Peersoft v1.5.5

Une extension Applesoft

Benoît GILON

26/07/2015

Une introduction à Peersoft

Peersoft se veut une extention au BASIC Applesoft. Aujourd'hui, dans sa version actuelle, Peersoft tourne sous DOS 3.3 mais une version ProDOS pourrait voir le jour si la demande est pressante.

Trois versions de Peersoft existent dépendant du CPU détecté dans l'Apple 2 utilisé (soit un 6502 de base, soit un 65C02 soit un 65802), avec les optimisations afférentes pour les deux derniers CPU cités.

Les efforts se sont surtout concentrés sur l'aspect des performances durant la phase de dessin de l'utilitaire. Voici les principales fonctionnalicaractère du nom de cette tés délivrées dans cette version:

- Nouvelles instructions pour définir le type par défaut de variable en fonction du premier caractère du nom de cette dernière:

Tableau 1: Nouvelles instructions pour définir le type par defaut de variables

Nouvelle instruction	Type de variable	Magnitude	Caractère en suffixe
DEFINT	Entier 16bits	De -32768 à 32767	%
DEFSNG	Virgule flottante (5 octets)		!
DEFSTR	Chaine de caractères	Longueur de 0 à 255	$

  - Ainsi le source d'un programme peut être plus compact en utilisant ces nouvelles instructions.

- Une nouvelle syntaxe pour fixer la nouvelle valeur d'une variable à partir du résultat d'une opération arithmétique.
  `LET <variableName> += <value>` or
  `LET <variableName> /= <value>` or
  `LET <variableName> -= <value>` or
  `LET <variableName> *= <value>`
  Si la variable spécifiée est de type Entier elle même, alors l'opération arithmétique le sera également, de plus the schéma += s'applique aussi aux chaines de caractères. L'un des bénéfices principaux, au delà de la vitesse apportée par l'introduction d'une opération entière par rapport à sa soeur en FP, est de minimiser le nombre de recherches d'adresses de variables à partir de leur nom si on comprare la nouvelle syntaxe avec celle de base `LET <variableName> = <variableName> <operator> <value>` particulièrement lorsque *variableName* déigne une variable tableau avec des valeurs égales pour chaque indice.

- Avant que la version 1.5 de Peersoft ne sorte, un élément du tableau A ne pouvait être référencé uniquement en uitlisant la syntaxe `A(<expression>, <expression>)` en préseumant que le tableau A ait été déclaré comme ayant deux dimensions (avec une instruction `DIM`), ceci est le moyen usuel d'utiliser des tableaux au sein du source d'une application.
  Avec la venue de la version 1.5 de Peersoft v1.5, le même élément peut être rappelé en utilisant une spécification sur une seule dimension. Les deux extraits suivant effectuent donc strictement le même travail:

```
10 DIM A(4,5):…:S = 0: FOR I = 0 TO 4: FOR J = 0 TO 5:S += A(I,J):
```

```
NEXT J,I
10 DIM A(4,5):…:S = 0: FOR I = 0 TO 29:S += A(I): NEXT I
```

Elles calculent la somme des éléments du tableau dans une variable S. Cette nouvelle syntaxe peut amener à une simplification des accès et donc une amélioration des performances.
De plus, au sein de l'interpréteur Applesoft d'origine, existe une limitation sur l'expression d'une dimension d'un tableau à 32767 mvaleur maximum. Au moment où Applesoft a été conçu, la plus petite taille d'élément intégrable dans un tableau était de 2 octets (taille d'un entier) et la taille de la mémoire adressable par le BASIC ne pouvait dépasser 64Ko: cette limitation est maintenant portée à 65535 rendant possible la mise en tableau de plus de 32768 éléments de 1 octet (dès que le type entier BYTE sera implémenté dans Peersoft).

- Une nouvelle pseudo variable (“`@`”) , utilisable dans toute expression arithmétique et qui retourne la valeur couramment stockée dans l'accumulateur flottant de l'Applesoft (FAC). Ainsi, la séquence d'instructions:

  ```
  S = 0
  FOR I = 0 TO 9: FOR J = 0 TO 9:S += A(I,J) *  @
  NEXT J,I
  ```

  va calculer la somme des carrés des éléments de la matrice 2x2 A. Vous noterez que le code se réfère à l'élément de tableau en une seule occurrence et que l'opérateur `*` est connu pour être (et l'est réellement) plus véloce que la mise au carré donné par l'expression `A(I,J) ^ 2`.

- Une nouvelle fonction `IIF()` est fournie afin d'évaluer juste une parmi deux expressions candidates, choix basé sur le résultat d'un test booléen. Voici un exemple:

  ```
  MA = IIF (A > MA,A,MA): REM will compute the MAX(A,MA) and store it into the MA variable
  ```

- Les variables entières sont maintenant permises au sein de boucles FOR/NEXT, les opérations d'incrémentation/comparaison lors du traitement de l'instruction NEXT seront alors du type entier dès que la variable de boucle est une variable entière elle même.

- Corrections de bugs sur le traitement de certaines instructions (`ONERR, RETURN et POP`): si vous visitez le listing de désassemblage de l'Applesoft généré par l'outil S-C Documentor et commenté par Bob Sander-Cederlof (http://www.txbobsc.com/scsc/scdocumentor/), alors vous découvrirez que l'Applesoft cache beaucoup de bugs dans son code.

- Des routines utilitaires sont fournies afin d'optimiser l'accès aux variables Applesoft au sein du programme. Pour les variables simples comme les tableaux, le temps de recherche d'une variable au nombre de variables du même type (i.e. simple ou tableau) qui ont déjà été définies (i.e. utilisées comme références pour les variables simples) avant que la variable recherchée (simple ou tableau) n'ait elle même été créée dans la zone mémoire. L'ordre de définition des variables/tableaux dans un programme suit rarement la fréquence d'utilisation de ces mêmes variables et tableaux: un ordre qui voudrait que les variables les plus utilisées soient créées en premier. Nous avons tendance à créer des variables accessoires comme `D$ =  CHR$( 4)` au tout début de notre programme bien que cette variable sera par la suite beaucoup moins utilisée qu'une autre variable. Le but de ces routines utilitaire est d'optimiser l'accès à ces variables après que celles-ci aient été créées. Deux approches alternatives sont mises à disposition:

- Réorganisation physique de la zone mémoire dévolue aux variables simples/tableaux où les contenus en mémoire sont réellement déplacés (à raison de 7 octets par variable simple);

- Référencement par cache: ici les variables restent immobiles dans leur emplacement respectif en mémoire. En lieu et place, Peersoft gère une petite zone mémoire dédiée pour conserver les noms et les adresses mémoire d'un petit nombre de variables ou de tableaux. Ces entrées sont auscultées en tout premier lieu et, uniquement si aucune correspondance n'est trouvée, la recherche séquentielle peut commencer. Les recherches de ces variables sont donc plus rapides au détriment des recherches sur les autres variables. Le cache de Peersoft permet de mettre en cache jusqu'à 4 variables simples et 4 tableaux.

- Mise à disposition de co-routines à l'intérieur d'un programme Applesoft: ceci est un nouveau paradigme pour tout auteur d'application sous Applesoft. Maintenant il est capable de concevoir des programmes **comme si** ces derniers tournent sous un environnement multi-tâches préemptif. L'application peut être considérée comme un assemblage de phases où le noyau est actif (co-routines s'exécutant "en parallèle") et de phases où le noyau est inactif (Juste un flux de programme est traité par l'interpréteur). Au delà de la description qui suit, un chapitre entier a été consacré aux co-routine dans la suite du document.

    - Parce que le basculement de contexte entre deux co-routines surviendra à des moments bien ciblés dans la boucle de l'interpréteur, le contexte de chaque co-routine peut être gardé de taille moindre que si le basculement avait lieu dans l'environnement d'un vrai moniteur multi-tâches.

    - Contrairement aux vrais moniteurs multi-tâches, aucune interruption matérielle n'est mise à contribution ici et donc les co-routines Peersoft fonctionnent également bien sur tous les modèles d'Apple 2 (de l'Apple II standard avec la ROM Autostart jusqu'au Apple //gs ainsi que tout environnement émulé dont j'ai connaissance).

- Peersoft offre une optimisation des temps de traitement pour les ordres GOTO/GOSUB par pré calcul des adresses cible.

- Possibilité est donnée de concevoir des sous-routines en BASIC Applesoft qui seront lancées en cascade d'interruptions souris ou VBL avec ouverture du dispositif pour prendre en compte d'autres natures d'interruption matérielles à l'avenir (port série, vidéo, horloge, ...).

- Possibilité de définir concurremment jusqu'à 10 fonctions utilisateur additionnelles (i.e. En plus de la fonction USR standard d'Applesoft). De plus, ces fonctions sont à même de recevoir jusqu'à deux arguments (contre un seul pour la fonction de base de Applesoft) + Il est possible de définir le corps de ces fonctions par des sous-routines écrites en Applesoft (ce qui permet une souplesse impossible avec l'instruction `DEFFN`).

Feuille de route de Peersoft

Les fonctionnalités décrites dans la section précédente sont aujourd'hui implémentées dans la version actuelle de Peersoft. Cependant, toutes les fonctionnalités imaginées depuis lors ne l'ont pas encore été. Certaines restent à développer afin de remplir le cahier des charges fixé par l'auteur à l'origine et continuellement enrichi...

Le tableau ci-dessous fournit quelques pistes sur les évolutions à venir et leur date de délivrance envisagée. Cependant, comme ce projet est basé sur le temps qu'il reste à l'auteur lorsque il est disponible sur les deux fronts en concurrence (i.e. la famille et le travail). Je vous conseillerais de ne pas retenir votre respiration.

Version	Fonctionnalité délivrée	Date de délivrance
1.7	Généralisation des fonctions utilisateur (`DEF FN`): plus d'args et type d'arg. (i.e. quelconques, pas juste FP)	08/09/2015
1.7	Nouveaux sous-types entiers comme BYTE, LNG24 et LNG32	08/09/2015
1.6	Tirer bénéfice de l'extension mémoire dans les //c, //gs et //e (128K) et options pour placer des tableaux dans les bancs additionnels.	31/06/2015
1.6	Tirer bénéfice de l'extension mémoire dans les //c, //gs et //e (128K). Traitement transparent des graphiques Double Hires et double lores.	31/06/2015
1.7	Fusion avec l'utilitaire Bananasoft (utilisant une technologie similaire mais se concentrant sur les fonctionnalités plutôt que sur l'aspect « performances »); le nom de l'application résultante pourra s'appeler « salade de fruits » mais je ne suis pas encore sûr de cette désignation.	08/09/2015
1.8	Arithmétique entière généralisée dans l'évaluation d' expressions (i.e. la sous-expression A% + 1 sera évaluée en premier avec une arithmétique entière et ne fera appel à l'arithmétique FP que en cas de dépassement de capacité).	31/12/2015
1.8	Compilation des fonctions Applesoft (introduites par DEFFN) en langage machine (appelablerouit par le nouveau pattern `USR<n>`)	31/12/2015
1.9	Généralisation de l'APAL et des routines de traitement d'interruption pour les autres sources (ports série, graphique //gs...).	01/03/2016

Contenu physique du package Peersoft

Peersoft consiste en une archive zip (Peersoftv1.5.5.zip) contenant:

- Un fichier image disque avec le suffixe the .do suffix (DOS 3.3 sector order) contenant un environnement fonctionnel Merlin 8 DOS3.3 (v2.48).
- Un fichier image disque avec le suffixe .do les sources complets de Peersoft dans le but de construire l'un des trois fichiers exécutables principaux (selon le modèle de CPU cible: 6502, 65C02 ou 65816) ainsi que les binaire compagnon.

Nom de fichier sur le disque	Objet
PEERSOFTV15.S	Fichier source principal pour constuire Peersoft avec Merlin
T.PEERINSTALL	Fichier include pour l'installation de Peersoft
T.PEERLIST	Traitement de l'instruction LIST modifiée
T.PEERINTEGARITH	Traitement des routines d'arithmétique entière
T.PEERAROMBA	Toutes les extensions à la rouitne FRMEVL d'origine
T.PEERMTK	Co-routines dans Peersoft
T.PEERGOTO	Précalcul des adresses cible pour GOSUB/GOTO
T.PEERMOUSTIME	Support des interruptions
T.PEERMOTIDATA	Segment de données pour support des interruptions
T.PEERGLOBALPAGE	Segment de données pour la page globale
T.PEERUTILR	Fonctions utilitaires diverses dans Peersoft
T.PEERGDATA	Segment de données pour le pré calcul des adresses cible
CRECON.S	Reconnaissance du CPU, utilisé part le programme HELLO.
TCPRECON.S	Source pour reconnaître une carte Thunderclock ou un //gs en vue de l'utilisation de l'horloge par le programme TF
SMTRECON.S	Source pour détection et exploitation d'une puce NSC.
TUTMC.S	Source pour les routines en langage machine utilisées par le programme Applesoft démonstration des co-routines TUTORIAL2

- Un fichier image disque avec le suffixe .do donnant une image bootable sous DOS 3.3 de Peersoft et de l'ensemble des programmes compagnon Applesoft ou langage machine qui lui sont rattachés.

