe les paramètres pour la gestion de la cartouche.
f : fréquence de 48 Hz à 614400 Hz.
init : adresse d'une routine d'initialisation.
inter : adresse de la routine d'interruption.

replay

exécute la routine d'initialisation, lance le TIMER A avec la routine d'interruption. Le clic droit de la souris permet l'arrêt. La routine d'interruption peut elle aussi y mettre fin, la fonction replay rend alors la main.

Les deux routines doivent être écrites en assembleur, les trois mots qui suivent permettent de générer les instructions donnant l'accès en lecture et en écriture du son ainsi que l'instruction mettant fin au traitement:

replay_in (effacé)

n replay_in : génère dans le dictionnaire l'instruction move $FB0000,Dn. C'est

donc le registre Dn qui contient le mot lu.

replay_out (effacé)

n replay_out : génère l'instruction tst ($FA8000,Dn.w). Donc écrit sur le port cartouche la valeur de mot contenue dans Dn.

replay_end (unutile)

génère les instructions qui redirigent l'interruption sur un simple RTE et qui mettent à 1 un flag interne mettant fin à replay.

3 Programmation des timers

Outre les routines de son, il peut être utile de programmer une routine en interruption à n'importe quelle fréquence:

timerA

Deux syntaxes possibles:

- **timer_stop** timerA : arrête le timer A, la constante timer_stop vaut en fait 2.

- rout mfp mode freq end timerA : lance le Timer A avec les paramètres suivants:

rout: adresse de la routine à exécuter périodiquement. Se termine par un RTE.
mfp: adresse des registres MFP à utiliser. On utilisera les deux constantes suivantes:
MFP_ST = $FFFA80
MFP_TT = $FFFA00. Sur un ST, seul le MFP ST existe!
mode: mode de comptage, on utilisera les deux constantes suivantes:
count=8 pour une synchro avec une source extérieure
delay=0 pour un comptage basé sur le temps.
freq: fréquence voulue en Hertz
end: mode de fin de routine, on utilisera les deux constantes suivantes:
auto_end=0 pour un retour automatique à la normale lors du RTE
soft_end=1 pour un retour programmé.

Exemple:

```
routine   MFP_ST   delay   9600    auto_end   timerA
```

lance le timer A du ST sur la routine spécifiée à la fréquence de 9600 hertz avec fin

de routine automatique.

```
routine   %hFFFA00   0   9600   0   timerA
```

a exactement le même effet que la ligne précédente sauf que les noms des constantes n'ont pas été utilisés.

Ensuite:

```
timer_stop timerA   \ (ou 2 timerA) stoppe l'exécution de la routine.
```

timerB, timerC, timerD

se programment de la même manière que le timer A.

timer_calc

freq timer_calc renvoie les valeurs internes Control et Data de la fréquence la plus proche de celle désirée. Control au sommet de la pile.

4 Souris, Joystick

joyst

n joyst : fixe le mode de surveillance des données Joystick.
n=1 : seul le joystick 1 est surveillé et la souris reste active. Note: le bouton de tir du joystick est équivalent au bouton droit de la souris.
n=2 : les deux joysticks sont surveillés et la souris n'existe plus.
Dans ces deux cas, deux vecteurs de Kbdvbase sont détournés. Il faut absolument utiliser **mouse** dès que l'accès aux joysticks est devenu inutile.

mouse

passe en mode de surveillance de la souris (mode normal). Ce mode est réactivé en quittant le programme, il réinitialise les vecteurs Kbdvbase.

mousex mousey mousek

renvoient respectivement les coordonnées x, y de la souris et l'état des deux boutons (bit0:gauche, bit1:droit). Ici, pas de vecteurs détournés, ce sont des appels VDI.

jx0 jy0 fire0

renvoient la position horizontale (-1:gauche, 0:centre, 1:droite), la position verticale (-1:haut, 0:centre, 1:bas) et l'état du bouton (1:appuyé, 0:relaché) du joystick 0 (celui qui prend la place de la souris). Joystick actif uniquement en mode **2 joyst**.

jx1 jy1 fire1

renvoient la position horizontale (-1:gauche, 0:centre, 1:droite), la position verticale (-1:haut, 0:centre, 1:bas) et l'état du bouton (1:appuyé, 0:relaché) du joystick 1. Instructions actives uniquement en mode **1 joyst** ou **2 joyst**.

Note: Sur l'**Apollo Vampire**, les deux joysticks DB9 sont toujours accessibles et le fire0 est équivalent à mousek=1. De plus, le Joypad USB est mappé sur le Joystick 1. Deux cas:

mode **1 joyst**: **fire1** renvoie un masque de bits avec tous les boutons du Joypad, la touche A étant identique au bouton de tir du joystick1.

0000 00sb 0000 YXBA (s=start et b=back)

mode **2 joyst**: **fire1** renvoie uniquement le bouton de tir ou la touche A.

5 Les autres

desk

Affiche une barre de menu, permet le réglage de la position et de la taille de la fenêtre FORTH, donne accès aux accessoires ou aux autres applications (sous MultiTOS). On revient au FORTH à l'aide de l'option Retour ou en réactivant la fenêtre (sous MultiTOS). La page de base du FORTH est redimensionnée et réactivée (car celles de l'utilisateur ne sont plus forcément bien ajustées).

ltype

" chaine" ltype : envoie une chaine de carctères à l'imprimante. On peut par ce moyen envoyer une série de codes de commande, mais, le caractère 0 marquant la fin de chaine, lui ne pourra pas être envoyé.

&cookie

&cookie XXXX : crée un mot XXXX dont le nom de 4 lettres correspond exactement au cookie recherché (penser aux majuscules si nécessaire). A l'exécution XXXX laisse sur la pile:

soit 0 si le cookie n'existe pas,

soit 1 si il existe. En dessous du 1 se trouve sur la pile la valeur du cookie (parfois une adresse sur un bloc de paramètres, parfois un numéro de version, etc...).

```
&cookie MiNT                        \ crée le mot
MiNT if                             \ si il y a 1
            ." Version " h.         \ si présent sa valeur est la version
else
            ." MiNT non installé"   \ si absent, on le signale
then
```

cache

x cache : fixe la valeur du registre CACR gérant le cache des 68030. Valeurs possibles de x:

%ha0a : vide en désactive les deux caches
%h101 : active les deux caches
%ha00 : vide et désactive le cache données
%h100 : active le cache données
%h00a : vide et désactive le cache instruction
%h001 : active le cache instruction

call

adr call : saute à une routine en assembleur à l'adresse adr. Elle doit se terminer par RTS (en utilisation normale). Il faut qu'elle ne modifie A4, A5, A6 qu'en connaissance de cause:

a6: pointeur de pile données
a5: pointeur d'interprétation
a4: pointeur de pile des retours.

trace_win

y h trace_win : fixe la fenêtre TRACE à la ligne y et sur une hauteur de h lignes (ceci en lignes de texte).

Le programme impose une valeur de 3 pour h au minimum.

Si y est négatif, la position est calculée à partir du bas de l'écran.

Pour tracer un programme en tout confort, on peut, avec desk, modifier la fenêtre FORTH pour qu'elle ne déborde pas sur les 3 dernières lignes et ainsi avoir deux zones séparées: exécution et traçage.

Par défaut, ce sont les 3 dernières lignes qui sont utilisées.

trace

f trace : passe en mode trace dès le mot suivant.

Si f=0 seuls les mots créés par l'utilisateur provoquent une pause.
Si f<>0 la plupart des mots provoquent la pause (sont exécutés automatiquement l'empilement des constantes entières, réelles, les codes internes des différentes variables, etc...)

Lors d'une pause, le mot est affiché à l'écran avec l'état de la pile juste **avant** son exécution.

[Enter] exécute les mots suivants et s'arrête au prochain selon f.
[S] (single step) exécute le mot affiché et s'arrête au mot suivant (indépendamment de tout autre réglage). Ceci permet d'observer la pile avant et après l'exécution du mot suivi.
[L] (leave) avance automatiquement jusqu'à une sortie de boucle ou d'un mot
[R] (until Return) si le mot affiché est un mot utilisateur, l'exécute entièrement et revient en pause à son retour.
[B] (breakpoint) lance l'éxécution jusqu'au retour à cette même position
[Esc] quitte l'exécution et revient à la ligne de commande.

untrace

arrête le traçage des programmes dès le mot suivant.

list watch

list 'mot1 'mot2... 'motn watch Permet de fixer une liste de mots à tracer.
Dans ce cas, l'arrêt se fait uniquement sur les mots de la liste (mots utilisateurs ou de base) indépendamment du réglage du flag "f" dans **trace**. La liste reste valide même après un **untrace**. Pour la supprimer, il faut entrer une liste vide:
list watch

illegal

insère l'instruction "illegal" du processeur qui déclenche une exception. Utile pour faire un point d'arrêt sous un debugger.
En mode interprété elle est suivie d'un RTS puis retour à la boucle d'interprétation.
En mode compilé, elle est insérée dans le programme.

À savoir:

- A5 est le pointeur d'interprétation sur l'instruction suivante (WORD)
- A4 est le pointeur de la pile des retours (utilisée aussi pour les boucles)
- A6 est le pointeur de la pile FORTH.

Les extensions mathématiques

1 Les fonctions

Une fonction n'est rien d'autre qu'un mot FORTH laissant son résultat sur la pile avec toutefois deux limitations:
- la fonction doit être un mot compilé (avec :: ... ;)
- la fonction prend sa ou ses valeurs dans des variables de type &float

Par exemple:

```
&float x
:: fonction x   f@    x^2    %f5    f-    ;
```

Ceci programme f(x)=x² - 5

solve

'FFF X %ferr solve : ce mot permet de résoudre une équation de type f(x)=0.
FFF est le nom d'une fonction
X est la variable utilisée par la fonction
%ferr est une valeur réelle correspondant à la précision souhaitée.

En retour, solve renvoie sur la pile la précision atteinte et dans X la solution trouvée.

Avec l'exemple ci dessus:

```
2 x f!                                  \ valeur de départ de x
'fonction x%f0.0001 solve f.            \ affiche la précision atteinte
x f@ f.                                 \ affiche la solution 2.236....,
                                        \ la racine carrée de 5.
```

gauss

'FFF X %fa %fb n gauss: ce mot permet de calculer l'intégrale de la fonction FFF pour la variable X allant de a à b. Le nombre d'intervalles n permet de fixer la précision du calcul.
En retour, gauss renvoie une approximation de l'intégrale.

Dans la pratique, toute fonction polynôme, jusqu'au degré 9, est intégrée exactement avec un seul intervalle (n=1). Pour les autres, passer n à 10, à 100 ou plus peut permettre d'obtenir une valeur plus précise.

Cette méthode ne prend pas les valeurs aux bornes, de ce fait, les intégrales généralisées (1/sqr(x) en 0) ou prolongées par continuité (x*log(x) en 0) sont acceptées.

Avec la fonction de départ:
```
'fonction x %f0 %f1   1 gauss f.
```
affiche -4.666..., c'est à dire l'intégrale de x²-5 entre 0 et 1.

maximum

'FFF X %fa %fb %ferr maximum

Calcule la valeur de x correspondant à un maximum local entre a et b avec l'erreur err demandée.

Il faut qu'il n'y ait qu'un seul maximum dans l'intervalle.

Renvoie l'erreur atteinte sur la pile et remplit la variable X.

minimum

'FFF X %fa %fb %ferr minimum

La fonction minimum fonctionne de la même manière que maximum pour un minimum local.

lambert

%fa lambert

La fonction de lambert qui renvoie la valeur w telle que $w \times e^{w} = a$, elle est définie pour $a \in [\frac{-1}{e} ; +\infty]$ et renvoie la valeur de la branche principale dans $[-1 ; +\infty]$. Cette fonction est un algorithme basé sur newton, donc assez lente en particulier sur une machine sans FPU.

lambert1

%fa lambert1

Même comportement que **lambert** sauf que si $a \in [\frac{-1}{e} ; 0]$, la valeur renvoyée est celle de la branche secondaire $[-\infty ; -1]$.

Exemple: pour résoudre $x^{x} = 0{,}75$, on utilise $x = e^{lambert(\log(0{,}75))}$

```
0.75 log lambert exp f.  -> renvoie x = 0,6362....
0.75 log lambert1 exp f. -> renvoie x = 0.1535....
```

Deux solutions !

2 Les courbes en 2D

Pour tracer une courbe, vous devez définir la page dans laquelle se fera le tracé à l'aide de **setpage**. Par défaut, la **page0** correspondant à la fenêtre FORTH est utilisée.

Puis, vous devez définir les limites des axes des abscisses et ordonnées avec **gr_window**. Il vous est possible de tracer les axes ainsi qu'une grille à l'aide de **gr_axes** et **gr_grid**.

Pour finir, vous pourrez tracer vos fonctions cartésiennes, paramétriques ou polaires.

gr_window

%fxmin %fxmax %fymin %fymax gr_window

Définit la fenêtre d'affichage des courbes pour la page en cours. Les quatre paramètres sont des réels et veillez à ce que xmin<xmax et ymin<ymax.

gr_axes

Sans paramètre, ce mot affiche simplement les axes s'ils sont visibles.

Effet de bord: la fonction utilise v_pline et, en sortie, applique un 0 0 vsl_ends qui annule les flèches.

gr_grid

%fxstep %fystep gr_grid

Affiche une grille avec une ligne verticale tous les "xstep" et une ligne horizontale tous les "ystep". Veillez à ce que ces deux valeurs réelles soient positives.

Effet de bord: la fonction utilise v_pline et, en sortie, applique un 0 vsl_style qui remet la ligne continue par défaut.

gr_y(x)

'FFF x %fx0 %fx1 n gr_y(x)

Trace la fonction FFF de variable x de x0 à x1 en calculant n points. La fonction FFF est compilée et renvoie simplement y(x) sur la pile.

Exemple: dessin d'une hyperbole

```
&float x

:: yx x f@ 1/x ;        \ la fonction y=1/x

cls
( %f-5 %f10 %f-4 %f4 ) gr_window
                       gr_axes
( 'yx x %f-5 %f10 99 ) gr_y(x)
```

gr_xy(t)

'FFF t %to %t1 n gr_xy(t)

Trace la fonction paramétrique FFF de variable t de to à t1 en calculant n points. La fonction FFF est compilée et renvoie x(t) puis y(t) sur la pile.

Exemple: dessin du symbole alpha:

```
&float t
\ x=-cos(2t) et y=sin(3t)

:: xyt
   t f@ fdup f+ cos fneg
   t f@ %f3 f* sin
;

cls
( %f-3.14 %f3.14 %f-1.1 %f1.1 ) gr_window
                                gr_axes
( 'xyt t %f-3.14 %f3.14 100 )   gr_xy(t)
```

gr_r(t)

'FFF t %to %t1 n gr_r(t)

Trace la fonction polaire FFF de variable t (l'angle) de to à t1 en calculant n points. La fonction FFF est compilée et renvoie r(t) (la distance algébrique à l'origine) sur la pile.

Exemple: dessin d'un trèfle à 4 feuilles:

```
&float t

:: rt t f@ sincos f* ;        \ la fonction r=cos(t)*sin(t)

cls
( %f-1.1 %f1.1 %f-0.7 %f0.7 ) gr_window
                              gr_axes
( 'rt t %f-3.14 %f3.14 100 )  gr_r(t)
```

gr_getpoint

n gr_getpoint

permet d'interagir avec le graphique sur la page en cours et de lire ou récupérer des coordonnées de points. Le pointeur de souris devient une croix lorsque vous êtes dans la zone graphique et redevient une flèche en dehors.

Dès que vous cliquez dans la zone graphique, une alerte s'ouvre avec les coordonnées réelles du point. Vous pouvez:

- **Empiler :** les coordonnées x et y du point restent sur la pile, si vous avez atteint le nombre maximal de points, la fonction se termine ici
- **Quitter :** la fonction se termine.
- **Vu ! :** le point est ignoré et vous pouvez cliquer à nouveau.

Le paramètre n règle le comportement de la fonction:

- n >0 nombre maximum de points à empiler (couples x et y réels)
- n = 0 seule la lecture des points est possible, on n'empile pas
- n = -1 aucune limite de nombre de points empilés, même si au delà de 99 l'affichage commencera à poser problème.

Renvoie le nombre de points empilés et les éventuelles coordonnées en dessous.

Effets de bord: la fonction utilise evnt_multi, form_alert et graf_mouse. En sortie, la forme de la souris est la flèche.

g_plot

%fx %fy gr_plot

Affiche un point dans la page courante en convertissant les valeurs réelles en coordonnées écran. Cette fonction utilise v_pmarker ainsi que ses réglages de couleur, taille et style.

gr_area

'test x y n gr_area

Remplit une zone selon le critère de la fonction test.

Cette fonction doit renvoyer une valeur entière 0 (faux) ou autre (vrai) correspondant à une condition dépendant de x et de y.

Toute la page FORTH courante est balayée, et la fonction est appelée pour chaque pixel (de ce fait, vous n'avez pas à initialiser x et y) et selon le retour, le point est marqué ou non.

La valeur n correspond à la densité de points souhaitée.

Si n=1, tous les points sont testés.

Si n=2, un point sur deux en largeur et en hauteur est testé (donc 1/4 des pixels)

Si n=3, un point sur trois en largeur et en hauteur est testé (donc 1/9 des pixels)

Etc.

En retour, vous obtenez dans x une approximation de l'aire coloriée (plus n est petit, plus l'approximation sera fine).

3 Les courbes en 3D

Pour tracer une courbe en 3D, il faut sélectionner la page d'affiche avec **setpage** puis définir les limites des axes avec **gr3_window**. On peut éventuellement tracer un cadre avec **gr3_grid** ainsi que les axes avec **gr3_axes**.

Finalement, on définit une fonction renvoyant les trois coordonnées x, y et z à partir d'un paramètre t et l'appel à **gr3_xyz(t)** fait apparaître la courbe.

gr3_window

%fxmin %fxmax %fymin %fymax %fzmin %fzmax gr3_window

Définit les limites des axes avec dans chaque cas la valeur min inférieure à la valeur max.

Cet appel réinitialise la valeur de **gr3_view** à zéro: projection x,y,z et la rotation **gr3_rot** à zéro.

gr3_grid

Affiche un cadre 3D montrant le fond de la zone de tracé.

gr3_axes

Affiche les trois axes x,y et z (en tenant compte de gr3_view). Si les limites fournies n'incluent pas l'origine, les axes peuvent ne pas apparaître.

gr3_xyz(t)

'FFF t %ft0 %ft1 n gr3_xyz(t)

Trace la fonction paramétrique FFF de variable t de t0 à t1 en calculant n points. La fonction FFF est compilée et renvoie x(t), y(t) puis z(t) sur la pile. Le tracé tient compte du paramètre gr3_view.

gr3_plot

%fx %fy %fz gr3_plot

Affiche un point dans la page courante en convertissant les valeurs réelles en coordonnées écran. Cette fonction utilise v_pmarker ainsi que ses réglages de couleur, taille et style.

