ANNEXE 1

Organisation d'une disquette

Cette annexe est destinée à ceux qui veulent en savoir un peu plus sur la manière dont le BASIC128 gère une disquette. Les informations que nous vous donnons ici sont un peu plus techniques, mais leur connaissance n'est pas du tout nécessaire, même pour faire un bon usage des fichiers.

Organisation physique

Une disquette est constituée de pistes concentriques de 16 secteurs chacune. Chaque secteur contient 128 octets utiles sur une disquette en simple densité ou 255 octets utiles en double densité. Les pistes sont numérotées à partir de 0 et les secteurs à partir de 1.

Le découpage des pistes en 16 secteurs est fait au moment de l'initialisation de la disquette par le logiciel du BASIC. On dit alors que la sectorisation est logicielle; ce qui va de pair avec le fait que la disquette ou plus exactement le support en mylar ne possède qu'un seul trou d'index sur la circonférence. Ce trou unique sert à repérer la position du 1er secteur de chaque piste.

	QDD	SD 5'1/4	DF DD 5'1/4	DD 3'1/2
Nb pistes	25	40	2 × 40	80
N° piste	0-24	0-39	0-39	0-79
Nb secteurs	400	640	2 × 640	1 280
Taille secteur (octets)	128	128	255	255
Capacité Ko	50	80	2 × 160	320

Organisation physique des disquettes MO6

QDD : Quick Disk Drive
SD : Simple densité
DD : Double densité
DF : Double face

Le répertoire

Pour des raisons de commodité d'accès, le répertoire est situé sur la piste 20 de la disquette. On diminue ainsi la distance entre les données d'un fichier et les informations générales qui le concernent et qui sont inscrites dans le répertoire.
Le répertoire commence au secteur 3 et occupe une place de 14 secteurs.
Chaque fichier inscrit au répertoire a une zone réservée fixe de 32 octets. Cette zone contient les informations suivantes:

Nombre d'octets	*Contenu*
8	Nom du fichier, cadré à gauche, complété au besoin par les blancs à droite. Si le fichier a été supprimé par KILL, le premier octet contient 0. Si la zone du répertoire n'a pas encore été utilisée, le premier octet contient FF (hexadécimal).
3	Suffixe du nom du fichier (BAS,DAT,BIN,...), cadré à gauche, complété par des blancs à droite si nécessaire.
1	Type du fichier: 0: programme BASIC 1: fichier de données BASIC 2: programme en langage machine 3: fichier de texte.
1	Type des données: 0: les octets contiennent du binaire FF: les octets contiennent des caractères codés en ASCII

1 le n° du premier bloc attribué au fichier (les blocs sont numérotés à partir de 0).
2 le nombre d'octets occupés dans le dernier secteur du fichier.
8 commentaire associé au fichier (lors de SAVE, COPY, NAME).
8 inutilisés.

Toutes les informations écrites dans le répertoire sont entièrement gérées par le BASIC. En particulier, le nombre d'octets occupés dans le dernier secteur ne peut être écrit qu'au moment de la fermeture: une bonne raison pour ne pas oublier CLOSE.

La table d'allocation mémoire de la disquette

Nous avons vu avec l'instruction DIR que la mémoire allouée à chaque fichier ne se comptait pas en secteurs mais en blocs de 8 secteurs soit 1 k-octets en simple densité et 2 k-octets en double densité.
Le secteur n° 2 de la piste 20 contient la table d'allocation mémoire, c'est-à-dire des informations sur les blocs de la disquette
Dans cette table, chaque bloc est représenté par un octet. Attention: le 1er octet du secteur n'est pas utilisé. Le 1er bloc est représenté par le 2e octet, le 2e bloc par le 3e octet, etc.
La valeur de l'octet donne des indications sur l'occupation de bloc correspondant (1):

Valeur (hexadécimale)	*Signification*
entre 0 et 4B	Le bloc fait partie d'un fichier, l'octet contient le n° du prochain bloc du même fichier.
entre C1 et C8	Le bloc est le dernier d'un fichier, l'octet contient le nombre de secteurs du bloc occupés par le fichier auquel est ajouté la constante C0.
FE	Le bloc est réservé et ne peut pas être utilisé pour un fichier.
FF	Le bloc est libre.

Cette table d'allocation est mise à jour au fur et à mesure de l'écriture dans un fichier. Cependant, elle n'est pas mise à jour à chaque écriture dans un fichier. C'est la raison pour laquelle il est obligatoire d'utiliser UNLOAD lorsqu'on arrête une opération d'écriture de façon impromptue sans qu'il y ait eu de fermeture du fichier. UNLOAD recopie cette table sur la disquette et vous évite ainsi des ennuis si vous la remplacez par une autre dans le lecteur.