Nom de fichier sur le disque	Objet
HELLO	Programme de démarrage affichant un menu de sélection et veillant au chargement d'une version de Peersoft compatible avec le CPU détecté.
CRECON	Fihier objet chargé par HELLO et détectant la variante de CPU hôte.
PEERSOFTV15_6502, PEERSOFTV15_65C02, PEERSOFTV15_65802	Un fichier binaire exécutable par CPU, chaque fichier résulte de l'assemblage du fichier source PEERSOFTV15.S par Merlin avec différentes options pour les macros KOPT et KOPT16
TF	Programme Applesoift de démontration des apports de Peersoft dans sa version actuelle.
TCPRECON	Reconnaissance et exploitation d'une carte horloge par TF pour mesure des temps d'exécution.
SMTRECON	Même objet mais pour détection d'une puce NSC dans l'Apple hôte
TUTORIAL2	Programme Applesoft illustrant les fonctionalités de Peersoft dans le domaine des coroutines
TUTMC	Routines en lngages machine pour l'utilisation des « tutoriels ». Ne sert juste que pour l'affichage des statuts des co routines ainsi que pour émettre un clic dans le HP lorsque du passage du CPU d'une co routine à l'autre..

- Documentation Peersoft sous forme de fichiers PDF (Français et Anglais) que vous êtes en train de lire.

Comment transférer le contenu des deux fichiers image disque vers des disquettes 5'1/4 pour être lues par du hardware Apple //

Afin de transférer le contenu des deux images disques vers du matériel Apple // (avec lecteur de disquette 5'1/4). Nous allons utiliser les outils suivant:

- Une image disque pré-formatée (avec les fichiers système ProDOS2.0.3 et lNuFX Shrink/Unshrink); Cette image disque va nous permettre de convertir les deux images disques démarrant en DOS 3.3 et de les placer dans un fichier archive ProDOS 8. Dans le cas où vous ne pourriez mettre la main sur une telle image, vous pouvez en obtenir une sur mon site sous l'URL suivante http://bgilon.free.fr/apple2/ShrinkIt.2mg.

- Un émulateur Apple // sur un ordinateur moderne pour faire tourner le programme NuFX Shrinkit sur votre ordinateur moderne. Dans les illustrations suivantes, j'utilise l'émulateur AppleWin 1.22 sous Win32.

- Le programme CiderPress toujours sous Win32 afin de placer le fichier archive résultant du lancement de NuFx sur une carte Compact Flash CFFA.

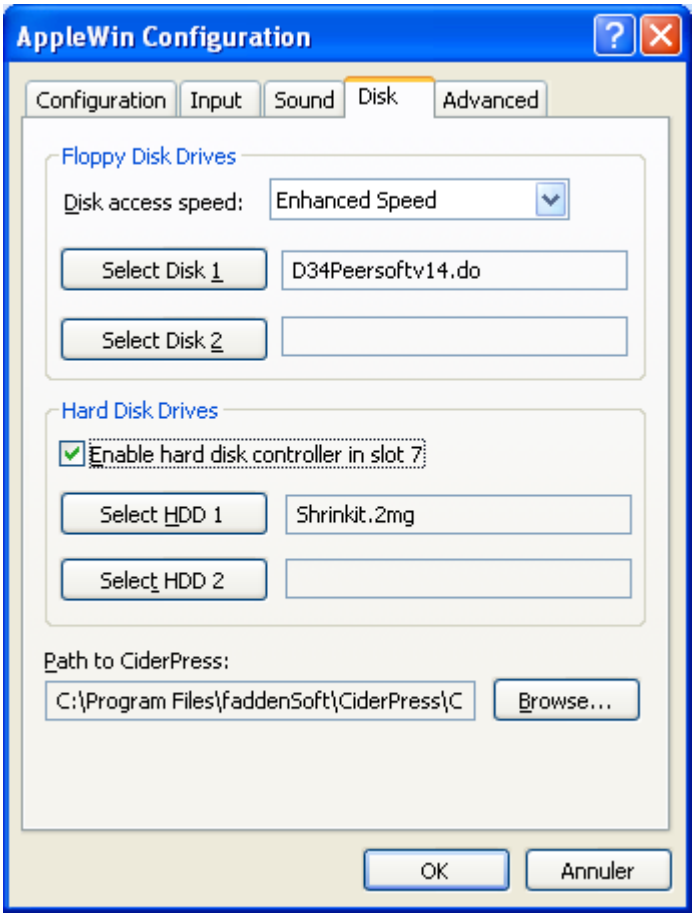
- Une carte interface CFFA pour Apple //e ou //gs (la mienne est une CFFA 2.0, mais toute version plus récente devrait être compatible. Notez que la dernière génération permet la lecture directe de fichier .po et .do) avec bien sûr une carte mémoire Compact Flash formatée ProDOS.

- Un hub pour lire/écrire les cartes compact flash sur la même machine que celle qui héberge le programme CiderPress (interface USB à priori).

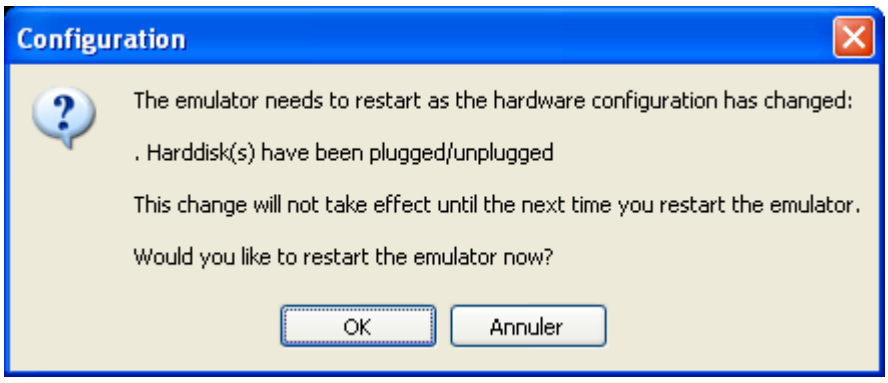
D'autres voies sont possibles pour réaliser les mêmes tâches (comme d'utiliser des liaisons série entre ordinateur « moderne » et Apple //).

Configuration de Applewin 1.22

Après avoir téléchargé l'image disque « bootable » contenant le système ProDOS et le programme NuFX ShrinkIt sur votre ordinateur et l'avoir placé à un endroit où vous souhaitez le conserver, démarrez l'émulateur Applewin et afficher la fenêtre de configuration en pressant la touche de fonction F8.

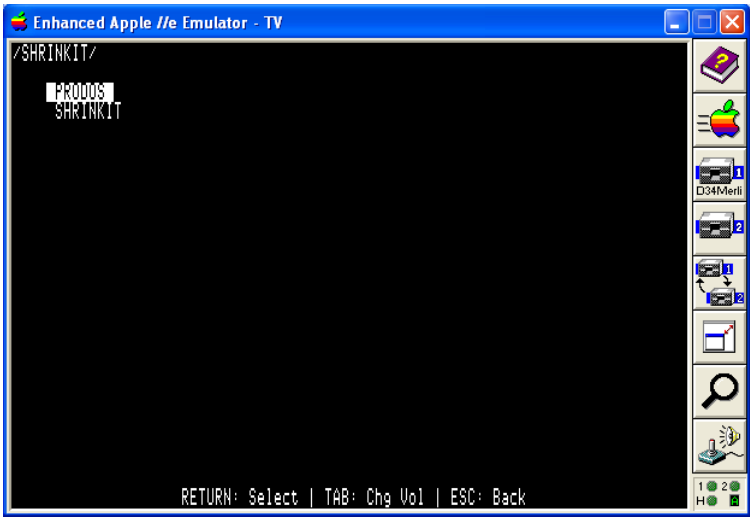


Sélectionnez la case à option libellée “Enable hard disk controller in slot 7” et cliquez sur le bouton libellé « Select HDD 1 ». Une boîte de dialogue pour la sélection d'un fichier va s'ouvrir. Naviguez alors jusqu'à l'emplacement incluant le fichier archive et choisissez ce dernier. Cliquez « OK » pour entériner votre choix. Une boîte de dialogue comme celle ci-dessous va alors surgir vous prévenant que un redémarrage de l'émulateur est nécessaire pour prise en compte du nouveau choix. Applewin va redémarrer pour cause de reconfiguration sur un de ses disques de démarrage.



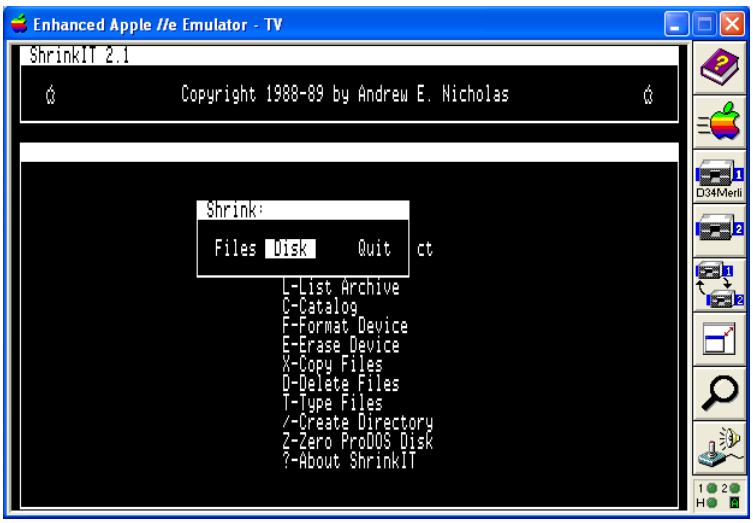
Procédure d'archivage des deux images disque

Dès que l'émulateur a redémarré, l'écran ci dessous s'affiche.



Sélectionnez le programme libellé « SHRINKIT », pressez la touche entrée pour valider et l'écran de bienvenue du programme (ci-dessous) devrait apparaître.

À partir du bureau Windows, glissez et déposez l'icône de l'image disque « D33Merlin – Peersoftv15.do » sur l'icône représentant le lecteur 1 de l'Apple //e émulé sur le panneau de droite. Puis sélectez l'option « Shrink » du menu principal.

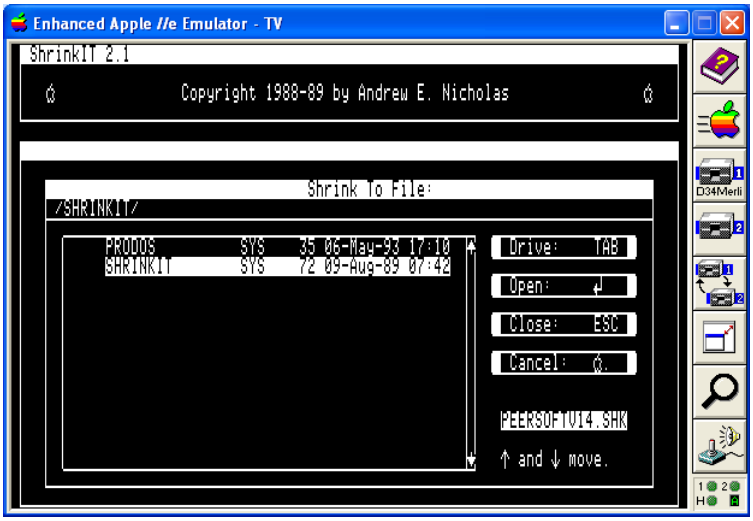


Lorsque le choix d'options s'offre à vous, choisissez l'option « Shrink Disk ».

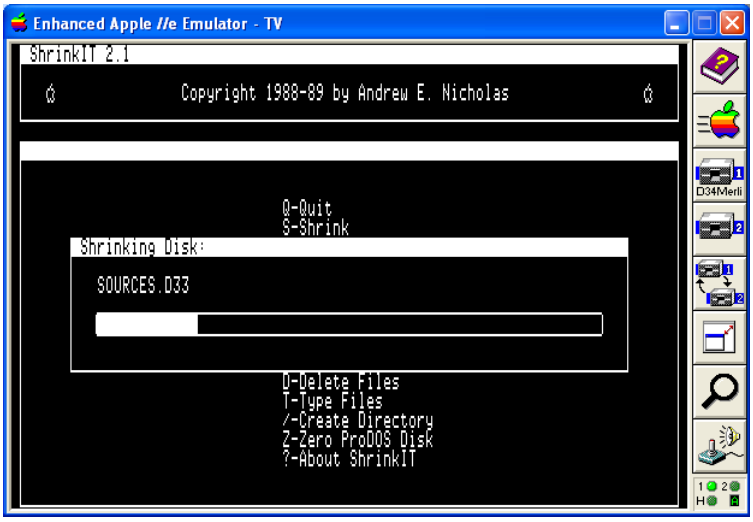
Sélectionnez maintenant la source à savoir ici l'option “Shrink Disk on S6, D1 140k Drive” ..

Renseignez alors le label qui permettra d'identifier cette « sauvegarde » au sein du fichier archive résultant (ici “SOURCES.D33”).

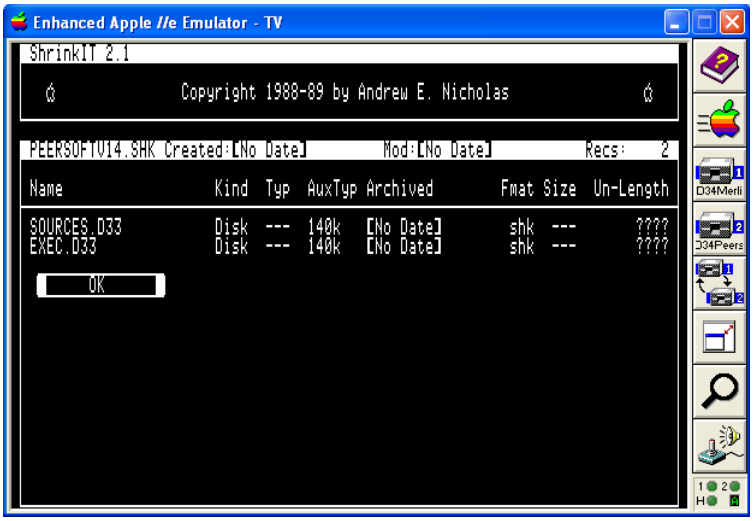
Comme sur l'écran ci-dessous, renseignez le chemin du fichier ProDOS archive qu isera créé pour l'occasion à l'issue de la sauvegarde.