Le tracé tient compte du paramètre gr3_view.

gr3_view

n gr3_view

Définit l'ordre des axes à l'écran. Par défaut, **n=0** pour une projection **x,y,z**.

C'est à dire x vers l'utilisateur, y vers la droite et z vers le haut.

Si **n=1**, c'est une projection **z,x,y**.

Si **n=2**, c'est une projection **y,z,x**.

Pour lire le mode en cours, on utilisera: **-1 gr3_view** qui renvoie la valeur 0, 1 ou 2. On peut donc modifier la perspective avec cette seule fonction sans modifier le reste du programme. Il faut bien sûr appeler de nouveau gr3_grid, gr3_axes et gr3_xyz(t)/plot pour obtenir la nouvelle vue.

4 Les surfaces en 3D

On bénéficie de deux fonctions pour tracer des courbes du type Z=f(X,Y), elles diffèrent dans leur vitesse et la qualité du rendu. Les autres fonctions relatives aux courbes 3D sont toujours disponibles (**gr3_window, gr3_grid, gr3_axes**) et l'orientation et la rotation du dessin peuvent également être modifiées avec **gr3_view** et **gr3_rot**.

La fonction à tracer, en mode assemblé, doit utiliser deux variables et renvoie la valeur de Z correspondante.

gr3_z(xy)l

'fn X Y n gr3_z(xy)l

Trace la fonction fn utilisant les deux variables X et Y par une série de lignes rapides sans tenir compte des traits cachés.

Le paramètre n correspond au nombre d'intervalles du maillage (si n=20, alors 21 lignes seront tracées).

La couleur est fixée par **vsl_color**.

gr3_z(xy)f

'fn X Y n gr3_z(xy)f

Trace la fonction fn utilisant les deux variables X et Y par une série de polygones en tenant compte des traits cachés.

Le paramètre n correspond au nombre d'intervalles du maillage (si n=20, alors 21 lignes seront tracées dans les deux sens).

La couleur du tracé est fixée par **vsf_color** puisqu'une fonction de remplissage est utilisée (**v_fillarea**). Les effets de bord sont la modification de **vsf_perimeter** et de **vsf_interior**.

gr3_rot

%fangle gr3_rot

Fixe l'angle (en radians) de rotation de la surface autour de l'axe Z. Si angle=0, il n'y a pas de rotation et le tracé est plus rapide.

5 Les graphiques

Voici les mots permettant de créer rapidement des **graphiques circulaires** ou en **barres**. Pour ces appels, un **paramètre "t"** définit le type de remplissage voulu.

Paramètre t

t type de remplissage qui est utilisé en boucle. Pour un diagramme circulaire ou en barres simples, on commence à la première valeur et on incrémente automatiquement. Pour des barres empilées, chaque nouvelle pile recommence la série de valeurs.

- t=0 conserver les réglages actuels de remplissage
- t=1 les couleurs (0-1 mono, 0-4 ST Basse et 2-15 autres modes)
- t=2 motifs de 9 à 24
- t=3 hachures de 1 à 12
- t=4 densité de 1 à 8
- 'XXX le type peut être le code d'un mot assemblé qui sera appelé à chaque nouveau secteur ou rectangle pour définir un style de remplissage personnalisé. À cet effet, la routine reçoit deux valeurs sur la pile:
  - Au sommet, la valeur n. Au premier appel, n≥100, cela veut dire qu'on commence une série, vous devez régler vos paramètres initiaux, préparer le premier style de remplissage et renvoyer n0 (valeur que vous choisissez). A l'appel suivant, vous recevrez n0+1. Vous préparez le remplissage suivant et pouvez renvoyer simplement n0+1. Ensuite vous recevrez n0+2, etc. Si une nouvelle série doit commencer, vous recevrez à nouveau n≥100.
  - En second paramètre, la valeur c. Pour les diagrammes en barres, elle correspond au numéro de la colonne en cours (à partir de 1). Pour un diagramme circulaire, elle correspond à l'angle de début du secteur en 1/10 de degré (par exemple pour un diagramme semi-circulaire avec 4 parts égales de 45°, on recevrait successivement: 0, 450, 900, 1350). Dans tous les cas, cette valeur doit être dépilée, seul "n" est renvoyé.

Les données

Ai représente l'adresse d'un tableau de données dont la taille dépend du réglage de **size** en cours.

- .b ou .w size les données sont des octets ou des mots, ils sont considérés comme non signés (0-255 ou 0-65535).
- .l size les données sont des mots longs et sont signés.
- .f size les données sont des nombres réels signés sur 8 octets.

Les labels combinés

Pour légender l'axe des abscisses d'un diagramme en barres ou les secteurs d'un diagramme circulaire, on combine l'ensemble des mots en une seule chaîne. Par exemple:

```
" LuMaMeJeVeSaDi" 2 7
```

Signifie qu'il y a 7 labels de 2 caractères. Si un label est plus court, vous devez l'aligner sur "c" avec des espaces. Au moment de l'affichage, les espaces de droite seront éliminés.

Taille de la fonte des labels

Le paramètre f correspond à la taille, elle est basée sur les 4 fontes standard du TOS. Pour un diagramme circulaire avec labels, vous devez spécifier ce paramètre. Pour un diagramme en barres, le FORTH adapte automatiquement la taille à l'espace disponible:

- f=0 petite fonte 6×6 (celle des icones du bureau)
- f=1 fonte 8×8 (celle de la résolution 320x200)
- f=2 fonte 8×16
- f=3 fonte 16×32 (sur le TT)

Les diagrammes (semi-)circulaires

Pour ces diagrammes, vous devez fournir des coordonnées, à vous de gérer l'espace selon la place disponible sur votre page.

gr_pie

x y r A1 n t gr_pie

trace un graphique circulaire:

- x y r les coordonnées du centre et du rayon du cercle.
- A1 n adresse des "n" données à représenter

gr_hpie

x y r A1 n t gr_hpie

trace un graphique semi-circulaire:

- x y r les coordonnées du centre et du rayon du demi-cercle.
- A1 n adresse des "n" données à représenter

gr_pie_label

x y r A1 n t "xxx" c f gr_pie

trace un graphique circulaire avec des labels:

- x y r les coordonnées du centre et du rayon du cercle.
- A1 n adresse des "n" données à représenter
- "xxx" c n labels combinés de c caractères chacun
- f taille de la fonte

gr_hpie_label

x y r A1 n t "xxx" c f gr_hpie

trace un graphique semi-circulaire avec labels:

- x y r les coordonnées du centre et du rayon du demi-cercle.
- A1 n adresse des "n" données à représenter
- "xxx" c n labels combinés de c caractères chacun
- f taille de la fonte

Les diagrammes en barres

Ces diagrammes utilisent toute la page en cours. Si vous voulez qu'ils ne débordent pas d'une certaine zone, vous devez créer une page avec **&page**, définir ses dimensions avec **defpage** et la sélectionner avec **setpage**. Par défaut, on trace sur la **page0**.

Ces fonctions travaillent sur la même base que les tracés de fonctions mathématiques 2D, elles font appel à **gr_window**, à **gr_axes** et éventuellement à **gr_grid** pour une grille de lecture.

gr_bar

A1 n t gr_bar

Trace un diagramme en barres:

- A1 n valeurs à représenter (peuvent être signées).
- t type de remplissage

gr_barg

dy adr n t gr_barg

Trace un diagramme en barres en ajoutant une grille de lecture:

- A1 n valeurs à représenter (peuvent être signées).
- t type de remplissage
- dy nombre réel (8 octets) donnant la taille d'une graduation sur l'axe des ordonnées (cela doit être cohérent avec vos données)
  - ► Si dy est négatif, la grille est tracée sous les barres.
  - ► Si dy est positif, la grille est tracée sur les barres.

gr_sbar

A1 A2 ... Ak k n t gr_sbar

Trace un diagramme en empilant "k" blocs par colonne:

- A1 A2 ... Ak k n k tableaux à empiler en n colonnes.
- t type de remplissage

Les valeurs de chaque tableau peuvent être positives ou négatives et seront empilées au-dessus ou au-dessous de l'axe des abscisses.

gr_sbarg

dy A1 A2 ... Ak k n t gr_sbarg

Trace un diagramme avec grille de lecture en empilant "k" blocs par colonne:

- A1 A2 ... Ak k n k tableaux à empiler en n colonnes.
- t type de remplissage
- dy nombre réel (8 octets) donnant la taille d'une graduation sur l'axe des ordonnées (cela doit être cohérent avec vos données)
  - Si dy est négatif, la grille est tracée sous les barres.
  - Si dy est positif, la grille est tracée sur les barres.

Les valeurs de chaque tableau peuvent être positives ou négatives et seront empilées au-dessus ou au-dessous de l'axe des abscisses.

gr_bar_hlabel

" xxx" c n gr_bar_hlabel

affiche les n labels combinés dans la chaîne "xxx" sous un graphique en barres. Pour cela, il faut qu'un graphique en barres ait été dessiné et que sa page soit encore sélectionnée. Sinon, le comportement est imprédictible.

Le nombre de labels "n" n'est pas obligé d'être celui du nombre de barres. Par exemple, vous venez de tracer un graphique annuel avec 12 barres pour les 12 mois. On pourra faire les appels suivants:

Pour l'initiale de chaque mois:

```
" JFMAMJJASOND" 1 12 gr_bar_hlabel
```

Pour grouper les mois par 3 en 4 saisons:

```
" HiverPrintÉté  Autom" 5 4 gr_bar_hlabel
```

Pour un seul commentaire:

```
" Sur une année" dup len 1 gr_bar_hlabel
```

gr_bar_vlabel

" xxx" gr_bar_vlabel

Ajoute une légende sur l'axe vertical du diagramme en barres. Le texte est tourné de 90°. Pour cela, il faut qu'un graphique en barres ait été dessiné et que sa page soit encore sélectionnée. Sinon, le comportement est imprédictible.

gr_bar_title

" xxx" gr_bar_title

Ajoute un titre au dessus du diagramme en barres. Pour cela, il faut qu'un graphique en barres ait été dessiné et que sa page soit encore sélectionnée. Sinon, le comportement est imprédictible.

Le Multitâche

Il s'agit d'un multitâche collaboratif. Chaque tâche décide du moment où elle passe la main aux autres. De plus, la bascule d'une tâche à l'autre n'est disponible qu'en mode interprété. Cela n'empêche pas l'utilisation du couple :: ;, mais on doit sortir de ces routines assemblées pour communiquer avec le reste du système.

1 Généralités sur les threads

Un thread est une tâche. Elle est définie par:

- un mot qui contient le programme à exécuter
- une pile de données personnelle définie avec **&stack**
- une page pour ses sorties textes ou graphiques définie par **&page**
- un identifiant unique, valeur entière, dont vous décidez.

Le reste des pointeurs internes pour bien séparer les tâches est géré par le FORTH.

Différents threads peuvent partager la même page, à vous de savoir comment. Sinon, le couple **setsubpage** et **getsubpage** est idéal pour créer des pages dans la même fenêtre.

Un thread peut être dans trois états différents:

- ✔ run : c'est l'état par défaut lors de sa création, lorsque vient son tour, il exécute son programme jusqu'à passer la main avec **thnext**
- ✔ pause : lorsque vient son tour, il est ignoré jusqu'à ce qu'une autre tâche le réveille avec **thwake**. Un thread peut se pauser lui-même ou l'être par une autre tâche avec la commande **thpause**.
- ✔ sync : il est en attente de synchronisation avec d'autres threads.

Lorsqu'un thread arrive à la fin de son programme, il est automatiquement effacé de la liste des tâches. Cet effacement peut également s'effectuer avec **thkill** ou **thkillall**.

2 Instructions générales

thread

page pile id 'mot thread

Définit le thread en associant la page, la pile, l'identifiant et le mot à exécuter. Ceci ajoute simplement la tâche dans une liste sans rien exécuter pour le moment.

vthread

page pile id 'mot vthread

identique à **thread** sauf qu'une station VDI est ouverte pour cette tâche. Cela lui permet d'avoir ses réglages indépendants des autres. La station est fermée en fin d'exécution ou par **thkill** ou **thkillall**.

thid

renvoie l'identifiant du thread en cours. Dans le mode direct, la valeur zéro est renvoyée. De ce fait, il est conseillé de ne pas donner zéro comme identifiant à un thread.

thrun

quitte le mode direct et lance l'exécution multitâche des threads définis.

On revient au mode direct de deux manières:

- ✔ soit tous les threads se sont terminés
- ✔ soit l'utilisateur a provoqué un arrêt avec Control+Alt+Shift (un seul Shift)

Si c'est l'utilisateur qui a provoqué la pause, l'exécution de thrun relance toutes les tâches là où elles s'étaient arrêtées.

thnext

à l'intérieur d'un thread, passe la main au suivant.

thpause

id thpause : met en pause le thread dont l'identifiant est fourni. Un thread peut se mettre en pause lui-même avec:

```
thid thpause
```

thwake

id thwake : réveille le thread dont l'identifiant est fourni. S'il n'était pas en pause, l'instruction n'a aucun effet.

thkill

id thkill : efface de la mémoire et de la liste des tâches de thread dont l'identifiant est fourni. Sa pile est toujours accessible et sa page existe encore.

thkillall

efface tous les threads. Au cas où l'instruction est appelée par un thread, on revient au mode direct.

thstat

affiche la liste des threads en cours avec leur identifiant ainsi que leur état.

3 La synchronisation

Le système de synchronisation est inspiré de celui des Transputers. Si plusieurs tâches ont besoin de se synchroniser afin d'échanger des données à un certain point :

1. on choisit un canal de synchronisation (il y en a quatre)
2. on indique le nombre de tâches qui doivent s'attendre avec **thnsync**
3. chaque tâche qui a fini sa partie et attend les autres appelle **thsync**
4. l'une d'elle doit appeler **thsyncmain**

Dès que toutes sont arrivées à leur point d'attente, le système les remet toutes en mode d'exécution et lance en priorité celle qui a appelé **thsyncmain**. Cette dernière doit se charger de récolter les données, préparer le travail suivant des autres tâches. Tout doit être à nouveau prêt lorsqu'elle appellera **thnext**.

thnsync

n c thnsync : indique que n threads doivent s'attendre sur le canal de synchronisation c (de 0 à 3).

thsync

c thsync : passe le thread courant en pause jusqu'à ce que le canal de synchronisation c indique que tous les threads sont bien arrivés.

thsyncmain

c thsyncmain : passe le thread courant en pause jusqu'à ce que le canal de synchronisation c indique que tous les threads sont bien arrivés. Un seul thread doit appeler cette fonction !

À ce moment-là, c'est le thread ayant appelé **thsyncmain** qui reprend la main avant les autres. Il est chargé de récupérer des données et de préparer la suite sachant que les autres threads vont repartir.

4 Les variables locales (ou presque)

Par défaut, les variables sont globales, c'est à dire qu'un nom de variable correspond toujours à la même cellule mémoire et donc à la même valeur.

Imaginons que plusieurs threads utilisent ce mot qui renvoie le niveau de gris d'un pixel à l'écran aux coordonnées X et Y:

```
: gris
      X @ Y @ v_get_pixel          \ renvoie l'index de couleur en X,Y
      intout w@ 1 vq_color         \ revoie la composition de cette couleur
      intout 2+  w@ 3 *            \ R*3
      intout 4 + w@ 6 * +          \ V*6
      intout 6 + w@ +              \ B
      10 /                         \ taux de gris =(3R+6V+B)/10
;
```

Si chaque thread travaille à des endroits différents, les valeurs de X et Y vont se télescoper.

On crée donc de nouvelles variables qui sont en fait des tableaux indexés selon l'ID du thread. Il va de soi que des ID consécutifs sont recommandés pour limiter la taille du tableau.

Ainsi, le thread ID=1 utilisera les valeurs d'indice 1 de chaque tableau, ceci de manière automatique. Le mode direct utilisera les valeurs d'indice zéro du tableau.

Si notre programme exécute deux threads et le mode direct, il nous faut trois valeurs sur 4 octets chacune, on définira X et Y ainsi:

```
3 4 thvariable X
3 4 thvariable Y
```

Le mot 'gris' défini plus haut fonctionnera avec des X, Y différents pour chaque thread.

thvariable

n t thvariable XXX : crée une variable XXX pour n threads (de 0 à n-1) et sur t octets chacune.

Si on désire des bytes (accédés par c@ et c!):

```
n 1 thvariable OCTET
```

Des words (accédés par w@ et w!):

```
n 2 thvariable MOTS
```

Des nombres réels (accédés par f@ et f!):

```
n 8 thvariable REELS
```

On peut également créer des chaînes de caractères. Il ne faudra pas oublier de compter le zéro final et les deux octets servant à la taille et d'aligner ceci sur une valeur paire.

Par exemple, pour des chaînes de 20 octets:

20 + zéro terminal + word taille = 23, donc 24 octets en alignant.

```
n 24 thvariable CHAINES
```

L'extension M_PLAYER / MP_STE

Cette extension permet de communiquer avec les logiciels M_PLAYER et MP_STE pour rejouer ou créer des vidéos. Le player doit être installé en tant qu'accessoire.

1 La lecture de vidéos

vd_set

" Nom" vd_set définit le nom du player (sur 8 caractères complétés par des espaces si nécessaire). Renvoie 0 si erreur ou l'identifiant AES du player si ok. Exemples:

```
" M_PLAYER" vd_set .    \ cherche la présence de M_PLAYER.ACC
" MP_STE  " vd_set .    \ idem avec MP_STE.ACC.
```

vd_info

" chemin\fichier" vd_info

renvoie l'adresse d'un bloc d'informations sur le fichier vidéo :

```
adr         WORD status       1=Ok
                              0=Fichier non trouvé (infos vides)
                              -1:type inconnu (seul le nom est valide)
                              -2:Erreur, M_PLAYER ne s'est pas lancé...
adr+2       CHAR file_name    (14 octets: max 12 + 0 + adresse paire)
adr+16      CHAR file_type    (28 octets: max 26 + 0 + adresse paire)
                               exemple 'Video for Windows (AVI)'
adr+44      WORD graph_status
                              1: supporté
                              0: pas d'images (infos suivantes vides)
                              -1:non supporté
adr+46      WORD width        largeur
adr+48      WORD height       hauteur
adr+50      LONG nombre d'images
adr+54      LONG compression  (4 octets comme 'cram', 'cvid'...)
adr+58      WORD sound_status
                              1: supporté
                              0: pas de son (infos suivantes vides)
                              -1: non supporté
adr+60      WORD sound bits   (le plus souvent 8 ou 16)
adr+62      WORD channels     (1:mono, 2:stéreo)
adr+64      LONG frequency    en Hertz
adr+68      LONG version      4 caractères représentant la version du
                              player, par exemple '4.28'
```

vd_play

" chemin\fichier" " options" vd_play

Lance la lecture du fichier vidéo spécifié. Les options sont:

+d/-d autorise l'affichage des dialogues ou non, défaut +d.