Comment voir le contenu de la disquette? DSKI$ et DSKO$

L'envie vous est peut-être venue de voir un peu plus précisément ce que contient la disquette, secteur par secteur. Vous pouvez le faire avec une fonction qui est l'analogue de PEEK( ) dans la mémoire centrale:

```
DSKI$0, 10, 3
```

vous donne le contenu (une chaîne de 128 ou 255 caractères) du secteur 3 de la piste 10 du lecteur 0. Avec une telle fonction, vous pourrez examiner à loisir comment sont enregistrées les données dans un fichier à accès direct.
Cette fonction a son pendant DSKO$ qui vous permet d'écrire ce qui vous plaît dans le secteur que vous choisissez. Aussi, pour écrire la chaîne de caractères SCT$, de longueur 128 ou 255 octets maximum, dans le secteur 3 de la piste 10 du lecteur 0, vous faites:

```
DSKO$0, 10, 3, SCT$
```

En utilisant la fonction DSKI$() ou l'instruction DSKO$, n'oubliez pas que les pistes sont comptées de 0 à 39 ou de 0 à 79 et les secteurs de 1 à 16. Un dernier conseil: n'employez pas DSKO$ sur une disquette qui contient des programmes ou des données qui vous sont chers, une fausse manœuvre pourrait rapidement lui être fatale. Entraînez-vous d'abord sur des copies, c'est moins risqué!

Exécution automatique d'un programme

Si vous utilisez souvent le même programme, il est agréable de pouvoir lancer son exécution immédiatement après la mise en service.

Pour ce faire, il vous faut sauvegarder le programme en question sous le nom "AUTO.BAT".
A la mise en service, quand vous choisirez l'option 1 du menu, l'interpréteur BASIC va chercher sur la disquette le programme AUTO.BAT, le charger et l'exécuter.
Pratique, non?

Cache disque

Le troisième paramètre de FILES permet de réserver une zone tampon en mémoire en vue d'optimiser les accès physiques à la disquette ou au QDD en lecture et en écriture.
Le principe est de précharger en mémoire à la lecture une piste entière au lieu d'un seul secteur, de façon à ce que l'accès au secteur suivant qui se trouve en mémoire vive soit instantané. A l'écriture, le cache disque mémorise les secteurs à écrire et les transfère par paquets sur la disquette. En contrepartie, pour avoir l'assurance que tous les fichiers ont été transférés, il est nécessaire de faire un CLOSE.
L'instruction DSKO$ ne passe pas par le cache.
Le paramètre à introduire est le nombre de pistes conservées en mémoire vive. Une piste représente 2 Ko en simple densité et 4 Ko en double densité. La taille maximum du cache disque est de 25 pistes. Par défaut, les valeurs sont de 0 piste avec un lecteur de disquettes et de 3 pistes avec un QDD.

ANNEXE 2

Organisation de la mémoire utilisateur

Géographie générale de la mémoire

L'espace mémoire est organisé de la manière suivante:

adresse	
0000	mémoire écran forme
	mémoire écran couleur
1FFF	
2000	
	registres du moniteur
20FF	
2100	tampons Basic
	mémoire utilisateur non commutable
5FFF	pile système
6000	
	banque n° 1 2 3 4 5 6
9FFF	
A000	
	contrôleur disque
	et E/S
AFFF	
B000	
	ROM Basic 128 ou Basic 1
EFFF	
F000	
	moniteur
FFFF	

La zone 2000 à 20FF est la page zéro du moniteur; elle contient tous les registres de travail.
La zone 2100 à 5FFF, non commutable, est utilisée par l'interpréteur

BASIC pour les zones tampon, la manipulation des données et tous les pointeurs sur les variables et le programme.
La zone des banques de mémoire A000 à DFFF contient le programme, les variables et les données chaînes de caractères soit 96 k-octets disponibles.

Connaître l'espace mémoire disponible: FRE()

La fonction FRE permet de connaître les différents paramètres de l'espace mémoire restant:

FRE(2) donne le volume en octets de l'espace libre dans les banques de mémoire, c'est-à-dire libre pour programme et données.

FRE(1) donne le volume de l'espace libre dans la mémoire hors banques, c'est-à-dire libre pour les tampons et les pointeurs.

