SMAKY NEWS

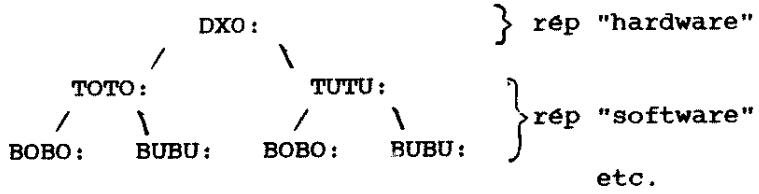
No 13 — 5 février 1981

SAMOS REVISION 1-F

Cette nouvelle révision est destinée exclusivement aux systèmes 64K. En effet, les nouveautés ont nécessité davantage de place en mémoire. SAMOS 1-F ne tient donc plus dans la limite des 4 K-bytes de ROM (10000-20000).
Il est absolument impératif de remplacer tous les anciens fichiers .SY par les nouveaux ou tout au moins de les supprimer, car les anciens drivers d'imprimante LP.SY et les anciens fichiers d'erreurs ER.SY détruisent une partie du nouveau SAMOS 1-F s'ils sont utilisés.

AUGMENTATION DU NOMBRE DE FICHIERS

La principale nouveauté est la possibilité d'avoir plus de 32 fichiers sur une disquette tout en conservant la compatibilité avec les anciennes disquettes.
Ceci est réalisé de la manière suivante:
Un nouvel appel ?CDIR permet de créer des fichiers répertoires qui peuvent contenir chacun 32 fichiers. Parmi ces fichiers, il peut y avoir également des fichiers répertoires qui peuvent contenir aussi d'autres fichiers répertoires etc... On peut donc créer une structure arborescente de répertoires et de sous-répertoires.



Traduction du schéma ci-dessus: sur la diquette en DX0 : il y a notamment deux fichiers répertoires TOTO.DR et TUTU.DR qui contiennent chacun entre autres deux fichiers sous-répertoires BOBO.DR et BUBU.DR (l'utilisation du même nom n'est pas interdite).

Appel ?CDIR

Cet appel crée un fichier dont le nom est pointé par DE, et dont la taille en blocs est dans BC. Si BC vaut zéro, la taille du fichier sera la moitié de la plus grande place disponible sur la disquette. L'appel remplit avec des zéros les trois premiers blocs du fichier, ce qui correspond à un directoire vide. Contrairement à l'appel ?CREATE, le fichier est déjà fermé à la fin de l'appel. Ceci permet de créer directement un fichier vide de longueur BC. Cet appel admet n'importe quelle extension au nom du fichier, mais dans son utilisation fondamentale qui consiste à créer des répertoires, l'extension devra obligatoirement être .DR.

Nom du fichier (pointé par DE dans les appels)

Rappelons tout d'abord l'ancienne définition: DXn : NOM.EX [RES]

avec:
- DXn : adresse du drive (DX0 : par défaut)
- NOM nom du fichier
- .EX extension au nom du fichier (optionnel)
- [RES] réservation de place pour la création de fichiers (moitié du plus grand trou par défaut)

Les utilisateurs de SAMOS ont l'habitude de spécifier soit dans leurs programmes, soit dans leurs lignes de commande l'adresse du drive DX1 : . Ce paramètre correspond en fait déjà à un répertoire particulier: un répertoire hardware. Maintenant, il ne seront plus limités à un répertoire hardware, mais ils pourront spécifier simplement plusieurs répertoires. En premier lieu un répertoire hardware, suivi de répertoires software constituant le lien au répertoire que l'on veut accéder. Cela nous donne donc la nouvelle définition suivante:

REP : NOM.EX [RES]

avec: REP : lien d'accès à un répertoire (DX0 : par défaut)

Imaginons que l'on veut accéder à un fichier TITI.SR qui, dans notre petit schéma plus haut, se trouverait dans le sous-répertoire BOBO.DR du répertoire TOTO.DR sur le drive DX0 : . Ce que nous appelons nom du fichier serait alors:

DX0 : TOTO : BOBO : TITI.SR

A titre d'exemple, imaginons maintenant que nous voulons créer le fichier TITI.SR dans ce sous-répertoire BOBO.DR, mais que nous n'avons pas encore créé le répertoire TOTO.DR ni le sous-répertoire BOBO.DR. Nous procéderons de la manière suivante:

.W ?CDIR avec DE pointant: DX0 : TOTO.DR [300]

Cette opération crée un répertoire TOTO.DR de 300 blocs. Nous ferons ensuite:

.W ?CDIR avec DE pointant: DX0 : TOTO : BOBO.DR [100]

Cette opération crée DANS TOTO.DR un sous-répertoire BOBO.DR de 100 blocs. Nous ferons finalement:

.W ?CREATE avec DE pointant: DX0 : TOTO : BOBO : TITI.SR [8]

Cette operation crée notre fichier TITI.SR avec un taille initiale de 8 blocs.