Toutes les autres options ne fonctionnent qu'en mode "-d", car lorsque les dialogues sont présents, ils prennent le pas sur les options de la ligne de commande.

+p/-p rejoue le son ou non, défaut +p.

+s/-s synchronise ou non les images avec les informations de temps, défaut +s.

+a/-a pour un fichier BAT, valide la création de l'animation, sinon c'est un simple slideshow des images. Pour une animation autre, valide l'entrée dans le mode de désassemblage d'animation. Défaut -a.

+e/-e affiche ou non les messages d'erreur, défaut -e

+i/-i valide ou non le mode répétition, défaut -i.

+xn/+yn force les coordonnées pour l'affichage. Si un changement de résolution est nécessaire, ceci est ignoré. Sur un TT en TT-Low, alignement de x sur 16. Par défaut, l'animation est centrée.

En cas d'**utilisation de ANIPLAY**, les options sont légèrement différentes:

+d/-d affiche ou non l'interface GEM, défaut +d.

+i/-i valide ou non le mode répétition, défaut +i.

+p/-p rejoue le son ou non, défaut +p.

+s/-s autorise le saut d'images ou non pour la synchronisation, défaut +s.

+xn/+yn comme pour M_PLAYER: position (centrée par défaut)

Contrairement à M_PLAYER, les réglages sont conservés d'un appel à l'autre.

2 La création de vidéos

Il y a deux manières de créer une animation avec M_PLAYER/MP_STE.

• Soit vous disposez déjà un jeu d'images, éventuellement un son, il vous suffit de vous reporter à la documentation du player pour fabriquer votre fichier BAT correspondant.
Ensuite, l'appel à vd_play est suffisant avec l'option +a:

```
" fichier BAT"   "  -d+a"  vd_play
```

Et votre vidéo sera créée.

• Soit vous voulez générer les images une par une et l'animation se monte en parallèle, le player appelle votre routine de génération pour chaque image et l'assemble à la volée. C'est ce que propose **vd_create**.

vd_create

'gen z " out" b w h f k q t (ou " son") vd_create
Lance la création d'une animation *(Sur **MP_STE** les possibilités sont réduites **(*)**)*.

gen nom d'un mot FORTH assemblé qui génère chaque image en 16 bits.
Il reçoit en entrée le numéro d'image désiré (de 0 à f-1)
Il doit renvoyer deux paramètres en sortie avec la fonction **vd_frame**.

z **(*)** facteur de zoom, trois possibilités:
%h0101 pas de zoom
%h0201 image finale doublée
%h0102 image finale réduite de moitié
En ajoutant %h8000 on autorise l'affichage de la boîte de progression pendant la création. Avec par exemple %h8102 on divise par deux l'image et on affiche la boîte.

" out" chemin et nom du fichier de sortie à générer.

b **(*)** nombre de bits et format des images de la vidéo:
- 8 8 bits, l'image sera tramée en 256 couleurs pour un MOV RLE8.
- 16 16 bits, pour un MOV RLE16.
- -8 8 bits, l'image sera tramée en 256 couleurs pour un AVI RLE8.
- -16 16 bits, pour un AVI CRAM16.
- 4 4 bits, vous envoyez une image au format DEGAS 320x200 16 couleurs pour un FLM 16 couleurs.
- -4 4 bits, vous envoyez une image au format NEOchrome 320x200 16 couleurs pour un FLM 16 couleurs.
- 1 monochrome, vous envoyez une image DEGAS 640x400 pour un FLM mono.

w,h largeur et hauteur de l'image source. Assurez-vous que la largeur finale (après un éventuel zoom) est multiple de 4.

f nombre total d'images.

k fréquence des key-frames, images reconstruisant entièrement l'écran et permettant de se resynchroniser en cas de replay trop lent (si k=5, les images 0,5,10,... seront des key_frames)

q qualité de la compression (de 1, mauvaise à 5, excellente).

t soit un temps en 1/200 de seconde pour chaque image (il doit être < 65536, c'est à dire limité à 5 min 27 sec par image), soit c'est une chaîne donnant le chemin et nom d'un son AVR ou WAV sur lequel se fera la synchronisation.

Le son doit être:
- 8 ou 16 bits non compressé
- mono ou stéréo
- fréquence proche des standards AVI: 11,025 ou 22,05 ou 44,1 kHz.

(*) **Sur MP_STE**, le zoom n'est pas disponible, seul le flag $8000 est lu. De même, les modes b=+/-8 et b=-16 ne sont pas disponibles.

Sur M_PLAYER, le zoom n'est disponible que pour la création MOV/AVI, il est ignoré pour les FLM qui ont une taille fixe. Mettre %h0101 dans ce cas.

vd_frame

buffer format vd_frame : dans la routine gen, renvoie l'adresse du buffer contenant l'image et en précise le format:

```
0     : pixels au format NOVA 16 bits     vvvbbbbb xrrrrrvv
-1    : pixels au format Falcon 16 bits   rrrrrvvv vvvbbbbb
4     : image format DEGAS (2 octets nuls, 32 octets palette XBIOS,
        32000 octets image écran 320×200×16).
-4    : image format NEO (4 octets nuls, 32 octets palette XBIOS,
        92 octets nuls, 32000 octets image écran 320×200×16).
1     : image format DEGAS (2 octets 0 et 2, 32 octets palette XBIOS,
        32000 octets image écran 640×400×2). Dans la pratique, la palette
        est inutile ici car remplacée par blanc/noir dans M_PLAYER.
```

Note : Pour envoyer la palette courante dans l'image on pourra utiliser **savevdipal** et **pal2xbios**. Exemple en 16 couleurs:

```
32034 allot constant DEGAS    \ l'image complète
  192 allot constant PALv     \ la palette VDI 16×6 octets

PALv savevdipal               \ la palette courante sauvée
PALv DEGAS 2+ 16 pal2xbios    \ transformée en XBIOS dans l'image
```

Précisions sur la routine gen:

Pour une image de 120×96 en 16 bits, on a besoin d'une zone de 120×96×2 octets:

```
23040 allot constant ZONE

:: gen                  \ le mot doit être assemblé !
      drop              \ on reçoit le numéro d'image, drop si inutilisé
      ....              \ ici on doit remplir la ZONE de pixels de 16 bits
      ZONE -1 vd_frame  \ renvoie l'adresse et le format
;

'gen %h8101 " C:\OUT.AVI" -8 120 96 50 10 5 20 vd_create
```

Ceci créera une animation AVI en RLE8, de 120×96 avec 50 images en qualité max et une vitesse de 20/200 sec par image, c'est à dire 10 images par seconde.

J'ai choisi k=10 ainsi on peut se resynchroniser à chaque seconde au cas ou le replay soit trop lent.

L'extension SuperCharger

Cette extension permet de communiquer avec le Superchager. Ce dernier est une carte PC branchée sur le port ACSI équipée d'un NEC V30 à 8MHz épaulé d'une mémoire vive allant jusqu'à 1 Mo.

1 La Tool Box

Afin de pouvoir communiquer, des routines spéciales doivent être chargées en mémoire et accessibles par le Trap #4.

Soit vous utilisez le programme résident SCTB.PRG fourni avec le Supercharger, soit on utilisera TOOL_BOX.BIN qu'on peut charger au moment de l'exécution du programme FORTH.

Tout périphérique du port ACSI se voit assigner un numéro de 0 à 7. Par défaut, c'est le DMA 3 qui est attribué au Supercharger. Si vous avez une configuration spéciale, vous devez modifier cette valeur avant tout appel aux autres fonctions:

sc_dma

n sc_dma : ce mot fixe à n le numéro de périphérique

sc_check

sc_check : teste la présence de la ToolBox.
Renvoie:
- un code d'erreur négatif si les routines ne sont pas présentes
- zéro et le numéro de version si les routines sont présentes
  la version est un long représentant 4 caractères en ASCII.

Codes d'erreur:

0	Ok
-1	routines ToolBox absentes
-2	(réservé)
-3	Supercharger non connecté
-4	Time Out
-5	Erreur de protocole
-6	Hotkey ABIO (émulation PC en cours!)
-7	(réservé)
-8	appel de fonction illégal
-12	fichier déjà chargé (avec sc_load)

Si sc_check écoue, vous devez charger les routines à l'aide de la fonction suivante:

sc_load

" chemin\TOOL_BOX.BIN" sc_load
charge et installe les routines sur le Trap #4

Renvoie zéro ou un code d'erreur. Celui-ci peut être inférieur à -32:
0 tout est ok
-33 fichier non trouvé
-39 mémoire insuffisante (il faut 18000 octets de ST-RAM)

sc_adr

sc_adr : renvoie sur la pile l'adresse où est chargée la ToolBox par sc_-load. Si l'appel n'a pas été exécuté, ou après sc_unload, zéro est renvoyé

sc_status

sc_status : renvoie l'état du Supercharger

0 si la Tool_Box est déjà en communication.
-6 ABIO Hotkey (MS Dos en exécution)

Toute autre valeur signifie un état indéfini.

sc_unload

sc_unload : désinstalle les routines de communication et libère le buffer en ST-RAM.
Renvoie un code d'erreur ou zéro.
Si une erreur s'est produite, le bloc mémoire n'est pas libéré.
Note: il faut absolument utiliser cet appel avant de quitter le programme FORTH sous peine de faire pointer le Trap #4 sur une zone ne contenant plus les routines.

II Le mode Esclave

Dans ce mode, c'est l'Atari qui a la main et qui pilote le Supercharger. C'est le premier mode dans lequel on doit basculer avec **sc_boot** afin de préparer le dialogue.

Adresses **côté PC**:

Le NEC V30 utilise deux registres 16 bits pour représenter une adresse.
Un registre de segment (bloc de 16 octets) de %h0000 à %hFFFF
Un registre d'offset à ajouter au segment de %h0000 à %hFFFF.
Ainsi, %h0200:045A représente l'adresse:
%h0200 x 16 + %h045A = %h0245A.
Côté Atari, on utilisera un mot long formé du segment et de l'offset.

La carte mémoire s'organise ainsi:

Supercharger 512 k

```
0000:0000 à 0000:2FFF   12 ko pour la ToolBox
0000:3000 à 6000:7FFF   404 ko libres
B000:8000 à B000:FFFF   32 ko libres
F000:0000 à F000:FFFF   64 ko libres
```

Supercharger 1 Mo

```
0000:0000 à 0000:2FFF   12 ko pour la ToolBox
0000:3000 à F000:FFFF   1012 ko libres
```

sc_boot

sc_boot : bascule en mode esclave.
Renvoie un code d'erreur ou zéro si Ok.

Dans ce mode, trois commandes sont possibles:

sc_sendm

atarisrc pcdest size sc_sendm : envoie size octets depuis l'adresse Atari atarisrc vers l'adresse PC pcdest.

Attention:
- l'adresse Atari doit se trouver en ST RAM !
- size est limité à 65535 octets
- le transfert est optimisé si les adresses sont paires et si size est un multiple de 512 octets.

sc_getm

pcsrc ataridest size sc_getm : reçoit size octets depuis l'adresse PC pcsrc vers l'adresse Atari ataridest.

Mêmes limitations que pour sc_sendm.

sc_exec

pcadr sc_exec : lance l'exécution d'un programme côté PC à l'adresse pcadr et bascule en mode coopératif.

Seuls les registres CS et IP seront initialisés pour le NEC V30:
CS = 16 bits de poids fort de pcadr
IP = 16 bits de poids faible de pcadr

A partir de là, Atari et PC doivent collaborer. C'est le PC qui décide de revenir en mode esclave.

III Le mode collaboratif

Dans ce mode, un programme s'exécute côté PC et chaque appel de l'Atari via le Trap #4 doit correspondre à un appel côté PC via Int 60h.

Au retour de ces appels, on peut recevoir régulièrement l'erreur -4 qui signale un Timeout. Cela vient simplement du fait que le NecV30 exécute une partie de son programme et ne répond pas immédiatement à une demande de transfert.

On pourra utiliser par exemple:

```
begin
            appel à une des
            quatre instructions de transfert
            dup -4 =
while
            drop
repeat
```

On ne sort de la boucle que si il n'y a pas de Timeout. Renvoie sur la pile 0 (tout est ok) ou un autre code d'erreur.

sc_xsendb

n sc_xsendb : envoie un octet n au PC.

Côté PC:

```
mov ah,9          fonction get byte
int 60h           au retour al = octet reçu
```

sc_xgetb

sc_xgetb : reçoit un octet du PC.

Côté PC:

```
mov al,octet           valeur à envoyer dans al
mov ah,10         fonction send byte
int 60h
```

sc_xsendm

atarisrc size sc_xsendm : envoie size octets depuis l'adresse atarisrc au PC.

Côté PC:

```
                       es:di adresse destination
mov cx,size       nombre d'octets à recevoir
mov ah,4          fonction Receive Data
int 60h
```

sc_xgetm

ataridst size sc_xsendm : reçoit size octets du PC à l'adresse ataridst.

Côté PC:

```
                       ds:si adresse source
mov cx,size       nombre d'octets à envoyer
mov ah,5          fonction Send Data
int 60h
```

Il existe **deux dernières fonctions** côté PC :

La première pour **revenir au mode esclave** tout en conservant la ToolBox:

```
mov ah,0          fonction terminate
int 60h           retour au mode esclave
```

La seconde pour **effacer complètement la ToolBox** de la mémoire PC. A la suite de cette dernière, l'Atari doit refaire un appal à sc_boot si il désire communiquer à nouveau:

```
mov ah,1          fonction kill tool box
int 60h           mode indéfini
```

IV Echange de données

Le NEC V30 stocke ses données en Little Endian. On utilisera les instructions suivantes pour communiquer lors d'échanges de données utilisant plusieurs octets:

iw! i! if!

Ces trois instructions sont les pendants de w!, ! et f! et stockent les données (mot, long, float) au format Intel à l'adresse indiquée.

Exemple:

```
%f2.36 adr if!          \ stocke un float Intel
adr 8 sc_xsendm .       \ et l'envoie au programme
```

iw@ i@ if@

Ces trois instructions sont les pendants de w@, @ et f@ et récupèrent les données (mot, long, float) au format Intel vers le format Atari.

Exemple:

```
adr 8 sc_xgetm    .     \ récupère 8 octets Intel
adr if@ f.              \ et l'affiche comme un float
```

Les modules M&E de Parx

Ce jeu d'instructions facilite l'utilisation des modules M&E définis par Parx Systems. Ce sont des modules d'importation/exportation d'images et des modules d'effets. A ceux-là s'ajoutent des modules audio que j'ai définis en respectant le modèle Parx.

On doit avant tout indiquer le chemin du dossier des modules, celui-ci suit l'organisation suivante:

Dossier PARX.SYS

\RIM	sous-dossier modules RIM (lecture image)
\WIM	sous-dossier modules WIM (sauvegarde)
\IFX	sous-dossier modules IFX (effets)
\RSN	sous-dossier modules RSN (lecture audio)
\FORTH	sous-dossier modules FOM (Forth Module)
PARX.TRM	module de tramage.
PARX.MEM	module de gestion mémoire.
PARX.Mnn	module MOT (motif sur nn plans)
PARX.Bnn	module BRO (brosse sur nn plans)
PARX.Pnn	module PAL (palette sur nn plans)

Le FORTH n'utilise pas le module mémoire mais va profiter de son organisation du dictionnaire comme une pile de blocs qu'on pourra réserver, étendre et libérer (voir **here**, **remember**, **restore**) au cours de l'utilisation des RSN, RIM, WIM et IFX.

Pour ce qui est du module même, il pourra être chargé dans le dictionnaire ou dans un bloc externe par malloc.

Gestion des modules

modset

" chemin" flag modset : définit les options générales.
flag : si le bit0 est à 1, alors les fonctions dorim/dowim/doifx et dotrm affichent l'abeille comme curseur de souris pendant leur travail.
Le chemin du dossier PARX, il doit se terminer par "\". Exemple:

```
" D:\PARX.SYS\" 1 modset
```

Ne renvoie pas de valeur.

modload

Charge un module et l'affecte à une variable.
Deux syntaxes possibles:

- " nom" flag var modload : charge un module.
  var : une variable qui restera associée au module, en entrée:
  var = 0 allouer un bloc avec malloc et y charger le module
  var = -1 charger le module dans le dictionnaire
  nom : chemin du module dans le dossier PARX., exemple: " WIM\XIMG.WIM".
  Pour ne pas ajouter le chemin du dossier PARX, c'est à dire si le nom est déjà complet, il suffit d'ajouter 128 au premier caractère de la chaîne.
- flag var modload
  var = adr le module est déjà en mémoire à l'adresse contenue dans var, attention cependant: cette adresse doit être précédée de 32 octets libres pour des variables temporaires.

flag : si le module est configurable, flag indique l'attitude à suivre:
flag = -1 ignorer la configuration
flag = autre lancer la configuration avec code = flag, par défaut le code doit être égal à 1, mais certains modules acceptent d'autres valeurs (exemple JPEG_68K.RIM avec 1 il affiche un copyright, avec 0 il se contente de reloger le programme en mémoire.)

Sur les 32 octets qui le précèdent, le module reçoit des informations qu'il pourra sauvegarder si nécessaire lors de son initialisation. Voir **Gestion des modules FORTH additionnels** pour plus de détails sur cette zone.

Dans le cas du TRM, il faut absolument lancer la configuration, flag=-1 est interdit, c'est une valeur réservée! (voir ***Carte Graphique et TRM 2.52***). Pour le cas courant:
flag : 0 pour un mode vidéo Atari standard
: 1 pour une carte graphique (celle réglée par EDIT_TRM)
Cette valeur peut être obtenue par la fonction **graphic_card**.

Retour : pour les cas var=0 ou -1, remplit la variable avec l'adresse du module. Cette variable sera nécessaire pour chacun des appels suivants ! A chaque module sa variable, vous pouvez gérer plusieurs modules à la fois, éventuellement dans un tableau.

Renvoie :

un code d'erreur GEMDOS si le chargement ne s'est pas déroulé correctement
0 : si tout s'est bien passé.
-1 : erreur, la configuration a échoué
-2 : le module ne supporte pas l'appel à la configuration
-3 : si le module est de type inconnu

modunload

var modunload S'il s'agit du TRM, la fonction TRM_END est appelée et uniquement dans le cas d'un module chargé avec var=0, la fonction libère le bloc mémoire et remet var à zéro. Pas de valeur renvoyée.

modmload

" masque" flag élément-tableau modmload

Charge tous les modules correspondant au masque et loge les adresses dans chaque élément du tableau à partir de celui indiqué. Renvoie le nombre de modules chargés.