FRE(0) donne le volume de l'espace total disponible:
FRE(0) = FRE(1) + FRE(2)

FRE(X$) donne le volume en octets de l'espace libre pour les chaînes de caractères.

Examen et modification d'octets en mémoire: PEEK et POKE

Il est possible de connaître la valeur d'un octet quelconque en mémoire par:

PEEK(adresse)

PEEK(&H4000) donne le contenu de l'octet situé à l'adresse 4000 (hexa).
La modification d'un octet se fait par:
POKE adresse,valeur
POKE&H4000,128 dépose la valeur décimale 128 (80 hexa) dans l'octet d'adresse 4000 (hexa).
Pour choisir la banque de mémoire à examiner, il faut utiliser l'instruction supplémentaire BANK.

BANK 2 oriente sur la banque n° 2 toutes les instructions qui agissent directement sur la mémoire.
Par la suite, l'exécution de PEEK(&H7000) donne le contenu de l'octet d'adresse 7000 (hexa) de la banque n° 2.
A l'initialisation du micro-ordinateur, c'est la banque d'ordre le plus élevé qui est sélectionnée, c'est-à-dire la banque n° 6.

Pour choisir la banque d'ordre le plus élevé, sans avoir à se préoccuper du nombre de banques en présence, il faut faire:

BANK 0

Le résultat est le même qu'après l'initialisation.
La banque de mémoire sélectionnée par BANK est alors implicitement désignée pour les instructions et fonctions suivantes: PEEK, POKE, LOADM, SAVEM, EXEC, DEF USR, CLEAR.

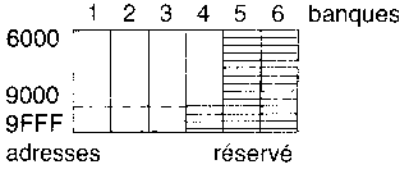
Réservation de zones mémoire: CLEAR

Toutes les modifications de l'espace disponible sont faites par l'instruction CLEAR.

1. *zone chaîne de caractères*
La modification la plus fréquente concerne l'espace disponible pour le stockage des chaînes de caractères.
Le volume de cet espace est fixé à l'initialisation à 300 caractères.
On le modifie avec le 1er argument de CLEAR.
CLEAR 2000 fixe à 2000 caractères cet espace

2. *réservation dans les banques de mémoire*
La réservation de mémoire est destinée à stocker des données en binaire ou des modules en langage machine. Cette réservation en zone fixe permet de les utiliser depuis un programme en BASIC.
L'adresse de réservation moins un, est fixée par le 2^{e} paramètre de CLEAR.
CLEAR,&H8FFF réserve la mémoire depuis l'adresse &H9000

jusqu'à &H9FFF dans la banque de mémoire courante et réserve entièrement les banques d'ordre supérieur.



la zone hachurée est réservée par:

BANK 4 : CLEAR,&H8FFF

3. *réservation hors de la zone commutable*
Elle est fixée par le 4e argument de CLEAR.
Elle n'affecte que l'espace compris entre &H2100 et &H5FFF.
CLEAR,,,&H4FFF réserve la zone comprise entre &H5000 et &H5FFF.

4. *réservation de caractères utilisateur*
Elle est fixée par le 3e paramètre de CLEAR.
Il suffit d'indiquer le nombre de caractères utilisateur maximum utilisés. Chaque caractère occupe huit octets.
CLEAR ,,10 fixe à 10 le nombre de caractères utilisateur et réserve la place nécessaire dans la zone des banques de mémoire.

Les réservations une fois fixées par CLEAR ne sont pas modifiées, même au chargement et à l'exécution d'un nouveau programme. De nouvelles réservations ne sont possibles que par l'exécution d'une instruction CLEAR avec les arguments appropriés.

Sauvegarde et chargement d'une zone mémoire: SAVEM, LOADM

Ce couple d'instructions est destiné à conserver en fichier une zone précise de la mémoire, contenant des données ou un module en binaire.
La sauvegarde s'écrit sous la forme:
SAVEM descr. de fichier, adr. début, adr. fin, adr. d'exécution
Le quatrième argument, adresse d'exécution, n'est utile que pour un programme en binaire, mais il est obligatoire.
Le chargement d'une zone s'écrit sous la forme:
LOADM descr. de fichier, déplacement ,R
Le deuxième argument, déplacement, quand il existe, indique le nombre d'octets dont il faut translater la zone mémoire par rapport à son adresse lors de la sauvegarde.
L'option ,R indique qu'il faut exécuter le module à l'adresse d'exécution sauvegardée dans SAVEM, en tenant compte du déplacement s'il y a lieu.
exemple de sauvegarde de l'écran (dessin et couleur):
(cf. manuel référence IV-181,182)

ANNEXE 3

Représentation des variables en mémoire

La fonction VARPTR donne l'adresse de la variable donnée en argument et permet donc de connaître et de modifier son contenu.