Ici, j'ai entré le nom de fichier PEERSOFTV15.SHK. Dès que la touche RETURN a été pressée pour valider l'entrée, la barre de progression du processus d'archivage apparaît.



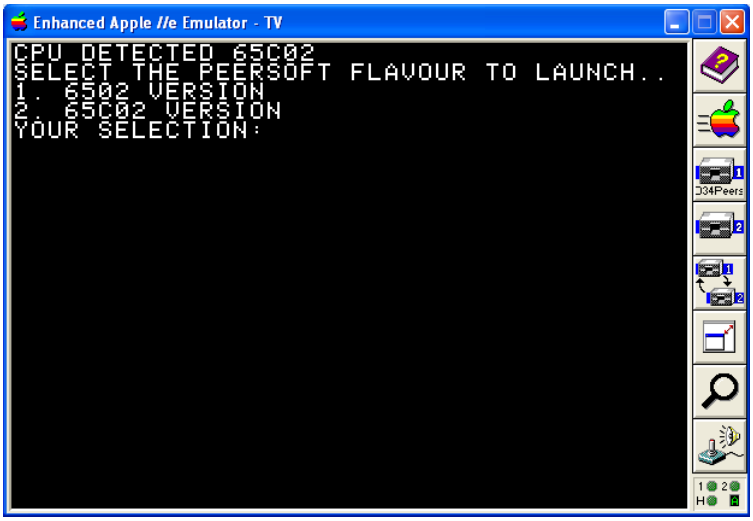
La prochaine étape sera de réitérer le processus ci-dessus pour l'image disque D34Peersoftv155.do. Pour vérifier que tout s'est bien passé à ce stade, vous pourvez lancer l'opération d'affichage du catalogue de l'archive qui devra faire apparaître les deux libellés saisis.


Enhanced Apple //e Emulator - TV
ShrinkIT 2.1
Copyright 1988-89 by Andrew E. Nicholas
PEERSOFTV14.SHK Created:[No Date] Mod:[No Date] Recs: 2
Name Kind Typ AuxTyp Archived Fmat Size Un-Length
SOURCES.D33 Disk --- 140k [No Date] shk --- ????
EXEC.D33 Disk --- 140k [No Date] shk --- ????
OK
D34Merli
D34Peers

Manuel utilisateur de Peersoft

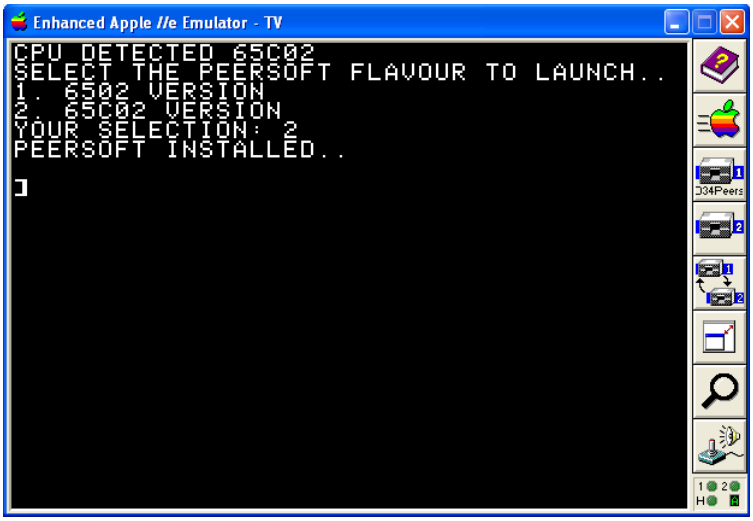
Les fichiers exécutables de Peersoft viennent au sein de l'image disque D34Peersoft155 (fichier archive ou disquette 5'1/4 selon que vous êtes en émulation ou non).

L'image disque est « bootable » en DOS3.3, insérez là dans le slot 6 drive 1 and re démmarez l'émulateur ou l'ordinateur. Un écran similaire à celui ci-dessous doit alors s'afficher.



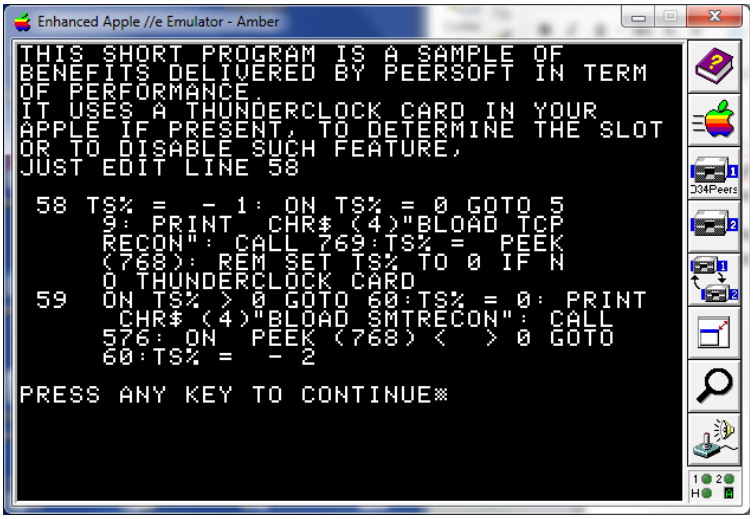
En fonction du CPU détecté sur l'environnement hôte, plus ou moins de choix sont offerts (ceci afin d'éviter de lancer une version 65816 de Peersoft sur un ordinateur ne disposant que d'un 6502 de base). À ce moment, vous avez la possibilité de taper la séquence de touches <Ctrl><C> pour ne pas installer en mémoire l'utilitaire Peersoft et simplement revenir au prompt Applesoft. Mais, supposons ici quue vous vouliez installer Peersoft dans sa version la plus optimisée possible compte tenu de l'hôte : vous sélectionnez donc l'option 2 pour ce faire.

La mention comme quoi Peersoft a été installé s'affiche alors.

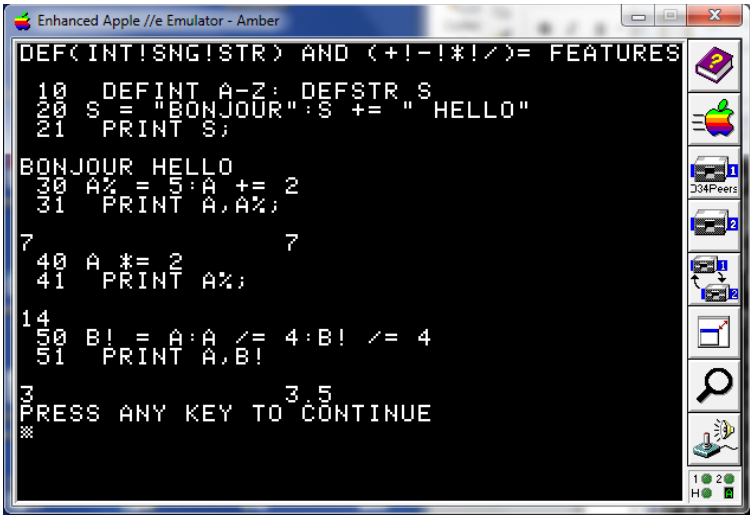


Il existe un programme Applesoft nommé TF qui permet à l'utilisateur de vérifier le bon fonctionnement de Peersoft (utile pour les tests de non régression pratiqués par l'auteur par exemple). De plus il offre une maigre démonstration des possibilités offertes à toute personne intéressée (qu'elle soit utilisateur ou auteur d'applications elle même).

Pour cela, vous avez juste à taper la commande `RUN TF` à l'invite "`]`". Cela mène à l'écran ci-dessous. Le programme TF peut exploiter l'horloge interne d'un //GS, une carte Thunderclock présente sans un slot d'Apple //e ou //GS (ou l'émulateur Virtual 2 sous Mac OS X), ou un circuit NSC SMT NSC (comme supporté par chaque émulateur qie j'ai utilisé pour débugger Peersoft) afin de mesurer le temps écoulé.



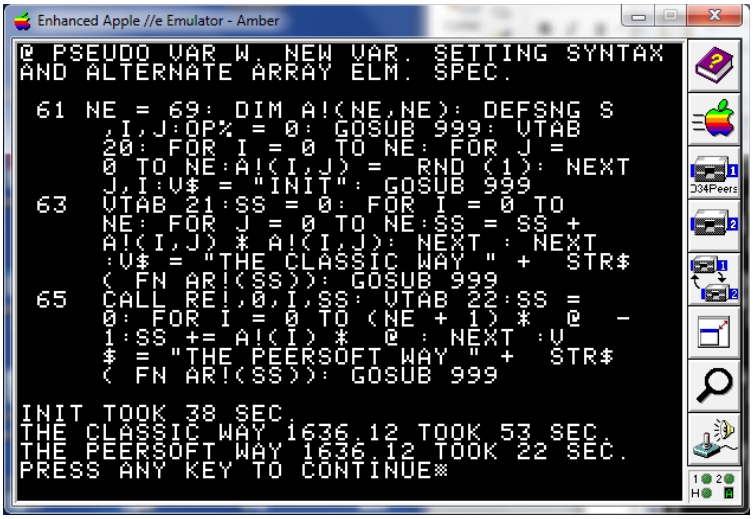
Pressez juste une touche pour passer à l'écran suivant…



Cet écran illustre certaines des fonctionnalités permises par Peersoft comme :

- Une nouvelle façon de concaténer des chaines de caractères à des variables ;
- Le typage par défaut des variables Applesoft (en utilisant les nouvelles instructions comme `DEFSTR`, `DEFINT` ou `DEFSNG`);
- Une nouvelle syntaxe pour altérer les valeurs de variables.

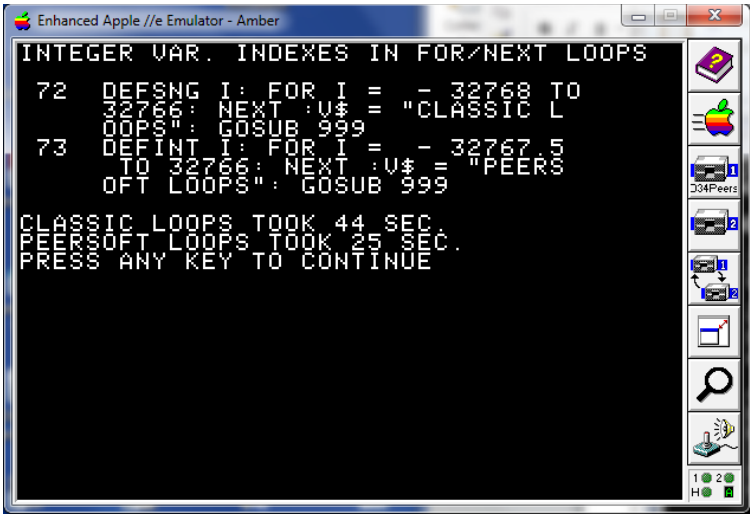
Pressez juste une touche pour passer à l'écran suivant…



Les fonctionnalités visibles sur l'écran ci-dessus sont :

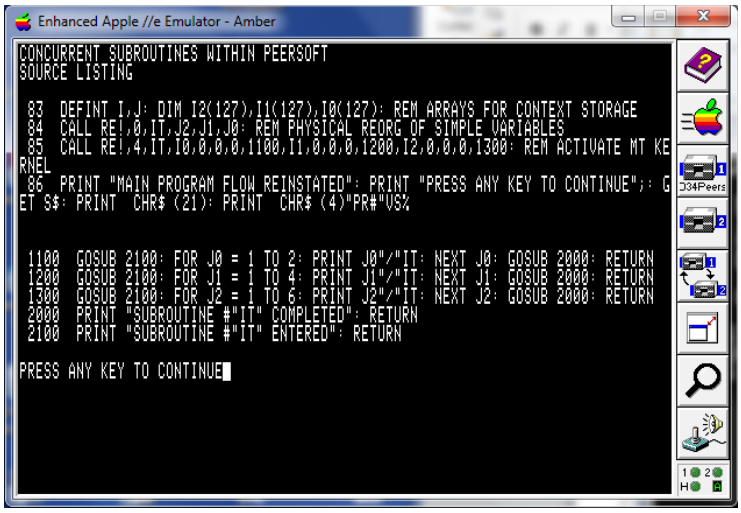
- L'utilisation d'une routine utilitaire (initiée par l'instruction CALL RE!) afin de réorganiser le segment mémoire des variables simples de façon que les variables “`J`”, “`SS`” et “`I`” sont positionnées au sommet du segment et sont donc recherchées en un minimum de temps;
- L'utilisation de la pseudo variable “`@`” afin d'éviter des re-calculs d'expressions identiques (ici les expressions « `NE + 1` » et « `A!(I)` » impliquant elles mêmes des recherches de références au tableau A et aux variables simples « `NE` » et « `I` » ;
- L'utilisation optionnelle de l'adressage simplifié (impliquant une seule variable d'indice) pour référence à un tableau à plusieurs dimensions.