REMARQUE:
Dans notre exemple, la réservation est spécifiée dans le nom du fichier. BC devra donc être égal à zéro. On aurait pu évidemment spécifier ces réservations dans BC.

Notons encore, que la structure que nous venons de décrire est valable pour tous les appels SAMOS. On peut donc par exemple compresser un répertoire, faire un transfert d'un répertoire à un autre etc.

AUGMENTATION DU NOMBRE DE CANAUX

Le nombre de fichiers disque ouverts simultanément a été porté à 8 en écriture et 8 en lecture au lieu de 4 précédemment. Cependant, le nombre de buffers à disposition pour l'accès par bytes est resté de quatre. On ne pourra donc pas avoir plus de 4 fichiers ouverts simultanément en accès par bytes.

MODIFICATION DE L'APPEL ?LIST

Cet appel délivre une information supplémentaire. Deux bytes de plus après les deux bytes qui donne la place libre sur la disquette, donnent la taille du plus grand trou. La seule incidence à remarquer est que la taille de la plus longue liste possible est augmentée de deux bytes.

LES DEUX VERSIONS DU BASIC REVISION 2-5

Le BASIC a été entièrement repris. Les fautes connues ont toutes été corrigées. D'autre part, plusieurs nouveaux ordres ont été rajoutés.

Il existe maintenant deux versions du BASIC:

BINBASIC:
Cette version correspond à l'ancienne revision 2-4. Elle contient un "package" mathématique binaire avec 7 chiffres. Cette version est la plus rapide.

BCDBASIC:
Cette nouvelle version contient un nouveau "package" mathématique BCD (binaire codé décimal) qui calcule avec 13 chiffres. Cette version est plus lente en calcul, mais elle permet d'aborder des problèmes commerciaux.

Ces deux versions sont cent pour cent compatibles du point de vue des ordres. Il faut cependant noter que les variables de la version BCD occupent 8 bytes au lieu de 4 dans la version binaire. Ceci signifie qu'un programme que l'on exécutera sur la version BCD prendra plus de place mémoire. Ainsi, un programme développé avec la version binaire peut ne plus passer avec la version BCD. On peut avoir un manque de place soit au niveau de la ligne, soit au niveau du programme.

OPTION POUR SMAKY 64K

Pour les SMAKY avec 64k de mémoire, le BASIC utilise maintenant la zone mémoire allant de 25000 à 37777 pour ses nombreux buffers internes, le stack, et une partie de ses variables en RAM. Ceci sauvegarde un maximum de mémoire pour l'utilisateur et permet un niveau d'imbrication considérablement plus élevé (plus de 400 imbrications).

ON ERROR THEN xxxx

La mise au point de cette ordre a nécessité de refaire entièrement la gestion des erreurs du BASIC. Maintenant cet ordre fonctionne bien et pour n'importe quelle erreur. L'utilisateur doit cependant savoir qu'une erreur provoque la réinitialisation du stack. Ceci a pour conséquence que l'on ne peut pas gérer des erreurs à l'intérieur d'une routine à moins que la gestion de l'erreur ne soit capable de remplacer l'ordre RETURN par un GOTO xxxx.

NOUVELLE VARIABLE INTERNE ERO

Cet variable permet une gestion plus efficace des erreurs avec l'ordre ON ERROR THEN xxxx. Cette variable contient le numéro de la dernière erreur produite. Elle vaut zéro avant la première erreur. En voici la liste:

erreurs système	erreurs BASIC
101 = WRITE PROTECT FILE	1 = UNKNOW LINE
102 = READ PROTECT FIL	2 = ARITHMETIC
104 = PERMANENT FILE	3 = SYNTAX
105 = LINE TOO LONG	4 = RETURN
106 = END OF FILE	5 = DATA
107 = FILE END OVERFLOW	6 = NEXT
108 = FILE IN USE FOR WRITING	7 = CONVERSION
109 = FILE ALREADY EXIST	8 = TOO LONG
110 = FILE DOES NOT EXIST	9 = OUT OF SPACE
111 = ILLEGAL FILENAME	10 = ARRAY
112 = ILLEGAL RESERVATION	11 = COMMAND
115 = OUT OF FILE	12 = CHANNEL
116 = FILE IN USE FOR READINGc	13 = RECORD OVERFLOW
117 = UNKNOWN DEVICE	14 = MODE
119 = FILE(S) IN USE	15 = UNDERFLOW
121 = DIRECTORY FULL	17 = OVERFLOW
122 = DISK FULL	
124 = DEVICE TIMEOUT	Remarque:
125 = WRITE PROTECT TAB SET	seules les erreurs système
126 = WRITE ERROR	plausibles avec le BASIC se
127 = READ ERROR.	trouvent dans cette liste

COMMANDE CON (CONTINUE)

La mise au point de cette commande a également nécessité un remaniement du BASIC. Anciennement on avait de nombreuses réinitialisations du stack qui rendaient par exemple impossible une commande CON après un STOP à l'intérieur d'une routine. Cette inconvénient n'existe plus. Le BASIC initialise le stack au départ, à chaque RUN et à chaque erreur.