Si un module ne se charge pas ou est de type inconnu, la mémoire n'est pas allouée et le module n'apparaît pas dans la liste.

Au départ, le premier élément du tableau contient 0 ou -1:
0: tous les modules seront chargés dans une zone allouée par malloc
-1: tous les modules seront chargés dans le dictionnaire.
(Dans la pratique, si la valeur n'est pas zéro, elle sera considérée comme -1).

Exemple:

```
100 array MODULES                    \ tableau de 100 éléments
0 0 MODULE !                         \ MODULES(0)=0, charger avec malloc
" rim\*.rim" 1 0 MODULES modmload     \ charge tous les RIM
constant NOMBRE                      \ nombre de modules chargés
```

modmunload

élément-tableau n modmunload

Efface n modules pointés par le tableau à partir de l'élément fourni. C'est la fonction réciproque de modmload. Pour effacer tous les modules de l'exemple précédent:

```
0 MODULES NOMBRE modmunload
```

modfmt

var modfmt

- pour un RIM/WIM/IFX :renvoie le format d'image du module.
  - 0 shifter, codage d'un écran Atari (plans entrelacés jusqu'à 8 bits, rrrrrvvvvvvbbbbb pour le 16 bits, RVB pour le 24 bits et xRVB pour le 32 bits)
  - 1 VDI, format indépendant (plans séparés jusqu'à 8 bits)
  - 2 peut gérer les deux selon les cas.
- pour un RSN : renvoie le format des sons gérés
  - 3 son contenant uniquement des data
  - 4 son contenant des data et du code exécutable
- pour le TRM : renvoie le nombre de formats de tramage disponibles.
- pour les BRO, PAL, MOT : renvoie le nombre de plans.

modsname

var modsname : renvoie dans pad (sur la pile) le nom court du module.

modlname

Plusieurs possibilités:

- pour un RIM/WIM/IFX
  var modlname : renvoie dans pad (sur la pile) le nom long du module.

- pour le TRM
  i var modlname : renvoie dans pad sur la pile le nom du tramage n°i

- pour les BRO, PAL, MOT et MEM
  var modlname : renvoie le nom court, il n'y a pas de nom long.

- pour les FOM
  i var modlname : si i=0, renvoie le long nom du module FORTH
  si i=1,2,3 renvoie les redéfinitions de modinfo, modpal et modexe.
  Avec un FOM étendu (modtype=n), alors de i=3 jusqu'à i=3+(n-1), renvoie la redéfinition de chaque sous-fonction de modexe.

modver

var modver : renvoie la version du module (en décimal).

Note: à partir de la *version 210* du **TRM**, en plus des tramages disponibles (voir **modfmt**), on dispose d'un **zoom** qualité (*tramage=-1*) et d'un zoom simple (*tramage=0*).

modtype

var modtype : renvoie le type de module.

Pour un RIM:

0	demande à avoir tout le fichier en mémoire.
1	peut travailler avec des segments du fichier (pas réellement supporté ici).
2	RIM générant, crée l'image lui même (fractales, scanner...), les dimensions de l'image doivent être spécifiées.
3	RIM générant, mais les dimensions sont fixes.
4	RIM animation.

Pour un WIM:

- 0 sauvegarde une image sur disque.
- 2 WIM générant, envoie une image vers un périphérique.
- 4 WIM animation.

Pour un IFX:

- 0 effet sur une seule image source
- 1 effet sur deux images sources

Pour un RSN:

- bit0 0 demande à avoir tout le fichier en mémoire.
  - 1 peut travailler par buffers
- bit1 0 toute RAM
  - 1 ST Ram seulement

Pour un FOM:

- 0 module compatible avec le format PARX (trois appels modinfo, mod pal et modexe, éventuellement redéfinis, voir **modlname**).
- n module FORTH étendu, l'appel modexe contient n sous-fonctions qu'on devra obligatoirement appeler avec leurs redéfinitions (voir **>fom** et **modlname**).

Pour les autres:

- -1 c'est le TRM
- -2 le module MEM
- -3 une brosse BRO
- -4 une palette PAL
- -5 un motif MOT
- -6 un module PARX non répertorié.

Gestion d'un RIM, lecture d'une image

Une fois le RIM en mémoire, vous allez pouvoir vous en servir pour charger et décoder une image.

Vous avez deux méthodes, la première automatique fait appel à **dorim** et se charge de l'ensemble de la procédure, la seconde est manuelle et fait appel au triplet **modinfo**, **modpal** et **modexe**.

Méthode automatique :

dorim

adr taille "nom" handle pal flag var dorim

Vous devez au préalable charger le fichier dans un nouveau bloc du dictionnaire obtenu avec allot et laisser le fichier ouvert.

adr : adresse du bloc fichier
taille : taille des données chargées
nom : chaîne nom du fichier (ou simplement son extension)
handle : celui du fichier encore ouvert
pal : zone de 1536 octets pour recevoir une éventuelle palette VDI, zéro si vous êtes certain qu'elle sera inutilisée.
var : correspondant au module chargé avec modload

Dorim présente le fichier au module RIM et tente de le décoder, on obtient:

2 : Ok, fichier décompressé et données dans mfdbs, PAL contient la palette
3 : Ok, fichier décompressé, données dans mfdbs, pas de palette
-1 : erreur dans modinfo, fichier non pris en charge par ce module
-2 : erreur dans modpal
-3 : erreur dans modexe, fichier abîmé.

En cas de succès, le dictionnaire contient dans l'ordre:

1) le bloc avec le fichier
2) suivi d'un éventuel bloc temporaire additionnel
3) suivi du bloc avec les données décompressées

Typiquement, le fichier et le bloc temporaire ne sont plus nécessaires mais occupent de la mémoire. C'est là le rôle du flag:

flag : 0 laisser les trois blocs tels quels
: 1 effacer le fichier et le bloc temporaire et déplacer les données vers cette nouvelle zone libre. Le pointeur de fin de dictionnaire est ajusté.

Il vous appartient de fermer le fichier.

Méthode manuelle :

Pour cela il faudra:

1. remplir le **mfdbs** avec les données de l'écran (voir **fillmfdb**)
2. ouvrir le fichier image et le charger en mémoire, en le laissant ouvert
3. appeler **modinfo** pour vérifier qu'il est reconnu par le RIM
4. appeler **modpal** pour gérer la palette et un éventuel buffer
5. appeler **modexe** pour exécuter le décodage.
6. fermer le fichier image.

modinfo

adr taille " nom" handle var modinfo:

vérifier l'image avec:

adr	: adresse où est chargé le fichier
taille	: taille du fichier
" nom"	: nom du fichier, ici on peut ne mettre que son extension, c'est elle qui est vérifiée par le module.
handle	: handle du fichier renvoyé par fopen.

renvoie :

0	: image non reconnue
1	: données insuffisantes (impossible si on a chargé tout le fichier)
2	: Ok, contient une palette
3	: Ok, pas de palette.

Si le fichier est reconnu alors **mfdbs** est rempli ainsi :

mfdbs	: LONG offset vers les données palette (ou 0)
mfdbs+4/+6	: largeur et hauteur de l'image
mfdbs+8	: Largeur en paquets de 16 pixels
mfdbs+10	: format
mfdbs+12	: nombre de plans
mfdbs+14	: LONG taille de la palette

À ce stade, vous pouvez décider de continuer ou non le chargement avec **modpal** selon les informations données.

modpal

pal var modpal

charge éventuellement la palette et donne des informations sur le buffer de travail nécessaire:

pal : adresse d'une zone de 1536 octets recevant la palette au format VDI, ou zéro si il n'y a pas de palette.

renvoie :

0 : erreur

1 : données insuffisantes (impossible si on a chargé tout le fichier)

2 : Ok !

3 : Ok, avec dépassement de bloc. Dans ce cas, en dessous est renvoyée la taille supplémentaire nécessaire à la suite de l'image comme bloc de travail. Il peut être judicieux de charger le fichier image dans le dictionnaire et d'utiliser **allot** pour prolonger la zone en cas de besoin.

modexe

dest var modexe

lance le décodage de l'image.

dest : adresse d'un buffer assez grand pour accueillir les pixels.
À cet effet, on pourra utiliser

```
mfdbs imagesize
```

pour obtenir la taille nécessaire.

renvoie :

0 : erreur

1 : données insuffisantes (impossible si on a chargé tout le fichier)

3 : Ok !

À l'issue de cet appel:

mfdbs : correctement rempli.

pal : contient les données de la palette (si présente) au format VDI

dest : contient l'image décodée

Gestion d'un WIM, écriture d'une image

Une fois le WIM en mémoire, vous allez pouvoir vous en servir pour sauver une image.

Vous avez deux méthodes, la première automatique fait appel à **dowim** et se charge de l'ensemble de la procédure, la seconde est manuelle et fait appel au triplet **modinfo**, **modpal** et **modexe**.

Méthode automatique :

Il faut remplir correctement le **mfdbs** avec les données du bloc à sauver puis appeler dowim:

dowim

" nom-fichier" PAL flag var **dowim**

tente une sauvegarde du bloc défini par mfdbs et l'éventuelle palette sur disque.
flag : bit0 = 1 pour gérer automatiquement la transformation de format avec vr_trnfm, ceci intervient uniquement si la seule différence entre votre bloc et ce qu'est capable de gérer le WIM est le format.
bit1 = 1 pour ajouter ou modifier l'extension du fichier avec celle conseillée par le module. Sinon, conserver celle donnée.
var : le pointeur vers le module
PAL : adresse d'une palette si besoin

renvoie:
0 : ok, tout s'est bien déroulé, le fichier est sauvé.
-1 : les dimensions du bloc ne conviennent pas, utiliser le TRM
-2 : le format du bloc ne convient pas, utiliser vr_trnfm
-4 : le nombre de plans ne convient pas, utiliser le TRM
... -7 : toute combinaison de -1, -2 et -4.
-8 : erreur dans MODPAL, le format est incompatible
-9 : erreur dans MODEXE
autre : autre code négatif, c'est un code d'erreur GEMDOS lors de la création ou de l'écriture sur le fichier.

Effets collatéraux:
- mfdbd est utilisé
- pad contient l'extension conseillée ou le nom final du fichier modifié
- le dictionnaire a été utilisé pour des blocs temporaires

Méthode manuelle :

1. remplir correctement le **mfdbs** avec les données du bloc à sauver
2. appeler **modinfo** pour connaître les possibilités du module
3. appeler **modpal** pour fournir la palette et voir la compatibilité
4. créer le fichier et le laisser ouvert
5. appeler **modexe** pour réaliser la sauvegarde
6. fermer le fichier

modinfo

var modinfo

fournit des informations sur ses possibilités en liaison avec votre **mfdbs** préalablement rempli.

renvoie :
- 0 : pas d'export de palette
- 1 : export de palette
- 2 : export de palette possible

renvoie également l'extension souhaitée dans **pad**.

De plus, il peut modifier le **mfdbs** pour vous :
- l et h pourvent être modifiés si l'export est limité à certaines tailles.
- format peut être modifié si le module ne supporte que le shifter ou le VDI.
- nombre de plans peut être modifié selon le format d'image.

C'est à vous de décider si les modifications vous conviennent pour poursuivre.

modpal

pal var modpal

relit le mfdb et l'éventuelle palette.
pal : adresse d'une palette au format VDI ou zéro si unitilisée.

renvoie :
- -1 : format incompatible
- autre : c'est la taille du buffer de travail demandé, tout est Ok.

modexe

travail handle var modexe
lance la sauvegarde de l'image dans un fichier que vous venez de créer.
travail : adresse du buffer de travail dont la taille était fournie par **modpal**.
handle : handle du fichier fourni par **fcreate**.

renvoie :
-1 : erreur
autre : c'est la taille restant à sauvegarder depuis le buffer de travail. Ne pas réaliser de fseek, le pointeur fichier est correct.
Le plus souvent, c'est zéro car le WIM a déjà tout écrit.

À l'issue de cet appel, il vous reste à fermer le fichier avec **fclose** et à libérer les éventuels buffers.

Gestion d'un IFX, effet sur image

Pour ce type de module, seul l'appel à modexe est utilisé. L'effet appliqué à l'image peut prendre un ou deux appels consécutifs.

Si le module est de type 0, l'effet ne nécessite qu'une image décrite dans mfdbs avec éventuellement une sortie vers une image destination stockée dans mfdbd.

Si le module est de type 1, l'effet nécessite deux images décrites dans mfdbs et mfdbd ainsi qu'un mfdb supplémentaire comme image de sortie ou image temporaire.

IFX de type 0

C'est le module le plus courant, une seule image source est nécessaire.

Méthode automatique:

doifx

PAL1 PAL2 var doifx
gère les deux appels consécutifs et renvoie:
0 : ok, effet appliqué au premier appel
1 : ok, effet appliqué au 2e appel sans palette destination
2 : ok, effet appliqué au 2e appel, PAL2 contient la palette destination
-1 : erreur au premier appel
-2 : erreur au second appel

Méthode manuelle:

modexe

PAL1 0 var modexe
tente d'appliquer l'effet à l'image décrite en mfdbs avec la palette PAL1 (celui-ci peut être nul pour une image monochrome ou TC).

Retour:
0 : ok, effet appliqué! Pas de second appel nécessaire.
-1 : erreur, l'effet ne peut pas s'appliquer sur l'image décrite
1 : le module a besoin d'un second appel sans palette, mfdbd décrit la seconde image nécessaire dont on devra réserver le bloc et fournir l'adresse dans mfdbd.
2 : le module a besoin d'un second appel avec palette, mfdbd décrit la seconde image nécessaire dont on devra réserver le bloc et fournir l'adresse dans mfdbd ainsi qu'une adresse d'une zone suffisante pour la seconde palette.

Pour le second appel: PAL1 PAL2 var modexe
PAL2 peut-être nul en cas de retour 2 au premier appel.

Retour:
0 : ok, effet appliqué!
-1 : erreur.

IFX de type 1

C'est un type de module plus rare, deux images source sont nécessaires.

Méthode automatique:

doifx

PAL1 PAL2 mfdb3 PAL3 var doifx
gère les deux appels consécutifs et renvoie:
0 : ok, effet appliqué au premier appel
1 : ok, effet appliqué au 2e appel sans palette destination
2 : ok, effet appliqué au 2e appel, PAL3 contient la palette destination
-1 : erreur au premier appel
-2 : erreur au second appel

Méthode manuelle:

modexe

PAL1 PAL2 adr 0 var modexe
Tente d'appliquer l'effet aux images mfdbs/PAL1 et mfdbd/PAL2, adr est l'adresse d'un troisème mfdb (20 octets) éventuellement utilisé.

Retour:
-1 : erreur, l'effet ne s'applique pas à ces images
0 : ok, effect appliqué, pas d'autre appel à effectuer.
1 : le module a besoin d'un second appel sans palette, adr décrit la troisième image nécessaire dont on devra réserver le bloc et fournir l'adresse dans mfdbs (?).
2 : le module a besoin d'un second appel avec palette, adr décrit la troisième image nécessaire dont on devra réserver le bloc et fournir l'adresse dans mfdbs (?) ainsi qu'une adresse d'une zone suffisante pour la seconde palette.

Pour le second appel: PAL1 PAL2 adr PAL3 var modexe
PAL3 peut-être nul en cas de retour 2 au premier appel.

Retour:
0 : ok, effet appliqué!
-1 : erreur.

Gestion du TRM, module de tramage

Ce module permet les transformations des données d'une image de et vers tous les formats possibles. On peut modifier le nombre de plans, le format VDI ou SHIfter, tramer en monochrome, en gris, effectuer des zooms.
S'il s'agit simplement d'adapter le nombre de plans à la résolution actuelle, voici:

```
mdfbs mfdbd fillmfdb    \ copie la source vers la destination
work_out 2- w@          \ récupère le nombre de plans actuel
mfdbd 12 + w!           \ et le modifie dans le mfdb de destination
```

On dispose de deux modes, le mode automatique avec la fonction **dotrm** et le mode manuel avec le triplet **modinfo**, **modpal** et **modexe**.

Méthode automatique :

dotrm

%bPRCCCID t pal-source pal-dest flag var dotrm

Vous devez au préalable remplir le mfdbs avec les données de l'image source puis le mfdbd avec les modifications attendues (sans vous préoccuper de l'adresse des données destination, elle sera allouée pour vous).

%bPRCCCID	: options, voir le détail dans modinfo ci-dessous
t	: numéro du tramage (voir modfmt)
pal-source	
pal-dest	: voir le détail dans modpal ci-dessous
var	: la variable associée au module TRM

Dotrm présente les mfdb au module TRM et tente la transformation, on obtient:

0	: Ok, données transformées dans mfdbd
-1	: erreur dans modinfo, options incompatibles
-2	: erreur dans modpal, transformation non prise en charge
-3	: erreur dans modexe

Si la transformation est un succès, vous avez dans le dictionnaire:

1. le bloc des données source pointé par mfdbs
2. suivi du bloc des données transformées pointé par mfdbd

Il est possible que les données sources ne soient plus utiles, c'est le rôle du flag:

flag : bit0=0 laisser les blocs mémoire tels quels
: bit0=1 dans le cas où le bloc source est le dernier alloué dans le dictionnaire, la fonction recouvre la source (perdue) par la destination et réajuste le pointeur de fin de dictionnaire.
: bit1=1 forcer la sortie en mode SHIFTER, si le TRM renvoie un bloc au format VDI, alors un appel à vr_trnfm est réalisé. Cette transformation recouvre le bloc destination.

Méthode manuelle :

modinfo

%bPRCCCID t var modfinfo
Fixe les paramètres de tramage.
Les options sont un masque de bits

D, direction	: 0 en import, la source provient d'un RIM et on veut afficher
	: 1 en export, la source va être exportée avec un WIM.
I, interaction	: 0 mode automatique
	: 1 interactif
CCC, coul.	: 000 palette de destination imposée par le programme
	: 001 palette proposée mais pas forcément retenue par le TRM
	: 010 palette de gris (elle sera copiée pour vous)
	: 100 palette courante imposée (elle sera copiée pour vous)
	: 101 palette courante proposée (elle sera copiée pour vous)
R, recouvre	: 0 le TRM peut exécuter le tramage par dessus l'image d'origine
	: 1 force l'utilisation d'un second buffer
P, palette	: 1 utiliser une palette par défaut du TRM (dans le cas où ccc=0/1)
	: 0 prendre la palette proposée
t, tramage	: numéro du tramage de 1 à n (voir **modfmt**) ou -1 pour le **zoom qualité** et 0 pour le **zoom simple** *(zoom disponible à partir du TRM 210).*
Renvoie	: 0 si erreur ou 1 si les paramètres sont acceptés.