Pressez juste une touche pour passer à l'écran suivant…

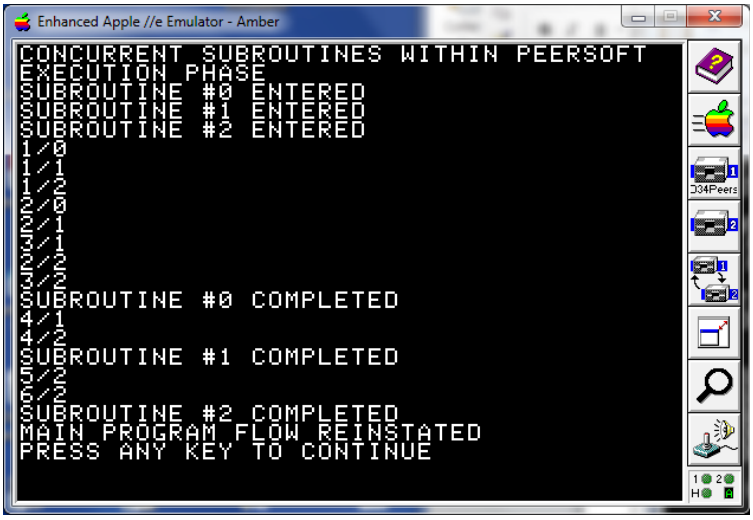


L 'écran ci-dessus montre quels bénéfices peuvent être tirés de l'utilisation de variables entières comme variables de boucles. Toutefois le bénéfice en terme de performance peut être réduit selon le niveau de l'utilisation de ces variables dans le corps des boucles.

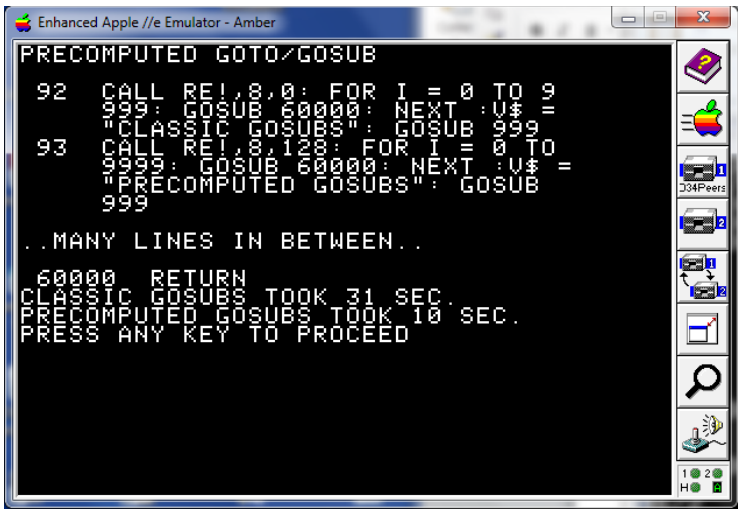
L'écran ci-dessous représente un code minimaliste illustrant l'utilisation de co routines dans Peersoft (une fonctionnalité inédite assurément). Les premières lignes préparent l'environnement et les dernières forment les corps des co routines ainsi que des sous routines appelées dans le contexte de co routines actives. Notons ici le rôle de la variable IT ici dont la valeur suit l'indice de la co routine (que l'on pourrait donc considérer comme « volatile » pour l'application proprement dite).



Le résultat de l'activation du noyau MTK à l'intérieur de Peersoft (gérant les co routines) est contenu dans l'écran suivant.



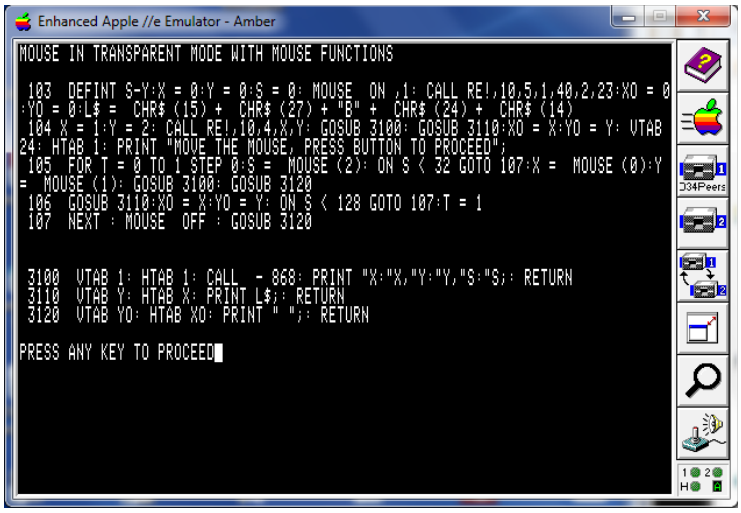
Chaque co routine est activée et désactivée selon un mécanisme circulaire de type « round robin » jusqu'à ce que toutes les co routines soient terminées auquel cas le noyau repasse le contrôle du CPU à l'instruction qui suit celle ayant lancé ces dernières. Un chapitre entier de ce document est consacré au détail du fonctionnement de ces co routines. Pour le moment, il vous suffit juste de presser une touche pour passer à l'illustration de la fonctionnalité suivante.



L'écran ci-dessus illustre une fonctionnalité nouvelle apparue dans la version 1.5 de l'outil, qui précalcule les adresses destinations des lignes mentionnées dasn des instruction `GOSUB` et `GOTO`. Le précalcul en lui même va coûter un peu de temps additionnel mais celui-ci sera vite plsu qu'amorti dès la deuxième rencontre de la même instruction de débranchement puisque l'adresse aura déjà été calculée (par exemple si l'instruction est incluse dans le corps d'une boucle.

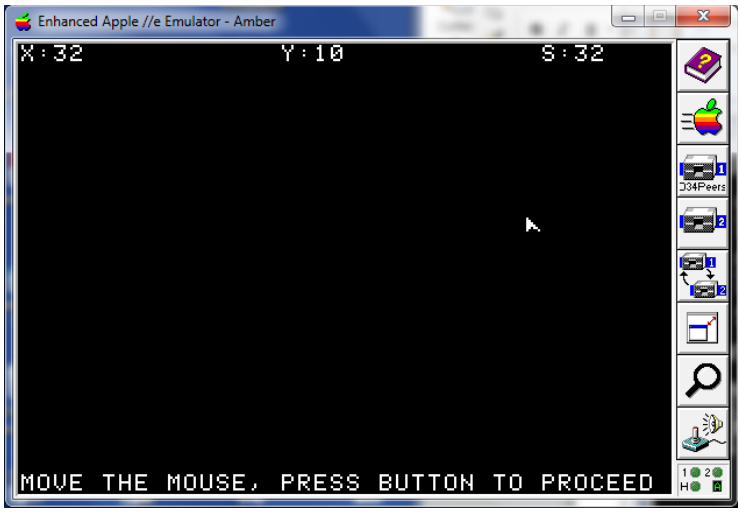
Le manuel de référence (plus loin dans ce document) donne des détails sur les routines utilitaires pour configurer le comportement vis à vis du précalcul des lignes destination (i.e. si oui ou non le précalcul automatique doit être activé). Sur un programme de taille moyenne (# de lignes entre 3 et 200), nous constatons que le temps de traitement des instructions `GOTO/GOSUB` sont réduits de moitié dès lors que leur exécution est multiple dans le flux du programme. Ces temps peuvent encore diminuer dans le cas de plus gros programmes avec un niveau de complexité supérieur (avec plus de références exxplicites à des lignes BASIC en vue de débranchement).

Les deux clichés d'écran à suivre illustrent une nouvelle fonctionnalité apparue avec la version 1.5 de l'outil.



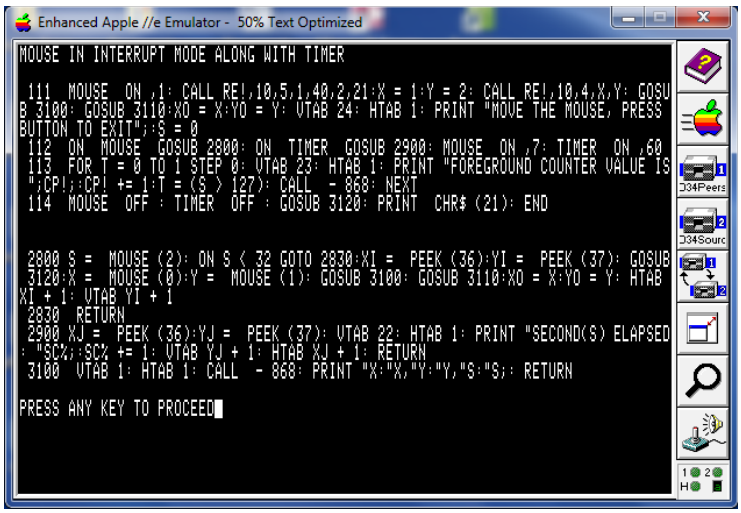
Bien sûr l'utilisation de la souris en mode transparent reste possible en Applesoft pur ou avec un mix Applesoft - 'langage machine' pour les fonctions évoluées. Ceci écrit, la couche d'interface offerte par Peersoft rendra l'écriture de programmes plus simple et plus proche de ce qui existe sur d'autre variantes

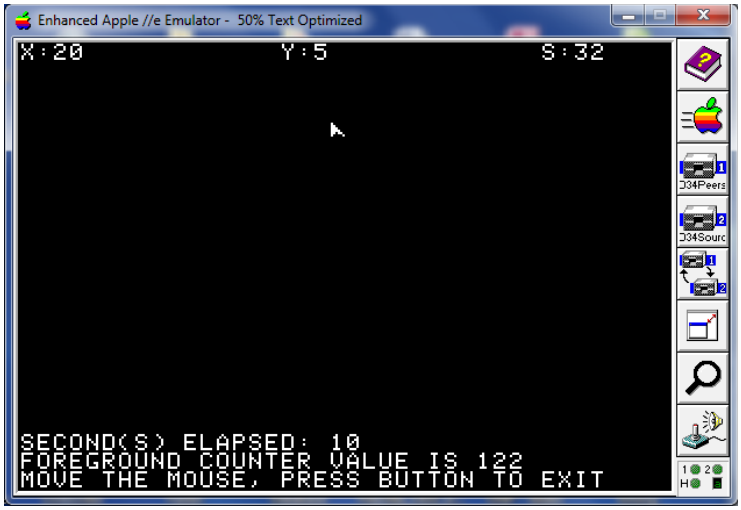
du BASIC Microsoft (utile dans le contexte de portage de programmes). Une nouvelle instruction `MOUSE` est utilisée pour configurer la souris, la fonction `MOUSE` pour retourner les valeurs *mouseX*, *mouseY* et *mouseButtonStatus*, et un jeu de sous routines pour les fonctions évoluées (toutes décrites dans le manuel Apple).



Le curseur souris suit le mouvement de cette dernière dans la fenêtre défini préalablement (excluant les première et dernière lignes de l'écran) . Presser le bouton de la souris interrompt le dispositif de suivi.

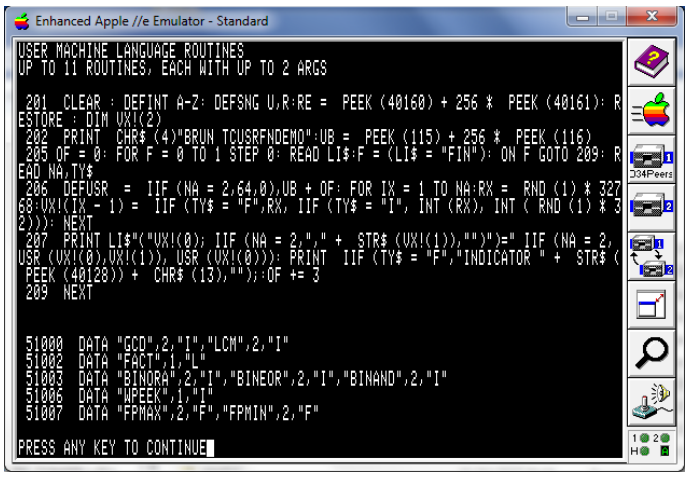
Un écran analogue mais non identique utilise cette fois-ci un mode d'interruption ce qui rend la programmation plus simple par l'omission de codes de redirection. À ma connaissance, le support de ce niveau de programmation pour les programmeurs Applesoft est une première. Vous devez disposer toutefois d'un Apple //e mis à niveau (//c like) (matériel ou émulation), d'un //c ou d'un //GS (tous modèles pour ces deux derniers) pour tirer profit de cette fonctionnalité. Ici vous voyez que les interfaces MOUSE et TIMER (traitements débutant en lignes 2800 et 2900 respectivement).

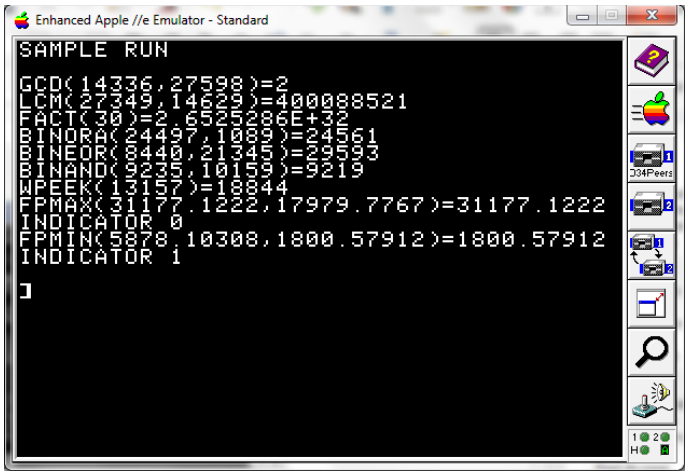




La ligne débutant avec la mention "*SECONDS ELAPSED*" est gérée par la routine enregistrée par l'instruction `ON TIMER GOSUB` ; Notez que le facteur de l'instruction TIMER activant la prise en compte des interruptions TIMER est de 60, cela implique que la routine Applesoft ne sera appelée que toutes les secondes (ou toutes les 60 ° 1/60 sec.) sur un Apple américain ou une émulation)..