Elle positionne automatiquement sur la banque de mémoire qui contient la variable en question.
Elle est utilisée le plus souvent pour interfacer un programme en BASIC avec un module en langage machine.

Variables entières

Ces variables sont représentées en binaire sur deux octets en complément à deux.
Le premier octet contient les bits de poids fort et le second les bits de poids faible.
Le bit de poids le plus fort du premier octet représente le signe:

0 nombre positif ou nul
1 nombre négatif

Les quinze autres bits permettent donc des valeurs absolues jusqu'à 2^15 soit 32 768.
Les nombres entiers s'étendent donc de −32768 à 32767.
La fonction VARPTR rend l'adresse du premier octet.

Exemple:

```
X%=12
?PEEK(VARPTR(X%)),PEEK(VARPTR(X%)+1)
0     12
```

Variables réelles ou simple précision

Ces variables sont représentées sous forme flottante (mantisse + exposant) sur 4 octets.

Le premier octet contient l'exposant.
L'exposant est noté en binaire signé et décalé de &H80.
Ainsi &H81 correspond à 2^1
et &H7E correspond à 2^−2.
Un exposant nul signifie un nombre nul.

Le 2e octet contient le bit de poids fort de la mantisse.
Le 3e octet contient le bit de poids moyen de la mantisse.
Le 4e octet contient le bit de poids faible de la mantisse.
Le bit de poids le plus fort de la mantisse représente le signe du nombre. Les autres bits représentent la mantisse en binaire, sauf le 1er bit de celle-ci toujours égal à 1, et qui donc est omis. La valeur de la mantisse peut donc s'écrire, en base 2:

0.1mantisse

En conséquence, le plus petit nombre positif vaut 2.9874 E-39 et le plus grand vaut 1.7014 E+38.
La fonction VARPTR rend l'adresse du premier octet, donc de l'exposant.

Exemples:

```
NB=3
ADNB=VARPTR(NB)
FOR I=0 TO 3: PRINT PEEK(ADNB+I);: NEXT
130 64 0 0
```

soit un exposant égal à 2^(130−128)=2^2
et une mantisse égale à 0.11000000 00000000 00000000

```
NB=-1
ADNB=VARPTR(NB)
FOR I=0 TO 3: PRINT PEEK(ADNB+I);: NEXT
129 128 0 0
```

soit un exposant égal à 2^(129−128)=2^1
et une mantisse égale à 0.10000000 00000000 00000000
et un signe négatif

Variables en double précision

Ces variables sont représentées sous forme flottante, de la même manière que les variables en simple précision mais sur 8 octets.
Le premier octet contient l'exposant, codé comme en simple précision, les sept suivants contiennent la mantisse.
Les valeurs minimum et maximum sont les mêmes que celles des variables en simple précision.

Variables chaînes de caractères

Ces variables sont constituées de deux parties, un descripteur sur trois octets et le contenu de la chaîne proprement dit.
Le premier octet du descripteur contient la longueur de la chaîne.
Les deux octets suivants contiennent l'adresse logique de la chaîne dans la zone chaîne de caractères.
L'adresse de la zone chaîne est contenue aux adresses &H2122 et &H2123. Le numéro de la banque mémoire qui contient la zone chaîne est indiqué en &H2124.
L'adresse physique de la chaîne est obtenue en ajoutant à l'adresse de la zone chaîne l'adresse logique de la chaîne.
Si l'adresse physique dépasse &HA000, il faut y retrancher &HA000 et y ajouter 6000 pour obtenir la bonne valeur, ceci pour tenir compte d'un chevauchement de la zone chaîne entre deux banques de mémoire.
La fonction VARPTR rend l'adresse du premier octet du descripteur.

Exemple:

```
5 ' CAS S MPLE SANS CHEVAUCHEMENT
10 CH$= "ABCDEF"
20 ADCH=VARPTR(CH$)
30 LCH=PEEK(ADCH): ADLOG=256*PEEK(ADCH+1)+PEEK(ADCH+2)
40 PRINT LCH,AD. OG
50 ADZCH=256*PEEK(&H2122)+PEEK(&H2123)
60 BCH=PEEK(&H2124)
70 PR NT HEX$(ADZCH),BCH
80 BANK BCH
90 PR NT CHR$(PEEK(ADZCH+ ADLOG))
RUN
6          0
9ED3       6
```

ANNEXE 4

Modules et fonctions en langage machine

Cette annexe intéresse surtout ceux qui ont quelques notions de programmation du microprocesseur 6809 en langage machine. Elle suppose une connaissance minimale de ce microprocesseur.
Il existe deux solutions pour utiliser le langage machine: les modules exécutables par EXEC, et les fonctions utilisateur USR.