Attention: **imposer** une palette peut être **long**, en particulier sur 68000.

modpal

pal-source pal-dest var modpal

gère les palettes et renvoie la taille des blocs nécessaires.

mfdbs	: doit être rempli avec les informations de l'image
mfdbd	: rempli également avec le nombre de plans souhaités mais l'adresse des données égale à zéro.
pal-source	: adresse de la palette de l'image, zéro sinon (pour les images TC)
pal-dest	: adresse d'un bloc de 1536 octets maxi avec la palette imposée ou proposée. Selon les modes CCC (voir modinfo), c'est une palette déjà prête (modes 000 et 001) ou juste une zone ou sera copiée automatiquement par la fonction modpal soit la palette courante (modes 100 et 101) soit une palette de gris (mode 010).

En retour:

<0	: erreur
0	: Ok, pas de bloc à réserver
>0	: Ok, la valeur est la taille du bloc à réserver, si cette valeur est égale à %h7FFFFFFF, il faut réserver le bloc le plus grand possible et indiquer la taille obtenue dans les 4 premiers octets du bloc.

Il faut en plus regarder de nouveau l'adresse mise dans le mfdbd:

0	: alors on doit allouer une zone pour recevoir l'image destination dont la taille se calcule avec: `mfdbd imagesize` et remplir l'adresse du mfdbd avec le début de ce bloc.
=mfdbs	: égale à l'adresse dans mfdbs, cela signifie que le TRM va écraser l'image source pour sa transformation.

modexe

buffer var modexe
exécute le tramage.

mfdbd	: doit être à jour avec les demandes de modpal.
buffer	: adresse du buffer demandé par modpal.

Renvoie:

<0	: erreur
0	: succès.

En retour, les données du tramage sont remplies et pal-dest contient la palette retenue pour cette nouvelle image. Ce n'est pas forcément la palette proposée, le TRM peut avoir adapté les couleurs pour ses besoins.

Carte Graphique et TRM 2.52

Ce TRM est configurable et vous permet d'éditer vos propres réglages pour les modes 15, 16, 24 et 32 bits. Mais, vous pouvez laisser le FORTH le faire pour vous! Dans ce cas, il faut que le TRM soit réglé sur la carte FORTHGFX et:

```
" nom"  -1   var  modload
```

En indiquant -1 comme flag, le FORTH utilisera la valeur de **screen_info** pour régler correctement le TRM sur l'écran actuel. Le plus simple est d'utiliser:

```
" nom"  graphic_card negate   var  modload
```

Ainsi, si **graphic_card** renvoie zéro, cela n'est pas changé et si il renvoie 1, vous obtenez bien le flag -1. Mettre -1 sur un autre TRM ou carte est équivalent à 1.

Gestion d'un RSN, lecture d'un son

Une fois le RSN en mémoire, vous allez pouvoir vous en servir pour charger et jouer un fichier audio. Le dialogue avec le module se fait en partie sur la pile et en partie à travers un bloc appelé sndb (sound buffer).

sndb

Dépose l'adresse du bloc son, de 128 octets, organisé ainsi:

0	LONG	identifiant du fichier supporté ("SNDH", "WAVE",...) ou zéro si le fichier n'est pas supporté.
4	LONG	taille du fichier source
8	LONG	taille du fichier décompressé (=taille si pas de compression)
12	LONG	handle du fichier
16	LONG	adresse du fichier chargé
20	LONG	nombre de pistes audio dans le fichier
24	LONG	n° de la piste en cours ou zéro si arrêté. bit 15 mis si morceau en pause.
28	LONG	MIDI masque ON/MUTE 32 pistes. Tout à ON après modpal.
32	LONG	MIDI paramètre 1 avec le code -2 dans modexe
36	LONG	MIDI paramètre 2 avec le code -2 dans modexe
40	WORD	MIDI vitesse de lecture (100 par défaut)
42	WORD	MIDI masque ON/MUTE des 16 channels (tous à ON modpal)
44...		non définis encore
60	STRING	titre du morceau (31 caractères max+zéro)
92	STRING	auteur du morceau (31 caractères max+zéro)
124	BYTE	libre
125	BYTE	libre
126	BYTE	utilisé par FORTH en interne
127	BYTE	utilisé par FORTH en interne

Ce bloc est rempli par le module de lecture, l'utilisateur peut le consulter.

modinfo

handle var **modinfo**
vérifie que le module prend bien en charge le fichier concerné.
handle: celui du fichier ouvert par l'utilisateur
var: le pointeur vers le module

Dans **sndb** le handle est écrit, la fonction remplit également la taille du fichier, la taille décompressée ainsi que son type:

Retour:
0 fichier inconnu, dans ce cas TYPE = 0
1 fichier pris en charge mais compressé
2 fichier pris en charge, non compressé

modpal

Note : *Bien qu'il ne s'agisse pas de palette, on a gardé le nom de la seconde fonction telle qu'elle était définie pour les modules images.*

adrs adrd var **modpal**
charge le fichier son et gère les différents buffers.
adrs adresse source où on le module chargera le fichier
si adrs=-1 ignorer ce bloc
si adrs=0 laisser FORTH gérer dans le dictionnaire
adrd adresse destination ou le fichier sera décompressé (ou adapté)
si adrd=-1 ignorer ce bloc
si adrd=0 laisser FORTH gérer dans le dictionnaire
var pointeur vers le module

L'utilisateur peut, au retour de modinfo, définir lui-même les buffers source et destination, le plus simple est de laisser le FORTH gérer avec 0 0 var modpal.

Le module remplit dans **sndb** le titre, l'auteur, le nombre de morceaux et l'adresse du fichier.

Retour:
0 erreur
1 fichier chargé et décompressé, si adrs=0, FORTH libère le bloc source.
2 fichier chargé
-2 fichier MIDI chargé mais plus de pistes que le module n'en gère (>32)
(dans ce cas en mettant 32 dans le WORD en fichier+10 on peut utiliser le module MIDI sur les 32 premières pistes).

modexe

n var **modexe**
rejoue la piste n du fichier défini dans le sndb.
n numéro de la piste (de 1 à nombre de pistes)
var pointeur vers le module

Cas particuliers:
n = 0 arrêter le son
n = -1 renvoyer le numéro de la piste en cours
n =%h8000 bascule entre play et pause. La valeur renvoyée a son bit 15 mis lors d'une pause. Exemple, 1 pour la chanson 1 en cours et %h8001 pour la chanson 1 en pause.

Valeur retournée:
-2 erreur, le fichier n'est pas reconnu (cela peut arriver après décompression)
-1 le matériel pour rejouer le son est absent
0 le son est arrêté
n>0 piste en cours

Dans le cas du **module MIDI**, d'autres options sont disponibles :

`n flag -2 p modexe`
avec 0 ≤ n < 32 met la piste n à ON (flag=1) ou à MUTE (flag = 0)
`n flag -2 p modexe`
avec 32 ≤ n < 48 met le canal MIDI n-32 à ON (flag=1)
ou à MUTE (flag = 0)

`%h100 s -2 p modexe`
fixe la vitesse de lecture à s (avec une valeur par défaut de 100, fixée par modpal)

`%h101 flag -2 p modexe`
autorise (flag=1) les messages sys_ex ou non (flag=0).

`%h102 flag -2 p modexe`
autorise (flag=1) les messages Pressure ou non (flag=0).

`%h103 flag -2 p modexe`
autorise (flag=1) les messages Program Change ou non (flag=0).

Dans le **module XLR8**, une option est disponible :

`%h102 flag -2 p modexe`

force (flag=1) le replay en 50Hz ou non (flag=0), dans ce dernier cas la fréquence dépend du VBL (50, 60 ou 70Hz).

Gestion des modules FORTH additionnels

Ces modules, bien qu'ils utilisent aussi le protocole PARX M&E, ne sont pas utilisables en dehors du FORTH, car ils sont intimement liés à lui. Du point de vue du FORTH, les trois appels **modinfo**, **modpal** et **modexe** transmettent de manière interne l'adresse du **sndb** ou du **mfdbs** aux routines, ça peut être une zone utilisée pour l'échange de données ou ignorée par le module.

Par ailleurs, lors de **modload**, le module reçoit sur les 32 octets qui précèdent son adresse adr :

- adr-4 : tableau des adresses des mots FORTH. Il pourra ainsi appeler n'importe quelle instruction en dialoguant par la pile de données pointée par le registre CPU A6.
- adr-6 : la handle VDI en cours
- adr-8 : le handle VDI interne géré par le FORTH
- adr-12 : l'adresse du tableau aespb (qui donne accès aux autres tableaux de l'AES)
- adr-16 : l'adresse du tableau vdipb (qui donne accès aux autres tableaux du VDI)
- adr-20 : l'adresse de la variable word contenant le code du langage
- adr-24 : adresse d'un bloc de données supplémentaires :
  - 0 : adresse de la liste des mots FORTH (byte taille + mot, fin avec taille=0)
  - 4 : nombre de mots FORTH de base
  - 8 : nombre de mots du début du dictionnaire créant d'autres
  - 12 : flag VDI (byte) FF mode relative, 00 mode absolute
  - 16 : version du FORTH (un LONG de la forme%h00xxyyzz pour x.y.z)
  - 20: adresse du début du bloc dictionnaire (zone de travail)
  - 24: taille du dictionnaire
  - 28: adresse des 16 flags Forth (16 octets FF=true, 00=false)
  - 32: adresse variable word contenant le nombre total de mots (base+user)

Les valeurs utiles au module sont à sauvegarder lors de la routine d'initialisation.

C'est donc chaque module qui peut définir l'usage des fonctions **modinfo**, **modpal** et **modexe** et spécifier si elles doivent ou non passer des paramètres sur la pile ou à travers sndb, et si elles en reçoivent en retour sur la pile ou à travers sndb.

Dernière possibilité, un module peut proposer des noms de mots pour redéfinir les appels standards, ceci se réalise grâce à la directive **>fom** à utiliser avant **modload**. Un module **FOM étendu** peut proposer plusieurs sous fonctions de modexe, dans ce cas l'utilisation de **>fom** est obligatoire.

Ces modules sont placés dans le dossier PARX.SYS, sous-dossier FORTH et portent l'extension FOM (Forth Module).

>fom

>fom mots1,mot2,...,motn

demande l'importation des mots correspondants dans les modules via modload. On n'est pas obligé de déclarer tous les mots proposés par un module, et si un mot n'existe dans aucun module, cela ne génère pas d'erreur. Il sera ignoré. La liste des mots doit être séparée par des virgules et sans aucun espace. Les mots doivent apparaître exactement comme à l'intérieur du module. Utiliser **modlname** pour vérifier.

Inclusion facile

Les modules FORTH peuvent être gérés entièrement à la main, mais une série de fichiers *.INC facilite leur gestion, ce sont des fichiers FOR contenant les instructions nécessaires pour charger et mettre les instructions du module à disposition.

Exemple avec le module DEBUG.FOM, nous pouvons utiliser simplement dans notre source:

```
9 >include "%DEBUG.INC"
```

Qui exécutera les instructions de ce fichier d'initialisation s'il est bien logé dans le dossier FOR par défaut. Ces fichiers utilisent le **flag 10**.

➔ Si le **flag 10 est mis** (défaut), le chargement se fait en mode **informations**, vous obtenez l'affichage du résultat du chargement par modload, le nom du module et diverses autres informations spécifiques au module.

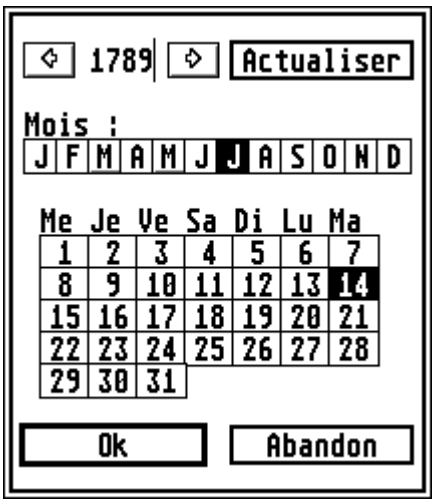
```
10 >true  \ mode informations
9 >include "%DEBUG.INC"
```

➔ Si le **flag 10 n'est pas mis**, le chargement est en mode **silencieux**.

```
10 >false \ mode silencieux
9 >include "%DEBUG.INC"
```

Il vous faudra sans doute éditer les fichiers *.INC afin d'ajuster le chemin d'accès aux modules FOM de votre disque dur. Le chargement du module se fait par inclusion directe, cela n'a pas trop d'incidence sous l'éditeur, mais si vous compilez votre programme, le module sera intégré à l'exécutable et vous n'aurez plus besoin d'un accès au PARX.SYS si vous diffusez votre logiciel.

►Module CALENDAR.FOM V1.00



Ce module affiche un dialogue avec le calendrier qui peut se régler à partir de l'année 1583. On peut modifier l'année, le mois, le jour, obtenir les jours de la semaine correspondants et grâce au bouton OK, récupérer une chaîne contenant la date sélectionnée au format utilisable par les fonctions FORTH.

Il propose les mots **set_calendar** et **calendar** pour remplacer modinfo et modexe.

D'abord, les appels standards :

Initialisation :

```
variable p                              \ pointeur sur mon module
" D:\PARX.SYS\" 1 modset                \ définit le chemin d'accès
0 p !                                   \ laisser FORTH gérer la mémoire
" FORTH\CALENDAR.FOM" 1 p modload . \ charger et initialiser
```

Renvoie 0 (Ok)

Réglage de la langue et du format :

```
code format p modinfo .                 \ code = code langue
                                        \ 0=US, 1=GE, 2=FR, 4=ES, 5=IT
                                        \ format = 0 (MM/JJ/AAAA),
                                        \        = autre (JJ/MM/AAAA)
```

Renvoie 2 (Ok)

Appel au calendrier :

```
0 p modexe              \ pas de date, ouvre avec la date du jour
if                      \ si renvoie <>0, bouton OK
      type              \ et affiche la chaîne de la date dans pad
else
      ." Annulé"        \ si 0, annulation pas de chaîne renvoyée
then

" 25/7/2024" p modexe   \ au lieu de zéro, l'appel peut se faire avec
                        \ une date de départ.
```

Utilisation de l'interface :

Indiquer la date et/ou le mois désiré et appuyez sur **Actualiser**. Les jours de la semaine ainsi que la taille du mois sont mis à jour.
Les boutons flèches (pour année précédente ou suivante) actualisent l'affichage.

Le bouton **Ok** renvoie la date dans **pad** sur la pile et une valeur non nulle au-dessus.
Le bouton **Abandon** renvoie simplement zéro.

Pour rendre plus lisible et aisée l'utilisation du module, ce dernier propose deux mots qu'on doit déclarer avec **>fom** :

set_calendar

code format set_calendar *(remplace p modinfo)*

calendar

date calendar *(remplace p modexe)*

Voici donc le nouveau programme :

```
>fom set_calendar,calendar          \ demande l'importation
variable p                          \ pointeur sur mon module
" D:\PARX.SYS\" 1 modset            \ définit le chemin d'accès
0 p !                               \ laisser FORTH gérer la mémoire
" FORTH\CALENDAR.FOM" 1 p modload . \ charger et initialiser

2 2 set_calendar drop               \ remplace modinfo, fixe FR et JMA

0 calendar if type then             \ ouvre calendrier et affiche la
                                    \ date sélectionnée.
```

►Module étendu BINARY.FOM V1.00

Ce module donne accès à toutes les **fonctions binaires** du 68000 qui n'étaient pas déjà intégrées au FORTH. Étant un module **FOM étendu**, l'utilisation de **>fom** est obligatoire pour accéder aux multiples sous fonctions de modexe.
Ces fonctions utilisent une valeur qui retient le dernier bit testé ou déplacé, donc 0 ou 1 qu'on obtient avec **Bcarry** (en assembleur, ce bit étant nommé carry). Certaines fonctions sont sensibles à la taille de l'opérande et peuvent s'appliquer à un octet, un mot ou un mot long, ceci se règle avec **Bsize**.
Lors de l'initialisation de **modload**, la taille est fixée à LONG et le carry à zéro.

Bsize

taille Bsize : taille pouvant être .b, .w ou .l. Fixe a taille des opérandes pour les fonctions de décalage et de rotation.

Bcarry

Bcarry : renvoie la valeur du dernier bit affecté par la dernière opération, valeur 0 ou 1.

Bcs

Bcs : Carry set : met à 1 le carry.

Bcc

Bcc : Carry clear : met à 0 le carry.

Blsl, Blsr, Basl, Basr

val #n Blsl : Logical shift Left
val #n Blsr : Logical shift Right
val #n Basl : Arithmetic shift Left
val #n Basr : Arithmetic shift Right

Ces fonctions réalisent le décalage sur val de n bits. Elles affectent uniquement la partie spécifiée par Bsize et carry contient le dernier bit décalé.

Le décalage arithmétique conserve le signe de val, le logique, non.

Brol, Bror

val #n Brol : Rotate left
val #n Bror : Rotate right

Ces fonctions réalisent une rotation de val de n bits. Elles affectent uniquement la partie spécifiée par Bsize et carry contient le dernier bit décalé.

Broxl, Broxr

val #n Broxl : Rotate left with extended
val #n Broxr : Rotate right with extended

Ces fonctions réalisent une rotation de l'ensemble (val+carry) de n bits. Elles affectent uniquement la partie spécifiée par Bsize et carry contient le dernier bit décalé.

Exemple sur un octet et une rotation à gauche, on colle carry à gauche de val :

C xxxx xxxx et la rotation s'effectue sur 9 bits.

C'est à dire que chaque bit de val passe par C, et la valeur précédente de C revient dans val.

Bclr, Bset, Bchg, Btst

val #n Bclr : bit test and Clear (met le bit n à zéro)
val #n Bset : bit test and Set (met le bit n à 1)
val #n Bchg : bit test and Change (inverse le bit n)
val #n Btst : bit test (remplit simplement carry)

Ces fonctions permettent d'agir sur le bit n d'un mot long, n allant de 0 à 31. Le bit #n est d'abord testé, sa valeur mise dans carry, et ensuite l'opération s'effectue sur val, qui est laissé sur la pile.