La ligne débutant avec la mention « *FOREGROUND COUNTER VALUE* » illustre le fonctionnement de l'application en avant plan. Rien de spectaculaire ici… La sous routine Applesoft débutant à la ligne 2800 est en charge de mettre à jour l'affichage des coordonnées et de collecter le nouveau statut du bouton pour une éventuelle sortie des traitements d'interruption.





À partir de la version 1.5.5, Peersoft met à disposition la définition concurrente de 11 fonctions utilisateur en langage machine (une initialement avec Applesoft), et la possibilité de traiter jusqu'à deux arguments en entrée pour ces fonctions (un argument uniquement avec Applesoft de base), ce qu'illustrent les deux clichés d'écran ci-dessus.

Un module exemple est mis à disposition et vous offre les fonctions usuelles (du point de vue de l'auteur) :

- PGCD et PPCM de deux entiers (cela inclue les expressions FP qui comprennent les entiers sur 32 bits) ;
- FACT qui permet le calcul d'une factorielle d'un entier (hélas, un entier en entrée supérieur à 33 donne un dépassement de capacité) ;
- Trois fonctions pour faire des opérations binaires bits à bit sur des entiers 16bits : AND, ORA et EOR ;
- Une fonction WPEEK pour connaître le mot 16 bits situé à une adresse en RAM ;
- Deux fonctions permettant de connaître les MAX et MIN de deux expressions FP ;

La véritable originalité introduite dans Peersoft vient du traitement des fonctions procédurales : imaginiez que vous désiriez implémenter la fonction retournant la somme des entiers compris entre 1 et la valeur transmise en paramètre.

1. Une telle fonction écrite par simple instruction `DEFFN` ne permet pas d'introduire des comptages ou des structures de contrôle de flux (même si l'apparition de fonctions comme `IIF` peuvent aider dans certains cas), de plus le nombre et le type d'argument sont contraints.

2. Un implémentation en assembleur 6502 reste possible mais demande des compétences autres aux auteurs d'applications Applesoft ; la nouvelle fonction codée pourra toutefois alors apparaître dans toute sous expression d'expression plus générale ce qui est un avantage à considérer pour la suite de l'évaluation.

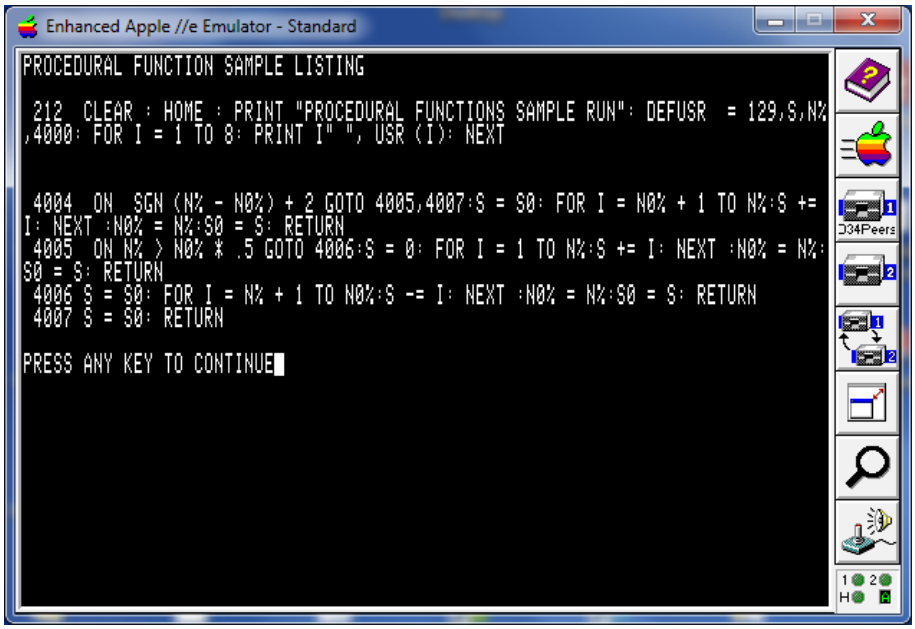
3. Il est aussi possible de sacrifier l'aspect « transparence » pour soit même gérer les segments de code d'appel comme le suivant :

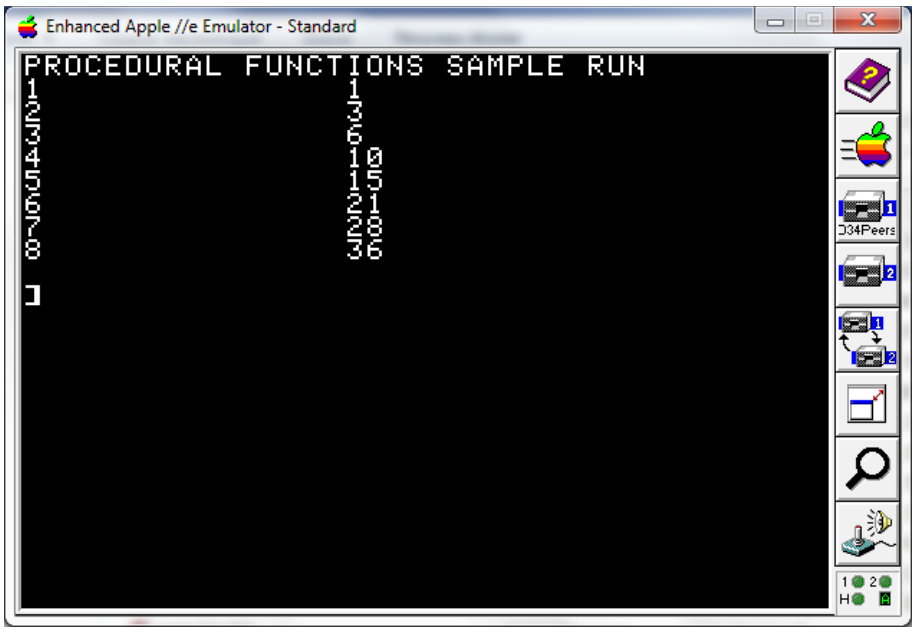
   ```
   P1= <expressionEntree1>:P2=<expressionEntree2>
   GOSUB <numeroDeLigne>
   REM Utilisation du résultat dans la variable R.
   ```

   Mais cela ne peut être utilisé comme sous expression ou être sollicité dans un `DEFFN` lui même.

4. Ce que permet les PF ici est la combinaison entre les approches 2 et 3 précédentes, tentant ainsi de permettre la programmation de fonction utilisateur avec du code Applesoft pur (approche 3) mais avec une sémantique d'appel similaire à l'approche 2.

En bref, on peut considérer les PF comme une alternative à la définition de fonctions utilisateur multi lignes supportées sur des versions plus évoluées du BASIC. Les deux écrans ci-dessous sont une illustration de l'approche des PF :





Nous utilisons dans cet exemple :

- un segment de données privé pour accélérer la recherche de variables dans l'exécution de la fonction (d'où la valeur `129` passé comme argument à `DEFUSR` au lieu de 128);
- un cache constitué des variables `N0%` et `S0` afin de ne pas recalculer depuis l'indice 1 de la boucle à à chaque occurrence.

Manuel de référence Peersoft

Assignation de type par défaut aux variables

DEFINT A, I-N,Z

Pour spécifier la liste des premiers caractères des variables impliquées par l'instruction DEF<type>, les caractères peuvent être spécifiés soit isolément (par ex. A et Z dans l'exemple ci-dessus) ou comme partie d'un intervalle (de I à N dans l'exemple ci-dessus).

À la suite de l'exécution de cette instruction, toute variable dont le nom commence par l'une des lettres A, I, J, L, L, M, N ou Z seront considérées comme entières si un caractère de type explicite n'est pas précisé.

Les autres instructions sont `DEFINT`, `DEFSNG` and `DEFSTR`.

Puisque Applesoft est un interpréteur, ces instructions peuvent apparaître à tout endroit dans le flux du programme et leur effet immédiat jusqu'à ce qu'une instruction `CLEAR` ou une autre intruction DEF<type> ne change la donne pour une lettre donnée.

Au démarrage du programme (ou lorsque une instruction RUN/CLEAR Applesoft est écévutée), toutes le variables adoptent par défaut un type implicite « réel virgule flottante ».

Toutefois et comme indiqué précédemment, un modificateur de type explicite (“%”, “$” or the new “!” à utiliser pour désigner des variables FP) comme suffixe à un nom de variable a priorité sur le type par défaut positionné. Ainsi la séquence `CLEAR : DEFINT I:I = 1: PRINT I!` imprimera 0 à l'écran.

Nouvelle syntaxe pour altérer les valeurs de variables

`A += 3`

Peersoft simplifie l'altération de valeur pour des variables, en rendant implicite la valeur d'origine de cette même variable. La nouvelle syntaxe `A += 3` est un raccourci pour `A = A + 3`.

Les quatre opérations arithmétiques peuvent être l'objet de telles simplifications. Ainsi `A -= 3` est un raccourci pour `A = A - 3` et `B /= 4` est un raccourci pour `B = B / 4`

Une construction avec la nouvelle syntaxe peut être utilisée à tout endroit où la syntaxe d'origine est autorisée. Cela incluse la séquence d' initialisation des boucles `FOR/NEXT`.

`FOR I += 5 TO 10` est un raccourci pour `FOR I = I + 5 TO 10`

Dès lors que la variable est de type entier, alors l'opération induite est elle même de type entier (ceci pouvant avoir un impact positif additionnel sur les performances). La nouvelle syntaxe `+=` est aussi valide pour les chaines de caractères.

`S$ = "BONJOUR": S$ += " HELLO": PRINT S$` affichera `BONJOUR HELLO` sur l'écran.

@ Pseudo variable

Ayant écrit un certain nombre d'applications moi même et ayant étudié le code et les commentaires d'applications d'autres auteurs, je me suis aperçu que une configuration qui ressortait souvent était l'utilisation d'une même sous expression/variable de multiples fois à l'intérieur d'une expression donnée.

Des fois, cela représente un coût en terme de performance de recalculer une sous expression de multiples fois à l’intérieur d'une expression plus globale (par exemple quand il existe des réréférences à un tableau multi-dimensionnel. D'où l'idée d'implémenter la pseudo varéelle ou un descripteur de chaine de caractères. La pseudo variable `@` prend la forme deu traitement le plus simple qui puisse s'imaginer. Tout ce qu'il fait est un simple RTS du CPU 6502, laissant les accumulateurs inchangés (en réalité c'est un peu plus subtil que cela mais pas trop: cf. code source code pour plus de détails ;-), cela va impliquer que la valeur retournée va rester inchangée de ce qu'elle était durant la dernière évaluation de facteur.

Ci dessous est un extrait de code tirant bénéfice de l'utilisation de cette facilité. Nous voyons ici que la pseudo variable peut concerner un scalaire de tout type (numérique ou chaîne de caractères)

`PRINT RIGHT$( A$, LEN( @ ) - 4)` imprimera la valeur courante de la variable/expression “`A$`” amputée de ses quatre premiers caractères.

L'utilisation de la pseudo variable `@` n'est pas conseillée pour référer à une expression évaluée dans une instruction précédente, principalement dans un environnement d'exécution en // de co routines (voir section plus loin dans ce document).

Nouvelle fonction IIF()

Cette fonction a comme objectif d'implémenter de façon simple et élégante un mécanisme de base en algorithmie : retourner une valeur parmi deux valeurs selon un critère booléen donné. Ujusqu'à maintenant, l'auteur d'application disposait de trois méthodes de base pour ce faire:

- Utiliser l'extrait de code Applesoft ci-dessous:

```
10  ON booleanExpr GOTO 20: LET returnVariable = returnValueIfFalse: GOTO 30
20  LET returnVariable = returnValueIfTrue
30  REM Flow of the program  continues from here
Pro: Peut s'appliquer pour retourner des chaines de caractères ou des valeurs
numériques
Pro: BooleanExpr n'est évalué qu'une fois (performance)
Pro: Juste une des deux sous expressions à retourner est évaluée (performance)
Con: Ne peux être inclus dans une fonction utilisateur simple (par DEF FN)
```

- Utiliser la sous expression Applesoft ci-dessous:

```
booleanExpr * returnValueIfTrue + (NOT booleanExpr) * returnValueIfFalse
Pro: Peut être incorporée dans une expression plus complexe (jouer le rôle de sous
expression).
Con: L'expression booléene est évaluée deux fois (performance)
Con: Les deux sous-expressions returnValueExpression sont évaluées même lorsque
une des deux est mis de côté (celle associée au « poids » de valeur 0)).
Con: Ne fonctionne que pour les sous expressions numériques
```

- Using a prefilled one dimension array (containing two elements)

```
10  DIM VR(1):VR(0) = returnValueIfFalse:VR(1) = returnValueIfTrue
20  REM Then you can use the VR(booleanExpr) as the subexpression.
Pro: Peut être incorporée dans une expression plus complexe
Pro: booleanExpr evaluée juste une fois
Pro: VR peut être de tout type (i.e. numérique or chaîne) et donc la valeur de
retour aussi.
Con: Ne marche que pour des valeurs constantes (l'expression retournée n'est pas
évaluée à chaque fois.
```

Vous voyez que chaque méthode a ses avantages et inconvénients propres, la fonction qui est mise à disposition ici a surtout des points forts par rapport à ses concurrentes:

- L'utilisation de la fonction `IIF( booleanExpr,returnValueIfTrue,returnValueIdfFalse)`

```
Pro: peut être considérée comme sous-expression d'une expression plus vaste
(utilisation dans un DEF FN par exemple).
Pro: booleanExpression évaluée juste une fois (performance)
Pro: Une seule des deux expressions candidates est évaluée, l'autre est simplement
analysée pour y trouver le caractère de clôture de l'expression. On gagne ainsi en
performance.
Pro: L'expression choisie est elle-même évaluée au moment de l'appel à IIF et donc
le choix n'est pas restreint à des constantes.
Pro: Peut traiter tout type d'expression à retourner (chaînes ou numériques). Les
deux expressions candidates peuvent même être de type différent (par exemple
lorsque IIF est utilisée comme argument de l'instruction PRINT)
```

En conclusion, la fonction IIF peut être un « time saver » pour l'auteur d'application comme pour l'utilisateur final.