L'instruction EXEC avec paramètres

La première chose à faire avant tout est de réserver la place nécessaire aux instructions machine. On utilise pour cela CLEAR qui réserve la zone mémoire utile:

```
CLEAR,&H9EFF
```

fixe l'adresse la plus haute du BASIC à &H9EFF, dans la banque courante, par défaut dans la dernière. Il reste donc 256 octets libres jusqu'au fond de la banque de mémoire en &H9FFF.

La deuxième chose est d'y déposer les instructions par des POKE, ou bien par LOADM si ces instructions ont été conservées en fichier, prêtes à être exécutées.

Enfin, on exécute le module en langage machine par:
EXEC &H9F00

si l'adresse de début du module est en &H9F00.

Toutefois, avant d'écrire le module en langage machine, il faut respecter certaines règles si l'on veut revenir à la suite dans le programme Basic:
— le module doit se terminer par RTS,
— les registres S (*Stack*) et DP (*Direct Page*) ne doivent pas avoir été modifiés. Pour pouvoir les utiliser dans le module, il faudra les sauvegarder à l'entrée et les récupérer à la sortie.

Il est possible de passer des paramètres au module en langage machine. Ces paramètres doivent suivre l'adresse, séparés par des virgules.
La récupération des paramètres dans le module se fait en appelant l'intrepréteur Basic.
Les points d'entrées utiles pour analyser les paramètres sont les suivants:

EFE6 RESBAN	routine à appeler avant le premier appel à l'une des six routines de lecture de paramètres de façon à commuter BASIC vers la RAM du progamme. A appeler aussi après PTRGET pour continuer l'analyse des paramètres, car PTRGET commute la banque contenant la variable
EFE9 LITINT	lit un entier signé, résultat dans le registre X
EFEC LITADR	lit un entier non signe (adresse), résultat dans le registre X
EFEF LITSGN	lit un réel simple précision, le registre X pointe sur le résultat
EFF2 LITSTR	lit une chaîne, en sortie le registre B contient la longueur, le registre X pointe sur le résultat
EFF5 FRMEVL	lit une donnée quelconque, en sortie on obtient le même résultat qu'avec une fonction USR, voir plus loin
EFF8 PTRGET	rend l'adresse du paramètre, en sortie le registre A contient le numéro de banque, le registre X pointe le contenu, l'octet VALTYP($6105) contient le type du résultat (2, 3, 4 ou 8)
21C6 IFEND	teste la fin de la ligne, en sortie le drapeau Z du registre d'état est à 1 si la fin est atteinte

Votre programme de récupération de paramètres doit être en RAM fixe. Si votre EXEC lance des routines d'interruption, votre routine de réception d'interruption doit être en RAM fixe.

Les fonctions USRn

On peut définir dix fonctions utilisateur en langage machine par l'instruction:

DEF USRn = adresse
(n est un chiffre (de 0 à 9)
adresse désigne l'adresse mémoire à laquelle commence la fonction.)

L'appel de la fonction se fait alors comme pour une fonction normale:
variable = USRn (expression)

L'avantage principal des fonctions USR est qu'elles permettent de passer des valeurs et d'en retourner au programme Basic par le résultat fourni.
Nous allons examiner comment ces valeurs sont passées aux fonctions utilisateur.
A l'appel de la fonction, l'accumulateur A contient le type de la donnée:
2 : nombre entier
3 : chaîne de caractères
4 : nombre en simple précision

8 : nombre en double précision

le registre d'index X pointe sur la valeur :
2,X : nombre entier
0,X : nombre en simple ou double précision
0,X : descripteur pour une chaîne

Dans le cas où la donnée est une chaîne de caractères, l'accumulateur B contient la longueur de la chaîne, le registre U pointe sur le 1er caractère de la chaîne.
Il faut remarquer que, sauf pour les chaînes de caractères, l'accumulateur A contient en même temps la longueur de la donnée en octets.
Rappelons que :
— les entiers sont codés sur deux octets : 1er octet poids forts, 2e octet poids faibles,
— les nombres en simple précision sont codés sur quatre octets : le premier contient l'exposant, les suivants la mantisse, poids forts en tête,
— les nombres en double précision sont codés sur huit octets : le premier contient l'exposant, les suivants la mantisse.
— le descripteur d'une chaîne de caractères comporte trois octets : le premier indique la longueur de chaîne, les deux autres adresse du 1er caractère.
On peut donc atteindre la longueur de la chaîne soit par l'accumulateur B, soit par le descripteur.
La valeur (nombre ou descripteur de chaîne) est toujours déposé au même endroit (FAC).