Exemple, inversons l'ordre des bits d'un mot long :

```
>fom Bsize,Broxl,Blsr   \ importe le nécessaire
variable p                          \ pointeur sur mon module
" D:\PARX.SYS\" 1 modset            \ définit le chemin d'accès
0 p !                               \ laisser FORTH gérer la mémoire
" FORTH\BINARY.FOM" 1 p modload .   \ charger et initialiser

.l Bsize                            \ taille LONG
%h12345678 0                        \ valeurs à inverser et de retour
32 ndo
      swap 1 Blsr       \ décale à droite et remplit Carry avec le bit
      swap 1 Broxl      \ décale à gauche en injectant carry
nloop
uh.                     \ affiche la valeur retournée bit par bit
drop                    \ efface valeur de départ
                        \ qui vaut 0 après 32 décalages.
```

►Module étendu CONFIG.FOM V1.01

Ce module permet de gérer un fichier texte de configuration contenant plusieurs lignes de la forme :

clé = valeur

On charge le fichier à l'aide de **CFGload**, on peut le sauvegarder avec **CFGsave** et accéder aux clés pour les lire (**CFGgetkey**), modifier/ajouter (**CFGsetkey**) , ou supprimer (**CFGdelkey**). Si le fichier n'existe pas encore, on peut créer un fichier vide en mémoire avec **CFGcreate**.

Comme dans tout module étendu, **>fom** est obligatoire.

- Dans le fichier, il ne faut qu'**une seule clé par ligne**, le signe = peut être entouré d'espaces ou non. Peu importe. La valeur de la clé court jusqu'à la fin de ligne.
- Le fichier peut avoir des séparateurs CR+LF ou seulement l'un des deux.
- Toute ligne commençant par **;** ou par **** est considérée comme un **commentaire**.

La taille du fichier ne peut excéder 64Ko (des instructions limitées aux mots de 16 bits sont utilisées dans les boucles.)

CFGset

extra flag CFGset
Fixe les paramètres généraux :
extra : taille supplémentaire en octets allouée lors du chargement du fichier pour prévoir son éventuelle expansion (en cas d'ajout de clés).
flag : 1 pour distinguer majuscules/minuscules dans la recherche des clés
0 pour ne pas distinguer.
Au démarrage, extra=1024 et flag=1. La fonction ne renvoie rien.
On peut fixer extra à zéro si le fichier ne doit être que lu et non modifié.

CFGload

" chemin\fichier" CFGload
charge le fichier de configuration spécifié dans un bloc avec « extra » octets en plus.
Renvoie : 1 Ok, fichier chargé
0 Erreur, fichier introuvable ou pas assez de mémoire.

On utilisera shel_find pour construire le chemin complet d'accès à un fichier se trouvant dans le répertoire de l'application.

```
" TEST.CFG" pad $!
pad shelfind
intout w@
if
      pad CFGload
      if
            ." Chargement ok"
      else
            ." Erreur de chargement"
else
      ." Fichier non trouvé"
then
```

CFGsave

" chemin\fichier" CFGsave
sauvegarde l'image du fichier tel qu'il est en mémoire.
Renvoie : 1 Ok, fichier écrit
0 Erreur, impossible d'écrire, ou emplacement inexistant.

CFGcreate

CFGcreate
crée un fichier vide de configuration en mémoire avec "extra" octets. Fonction à utiliser si le fichier de configuration est introuvable, ou lors d'une première utilisation du logiciel.
La fonction renvoie : 0 erreur (extra=0 ou bien pas assez de mémoire)
1 Ok, bloc créé.

CFGfree

CFGfree
Libère le bloc mémoire réservé au fichier de configuration. Vous pouvez à nouveau utiliser CFGload pour un autre fichier. Renvoie 0 si erreur, 1 si Ok.

CFGgetkey

" clé" CFGgetkey
Recherche la clé dans le fichier et renvoie:
0 : clé non trouvée
1 : clé trouvée et dessous, pad contient la valeur de la clé.
Une clé peut avoir la forme :
CLEF=
Avec une valeur vide, CFGgetkey renvoie bien 1 (clé trouvée) et une chaîne vide dans pad. Ceci est l'équivalent de DEFINE en C, le mot est défini mais sans valeur. Il sert de flag.

Pour la valeur renvoyée, le module supprime les espaces en début et en fin de chaîne.

CFGsetkey

" clé" " valeur" CFGsetkey
Donne la valeur à la clé.
Si la clé n'existe pas, alors elle est ajoutée au fichier.
Si elle existe, elle est modifiée et l'ancien contenu perdu.
Renvoie 0 pas assez de place pour la nouvelle clé
1 Ok, clé modifiée/ajoutée.

En cas d'erreur, il vous appartient de sauver et rouvrir le fichier avec à nouveau extra octets libres.

```
" clé"     " valeur"    CFGsetkey
0= if
      " chemin\fichier" CFGsave drop
      CFGfree
      " chemin\fichier" CFGload drop
then
```

CFGdelkey

" clé" CFGdelkey
Efface la clé et sa valeur.
Renvoie 0 la clé n'existait pas
1 clé bien effacée

►Module étendu RTF.FOM V1.03

Ce module permet l'export du contenu de l'éditeur (ou du bloc marqué) au format **RTF (Rich Text Format)** à la fois pour le monde Atari (Atari Works par exemple) et pour le monde PC (Wordpad par exemple).

L'aspect est celui-ci :

✔ les **mots du FORTH** apparaissent en gras
✔ les mots créés sont soulignés lors de leur création
✔ les *commentaires* apparaissent en italique

Pour la sauvegarde **Atari**, deux choix possibles :

◆ on utilise soit le mode par défaut (System Font) et ce sont des attributs de cette seule fonte qui modifient l'aspect des différents mots.
◆ on utilise une liste de trois fontes à spécifier pour le mode normal, gras et italique.

Pour la sauvegarde **PC** :

◆ on utilise soit le mode par défaut (fonte Arial)
◆ on peut changer la famille de fonte (Courier par exemple)
◆ dans ces deux cas, les attributs permettent au PC de choisir la fonte correcte dans la famille demandée.

RTFmode

n RTFmode : choisit le mode Atari (n=0, défaut) ou PC (n<>0). Cet appel modifie le comportement de RTFfont, il doit donc le précéder ! Renvoie 2 pour Ok.

Note : dans le mode PC, les codes ASCII de la plupart des caractères sont adaptés (par exemple, les minuscules accentuées).

RTFfont

Selon le mode de l'instruction précédente, pour l'Atari :

0 RTFfont : choisit le mode par défaut (Système Font et attributs)
" nom" Id n RTFfont : fixe la fonte pour le mode normal (n=1), le mode gras (n=2) ou le mode italique (n=3).

Le nom et l'Id sont exactement ceux renvoyés dans pad et intout par **vqt_name**.

Pour le PC :

0 RTFfont : choisit le mode par défaut (Famille Arial et attributs)
" nom" RTFfont : fixe un nouveau nom de famille de fontes.

Renvoie 2 si Ok, et 0 si erreur (valeur n en dehors de 0,1,2,3).

RTFsave

" chemin\fichier" RTFsave : sauvegarde le source (ou le bloc) dans le format spécifié par RTFmode et RTFfont.

Si vous ne désirez pas sauvegarder le source présent sous l'éditeur, mais un autre fichier FOR chargé à une autre adresse, on utilise alors la zone **sndb** comme dialogue avant l'appel :

```
" .RTF" @ sndb !  \ 4 premiers octets de sndb avec une valeur magique
adr sndb 4 + !    \ les 4 suivants avec l'adresse d'un fichier FOR complet
                  \ suivi d'au moins deux octets nuls (si version<1.03,
                  \ ensuite inutiles)
```

Renvoie 2 si Ok et 0 si erreur (erreur d'écriture sur le fichier), si le sndb a été utilisé la valeur magique est effacée et doit être remise à chaque appel de ce type.

RTFtab

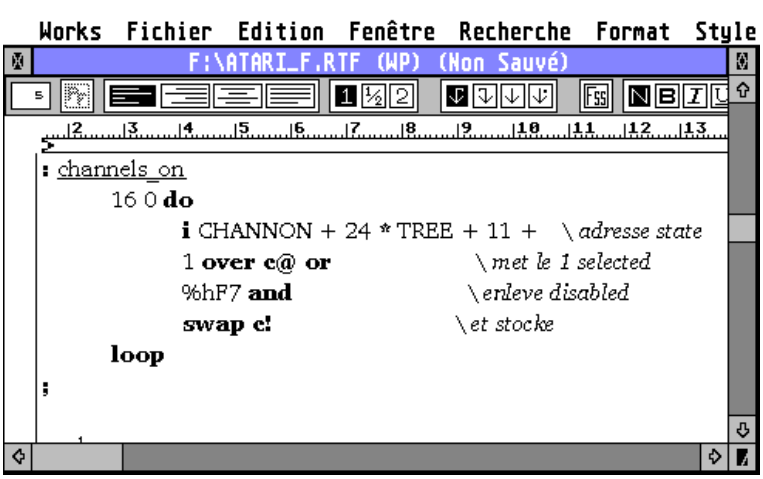
val RTFtab : définit la taille de la tabulation. Par défaut 720, mais une valeur proche de 400 rend mieux l'aspect du source tel qu'il est sous l'éditeur.

Un exemple : La première partie charge le module et crée le mot « go » en mémoire.

Ensuite, l'éditeur est effacé avec buf! et on charge le programme à transformer en RTF. L'exécution de go lance le processus dans les 4 cas de figure.

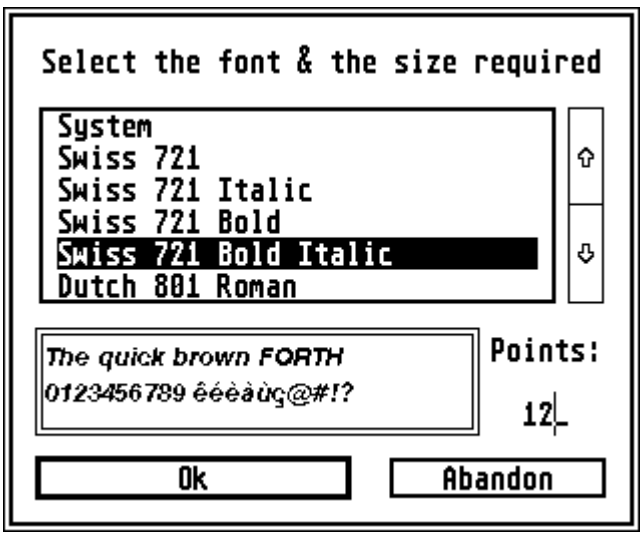
```
variable p
>fom RTFsave,RTFmode,RTFfont
" D:\PARX.SYS\" 1 modset
" FORTH\RTF.FOM" 1 p modload .

: go
   ." Sauvegarde Atari : "
   0 RTFmode . space
   0 RTFfont . cr
   ."    Par défaut : "
   " F:\ATARI_D.RTF" RTFsave . cr
   ."    Avec fontes: "
   " Bitstream Charter"        5648 1 RTFfont . space
   " Bitstream Charter Black"  5709 2 RTFfont . space
   " Bitstream Charter Italic" 5649 3 RTFfont . space
   " F:\ATARI_F.RTF" RTFsave . cr cr
   ." Sauvegarde PC    : "
   1 RTFmode . space
   0 RTFfont . cr
   ."    Par défaut (Arial) : "
   " F:\IBM_D.RTF" RTFsave . cr
   ."    Avec fonte         : "
   " Courier" RTFfont . space
   " F:\IBM_F.RTF" RTFsave . cr
;
>comp
buf! loadb go
```

Voici ce que l'on peut obtenir sous **Atari Works** avec la sauvegarde Atari et les 3 fontes Bitstream Charter.



►Module FONTSEL.FOM V1.02

Ce module permet de sélectionner une police de caractères GDOS par l'intermédiaire d'un formulaire.



Il ne fonctionne qu'en présence d'un gestionnaire de fontes installé (GDOS, SpeedoGDOS, NVDI,...).

Il nécessite d'abord l'appel obligatoire à **FSinit** pour fixer son comportement, un appel facultatif à **FStitle** pour personnaliser le titre du formulaire et finalement, l'appel principal à **FSopen** qui ouvre le sélecteur.

FSinit

flag var FSinit : fixe les paramètres.
flag 0 si le sélecteur doit faire lui-même l'appel à vst_load_fonts
n vst_load_fonts a déjà été appelé, n est sa valeur de retour
var 0 pas de variable d'échange
var adresse d'une variable 4 octets qui contiendra en entrée comme en sortie la taille sélectionnée (max 999).

Renvoie : 0 erreur (GDOS non installé)
f nombre de fontes disponibles (fonte système comprise)

FStitle

" chaîne" FStitle : personnalise le titre du formulaire. 35 caractères maxi.

Renvoie 2 Ok.

FSopen

flag FSopen : ouvre le sélecteur.
flag 0 pas de fonte présélectionnée
n sélectionner la fonte n à l'ouverture.

Renvoie 0 Abandon, ou aucune fonte sélectionnée
n numéro de la fonte sélectionnée (utilisable dans **vqt_name**).

►Module étendu DIGIDRUM.FOM V1.00

Ce module permet de rejouer des partitions de batterie sur deux voix. La sortie se fait sur le Yamaha ou sur diverses cartes son. On peut définir de simples patterns rejoués en boucle ou un enchaînement de ceux-ci constituant une chanson.

Outre le chargement du module, il faut sélectionner un kit de batterie composé d'un maximum de 16 samples de 16Ko échantillonnés à 20KHz, 8 bits, mono, ceci se fait avec **DDkit**.

On doit ensuite créer un tableau de chaînes de caractères qui recevront les patterns, une description de la partition au format texte, tableau qu'on fournit à **DDpatterns**. Cela peut être suffisant pour rejouer des motifs en boucle. Une possibilité plus élaborée est de créer des chansons avec un enchaînement programmable de patterns, à cet effet, on définit un tableau d'entiers qui recevront les codes de la chanson, tableau qu'on fournit à **DD-songs**.

DDkit

adr DDkit fixe l'adresse du kit batterie. En retour, on obtient dans pad une liste des différents échantillons avec leur code lettre (de A à P maximum). Pad n'est pas laissé sur la pile.

DDtempo

n DDtempo fixe le tempo en 1/50e de seconde, n peut varier de 1 à 49. Il donne le temps pour chaque beat. La valeur par défaut est 6 (8,33 beats par seconde).

DDfreq

n DDfreq fixe la fréquence, et donc la qualité du replay.
- n=0 6,6 kHz, qualité basse mais CPU peu occupé.
- n=1 10 kHz, qualité moyenne, charge CPU moyenne.
- n=2 20 kHz, haute qualité mais charge CPU importante.

La valeur par défaut est n=2.

DDoutput

n DDoutput sélectionne le périphérique de sortie pour le son.
- n=0 Yamaha (Qualité moindre, consomme du CPU)
- n=1 carte Psound (Qualité moyenne)
- n=2 carte ST Replay 16 (Bonne qualité)
- n=3 cartes ST Replay 8 ou MV16 (Bonne qualité)

La valeur par défaut est n=0.

DDpatterns

adr DDpatterns fournit l'adresse du premier élément du tableau de chaînes des patterns. Les chaines sont utilisées par paire car il y a deux canaux pour la partition, voir plus loin. La taille donnée aux chaînes définit la taille maximale d'un pattern en termes de beats.

Exemple :

```
32 100 array$ PATT        \ 100 chaines de 32 beats maxi
0 PATT DDpatterns         \ définit l'adresse de la première chaine
```

DDplaypatt

n DDplaypatt rejoue en boucle le pattern défini par les chaines n et n+1. Ces deux chaînes doivent avoir la même taille !

Exemple (avec le kit de base) :

```
" --E--E" 2 PATT $!             \ canal 1 le hit hat
" A--B--" 3 PATT $!             \ canal 2, grosse caisse et snare
2 DDplaypatt                    \ rejoue un boogie woogie
```

DDstop

n DDstop arrête le son en cours selon n :
- n=0 immédiatement
- n=1 à la fin du pattern en cours

DDsongs

adr DDsongs fixe l'adresse du premier élément du tableau pour loger les chansons. On peut estimer la taille nécessaire selon la complexité de la chanson. Les 10 premiers éléments sont réservés à la gestion des labels pour les boucles et chaque instruction de programmation peut prendre de 1 à 2 éléments. On dira que par sécurité, 2×nombre d'instructions + 10 est une bonne valeur.

Exemple :

```
100 array SONG      \ définit le tableau de 100 valeurs
                    \ assez pour 45 instructions.
0 SONG DDsongs      \ et en fournit la première adresse
```

DDplaysong

n DDplaysong rejoue la chanson définie à partir de l'élément n du tableau.

Programmation d'une chanson :

Pour programmer une chanson on se base sur des patterns existants dont on définit les répétitions et l'enchaînement. De plus, l'utilisation de plusieurs pédales est possible, elles se branchent sur le port Joystick et doivent être rattachées aux mouvements haut, bas, droite, gauche.

#new

flags #new : efface le tableau des chansons et initialise les pointeurs de programmation. Le flag correspond à un masque de bits définissant la position initiale des pédales afin de s'adapter à certains dispositifs dont la valeur au repos est 1 au lieu de zéro.

```
flag    right  =    %b1000
        left   =    %b0100
        up     =    %b0010
        down   =    %b0001
```

#here

l #here : définit l'emplacement courant comme étant le label l (valeur de 0 à 18).

#where

l #where : renvoie l'index dans le tableau de chanson du label l.

#patt

r n #patt : répète r fois le pattern n.

#gosub

r l #gosub : exécute r fois les instructions se trouvant au label l.

#ret

#ret : retourne à l'instruction gosub ou met fin à la chanson si aucun niveau d'appel n'est en cours.

#tempo

n #tempo : modifie le tempo en cours de chanson, voir DDtempo.

#tempo+ #tempo-

n #tempo+ : ajoute n au tempo en cours (ou le retire pour #tempo-).

#mute

n #mute : insère un silence de n beats avant l'instruction suivante.

#pedal

" masque" l #pedal : retourne vers le label l jusqu'à ce que la condition spécifiée par le masque soit vraie. Le masque est une chaîne d'exactement 4 caractères pour les 4 directions du joystick. Pour chaque caractère on peut avoir :

x bit ignoré
0 le bit doit obligatoirement être nul
1 le bit doit obligatoirement être à 1
o (lettre O minuscule) définit un ensemble de bits dont l'un au moins doit être non nul.

Exemples :

```
" xx0x" l #pedal   :retour vers l en attendant le bit UP à zéro
" xxx1" l #pedal   :retour vers l en attendant le bit DOWN à 1
" ooxx" l #pedal   :retour vers l en attendant RIGHT ou LEFT à 1
" oooo" l #pedal   :retour vers l en attendant l'un des 4 bits à 1
```

#goto

l #goto : saute vers le label l.