Variables entières comme variables de boucles FOR/NEXT

J'ai été surpris de voir que, dans Applesoft, les variables entières étaient bannies de l'utilisation comme variables de boucles FOR/NEXT par l'interpréteur.

Ainsi, si vous essayez le code ci-dessous sous l'interpréteur Applesoft « plain vanilla », tout ce qui est retourné sera un message "`?SYNTAX ERROR`" .

```
FOR I% = 1 TO 10: PRINT I%: NEXT I%
```

Je croyais que l'utilisation d'arithmétique entière pour le traitement des incréments et des tests vis à vis de la valeur finale (constituant les traitement de l'instruction NEXT), pouvait être bénéfique en terme de performance (comme cela l'est sur d'autres interpréteurs y compris de Microsoft).

Hélas, à partir du moment où la variable de boucle apparaît dans le corps de la boucle, alors tous les bénéfices tendent à s'annuler car la gestion des variables entières au sein d'expression est plus couteuse que celle de variables FP par le simple fait que les variables entières sont converties en FP avant même que toute opération soit effectuée sur les valeurs FP résultantes.

Quoiqu'il en soit, avec Peersoft installée, vous pouvez maintenant disposer de variables entières comme variables de boucle et les opérations d'incrément et de test de valeur finale utilisent elles aussi l'arithmétique entière.

Une dernière chose, soyez prévenu que Applesoft est buggé dans sa gestion des variables entières. Avez-vous déjà essayé de soumettre l'instruction d'affectation `A% = - 32768` et de se voir insulté par un message d'erreur `?ILLEGAL QUANTITY ERROR ?`

Les utilisateurs curieux d'en savoir plus peuvent se reporter à l'excellente référence sur le Web déjà mentionnée dans ce document (http://www.txbobsc.com/scsc/scdocumentor/). Il suffit ici juste de préciser que `A% = - 32767.5` marche bien et fournit le même résultat attendu (i.e. affectation de la valeur -32768).

Une autre limitation que le lecteur doit mémoriser est la suivante : la valeur finale d'une boucle mettant en œuvre des variables entières ne peut être 32767 (qui est la valeur algébrique la plus élevée qu'un entier signé sur 16bits peut atteindre). Ceci parce que, comme la dernière itération d'une telle boucle (la variable de boucle atteignant alors la valeur 32767) se termine, la première opération effectuée de l'instruction NEXT sera d'incrémenter cette valeur de 1 (pas de progression de la boucle par défaut), cette opération d'incrément va causer un dépassement de capacité (OVERFLOW) à l'intérieur du 6502 et, par cascade, une condition d'erreur au niveau Applesoft "`?OVERFLOW ERROR`" signalé au programme Applesoft.

Pour compléter cette section sur une note optimiste, ceci est mon conseil en regard du placement des variables de boucles FOR/NEXT. Le point à garder en mémoire est que:

- Il est sans objet de laisser une variable de boucle au sommet de la table des variables simples, à moins que, soiit vous l'utilisez intensément dans le corps de la boucle ou, pour une raison qui vous est propre, voulez utiliser la structure `NEXT <variableName>` syntaxique pour itérer à l'intérieur de cette boucle. Le placement de la variable de boucle ne joue aucun rôle dans les aspects performance lors du traitement de l'instruction `NEXT` sans précision de nom de variable.

La référence (i.e . l'adresse physique de la variable de boucle est alors prélevée de la pile du 6502). Cette pile est construite une fois pour toute lors du traitement de l'instruction `FOR`.

Corrections d'anomalies sur les instructions Applesoft existantes

Chaque correction d'anomalie fournie ici est issue de la lecture (et j'espère la compréhension) de la ressource Web (http://www.txbobsc.com/scsc/scdocumentor/).

Instruction ONERR

L'implémentation courante par Applesoft pour l'instruction `ONERR` résultait au fait que les instructions suivantes sur la même lignes étaient ignorées. L'exécution reprenait à la ligne physique suivante. Ceci est corrigé dans cette version de Peersoft.

Instructions RETURN et POP

Gestion de la souris et du timer : une nouveauté Peersoft

Le support pour la gestion de la souris et du timer (vertical blanking) à l'intérieur de Peersoft se base sur les composants suivant:

- Les nouvelles instructions `MOUSE` et `TIMER` pour activer et désactiver les interfaces respectives, ainsi que les configurer en mode «transparent » comme en mode d'interruption;
- Les nouvelles instructions `ON MOUSE GOSUB` comme `ON TIMER GOSUB` pour établir les sous programmes Applesoft qui seront lancés sur apparition d'interruption ;
- La nouvelle fonction `MOUSE` pour retourner le statut de l'interface souris (quelque soit le mode d'exécution, i.e. transparent ou basé sur des interruptions).
- De même, la fonction `TIMER` est utilisé to pour retourner quelques paramètres utilisés pour le déclenchement de l'interruption TIMER.

Merci de noter que seul le mode transparent de la souris est supporté sur ces configurations matérielles: Apple 2, 2+ ou //e with ROM non mises à niveau; Aucune fonctionnalité de type TIMER n'est supportée sur ces configurations. La raison tient dans la gestion du dispositif de traitement d'interruption et l'utilisation d'emplacements stratégique dans la page zéro par ce dernier non dépourvu de conflit avec le DOS 3.3. je conseille aux possesseurs d'Apple //e anciennes ROM de mettre à niveau ce dernier.

ON MOUSE GOSUB <numérodeLigne>

L'utilisation de cette construction syntaxique va déterminer quelle sous-routine Applesoft va traiter les interruptions provenant du déplacement de la souris ou de l'appui sur le bouton de cette dernière. Cette instruction au sein du flux d'un programme, doit précéder une instruction « MOUSE ON, mode » où mode est > 0 sans quoi une erreur sera générée par le programme. Le lancement du programme (instruction Applesoft RUN ou commande DOS RUN/LOAD) réinitialisera la configuration mise à jour par cette instruction..

MOUSE OFF

Cette instruction va éteindre l'interface souris, rendant toute action sur celle-ci inopérante, ou remontant une erreur.

Lorsque le programme démarre (toujours par l'utilisation de la commande Applesoft RUN ou de commande DOS RUN/LOAD), l'interface souris est considéré comme éteinte. Comme la carte périphérique de la souris traite à la fois les interruptions souris et « timer », cette carte est elle même « mis au repos » dès lors que les deux sources d'interruption sont éteintes.

MOUSE STOP

Cette instruction sert à temporairement inhiber la cascade des événements du gestionnaire d'interruption souris à l'intérieur de Peersoft (écrit en ML 6502) vers la sous routine Applesoft dévolue à ce traitement.

Ainsi le code en cours d'exécution peut se conclure avant d'être interrompu par une nouvelle interruption.

À noter que une instruction `MOUSE STOP` est exécutée implicitement dès l'entrée de la routine Applesoft de gestion d'interruption souris (celle annoncée par l'instruction `ON MOUSE GOSUB`) . Ceci afin d'éviter des boucles sans fin si Applesoft met trop de temps à traiter l'événement. Une instruction `MOUSE ON` est alors implicitement exécutée en sortie d'une telle routine. Notez que dans ce cas, les interruptions sont toujours traitées dans Peersoft mais l'appel à la sous routine Applesoft est temporairement désactivé.

MOUSE ON[, mode]

Cette instruction initialise l'interface souris. Les valeurs permises pour le mode sont les suivantes:

Value	Meaning
1	Mode transparent: aucune interruption liée à la souris (i.e. mouvement ou pression sur le bouton) n'est considéré plus avant dans le flux du programme
3	Mode interruption: juste les mouvements de souris sont reportées par interruption
5	Mode Interruption: juste les appuis sur le bouton de souris sont reportés par interruption
7	Mode Interruption: report d'alafois les mouvements de souris et les pressions ssur le bouton sont notifiés

Fonction MOUSE

Cette fonction accepte un unique argument et retourne soit l'abcisse X du curseur de la souris, soit l'ordonnée Y, soit le statut des boutons (en cours et précédents) de la souris.

Arg. Value	Signification
0	Coordonnée en X (valeur signée sur 16 bits)
1	Coordinée en Y (valeur signée sur 16bits)
2	Statut du bouton (valeur non signée sur 8bits)

Fonctions auxiliaires pour gérer d'autres aspects de la souris dans Peersoft

A partir de l'étude de l'API de l'interface souris que vous pouvez trouver dans tout manuel de référence, vous découvrirez que les fonctions décrites dans les paragraphes précédents ne sont qu'une partie des fonctions de l'API.

Pour couvrir l'intégralité de l'API dans Peersoft, un nouveau « reason code » pour la fonction utilitaire principale a été mis en place. L'extrait de code suivant en est une illustration :

```
10 RE = PEEK (40160) + 256 * PEEK (40161)
20  MOUSE   ON ,1: REM Mouse in transparent mode
30  CALL RE,10,5,1,40,2,23: REM Clamp limits set from (1,2) to (40,23)
40 X = 1:Y = 2: CALL RE,10,4,X,Y: REM Position the mouse cursor to (1,2)
50  FOR T = 0 TO 1 STEP 0:S =  MOUSE (2): ON S < 32 GOTO 90:X =  MOUSE
```

```
(0):Y =  MOUSE (1)
60  VTAB 1: HTAB 1: CALL - 868: PRINT "X:"X,"Y:"Y,"S:"S;: ON S < 128 GOTO
90:T = 1
90  NEXT : MOUSE OFF : END
```

Le tableau ci-dessous donne une liste complète de l'API offerte par la fonction utilitaire principale..

Reason subcode	Arguments	Signification
2 (READ)	Xm variable, Ym variable, Sm variable	Peuple les trois variables avec les coordonnées X, Y de la souris ainsi qu'avec le bouton status. Xm, Ym and Sm doivent être des variables entières .
3 (CLEAR)	Aucun	Initialise les coordonnées de la souris à zéro.
4 (POSMOUSE)	X expression, Y expression	Force la position du curseur souris au point (X, Y).
5 (CLAMPMOUSE)	Xmin expression, Xmax expression, Ymin expression, Ymax expression	Force la fenêtre d'évolution de la souris au cadre de (Xmin, Ymin) à (Xmax, Ymax)
6 (HOME)	Aucun	Réinitilise le curseur de la souris à l'ogine du acdre d'évolution (en haut à gauche)

ON TIMER GOSUB <numérodeLigne>

L'utilisation de cette construction syntaxique va déterminer quelle sous-routine Applesoft va traiter les interruptions provenant de l'échéance d'un timer (lui même basé sur l'interruption VBL de la carte périphérique souris). Cette instruction au sein du flux d'un programme, doit précéder une instruction « TIMER ON, factor» sans quoi une erreur sera générée par le programme. Le lancement du programme (instruction Applesoft RUN ou commande DOS RUN/LOAD) réinitialisera la configuration mise à jour par cette instruction.

TIMER OFF

Cette instruction va éteindre l'interface VBL, rendant toute action sur celui-ci inopérante, ou remontant une erreur.

Lorsque le programme démarre (toujours par l'utilisation de la commande Applesoft RUN ou de commande DOS RUN/LOAD), l'interface timer est considéré comme éteinte. Comme la carte périphérique de la souris traite à la fois les interruptions souris et « timer », cette carte est elle même « mis au repos » dès lors que les deux sources d'interruption sont éteintes.

TIMER STOP

Cette instruction sert à temporairement inhiber la cascade des événements du gestionnaire d'interruption VBL à l'intérieur de Peersoft (écrit en ML 6502) vers la sous routine Applesoft dévolue à ce traitement.

Ainsi le code en cours d'exécution peut se conclure avant d'être interrompu par une nouvelle interruption.

À noter que une instruction `TIMER STOP` est exécutée implicitement dès l'entrée de la routine Applesoft de gestion d'interruption souris (celle annoncée par l'instruction `ON TIMER GOSUB`) . Ceci afin d'éviter

des boucles sans fin si Applesoft met trop de temps à traiter l'événement. Une instruction `TIMER ON` est alors implicitement exécutée en sortie d'une telle routine. Notez que dans ce cas, les interruptions sont toujours traitées dans Peersoft mais l'appel à la sous routine Applesoft est temporairement désactivé.

TIMER ON[, factor]

Cette instruction initialise l'interface timer, bien sûr sur le mode interruptif. L'argument optionnel permet de ne générer un événement vers la routine Applesoft que toutes les <factor> interruptions détectées (le défaut de 1 donne une cascade toutes les 1/60èmes de secondes, mais une valeur 60 permet de ne déclencher cette dernière que toutes les secondes).

Fonction TIMER

Deux valeurs possibles pour l'unique argument de cette fonction.

- `TIMER(0)` retourne la valeur du facteur telle qu'elle est prise en compte à partir du traitement d'une instruction `TIMER ON`.
- `TIMER(1)` retourne la valeur courante du compteur interne de 16 bits.