A la sortie du module assembleur, il faut également rendre les contenus de l'accumulateur A et du registre X compatibles avec le type de la variable qui va recevoir la valeur de sortie.
En sortie, X doit donc pointer sur la même adresse qu'à l'entrée.
Dans tous les cas, les registres S et DP doivent être rendus avec les mêmes contenus qu'à l'entrée.

Exemple : Ajouter 256 à un entier.
Après avoir réservé la place (largement suffisante) en ligne 20, on lit les différentes instructions machine, introduites en DATA, et on les dépose en mémoire (lignes 40 à 70).

Pour pouvoir mettre un nombre variable d'instructions en DATA, la dernière valeur, fictive, est supérieure à 255 (&H100) et sert au test de fin de lecture.

Le module lui-même ne comporte que deux instructions machine (on ne peut faire plus simple) :

```
6C 02 INC 2,X ; incrémenter en (X)+2
39 RTS ; retour
```

L'incrémentation de l'octet de poids fort revient à ajouter 256. Notez que pour les entiers, la valeur n'est pas en (X) mais (X)+2.

La nouvelle variable R% reçoit alors la nouvelle valeur qui est affichée avec la précédente.

```
10 ' AJOUTER 256
20 CLEAR,&H9EFF
30 DEFUSR1=&H9F00
40 ADR=&H9F00
50  READ OCT: IF OCT>255 THEN 80
60  POKE ADR,OCT
70  ADR=ADR+1: GOTO 50
80 ' UTILISATION
90 N%=10
100 R%=USR1(N%)
110 PRINT N%,R%
120 DATA &H6C,&H02,&H39,&H100
```

ANNEXE 5

Liste des mots réservés

ABS
AND
ASC
ATN
ATTRB
AUTO

BACKUP
BANK
BEEP
BOX

CDBL
CHAIN
CHR$
CINT
CIRCLE
CLEAR
CLOSE
CLS
COLOR
COMMON
CONSOLE
CONT
COPY
COS
CRUNCH$
CSNG
CSRLIN
CVD
CVI
CVS

DATA
DEF
DEFDBL
DEFINT
DEFSNG
DEFSTR
DELETE
DENSITY
DEVICE
DIM
DIR
DO
DOS
DSKI$
DSKF
DSKO$

ELSE
END
EOF
EQV
ERL
ERR
ERROR
EVAL
EXEC
EXIT
EXP
FIELD
FILES
FIX
FKEY
FN
FOR
FRE

GET
GO
GRS

HEAD
HEX$

IF
IMP
INKEY$
INMOUSE
INPEN
INPUT
INSTR
INT
INTERVAL

KEY
KILL

LEFT$
LEN
LET
LINE
LIST
LOAD
LOC
LOCATE
LOF
LOG
LOOP
LSET

MAX
MERGE
MKD$
MKI$
MKS$
MID$
MIN
MOD
MOTOR
MOUSE
MTRIG

NAME
NEW
NEXT
NOT

OCT$
OFF
ON
OPEN
OR

PAINT
PALETTE
PATTERN
PEEK
PEN
PLAY
POINT
POKE
POS
PRINT
PSET
PTRIG
PUT

READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
ROT
RSET
RUN

SAVE
SCREEN
SEARCH
SGN
SHOW
SKIPF
SPACE$
SPC(
SQR
STEP
STICK
STOP
STRIG
STRINGS
STR$
SUB
SWAP

TAB(
TAN
THEN
TO
TRACE
TROFF
TRON
TUNE
TURTLE

UNLOAD
UNMASK
USING
USR

VAL
VARPTR

WAIT
WINDOW
WRITE

XOR

ZOOM

ANNEXE 6

Messages et codes d'erreurs

1 **Next Without For**
Une instruction NEXT a été rencontrée avant le FOR correspondant.

2 **Syntax Error**
Erreur de syntaxe: l'instruction n'est pas connue ou mal rédigée.

3 **Return Without Gosub**
L'instruction RETURN a été rencontrée sans qu'il y ait appel par GOSUB.