Exemple :

```
" --F--F" 0 PATT $!      \ boogie woogie hit hat ouvert
" A--B--" 1 PATT $!
" --E--E" 2 PATT $!      \ boogie woogie hit hat fermé
" A--B--" 3 PATT $!

%b0010 #new              \ nouvelle chanson
0 #here
   1 2 #patt             \ couplet fermé jusqu'à pédale UP
   " xx0x" 0 #pedal
2 #here
   1 0 #patt             \ refrain ouvert jusqu'à pédale UP
   " xx0x" 2 #pedal
   0 #goto
1 #here                  \ début de chanson
   8 #tempo              \ fixe le tempo
   0 #goto               \ et démarre indéfiniment
1 #where DDplaysong      \ démarre au label 1
```

►Module étendu DEBUG.FOM V1.03

Ce module permet d'utiliser un second écran afin d'afficher des informations de débuggage pour la mise au point d'un programme sans interférer avec l'affichage de l'application en cours. Le matériel nécessaire :

- Un TT avec carte graphique, l'affichage se fait sur la sortie standard en monochrome
- Un MegaSTE avec une carte graphique, l'affichage se fait en ST Haute ou ST Moyenne selon le second moniteur connecté.

Si la machine ne comporte pas de carte graphique, alors un buffer de 32000 octets est réservé pour un second écran vers lequel on peut basculer avec la combinaison de touches Alt+HELP:

- Un TT, le second écran est toujours en ST Haute
- Un ST(e), le second écran est en ST Haute sur moniteur monochrome et en ST Moyenne sur un moniteur couleur.

L'appel à **modload** initialise le second écran et renvoie un masque de bits :
bit#0 : 1=fonte 8x8 disponible
bit#1 : 1=fonte 8x16 disponible
bit#2 : 1=second écran monochrome, 0=ST Moyenne
bit#3 : 1=GEM sur carte graphique

En sortie de programme, ou lors de l'appel de **modunload**, un message vous informe de la désinstallation du module (routine **do_exit** présente) et vous demande de valider par CapsLock.
Si un buffer était alloué, il est libéré.

Dfont

n Dfont change de fonte selon n :
n=8 demande la fonte système 8x8
n=16 demande la fonte système 8x16
n=adr donne l'adresse d'un header de fonte 8xh avec 256 caractères définis

renvoie 2 si Ok, 0 si fonte non disponible ou si la taille/nombre de caractères ne convient pas.

Dauto

masque Dauto fixe les comportements automatiques pour l'affichage :
bit#0 : 1=espace automatique après un nombre (défaut 1)
bit#1 : 1=saut de ligne automatique après un nombre (défaut 0)
bit#2 : 1=auto dup pour les nombres (défaut 1), sinon dépile
bit#3 : 1=auto dup pour les chaînes (défaut 0), sinon dépile
bit#4 : 1=auto Dback pour les dump/+/- (défaut 0)
bit#5 : 1=intercepter les erreurs de bombes (défaut 0)

-1 Dauto renvoie la valeur courante du masque.

Dconfig

affiche un menu interactif pour les réglages de Dauto. Flèches haut/bas pour sélectionner la ligne et Espace, Enter, ou flèches droite/gauche pour modifier ON/OFF. **Escape** pour quitter ce menu.

Dcls

efface le second écran

Dcr

ligne suivante sur le second écran

Dspace

affiche un espace

D. Df.

affiche le sommet de la pile (conservé ou non, voir Dauto) en décimal (pour D.) ou réel (Df.).

Dh.

affiche le sommet de la pile (conservé ou non, voir Dauto) en hexadécimal sur 8 chiffres.

Dtype

affiche la chaîne (conservée ou non, voir Dauto)

D.s

affiche le contenu de la pile en décimal.

Dnewl

revient au début de la ligne en cours et l'efface.

Dkey

pause et attend l'appui de CapsLock (dont l'état est restauré ensuite), l'affichage d'une flèche vers la droite vous indique cette action à l'écran. La flèche disparaît dès qu'on a appuyé sur CapsLock.

Sur un système sans carte graphique, l'appel à Dkey bascule vers le second écran si on y est pas encore. Et dans ce cas, l'appui de CapsLock revient à l'écran normal sauf si on a utilisé Alt+Help entre l'entrée et la sortie de Dkey.

Dump

adr n Dump affiche le contenu de la mémoire à partir de l'adresse adr sur n lignes de 16 octets. Les octets sont affichés à la fois en hexadécimal et sous forme ASCII.

La lecture se fait en superviseur, les zones protégées sont accessibles.

Dump+ Dump-

selon les réglages de Dump, avance ou recule de n lignes et affiche la mémoire.

Dsave Dback

sauvegarde la position actuelle du curseur (Dsave) ou la restaure (Dback). Voir aussi Dauto pour le fonctionnement automatique avec Dump. Ce système n'a qu'un seul niveau de sauvegarde. Si vous appelez Dsave deux fois de suite, la première position est perdue.

Interception des bombes:

Lorsqu'une erreur est détectée, s'affiche sur l'écran de debug le type d'erreur, son point d'arrêt, la valeur de 16 registres du processeur ainsi que des informations sur la localisation du compteur programme du CPU et du registre A5 qui sert de pointeur d'interprétation en FORTH.

La touche CapsLock est attendue pour revenir à la ligne de commande.

Lors de son appui, les piles sont réinitialisées et le programme en cours stoppé.

À noter:

- La plupart du temps l'adresse PC du CPU indique l'instruction suivant celle de l'erreur (sauf pour "illegal instruction").
- Sur un 68000 la valeur du PC pour "bus error" et "address error" est aléatoire peut être décalée de plusieurs octets par rapport à l'erreur.

►Module QRCODE.FOM V1.00

Ce module permet de générer un QR-Code à partir d'un texte. Il supporte les 40 versions possibles du QR-Code, les 4 niveaux de récupération d'erreur et les 8 masques pour améliorer la lisibilité.

Il permet l'encodage en **NUM** (uniquement des nombres, codage très compact), en **ALPHA** (les majuscules, les nombres et les symboles d'une URL, codage compact) et en **BIN** (mode binaire, tout octet est valide, codage moins compact). Pour l'instant, les caractères Kenji ne sont pas supportés.

On fournit le message à **QRinfo** qui renvoie la taille nécessaire pour le calcul. On lance la création avec **QRcode** qui génère une image monochrome (avec une éventuelle icone ajoutée au centre) utilisable directement avec un appel VDI pour affichage. Pour finir **QRsave** permet de garder un fichier IMG ou BMP sur le disque.

QRinfo

mess err QRinfo : prépare l'encodage du message mess (chaîne de caractères finie par zéro) avec le niveau de récupération d'erreur err, qui permet de lire le QR-Code malgré une partie manquante ou abîmée de l'image.

Attention: l'adresse du message ne doit pas changer entre QRinfo et QRcode, il ne faut pas que ça soit une constante.

err = 0 pour 7% de récupération
1 pour 15%
2 pour 25%
3 pour 30%

Plus la récupération est efficace, plus cela demande de données et donc génère un QR-Code plus grand.

Valeur retournée:

-2	valeur err erronnée (en dehors de 0-3)
-1	message vide
2	Ok

De plus, le **mfdbs** contient les informations suivantes:

+0	LONG	taille buffer à réserver pour **QRcode**
+4	WORD	version choisie (1-40)
+6	LONG	4 octets du nom de l'encodage NUME, ALPH, BYTE, KENJ
+10	WORD	encodage (1 num, 2 alp, 3 byt, 4 ken)
+12	WORD	err

QRcode

buf QRcode : lance la génération du QRcode en utilisant le buffer fourni, sa taille vous a été donnée via le **mfdbs**. D'une manière simple, si vous voulez englober tous les cas possibles et que la taille mémoire n'est pas une limitation pour vous, un buffer de 80ko sera largement suffisant.

En entrée si :

buf+0 = 'ICON'

buf+4 = adresse

Alors cette adresse correspond à celle d'une **structure ICONBLK** décrivant une icone qui sera ajoutée au centre du QR-Code (avec la gestion du masque et des données).

Valeur retournée:

-1 mode non supporté

2 Ok

De plus le **mfdbs** est rempli et pointe vers le bloc monochrome de l'image.

+0	LONG		adresse des données (quelque part dans le buffer)
+4	WORD	l	largeur en pixels
+6	WORD	h	hauteur en lignes
+8	WORD	L	largeur en mots = (l+15)/16
+10	WORD	0	format shifter
+12	WORD	1	nombre de plans

Pour l'afficher à l'écran, à la position x,y

```
0 mfdbd !                       \ destination écran
0 0                             \ position 0,0 dans la source
x y                             \ position à l'écran
mfdbs 4 + w@ mfdbs 6 + w@       \ largeur et hauteur
1 1 0                           \ mode replace, couleurs noir/blanc
vrt_cpyfm                       \ copie bloc
```

QRsave

" chemin\fichier" i QRsave : enregistre le QR-code sur disque avec le nom spécifié et selon i:

i = 0 format Atari IMG monochrome

1 format PC BMP monochrome

Valeur retournée:

-2 erreur fichier

-1 valeur i non supportée

2 Ok, fichier sauvé.

Le Pré-processeur

1 Inclusion d'un fichier lors de la compilation

On peut assembler au préalable une routine et l'inclure dans un programme FORTH au moment de la compilation, en liaison avec COMP.PRG, cecla donnera un programme monobloc (le fichier est inclus une fois et fait partie de l'exécutable). Il est bien sur possible d'inclure autre chose que du code exécutable, ne serait-ce qu'un autre fichier FOR !

Pour le code exécutable, plutôt que de chosir un format objet (de certains compilateurs) pusiqu'ils ne sont pas tous équivalents, j'ai choisi d'inclure directement des fichiers éxécutables Atari.

>include

```
m   >include   "chemin\fichier.ext"
```

inclut un fichier à l'intérieur du programme à l'emplacement courant du dictionnaire. Le mot de commande m est un masque de bits:

```
bit 0       : 0=un fichier PRG
              1=Autre
bit 1       : 0=s'assurer de la parité du pointeur après chargement
               1=ignorer parité
bit 2       : 0=reloger les adresses absolues
              1=ne pas reloger
bit 3&4     : pris en compte uniquement si bit0=1 (Autre)
              00    = fichier quelconque
              01    = fichier FOR à inclure
              10    = module PARX
              11    = fichier DEF (constantes d'un RSC)
```

Dans le cas de l'**inclusion d'un fichier FOR**, il s'agit simplement d'y poursuivre la compilation, le fichier n'est pas réellement inclus, mais interprété. S'il contient des parties exécutables, elles seront exécutées, si ce sont des définitions, elles seront ajoutées.

Plusieurs niveaux d'inclusion FOR sont possibles.

Pour l'**inclusion d'un fichier DEF**, des constantes sont créées selon les définitions d'un fichier RSC sous INTRFACE.PRG ou ORCS.PRG avec 8 caractères maxi.

Pour tous les autres cas, voir ci-dessous les différentes possibilités offertes par cette commande.

Inclusion d'une routine dans un mot assemblé

Exemple, dans un mot je veux décaler une valeur de 3 bits vers la droite (division rapide par 8). Je prépare en assembleur le programme suivant:

```
      text

      move.l (a6),d0    ; la valeur prise sur la pile
      asr.l #3,d0       ; décale
      move.l d0,(a6)    ; remet sur la pile

      end
```

Que j'assemble vers C:\DIV8.PRG

Il ne reste plus qu'à écrire en FORTH:

```
:: essai
      p @
      0 >include "C:\DIV8.PRG"
      p !
;
```

Ce mot divisera la variable p par 8!

A noter:

** pas d'espaces dans la chaine du nom de fichier*

** les sections TEXT et DATA sont incluses dans la définition, à vous de 'sauter' par dessus les datas si le mot doit se poursuivre.*

Inclusion de définitions de mots en assembleur

Si le mot doit être défini entièrement en assembleur, on doit utiliser une structure particulière du fichier PRG:

* la zone TEXT contient les définitions et les données

* la zone DATA contient les adresses et les noms des mots créés, elle est donc réservée.

Exemple, on peut préparer en assembleur les mots suivants:

```
      text

plus8:
      dc.w 437          ; tous les mots assemblés en FORTH commencent
            ; par ce code.
      addq.l #8,(a6)    ; ajoute 8 à la valeur sur la pile
      rts

moins8:
      dc.w 437
      subq.l #8,(a6)    ; soustrait 8 à la valeur sur la pile
      rts

salut:
      dc.w 437
      lea chaine(pc),a0
      move.l a0,-(a6)   ; empile l'adresse d'une chaine
      rts

      dc.w 26           ; en FORTH, une chaine est précédée de sa
                              ; longueur maximale.
chaine: dc.b "Salut, je suis assemblé!",0
      even
      data

      dc.b "ADD_DEFS"   ; texte obligatoire signalant une série de
                              ; définitions
      dc.l plus8,moins8,salut,-1
                              ; suite des adresses et -1 pour terminer
      dc.b "8+",0,"8-",0,"salut",0
                              ; suite des noms séparés par un '0'

      end
```

Que j'assemble vers "C:\DEFS.PRG"

Il suffit d'ajouter dans le programme FORTH:

```
0 >include "C:\DEFS.PRG"
```

pour disposer des mots '8+', '8- 'et 'salut'.

A noter:

** pas d'espaces dans la chaine du nom de fichier*

** seule la zone TEXT est incluse, la zone DATA est simplement lue mais non conservée.*

** laisser toujours à 0 le bit 2 du mot de commande si votre programme n'est pas relogeable (accès non relatif au PC).*

Inclusion 'brute' d'un fichier autre que PRG

Le problème réside dans la récupération de l'adresse à laquelle a été chargé le fichier. Le plus simple est d'utiliser l'un des codes internes du FORTH, il s'agit de [var], utilisé dans une variable.

Une variable est construite ainsi:

[var]

+valeur sur 4 octets

L'action de [var] est de fournir sur la pile l'adresse de la zone qui le suit immédiatement! Ainsi:

```
: bloc
      [var]
      1 >include "C:\IMAGE.IMG"
;
```

incluera une image dont l'adresse est obtenue en exécutant bloc. Malgré l'utilisation de : ... ; (mode interprété), le mot bloc est parfaitement assemblable car il est reconnu comme une variable (il commence par [var]!).

Aucun mot ne sera exécuté après [var] car il contient en lui même l'équivalent de ';' (fin de mot).

Ainsi:

```
: bloc
      [var]
      1 >include "C:\IMAGE.IMG"
      16 +
;
```

ne renverra pas l'adresse de l'image augmentée de 16, car la fin du mot est atteinte avec [var].

Inclusion d'un module PARX

Un module PARX nécessite d'être précédé de 32 octets libres pour les variables locales. L'inclusion permet de rendre portable son application sans qu'elle dépende de la présence du dossier PARX.SYS. La procédure est simple, on crée un mot contenant le module (**>include**), ce dernier renvoie l'adresse du module qu'on récupérera pour l'initialisation avec **modload**.

```
variable p        \ pour pointer sur le module
: mon-module
      17 >include "chemin\module"   \ 17 pour le chargement d'un module PARX
;
mon-module p !    \récupère l'adresse dans p
1 p modload .     \et lance l'initialisation du module
```

En résumé

** pour **inclure un programme** le mot de commande est 0*
** pour poursuivre la compilation dans un **fichier FOR**, le mot est 9*
** pour inclure un **fichier DEF**, le mot est 25*
** pour inclure un **module PARX**, le mot est 17*
** pour inclure un **autre fichier**, le mot de commande est 1*
** s'il s'agit d'un programme:*

*la section DATA commence par "**ADD_DEFS**":*
il s'agit d'inclure les définitions de la zone TEXT dont les adresses et les noms figurent dans la zone DATA.

la section DATA ne commence pas par "ADD_DEFS":
il s'agit d'inclure les zones TEXT et DATA comme une partie d'une définition déjà commencée.

** n'utiliser le bit 1 du mot de commande (ignorer parité du dicco) qu'en connaissance de cause: sur un 68000 les accès mots à une adresse impaire provoquent une erreur de bus, sur tous les 680X0, une instruction assembleur doit commencer à une adresse paire.*

Pour aller plus loin

On peut définir des variables dans un fichier PRG:

```
      text

var:
      dc.w 140          ; c'est le code de [var]
      dc.l 1234         ; la variable est initialisée à 1234

      data
      dc.b "ADD_DEFS"
      dc.l var,-1
      dc.b "ma_variable",0

      end
```

On peut faire du FORTH en assembleur...

```
      text

forth:
      dc.w 96           ; code de 'pad'
      dc.w 92           ; code de 'len'
      dc.w 124          ; code de '.'
      dc.w 79           ; code de ';'

      data
      dc.b "ADD_DEFS"
      dc.l forth,-1
      dc.b "pad_len",0
      end
```

Est équivalent à:

```
: pad_len
      pad len .
;
```

qui affiche la longueur de la chaîne contenue dans pad.

On peut également mêler dans un même mot une partie interprétée et une partie assemblée:

```
      text

      move.l (a6)+,d0   ; la valeur sur la pile
      btst #0,d0        ; le chiffre des unités
      beq.s .lb0
      moveq #0,d0       ; renvoie 0 si le nombre est impair
      move.l d0,-(a6)
      rts
.lb0:
      moveq #1,d0       ; renvoie 1 si il est pair
      move.l d0,-(a6)
      rts

      end
```

Assemblé vers C:\PAIR.PRG.

Ensuite:

```
: parité
      p @ [ass]
      0 >include "C:\PAIR.PRG"
;
```

renverra 0 ou 1 selon que la valeur de p est impaire ou paire. De même que pour [var], le code [ass] (celui qui vaut 437) contient en lui-même l'équivalent de ';', ainsi, c'est le dernier mot exécuté dans parité. On ne pourra pas appeler parité depuis un mot assemblé (puisque le début de sa définition est interprété).

Utiliser des variables définies en FORTH

Si la variable est définie dans la partie FORTH du programme, il se pose le problème de fournir son adresse à la partie assembleur si elle doit y accéder. Le plus simple est de définir une routine 'init' à qui on donnera ces précieux renseignements qui seront sauvés et accessibles aux autres routines:

```
      text

init:
      dc.w 437
      lea adresses(pc),a0
      move.l (a6)+,(a0)+       ; récupère deux adresses et les range
      move.l (a6)+,(a0)+
      rts

adresses: dc.l 0,0            ; place pour deux adresses: une variable et pad

majusc:
      dc.w 437
      lea adresses(pc),a0
      move.l (a0)+,a1   ; adresse de la variable
      move.l (a0)+,a2   ; adresse de pad
      move.l (a6)+,a0   ; adresse d'une chaine
      move.l a2,-(a6)   ; renvoie pad sur la pile
      moveq #0,d1       ; 0 caractères au départ
.lb0:
      move.b (a0)+,d0   ; un caractère
      beq.s .lb1        ; si nul, fin de chaine
      addq.w #1,d1      ; compte
      and.b #$DF,d0     ; en majuscule
      move.b d0,(a2)+   ; dans pad
      bra.s .lb0        ; continuer
.lb1:
      move.l d1,(a1)    ; longueur de chaine renvoyée dans la variable
      rts

      data
      dc.b "ADD_DEFS"
      dc.l init,majusc,-1
      dc.b "init",0,"majusc",0

      end
```

Assemblé vers "C:\DEFS.PRG". Ainsi:

```
variable Longueur

0 >include "C:\DEFS.PRG"

>comp
pad Longueur init       \ permet au mot 'majusc' de tourner correctement
" Coucou" majusc        \ renvoie sur la pile pad contenant "COUCOU" et la
                              \ longueur 6 dans la variable Longueur.
```

2 Préparer le source assembleur sous l'Editeur

Pour faciliter la mise au point d'un programme, le texte assembleur peut-être écrit sous l'éditeur FORTH, exporté et assemblé lors de la compilation:

>export

```
m >export "chemin\fichier.ext"
....
>comp
```

Le mot >export permet d'envoyer dans un fichier texte toutes les lignes le suivant jusqu'au prochain >comp (qui doit être le premier mot de sa ligne!) Le mot de commande m ne contient que la valeur du flag (voir >true et >false).