SAISIE DES LIGNES ET ECRAN

Vous vous souvenez certainement de nombreuses difficultés avec le problème de la longueur des lignes et le problème du nombre de lignes dans l'écran, différent en mode commande (19 lignes) et en mode RUN (20 lignes). Cette partie a été entièrement refaite. Maintenant, on travaille toujours avec un écran de 20 lignes et l'on peut saisir dans tous les cas sans aucun problème (création de programme, correction en mode EDI, INPUT de données à l'exécution etc...) des lignes de 256 caractères, ce qui correspond à la plus longue ligne autorisée en BASIC. D'autre part, le désagréable papillottement des lignes longues à disparu lorsque l'on déplace le curseur. Enfin ce nouveau "driver" de ligne permet la saisie des accents flottants tout comme dans nos éditeurs. Les ordres sont également acceptés en minuscules.

NOUVEAUX ORDRES ALPHA ON, ALPHA OFF, GRAPH ON, GRAPH OFF

Ces nouveaux ordres permettent à l'utilisateur de contrôler lui-même depuis le BASIC les écrans du SMAKY. Ceci permet des effets nouveaux, tel que par exemple la préparation cachée d'un écran, puis son affichage uniquement lorsque celui-ci est prêt. Ils permettent également de résoudre les problèmes rencontrés avec l'ordre PSCR pour les "dump" d'écrans. Signalons ici la seule légère incompatibilité avec la version précédente du BASIC: Dans la version 2-4 on avait toujours l'écran graphique actif. Maintenant le BASIC démarre avec uniquement l'écran alpha actif. Il y aura donc lieu dans les programmes utilisant le graphique de rajouter l'ordre GRAPH ON dans le programme ou de taper cet ordre en mode commande avant l'exécution.

FORMATAGE DES NOMBRES, FONCTION #(n,d)

L'affichage par défaut des nombres est identique à la précédente version. On peut cependant à l'aide de cette nouvelle fonction sélectionner le nombre "n" de digits et le nombre "d" de décimales. Le format choisi est respecté tant que les nombres à afficher rentrent dans ce format. Si ce n'est pas le cas, le BASIC passe automatiquement à la notation scientifique. Cette fonction se place devant chaque variable que l'on désire formater.

EXEMPLE: PRINT #(8,2)A,#(4,0)B,#(0,0)C

Donne avec A = 123456.12345
B = 123.20453
C = 1234.2345

..123456.12 .123 1.2342345 E+003

REMARQUE: Les points au début sont en réalités des espaces. Zéro digit force la notation scientifique.

Afin de pouvoir également convertir un nombre en un string en format fixe, cette fonction peut être ajoutée à l'ordre:
A$=NUM$(X)
On écrira: A$=NUM$(#(n,d)X)

NOUVEL ORDRE NAME "oldname.ex,newname.ex"

Cet ordre permet de renommer un fichier soit en mode commande ou à l'intérieur d'un programme. Le séparateur entre les deux noms à l'intérieur du "string" est obligatoirement une virgule.

NOUVEL ORDRE COMPRESS "REP:"

Cet ordre permet de compresser un répertoire directement depuis le BASIC soit en mode commande ou à l'intérieur d'un programme. Il faut cependant remarquer que ce "compress" peut être très long, car il ne dispose que de 3 blocs comme buffer.

NOUVEL ORDRE LOAD "filename.BS"

Cet ordre permet de charger des lignes de programme durant l'exécution d'un programme. Ceci permet de réaliser en quelque sorte des "OVERLAYS". Ce chargement n'est évidemment pas très rapide, puisque l'on charge du source.

EXEMPLE: Supposons deux programmes BASIC DEMO1.BS et DEMO2.BS sous forme de routines commençant les deux en 100 avec des par de programmation identiques de 10 (100,110, etc.)

Le petit programme suivant exécutera successivement ces deux démonstrations:

```
10 LOAD "DEMO1.BS"
20 GOSUB 100
30 LOAD "DEMO2.BS"
40 GOSUB 100
50 PRINT "DEMONTRATION TERMINEE"
60 END
```

NOUVELLE COMMANDE DUMP filename

Cette commande crée un fichier "filename.SM" qui contient un "dump" de la mémoire avec le BASIC, son programme et ses variables. Lorsque l'on exécute ce programme, on charge donc directement le BASIC avec un programme BASIC qui est effectué directement. Attention, ce dump va du début du binaire du BASIC jusqu'à MAXMEM. Les éventuelles routines en assembleur ne seront donc pas prises, il faudra donc les sauver et les charger séparément. Il faut aussi prendre garde que MAXMEM ait la même valeur au chargement du fichier que lors de sa création.

SMAKY NEWS

ADRESSEZ VOS COMMUNICATIONS A:

EPSITEC-system sa

Chemin de la Mouette, CH - 1092 Belmont