4 **Out Of Data**
Il n'y a plus assez de données dans les lignes de DATA pour le READ à exécuter.

5 **Illegal Function Call**
Une fonction ou une instruction est appelée avec des valeurs interdites.

6 **Overflow**
La valeur numérique obtenue est trop grande.

7 **Out Of Memory**
Il n'y a plus assez de place en mémoire centrale.

8 **Undefined Line Number**
On ne peut pas faire de GOTO ou de GOSUB à une ligne qui n'existe pas.

9 **Subscript Out Of Range**
L'indice d'un élément de tableau dépasse la valeur maximum ou est négatif.

10 **Duplicate Definition**
Il n'est pas possible de déclarer deux fois le même tableau.

11 **Division By Zero**

12 **Illegal Direct**
L'instruction ne peut pas être utilisée en mode direct.

13 **Type Mismatch**
Il n'est pas possible de mettre un nombre dans une variable chaîne et inversement.

14 **Out Of String Space**
Il n'y a plus assez de place en mémoire centrale dans la zone des chaînes de caractères.

15 **String Too Long**
Une chaîne de caractères ne peut pas dépasser 255 caractères.

17 **Can't Continue**
La commande CONT ne permet pas de poursuivre l'exécution du programme.

18 **Undefined User Function**
La fonction USR utilisée n'a pas été définie.

19 **No Resume**
Il n'y a pas d'instruction RESUME dans la partie qui traite les erreurs (ON ERROR GOTO).

20 **Resume Without Error**
L'instruction RESUME est rencontrée alors qu'il n'y a pas eu d'erreur.

21 **Undefined Error**
Erreur non définie (simulée par ERROR).

22 **Missing Operand**
Il manque un opérande dans une opération comme addition, soustraction,...

23 **For Without Next**
Aucun NEXT n'a été rencontré après l'exécution de FOR.

24 **Can't Exit**
L'instruction EXIT est utilisée en dehors d'une boucle.

25 **Do Without Loop**
Aucun LOOP n'a été rencontré après l'exécution de DO.

26 **Loop Without Do**
L'instruction LOOP est rencontrée alors que le DO correspondant n'a pas été exécuté. Erreurs portant sur les fichiers.

50 **Bad File Number**
Ce numéro de fichier n'est pas utilisé.

51 **Bad File Mode**
Le mode d'utilisation du fichier est incorrect.

52 **File Already Open**
Un fichier déjà ouvert ne peut pas être ouvert à nouveau.

53 **Device I/O Error**
Problème matériel d'accès au périphérique.

54 **Input Past End**
Il n'est pas possible de lire au-delà de la fin du fichier.

55 **Bad File Descriptor**
Le descripteur de fichier n'est pas correct.

56 **Direct Statement In File**
Le fichier en cours de chargement contient une commande d'exécution directe.

57 **File Not Open**
Le fichier sur lequel on veut lire ou écrire n'est pas ouvert.

58 **Bad Data In File**
Les données lues dans un fichier ne sont du type de celles qui sont attendues.

59 **Device In Use**
Le périphérique est déjà en fonctionnement.

60 **Device Unavailable**
Le périphérique n'est pas disponible.

61 **Protected Program**
Le programme est protégé et ne peut pas être listé ou sauvegardé.

62 **File Not Found**
Le fichier n'existe pas sur la disquette.

63 **Disk Full**
Il n'y a plus de place disponible sur la disquette.

64 **Too Many Open Disk Files**
On ne peut pas ouvrir plus de fichiers qu'il n'est indiqué dans l'instruction FILES.

65 **Directory Full**
Il n'y a plus de place sur le répertoire.

66 **File Already Exists**
Le nom du nouveau fichier dans NAME correspond à un fichier existant.

67 **Field Overflow**
La somme des longueurs des variables de champ dans un FIELD dépasse la longueur de l'enregistrement.

68 **String Fielded**
Une variable de champ ne peut être affectée que par LSET ou RSET.

69 **Bad Record Number**
Le numéro d'enregistrement dans PUT ou GET n'est pas possible.

70 **Bad File Structure**
Les indications concernant les blocs du fichier dans la table d'allocation ne sont pas cohérentes.

71 **No Disk**
Le lecteur de disquettes est vide.

72 **Disk Write Protected**
La disquette est protégée en écriture.

73 **Out Of Fielded Buffers**
Il n'y a plus assez de place dans la zone réservée aux tampons des fichiers à accès direct.