3 Utilisation d'un assembleur extérieur

Une fois le code tapé sous l'éditeur et exporté, il peut être assemblé directement à partir du FORTH lors de la compilation, c'est le mot >exec qui se charge d'appeler l'assembleur.

>exec

```
m >exec "chemin\prog.ext"
"ligne de commande"
```

exécute le programme spécifié en lui transmettant la ligne de commande (qui se trouve obligatoirement sur la ligne suivante).

le mot m:

```
bit 0   : 0=appui d'une touche après l'exécution du programme
          1=ne pas attendre
bit 1   : 0=ERREUR si le programme renvoie une valeur positive
          1=ne pas signaler d'erreur
bit 2   : 0=ERREUR si le programme renvoie une valeur négative
          1=ne pas signaler d'erreur
bit 3   : 0=fermer la fenêtre FORTH le temps de l'exécution
          1=ne pas tenir compte de la fenêtre
```

Dans tous les cas, une valeur nulle renvoyée par l'assembleur ne génère pas d'erreur. La règle voudrait que seules les valeurs négatives correspondent à des erreurs, mais certains programmes renvoient un code positif. Ainsi, les bits 1 et 2 permettent de s'adapter à

toutes les situations.

En cas d'attente de touche (bit 0 = 0), l'appui sur 'ESC' interrompt le traitement comme une erreur.

4 Compilation conditionnelle, les Flags

Si on compile plusieurs fois de suite un programme FORTH sans avoir modifié les parties exportées, il est inutile de les re-exporter et de les re-assembler avec les commandes >export et >exec.

Ainsi, on dispose d'une compilation conditionnelle rudimentaire, mais suffisante. Le mot de commande 'm' des trois instructions >export, >exec, >include, peut contenir le numéro d'un flag à tester pour valider ou ignorer l'action. Ce flag est signalé dans le mot supérieur (bits 16 à 31) de 'm':

>true

n >true : valide le flag n (de 1 à 15). Au démarrage, tous les flags de 1 à 12 sont 'vrais' par défaut.

Flag 13: vrai sous l'interpréteur et faux sous le compilateur.
Flag 14: vrai sur une version 68030, faux sur une version 68000
Flag 15: vrai sur une version française, faux sur l'anglaise.
Les flags 13 à 15 peuvent être redéfinis.

>false

n >false : invalide le flag n (de 1 à 15).

exemples:

```
%h40000 >export "f:\test.s"
```

n'exportera que si le flag 4 est vrai. En prenant l'opposé de 'm', on ne valide l'opération que si le flag est faux:

```
%h-20008 >exec "F:\assemble\asm.ttp"
"f:\test.s"
```

n'assemble que si le flag 2 est faux, le 8 correspond au bit 3, ne pas fermer la fenêtre FORTH lors de l'exécution.

>ask

n >ask "chaîne" : au moment de la compilation affiche la chaîne suivie de (O/N) et attend une touche pour valider (O) ou non (N) le flag n.

>ifflag

>endf

n >ifflag : poursuit la compilation si le flag n est vrai, sinon saute au prochain >endf.

-n >ifflag : poursuit la compilation si le flag n est faux, sinon saute au prochain >endf.

Dans les deux cas, >endf doit être le premier mot de sa ligne!

5 Autres Liens Extérieurs

On peut très bien exporter autre chose que de l'asembleur, exécuter autre chose que ASM.TTP. On peut imaginer qu'un programme FORTH ait besoin d'une image calculée par POV:

```
0 >export "F:\POV\image.pov"              \ envoie le script sur disque
...
ici se trouve le script de l'image
...
>comp

0 >exec "F:\POV\pov30_82.ttp"             \ calcule l'image
"-iIMAGE.POV -oIMAGE.TGA +w100 +h100"     \ en 100x100

: image
      [var]
      1 >include "F:\POV\IMAGE.TGA"             \ l'inclut au programme!
;
```

Le mot image déposera alors l'adresse du fichier en RAM.

6 Chemin par défaut

Dans les directives >include, >export et >exec, on n'est pas obligé de spécifier le chemin complet si il correspond au chemin par défaut des fichiers FOR (ou au dernier chemin spécifié dans le sélecteur de fichiers pour saveb).

Pour que FORTH ajoute le chemin, il suffit d'inclure le caractère %. Par exemple, si mon chemin par défaut est "F:\FORTH\SOURCES*.FOR":

```
0 >include "%HELLO.FOR"
```
chargera: F:\FORTH\SOURCES\HELLO.FOR

```
0 >export "%ASM.S"
```
exportera F:\FORTH\SOURCES\ASM.S

```
0 >exec "F:\ASSEMBLE\ASM.TTP"
"%ASM.S"
```
donnera F:\FORTH\SOURCES\ASM.S comme ligne de commande à ASM.TTP.

Pour >exec, ce n'est que dans la ligne de commande que le caractère % est possible.

Si on doit absolument mettre le signe % dans une ligne, il faut alors le doubler: %%.

Le compilateur

Ce programme séparé permet, avec la librairie FORTH.LIB, de générer un programme exécutable indépendant à partir d'un fichier source créé sous l'interpréteur FORTH.

1 Installation

Il faut pour cela disposer de (version 68030 / 68000):

COMP.PRG	/	COMPSTE.PRG	le compilateur
FORTH.LIB	/	FORTHSTE.LIB	la librairie
COMP.INF	/	COMPSTE.INF	quelques paramètres généraux

À l'instar de l'interpréteur, vous pouvez utiliser **FORTHINS.PRG** pour créer et modifier le fichier de paramètres du compilateur, COMP.INF.

comp.inf compste.inf

fichier texte de 4 ou 5 lignes contenant les infos suivantes:

Ligne facultative
Elle commence par le signe **#** et contient une série de flags. Si elle est absente, les valeurs par défaut sont appliquées.

"**L**" flag demandant la création d'un fichier LOG (COMP.LOG ou COMPSTE.-LOG) dupliquant tout ce qui est affiché à l'écran (par défaut, pas de fichier LOG).

Première ligne
Taille à réserver pour la compilation (le source+FORTH.LIB+la table de relogement+la partie compilée elle même) au format:
%xxx+ (ou -)yyyyy, c'est à dire xxx% de la mémoire libre plus (ou moins) yyyyy Ko. Par exemple %100-400 réservera tout moins 400Ko. Si le code contient la commande >exec (appel d'un programme externe), il faudra absolument laisser de la place.

Deuxième ligne
chemin d'accès aux fichiers sources *.FOR

Troisième ligne
chemin complet d'accès à forth.lib

Quatrième ligne
chemin d'accès au répertoire des exécutables (*.PRG) générés.

Un fichier possible est:

```
%50+100
F:\FORTH\SOURCES\*.FOR
F:\FORTH\FORTH.LIB
F:\FORTH\PROGRAMS\*.PRG
```

Assurez-vous d'avoir des **chemins semblables a l'interpréteur** en particulier si vous utilisez les chemins par défaut dans les directives >include, >exec ou >export.

2 Utilisation

Pour compiler un programme, il suffit d'avoir créé un fichier source sous l'interpréteur (*.FOR) et de lancer COMP:

1) dans le sélecteur, choisissez le fichier source à compiler (s'il n'est pas trouvé, un second essai est réalisé en forçant l'extension à .FOR)

2) dans la fenêtre s'inscrivent les opérations réalisées:

- si une erreur se produit elle est signalée et le traitement interrompu
- si tout va bien, un sélecteur s'ouvre à nouveau

3) indiquez le nom et le chemin pour le programme à écrire

Le programme est maintenant disponible sur disque. Il ne faut pas s'inquiéter de sa taille impressionnante pour un source de quelques lignes, la taille augmente peu par la suite avec un source plus gros.

Attention: Sous le compilateur, on ne peut pas utiliser les fichiers *.TXT obtenus avec export, il faut impérativement utiliser les fichiers *.FOR obtenus avec saveb ou append.

Si votre source utilise des définitions assemblées (avec :: xxx ;), alors la ligne du titre de la fenêtre affiche (s'il y a eu des optimisations) :

```
Optim #Imm:00000 1>2:00000
```

donnant les statistiques d'optimisation. Le premier nombre sont les optimisations liées à l'utilisation de constantes comme argument et le second les optimisations de chaînage où la valeur de sortie d'une fonction est récupérée par la suivante sans passer par la pile.

Astuce: *si au démarrage du programme, l'une des **touches SHIFT** est appuyée, alors cela force la création d'un **fichier LOG** sans avoir à modifier les paramètres du fichier INF.*

3 Pièges à éviter

Un programme tournant parfaitement sous l'interpréteur peut parfois avoir un comportement surprenant une fois compilé. Voici des erreurs classiques:

1°) J'écris sous l'éditeur ceci:

```
: plus
            fastopen drop
            1 2 + .
            key drop
;
```

Je le compile avec compilb puis je le lance, il m'affiche bien 3 sous l'interpréteur. Je le passe à COMP et le programme exécutable ne fait rien (retour immédiat au bureau).

Ce qui manque à mon source, c'est l'opération 'lancer le mot plus' que j'exécutais à part sous l'interpréteur. Il faut donc ajouter à mon source une ligne contenant le mot à exécuter:

```
plus
```

En fait un programme compilé réagit comme l'interpréteur, et ce source ne comportait aucune partie exécutable, seulement une définition. De ce fait, le programme s'est contenté d'enrégistrer la définition et n'a rien trouvé à exécuter!

2°) Le compilateur (ainsi que l'interpréteur) ne s'occupent que des lignes validées par Enter, il faut donc qu'il y ait une ligne vide qui suive la dernière ligne de texte. En l'absence de cette ligne fantôme, les derniers ordres du programme pourraient être oubliés.

3°) Sous l'interpréteur il existe toujours une fenêtre ouverte, l'utilisation de Fastopen n'est donc pas nécessaire, par contre pour un programme indépendant ceci est indispensable avant toute sortie graphique (textes ou autres), de plus Fastopen renvoie le handle de la fenêtre, ce qui permet de la personnaliser (en changeant le titre par exemple avec wind_set).

4 Programmes "normaux"

On dira qu'un programme est "normal" lorsqu'il est exécuté à partir du bureau et non à partir du dossier AUTO ou en tant qu'accessoire. Dans ce cas une station VDI interne est ouverte (pour les sorties textes simples) ainsi qu'une station externe pour l'utilisateur avec l'identificateur dans la variable mot handle.

Dans ce cas, fastopen ouvre une fenêtre semblable à celle de l'interpréteur FORTH et fastclose la referme (inutile en fin de programme, car la routine de sortie assure la fermeture avant de revenir au bureau).

En général, la taille de la zone réservée au dictionnaire est suffisante pour toutes les applications, mais dans certains cas l'utilisation de "reserve" pourra être nécessaire (voir le paragraphe suivant). Cette **taille par défaut** est égale aux **trois quarts** de la mémoire libre.

5 Programmes TOS/TTP

Contrairement à un programme normal, ni la VDI, ni l'AES ne sont initialisés. Le programme se lance en mode texte en plein écran. Les instructions d'affichage de nombres ou de chaînes sont redirigés sur des appels GEMDOS ainsi que les instructions d'entrées comme **input** et **input$**.

L'instruction **expect** ne permet plus d'éditer une chaîne.

Pour générer un programme TOS/TTP, il faut indiquer la valeur zéro pour le pourcentage d'écran de la fenêtre VDI dans la directive **>prgflags**.

Lors de la compilation l'extension **.TOS** vous est proposée, et si en plus votre programme fait appel à ARGC/ARGV, alors l'extension **.TTP** sera proposée.

6 Accessoires

Comme pour un programme normal, deux stations VDI sont ouvertes et fastopen/ fastclose ne gèrent que la fenêtre simple.

Un accessoire, par nature, reste présent en mémoire attendant qu'on le réveille. De ce fait, si les 3/4 de la mémoire lui sont alloués, il ne restera pas grand-chose pour les autres applications. Un mot gérant la taille du dictionnaire est donc indispensable:

reserve

n p reserve : fixe la taille du dictionnaire à p% de la mémoire libre plus n koctets. Par exemple:

```
10   0   reserve         : réserve 10 ko
0    100 reserve         : réserve toute la mémoire
-50  100 reserve         : toute la mémoire moins 50Ko
0   75   reserve         : les 3/4 de la mémoire
```

renvoie sur la pile un code d'erreur GEMDOS (négatif) ou la taille en octets allouée (positif).

Effet de bord: reserve re-alloue une table de gestion des variables dynamiques de 1028 octets. Voir les structures et l'instruction settable. De ce fait il faut au moins laisser 1028 octets libres avec reserve!

En l'absence d'instruction reserve, les 3/4 de la mémoire sont alloués comme pour un programme normal. Si on utilise reserve, cette instruction doit précéder toutes celles utilisant le dictionnaire (déclaration de variables, création de mots, instructions assembleur etc...), le plus simple est de mettre reserve sur la première ligne du source!

Pour déterminer la taille nécessaire, il suffit de lancer l'accessoire sans reserve et, au moment ou vous jugez que l'utilisation qu'il fait du dictionnaire est maximale, lui faire afficher la valeur de full (mémoire occupée dans le dictionnaire).

>accname

>accname chaine : permet de forcer un menu_register précoce.

En effet, les initialisations du FORTH prennent du temps et il se peut que l'AES ait déjà ouvert son bureau avant que l'instruction menu_register ne soit rencontrée. De ce fait, l'accessoire n'apparaît qu'après l'exécution d'une application et la mise à jour du menu du bureau.

Pour pallier ce défaut, cette directive force un appel système juste après appl_init alors que le dictionnaire n'est pas encore réservé, que les variables ne sont pas créées. De ce fait, vous devez malgré tout appeler menu_register par la suite, pour récupérer votre identifiant. Ce sera un appel simulé qui renverra la valeur attendue dans intout.

Le paramètre chaîne est celui de menu_register. Mettez bien le même !

7 Les programmes AUTO

Ce sont les programmes exécutés à partir du dossier AUTO avant l'installation de l'AES. De ce fait les instructions AES ne sont pas disponibles! Lors du lancement, aucune station VDI n'est ouverte. C'est fastopen qui le réalisera (une station physique interne pour les sorties de texte plus une station virtuelle pour l'utilisateur). Ceci permet de modifier la résolution de l'écran avant puis d'ouvrir proprement une station VDI aux nouvelles dimensions. Fastopen ouvre la pageo sur tout l'écran et l'efface par la même occasion, pour l'ins-

tant, la valeur renvoyée par fastopen est toujours 1 dans un programme AUTO, et donc inutile.

Fastclose referme les deux stations (attention, plus d'instructions VDI utilisables! plus de sorties de textes non plus (sauf avec les mots GEMDOS)). Ceci permet par exemple de commencer un programme en basse résolution puis de passer à la moyenne résolution, toujours en accédant proprement au VDI avec plusieurs couples fastopen et fastclose successifs. Fastclose n'est pas nécessaire en fin de programme car la routine de sortie s'en charge.

Rappelons que la souris (mousex, mousey, mousek, v_show_c, v_hide_c) est gérée par le VDI, donc accessible dans un programme AUTO dès que fastopen est appelé.

8 Programmes STE ou TT/Falcon

En utilisant COMPSTE.PRG et FORTHSTE.LIB on génèrera des programmes compatibles avec le STE (et peut-être avec le STF). C'est à dire qui évitent l'utilisation de codes réservés au MC68030 et au coprocesseur MC68882. Ces programmes tournent également sur TT et Falcon. Avec COMP.PRG et FORTH.LIB les programmes ne tourneront que sur TT/Falcon.

Il faudra également créer des fichiers FORTHSTE.INF et COMPSTE.INF pour fixer les paramètres généraux.

9 Instructions complémentaires

basepage

renvoie l'adresse de la Basepage du programme, sous l'éditeur il s'agit de la basepage du FORTH.

vq_aes

renvoie 0 si l'AES est disponible et -1 si le programme est lancé à partir du dossier AUTO. Sous l'interpréteur on obtient toujours 0.

_app

renvoie 0 si le programme est lancé en tant qu'accessoire (ACC) et une valeur non nulle si il est lancé en tant qu'application (PRG, TOS, TTP...). Sous l'interpréteur on obtient toujours une valeur non nulle.

10 Les flags GEMDOS de l'en-tête

Dans l'en-tête d'un programme se trouve un mot long à l'offset 22 déterminant le comportement de certaines fonctions du programme.

bit 0 : 1 pour n'effacer que la partie BSS de la RAM (fast load)
bit 1 : 1 pour charger le programme en TT Ram si disponible
bit 2 : 1 pour diriger Malloc vers la TT Ram si disponible
bit 3 : inutilisé
bits 4 & 5 : gèrent l'accès à l'espace mémoire du programme dans un environnement multitâche:

00 mémoire privée (réservée à l'application)
01 mémoire entièrement partagée
10 mémoire partagée uniquement en mode superviseur
11 mémoire partagée en lecture mais écriture privée

>prgflags

l p f >prgflags fixe trois valeurs du programme:
f : les flags du programme généré par le compilateur
p : le pourcentage de l'écran occupé par la fenêtre ouverte par fastopen.
une valeur de **zéro** indique qu'on génère un **programme TOS/TTP**.
l : le code langue (0=US, 2=FR, ...)

f et p sont simplement ignorés sous l'interpréteur.
Pour conserver la valeur par défaut d'un paramètre, on le remplace par -1.

Par défaut:
p = 100
l = code langue de la ROM du système
f = les flags fixés dans FORTH(STE).LIB, typiquement, on a:

FORTH.LIB flags à 01x111
FORTHSTE.LIB flags à 00x001

11 Version anglaise ou française

Selon que vous utilisez le compilateur français ou anglais, certaines chaînes (messages d'alerte) seront adaptées dans le programme généré.

Votre librairie FORTH.LIB ou FORTHSTE.LIB reste la même quelle que soit la version du compilateur mais le programme généré aura des textes différents.

Table des matières