Extrait de code utilisant l'instruction TIMER related instructions

L'extrait ci-dessous affiche un chronomètre pendant que le programme effectue en avant plan d'autre calculs.

```
10  CLEAR : ON  TIMER  GOSUB 100: REM SETS UP THE HANDLING ROUTINE
20 S%= 0: TIMER  ON  ,60: REM ONLY CALLS THE ROUTINE EVERY SECOND.
30  FOR …. : REM MAIN PROCESSING (SPENDING SOME *USER* TIME)
40 VTAB 2: HTAB 1:CALL – 868: PRINT ”THE MAIN PROCESSING TOOK ”S%”
SECONDS TO COMPLETE”
100 S% += 1: VTAB  1: HTAB 1:CALL – 868: PRINT ”SECONDS ELAPSED:”S%;:
RETURN
```

Nouveaux messages d'erreurs liés à laa gestion de la souris et du timer

Le tableau ci-dessous les résume :

Erreur #	Message	Description
32	MOUSE HARDWARE NOT DETECTED	Remonté si aucune carte périphérique souris reconnu au démarrage de Peersoft alors que le programme en cours utilise une fonctionnalité afférante
33	UNSUPPORTED HARDWARE CONFIGURATION	Tentative d'utiliser un mode par interruption sur un Apple 2, 2+ or unenhanced //e
34	UNKNOWN APPLESOFT MOUSE EVENT HANDLER	Rencontre d'une instruction MOUSE ON sans qu'un ON MOUSE GOSUB n'ait été traité auparavant
35	UNKNOWN APPLESOFT TIMER EVENT HANDLER	Rencontre d'une instruction TIMER ON sans qu'un ON TIMER GOSUB n'ait été traité auparavant
36	ILLEGAL MOUSE MODE	Une valeur invalide a été transmise sur l'instruction MOUSE ON.
37	ILLEGAL MOUSE OPERATION	Réponse par un flag Carry positionné lors de l'appel à une routine de l'Apple mouse firmware.

Note aux auteurs en langage assembleur

Dès lors que Peersoft traite une instruction `MOUSE ON` ou `TIMER ON` avec comme effet de passer d'un mode de gestion transparente (valeur < 2) vers un mode de gestion par interruption de la carte souris (valeur > 1), Peersoft positionne alors sur le vecteur IRQV en page 3 ($03FE-$03FF) son propre vecteur et stocke le contenu d'origine dans un endroit au froid. Dès que Peersoft traite une interruption qu'il ne reconnaît pas lui même, alors il passe le contrôle au vecteur d'origine qu'il aura trouvé, sauf si ce dernier est l'adresse $FF65 en ROM, auquel cas, le gestionnaire Peersoft effectue un retour par l'instruction 6502 RTI.

Réciproquement, le vecteur IRQV est restauré à sa valeur d'origine dès que le mode souris/timer passe d'une valeur > 1 (mode de gestion par interruption) à un mode transparent ou éteint (communément par le programme se terminant et les instructions `TIMER OFF`/`MOUSE OFF` étant rencontrées).

Rappel : Le système de gestion d'interruption de Peersoft est seulement en place sur des hôtes cette configuration (interface Applemouse // présente avec enhanced //e or //c, //c+ ou //GS).

Note aux auteurs d'applications Applesoft

Quelques instructions Applesoft peuvent potentiellement prendre un temps (macroscopique) certain avant de rendre le contrôle à la boucle d'évaluation. J'entrevois trois cas en particulier :

- L'instruction `WAIT` qui attend indéfiniment jusqu'à ce que une cellule dans la RAM (ou sur une carte d'interface) se conforme à une configuration donnée. L'interface en question pouvant par exemple être le clavier ou le joystick ($C0nx range) ou une cellule dans la mémoire RAM principale (premiers 48K) et fixé par une routine d'interruption donnée;
- L'instruction `GET` lorsque le flux d'entrée en cours provient du clavier;
- l'instruction `INPUT` lorsque le flux d'entrée en cours provient du clavier;

Peersoft dans son incarnation actuelle fournit une version patchée de la routine de traitement de 'l'instruction `WAIT` de sorte que, si une interruption survenait alors que une instruction `WAIT` était en attente, alors:

- ce qui tient lieu de contexte pour l'instruction `WAIT` est sauvegardé;
- La sous routine Applesoft en charge du traitement en cascade de l'interruption est immédiatement lancée;
- Dès que la routine Applesoft redonne la main, alors, au lieu d'aller se brancher directement àau début de la boucle d'évaluation Applesoft (NEWSTT) , contrôle est redonné à l'instruction WAIT par restauration du dit contexte préalablement sauvegardé. La boucle interne à WAIT peut alors se poursuivre.

L'implémentation de patches similaires pour les deux autres cas d'usage nécessitent je pense de creuser plus avant dans les couches successives du firmware Apple et je préfère passer mon temps sur d'autres sujets plus satisfaisants...)

Co routines dans Peersoft

Un petit avertissement: ce paragraphe décrit le fonctionnement des co routines apparues avec la version 1.4 de Peersoft. Des versions futures pourront exposer d'autres API que celles décrites dans ce paragraphe. Si cela devait arriver alors cette section serait mise à jour en conséquence.

Jusqu'à maintenant (avant que la version 1.4 de Peersoft ne sorte), alors, les programmes Applesoft ne pouvaient être considérés que comme un bloc monolithiques exécutant du code en séquence sous la direction d'un interpréteur Applesoft basique.

À partir de maintenant (i.e. après la délivrance de la version 1.4 de Peersoft), un programme Applesoft peut être considéré comme une succession :

- de phases pendant lequel il garde son fonctionnement d'avant ;
- pendant lesquelles il peut exécuter en parallèle plusieurs segments de codes indépendants, et ceci en //, ces segments, sont appelés dans la suite du document présent des co routines car ils prennent la forme de sous routines Applesoft classique.

Peersoft fournit des outils pour passer d'un mode de fonctionnement à un autre.

Activation du mode « co routines » (exemple simplissime)

L'extrait suivant donne la façon de passer du mode classique au mode avec co routines:

```
5 DIM I0%(127),I1%(127), I2%(127)
10 RE! = PEEK( 40160) + 256 * PEEK( 40161)
20  PRINT "ACTIVE CO ROUTINES PHASE ABOUT TO BEGIN ON LINES 1000, 2000
AND 3000"
30  CALL RE!,4,IT%,I0%,0,0,0,1000,I1%,0,0,0,2000,I2%,0,0,0,3000
40  PRINT "ACTIVE CO ROUTINES PHASE ENDED": END
1000  GOSUB 5000: FOR J0% = 1 TO 2: PRINT J0%;"/";IT%: NEXT
1010  GOSUB 5010: RETURN
2000  GOSUB 5000: FOR J1% = 1 TO 4: PRINT J1%;"/";IT%: NEXT
2001  GOSUB 5010: RETURN
3000  GOSUB 5000: FOR J2% = 1 TO 6: PRINT J2%;"/";IT%: NEXT
3001  GOSUB 5010: RETURN
5000  PRINT "CO ROUTINE #";IT%;" ENTERED": RETURN
5010  PRINT "CO ROUTINE #";IT%;" ABOUT TO QUIT": RETURN
```

La cellule aux adresses 40160/40161 contient un pointeur vers la routine générale utilitaire à l'intérieur de Peersoft.

Les arguments sont décrits dans le tableau suivant :

Table 2: Arguments pour activer les co routines

Argument	Description
4	Reason code signification: Je voudrai activer les co routines avec les paramètres spécifiés ci après.
IT%	Nom de la variable entière Applesoft qui contiendra l'indice de la co routine de 0 à *NumCoRoutines* – 1. Peersoft met à jour cette variable à chaque commutation de contexte).

Argument		Description
1st co routine	I0%	Nom du tableau entier qui contiendra le contexte associé à la co routine (ce contexte incluera : quelques cellules mémoires tirées de la page zéro du 6502 (gestion Applesoft), un segment dans la pile harware.
	0	This parameter defines whether the co routine has a private error handling routine of its own. This parameter should be considered as a bit string here where, for our purpose only the two lsb interest use. Three values are possible here: 0: implies that no error handling at all while the co routine is the one run by CPU. That means that no segment exists in the context dealing specifically with the error handling, making its size smaller and its store and retrieval faster. Whenever the context is restored, a zero is stored in the ERRFLG flag page zero location. 1: Private error handling which instructs Peersoft to cater for dedicated error handling segment within stored context for this co routine. The co routine should however, execute an ONERR GOTO nnn instruction in its own flow of control. 2: The co routine relies on the status of the “global environment” (ie error handling status as the CALL RE!, 4,... is run), a context segment for dealing with error handling is created iif the ERRFLG (page zero location $D8 meaning an ONERR handler is active) is set upon the CALL RE!,4,... is processed by Peersoft. The role of other bits (b2b7 from the value are described in a subsequent section).
	0	This parameter is the address of a machine language subroutine (ending with a RTS instruction) called whenever the co routine is about to be active (gain the 6502 CPU). The sub routine must not change any register value (cf. Push and Pull 6502 instructions)
	0	This parameter is the address of a machine language subroutine (ending with a RTS instruction) called whenever the co routine is about to release control and the corresponding context be stored in the context storage area (see array I0% description above). The sub routine must not change any register value (cf. Push and Pull 6502 instructions)
	1000	This is the co routine starting Applesoft BASIC line number. Consider that, internally, the CALL RE!, 4... does a GOSUB to this line number upon co routine activation.

Les descriptions des champs pour la 2ème et 3ème co routines sont similaires à la description donnée ci-dessus pour la 1ère co routine. Jusqu'à 8 co routines peubt être actives en même temps.

Il est possible, à travers l'utilisation conjointe :

1. de la variable IT % (spécifiée comme un des premiers paramètres du tableau ci-dessus) ;
2. de tableaux mono dimensionnels (pour faire simple) ;
3. des routines d'entrée et de sortie de contexte (définies par co routine dans le tableau ci-dessus) ;

Par exemple, une application désirant conserver une gestion multi fenêtres en mode texte pourra le faire en prévoyant des copies privées des cellules en pages zéro suivantes :

WNDLFT, WNDWDTH, WNDTOP, WNDHGHT pour la position et le dimensionnement des fenêtres sur l'écran,

CH, CV, BASL, BASL+1 pour la gestion du curseur texte à l'intérieur de chaque fenêtre.

Désactiver les co routines

Au delà de la méthode normale et naturelle de revenir à un unique flux d'exécution (celui qui reprendra après l'instruction `CALL RE!,4,…` (par utilisation d'une combinaison de d'instructions `RETURN/POP` au sein de chaque co routine encore active).

Un moyen beaucoup plus radical existe dans l'appel par l'une des co routines de l'instruction: `CALL RE!,5` . Une telle instruction doit être lancée alors que les co routines sont actives.

Structures de données et astuces pour réaliser les tâches usuelles relatives aux co routines

Comme référence à utiliser par les programmeurs en assembleur, voici la structure du segment de la page globale de Peersoft (placée à la page physique $9C sous DOS 3.3)[1].

Table 3 : Peersoft global page

Address (decimal)	Address (hexadecimal)	Description
40159	$9CDF	A call to this address will branch to the Peersoft "general utility" routine already described in a previous section. An alternate way is to get the vector stored at (40159+1, 40159+2) and calling it directly (cf. sample Applesoft above)
40158	$9CDE	Peersoft version byte: currently a $15 value is stored at this location (meaning 1.5)
40157	$9CDD	Number of instructions between two context switches (default to 10, setup whenever Peersoft is loaded from disk).
40156	$9CDC	Bit 7 set iif the MT kernel is active. A call to CALL RE!,4,… will set it up. This flag is reset whenever the MT kernel is terminated, usually as the last co routine returns to the global environment.
40155	$9CDB	Number of ticks that the currently running co routine will last before next context switch. At every context switch, Peersoft copies the $9CDD slot into this slot, upon running an Applesoft instruction, the context switch occurs only if the value from this slot, decremented by one, reaches zero.
40154	$9CDA	Bit 7 set if context switch temporarily inhibited while a critical section of code is run by the current co routine.

Que se passe-t-il lors de l'activation de co routines?

Un segment de pile tout à fait similaire à celui posé lors de la rencontre d'une instruction `GOSUB` est créé. Pour chaque co routine la portion de la pile depuis son sommet courant (propres à l'activité de la co routine) jusqu'à ce point fait partie du contexte sauvegardé par co routine. Cependant, le segment indique que l'emplacement de retour sera celui de l'instruction qui suivra l'instruction `CALL RE!,4` qui aura activé ces co routines. Bien entendu le pointeur de pile vu par chaque contexte est prédécrémenté de 5 positions qui correspond à la taille du segment « GOSUB ».

Peersoft marque une co routine comme terminée dès lors que, une co routine étant active, son pointeur de pile courant revient dépasse (c'est à dire que le pointeur de pile devient inférieur ou égal à celui d'origine).

1 Le segment de cette page allant de $9CEO à 9CFF est réservé à un autre utilitaire de l'auteur, ainsi les deux utilitaires peuvent co exister pleinement du point de vue des réservations au sein de cette page.

Trucs et astuces

Comment faire pour rendre le contrôle aux autres co routines avant l'échéance fatidique ?

Il suffit pour cela d'insérer un `POKE 40155,1` juste avant l'instruction sur laquelle le contrôle doit être rendu à une autre co routine, Peersoft va décrémenter cette valeur à 0 et ainsi une commutation de contexte sera déclenchée (saauvegarde du contexte courant et restauration de la prochaine co routine active enregistrée). Soyez prévenu toutefois que cela peut être la même co routine que celle qui est couramment active si il s 'agit de la dernière encore active.