74 **End Of Record**
La lecture ou l'écriture dans un fichier à accès direct dépasse la fin de l'enregistrement.

75 **Verification Failure**
La relecture après écriture (VERIFY ON) indique une différence.

76 **Unreadable diskette**
La disquette n'est pas formatée.

78 **Bad Picture**
L'image ne peut pas être chargée ou sauvegardée.

ANNEXE 7

CODE ASCII

Code décimal		Code décimal		Code décimal		Code décimal	
000	NUL	032	Espace	064	@	096	
001		033	!	065	A	097	a
002	STOP touche STOP clavier	034	"	066	B	098	b
003	BREAK (CNT C)	035	#	067	C	099	c
004		036	$	068	D	100	d
005		037	%	069	E	101	e
006		038	&	070	F	102	f
007	SONNETTE	039	'	071	G	103	g
008	BS (Recul arrière) ← clavier	040	(	072	H	104	h
009	HT (Tabulation H) → clavier	041	)	073	I	105	i
010	LF (Saut de ligne) ↓ clavier	042	*	074	J	106	j
011	VT (Tabulation V) ↑ clavier	043	+	075	K	107	k
012	FF (Saut de page) RAZ clavier	044	,	076	L	108	l
013	CR (Retour à la ligne) ENTREE	045	-	077	M	109	m
014	SO Mode semi-graphique	046	.	078	N	110	n
015	SI Mode alphanumérique	047	/	079	O	111	o
016		048	0	080	P	112	p
017	DC1 Clignotement curseur	049	1	081	Q	113	q
018	DC2 Répétition	050	2	082	R	114	r
019		051	3	083	S	115	s
020	DC4 Arrêt curseur	052	4	084	T	116	t
021		053	5	085	U	117	u
022	SS2 Touche caractères accentués	054	6	086	V	118	v
023		055	7	087	W	119	w
024	CAN Efface la fin de la ligne	056	8	088	X	120	x
025		057	9	089	Y	121	y
026		058	:	090	Z	122	z
027	ESC Appel d'une séquence	059	;	091	[	123	{
028	INS (INS clavier)	060	<	092	\	124	⁞
029	DEL (EFF clavier)	061	=	093	]	125	}
030	RS Touche clavier	062	>	094	↑	126	~
031	US Séparateur d'article	063	?	095	—	127	■

ANNEXE 8

Index thématique

Cet index thématique vous permet de retrouver une instruction Basic dans le domaine qui vous intéresse. Le numéro de page correspondant à l'instruction figure dans l'index général.

FILES
GET#
INPUT#
KILL
LINEINPUT#
LSET
MOTORON
MOTOROFF
NAME
OPEN
PRINT#
PRINT# USING
PUT#
RSET
SKIPF
UNLOAD
VERIFYON
VERIFYOFF
WRITE#

Fonctions

CVD
CVI
CVS
DSKF
DSKI$
EOF
LOC
LOF
MKD$
MKI$
MKS$
POS

AFFICHAGE CARACTERE

Instructions

ATTRB
CLS
COLOR
CONSOLE
DEFGR$
LOCATE
PALETTE
PRINT
PRINT USING
SCREEN

Fonctions

CSRLIN
GR$
POS
SPC
TAB
SCREEN

AFFICHAGE GRAPHIQUE

Instructions

BOX
BOXF
CIRCLE
CIRCLEF
GET
LINE
PAINT
PATTERN
PSET
PUT
TUNE
WINDOW

Fonction

POINT

TORTUES

Instructions

FWD
HEAD
INPUTTURTLE
ROT
SHOW
TRACE
TURTLE
ZOOM

Fonctions

HEAD
ROT
SHOW
TRACE
ZOOM

AUTRES ENTREES-SORTIES

Instructions

INMOUSE
INPEN
INPUTMOUSE
INPUTPEN
ONPEN ...GOTO
ONPEN ...GOSUB
PEN
PLAY

Fonctions

MTRIG
PTRIG
STICK
STRIG

ANNEXE 9

Index général

Le premier renvoi de page concerne la partie Références, le deuxième, la partie Initiation.

Sommaire

Installation

Guide du BASIC

Première partie: Initiation

Deuxième partie : Références

Généralités

Annexes

THOMSON se réserve le droit d'apporter des modifications aux produits et aux programmes décrits dans ce manuel à tout moment et sans avoir à le notifier. Les informations contenues dans ce document ne constituent en aucun cas un engagement de la part de THOMSON.

Imprimé en France par Pollina, 85400 Luçon - N° 8214
Dépôt légal : Juin 1986