Comment temporairement inhiber la commutation de contextes?

Alors que une section de code jugée critique est en cours d'exécution dans le flux d'une co routine, il peut être intéressant d'inhiber provisoirement la prochaine commutation qui doit avoir lieu. Une faço simple d'y arriver est d'insérer une instruction `POKE 40154, 128` au début de votre code critique. Afin de revenir à un contrôle normal et de permettre de futures commutations de contextes à partir de cette co routine, un `POKE 40154, 0` en conclusion de la section critique fera son office.

Aussi, comme l'opération de décrémentation du compteur n'est pas opérée pendant que la commutation est désactivée, ce serait peut être une bonne idée d'insérer une instruction `POKE 40155, low_value` juste au voisinage de l'instruction POKE précédente.

Having private variable sets (no collision between co routines)

The current solution I propose is to get arrays of variables with at least one dimension indexed by the context index value. In the tutorials from the disk, I used two arrays (`XH()` and `XV()`) to store cursor data (line and column where cursor lies in two dedicated integer arrays) and all PRINT statements or cursor position setting statements being run in critical sections of code.

Comment se suicider ou commettre un meurtre (d'autres co routines)

Au delà de la façon usuelle pour marquer une co routine comme terminée (i.e. par utilisation d'instructions POP/RETURN en nombre suffisant pour que le pointeur de pile atteigne sa valeur d'origine), une alternative plus intrusive serait de forcer un octet particulier de la mémoire à la valeur $FF, de sorte que Peersoft considère désormais la co routine comme terminée d'office. Voici le segment de code qui réalise l'action.

```
AD =  PEEK( 40152) + 256 * PEEK( 40153): POKE AD + 8 + IT%,255
```

Où IT% désigne l'indice de la co routine actuelle pour indiquer un suicide ou une autre valeur pour indiquer un assassinat de la co routine de valeur d'indice IT % (devant être entre 0 et 7).

Structure accueillant le contexte d'un co routine

Chaque contexte de co routine est stockée dans un tableau d'entiers (une dimension) dont la structure est fournie dans le tableau ci-dessous :

Table 4 : Structure du stockage de contexte

Offset	Page zero	Description
Header for housekeeping by Applesoft		
0 and 1	N/A	Name of the array (two bytes)
2 and 3		Offset from the beginning of this array to next array variable or to end of memory area
4		Number of dimensions (must be 1 for Peersoft usage).
5 and 6		Value of first (and last) dimension
Constant segment (general use)		
7	N/A	Offset to stack segment (always populated)
8	N/A	Operation mode for context. B0b1 provides an indication whether the local error handling is in use or not. In case local error handling is in use, whether the global environment is used for such context or not; B2b7 provides options for additional context switch operations. The one being shown within the tutorial is the display cursor backup/restore operations.
Constant segment for monitoring context switches		
9 and 10	N/A	Address of machine language routine to be called whenever the co routine is paged in. This routine must not alter register values from the calling environment (unless pushed on stack) and must return with a RTS (after possible Pull from stack instructions). High byte is $FF if no routine registered.
11 and 12	N/A	Address of machine language routine to be called whenever the co routine is paged out. This routine must not alter register values from the calling environment (unless pushed on stack) and must return with a RTS (after possible Pull from stack instructions). High byte is $FF if no routine registered.
Core segment (always populated)		
13	REMSTK ($D8)	Current stack pointer for this pointer (only byte at offset 8 is meaningful)
14 and 15	CURLIN, CURLIN+1	Current Applesoft line # for the co routine
16 and 17	TXTPTR ($B8), TXTPTR+1	Current text pointer within program text for the co routine
18 and 19	OLDTEXT, OLDTEXT+1	Text pointer of last instruction parsed by interpreter exec loop
Local Error handling segment (optional: see value at offset 8)		
20 and 21	TXTPSV ($F4), TXTPSV+1	Points to the first character of line # as ONERR GOTO statement is parsed.
22 and 23	CURLSV ($F6), CURLSV+1	Line # where the ONERR GOTO is located
24	ERRNUM ($DE)	Error # when an error occurs
25	ERRSTK	Stack pointer as the error occurs (so that RESUME could branch back to the faulty statement)
26 and 27	ERRLIN ($DA), ERRLIN+1	Applesoft line # where the error occurred (so that RESUME could branch back to the faulty statement)
28 and 29	ERRPOS ($DC), ERRPOS+1	TXTPTR pointer of the statement raising the error.
30	ERRFLG ($D8)	Only bit 7 is meaningful here.
Stack segment (variable size)		
<ValueAt Offset 7> and above	N/A in page zero: within hardware page 1	From private stack pointer to global environment stack pointer value, bytes extracted from hardware stack (page 1) from offset given by REMSTK value at offset 8 from this structure) to the REMSTK known as the `CALL RE!, 4, …` was parsed.

Tutoriel sur les co routines Peersoft

À l'intérieur du listing de programme Applesoft ci-dessous,

- Les tableaux `I0`, `I1` et `I2` servent au stockage des contextes (mémorisation des données considérées comme privées entre deux commutations impliquent la co routine;
- Les tableaux `XH` and `XV` servent de support pour y placer les coordonnées X et Y du curseur texte ceci pour chaque co routine impliquée dans cet exemple (ici au nombre de 3).
- La variable `XC` sert comme indicateur du fait que l'utilisateur a tapé la combinsaison de touches Control-C alors que le programme tournait, indiquant par là sa volonté d'arrêter ce programme. C'est pourquoi sa valeur devient non égale à zéro en ligne `2901` (partie du gestionnaire d'erreurs partagé par les co routines et commencant à la ligne `4000` (voir l'instruction `ONERR` à la ligne `2` et aussi la partie du gestionnaire d'erreur dédié pour la co routine 1 commencant à la ligne `2900`);
- La variable RE contient l'adresse où appeler a routine utilitaire générale de Peersoft avec des paramètres appropriés.

```
1 CLEAR : DEFINT I-N,X: DIM I0(127),I1(127),I2(127),XH(2),XV(2)
2 PRINT CHR$ (4)"PR#0": TEXT : HOME :XC = 0: ONERR GOTO 4000
3 PRINT CHR$ (4)"BLOAD TUTMC": VTAB 1: HTAB 15: PRINT "TUTORIAL 2"
4 XH(0) = 1:XV(0) = 2:XH(1) = 1:XV(1) = 21:XH(2) = 1:XV(2) = 6: DEF FN DR(A) = PEEK (A) +
256 * PEEK (A + 1): DEF FN AR(CX) = INT (CX * 100) * .01
5 RE = FN DR(40160): POKE 40157,4: REM # OF APPLESOFT INSTRUCTIONS RUN BETWEEN TWO
SWITCHES
6 CALL RE,4,IT,I0,2,0,0,1000,I1,1,768,774,2000,I2,2,774,768,3000
7 VTAB 1: HTAB 1: PRINT "PROGRAM ENDED, PRESS ANY KEY";: GET A$: HOME : END
999 REM FIRST COROUTINE: MONITOR EVERY CONTEXT INCLUDING ITSELF
1000 AD = FN DR(40152):OF = 0:NT = 0:SO = PEEK (AD + 17):SL = 0:LX = - 1: GOSUB 5010:
PRINT " RUNNING TASKS STATUS (";SO"/";: GOSUB 5000:XH = XH(IT):XV = XV(IT)
1002 FOR JT = 0 TO 7: ON PEEK (AD + 8 + JT) < 255 GOTO 1003:NT = JT - 1:JT = 7
1003 NEXT JT: FOR J0 = 0 TO 1 STEP 0: GOSUB 1100:JF = 1
1004 FOR JT = 0 TO NT: GOSUB 1200: NEXT JT
1005 J0 = JF: NEXT J0: RETURN
1099 REM
1100 ON PEEK (40157) = LX GOTO 1102: POKE 40154,128: HTAB XH: VTAB XV:LX = PEEK (40157)
1101 PRINT LX;")";: CALL - 868: POKE 40155,1: POKE 40154,0
1102 RETURN
1199 REM PRINT A CONTEXT CONTENT (JT)
1200 IF PEEK (AD + JT + 8) < 255 AND JT < > IT THEN JF = 0
1201 OF = PEEK (AD + JT + 8) * 256 + PEEK (AD + JT):XV(IT) = 3 + JT:XH(IT) = 1: GOSUB
5010: CALL 777,OF,JT: GOSUB 5000
1202 RETURN
1999 REM SECOND CONTEXT: PROCESS SOME KEYBOARD INPUT FROM USER
2000 BS$ = CHR$ (8):CU$ = CHR$ (127) + BS$: POKE 49168,0: ONERR GOTO 2900
2001 GOSUB 5010: PRINT SPC( 6);"DIVISION EXEMPLE": GOSUB 5000:LY = XV(IT): FOR J1 = 0 TO
1 STEP 0
2002 XH(IT) = 1:XV(IT) = LY: GOSUB 5010: CALL - 958: PRINT "ENTER NUMERATOR: "CU$;: GOSUB
2801: ON M$ = "" GOTO 2004:VN = VAL (M$)
2003 GOSUB 5010: PRINT "ENTER DIVISOR: "CU$;: GOSUB 2801: ON M$ < > "" GOTO 2005
2004 J1 = 1
2005 ON J1 = 1 GOTO 2007:VD = VAL (M$):VR = FN AR(VN / VD): GOSUB 5010: PRINT "RESULT:
";VR;" <RET> TO PROCEED"CU$;: GOSUB 2851: ON XC = 1 OR ES% = 1 GOTO 2004: GOTO 2007
2006 POKE 40154,128: VTAB 24: HTAB 1: PRINT MO$;
```

```
2007 NEXT : RETURN
2800 REM INPUT SUBROUTINE
2801 GOSUB 5000:M$ = "":LM = 0:ES% = 0: FOR JS = 0 TO 1 STEP 0
2802 GOSUB 2861: ON ES% = 0 AND XC = 0 GOTO 2803:M$ = "":LM = 0: GOTO 2809
2803 ON JS = 1 GOTO 2809: ON A < > 8 OR LM = 0 GOTO 2804:LM -= 1:M$ = LEFT$( M$,LM + (LM
= 0)): PRINT " "A$A$;CU$;: ON LM > 0 GOTO 2804:M$ = ""
2804 ON A < 31 GOTO 2809:LM += 1:M$ += A$: PRINT A$;CU$;
2809 GOSUB 5000: NEXT
2810 GOSUB 5010: CALL - 868: PRINT : GOSUB 5000: RETURN
2850 REM GET RETURN SUBROUTINE
2851 GOSUB 5000:ES% = 0: FOR JS = 0 TO 1 STEP 0
2852 GOSUB 2861:JS = (ES% = 1) OR (XC = 1) OR (A = 13): GOSUB 5000: NEXT : GOSUB 5010:
CALL - 868: GOSUB 5000: RETURN
2860 REM GET KEYBOARD ENTRY
2861 ON PEEK (49152) > 127 OR XC = 1 GOTO 2862: POKE 40155,1: GOTO 2861
2862 GOSUB 5010: IF XC = 0 THEN GET A$:A = ASC (A$)
2863 ON XC = 0 GOTO 2864: PRINT "#ABORTED#!";:JS = 1
2864 ON A < > 27 GOTO 2865: PRINT "<ESCAPED>";:JS = 1:ES% = 1
2865 ON A < > 13 GOTO 2866:JS = 1
2866 RETURN
2900 ON PEEK (222) < > 255 GOTO 2902
2901 XC = 1:A$ = CHR$ (3):A = 3: PRINT CHR$ (7);: RESUME
2902 ON PEEK (222) < > 133 GOTO 2903:EL = FN DR(218): ON EL < > 2005 GOTO 2903:MO$ =
"DIVIDE BY ZERO ERROR":J1 = 1: CALL - 3288: GOTO 2006
2903 PRINT CHR$ (7);: GOTO 4003
2998 REM 3RD CONTEXT MAIN ROUTINE, JUST PRINT SOME STAR CHARACTERS
2999 REM AS A BACKGROUND ACTIVITY
3000 FOR J2 = 0 TO 1 STEP 0:J2 = J1: GOSUB 3008
3001 PRINT "*";: GOSUB 5000: NEXT
3002 FOR J2 = 1 TO 4: GOSUB 3008: PRINT MID$ ("OVER",J2,1);: GOSUB 5000: NEXT : RETURN
3008 XV(IT) = INT ( RND (1) * 15) + 6:XH(IT) = INT ( RND (1) * 40) + 1: GOSUB 5010:
RETURN
4000 IF PEEK (40156) < 128 THEN VTAB 23: HTAB 1: CALL 771: END
4001 ON PEEK (222) = 255 GOTO 2901
4003 XH(IT) = 1:XV(IT) = 23: GOSUB 5010: CALL 771: GOSUB 5000: CALL RE,5
4998 REM STORE CURSOR POSITION INTO CONTEXT AND RELEASE CONTROL TO MT
4999 REM EXPECTS TO BE CALLED WHILE CONTEXT SWITCHES INHIBITED
5000 XH(IT) = PEEK (36) + 1:XV(IT) = PEEK (37) + 1: POKE 40155,1: POKE 40154,0: RETURN
5009 REM INHIBIT CONTEXT SWITCH AND RESTORE CURSOR POSITION FROM STORED CONTEXT
5010 POKE 40154,128: VTAB XV(IT): HTAB XH(IT): RETURN
```

Gestion concurrente de 11 fonctions utilisateur

Onze instructions `DEFUSR[n]` et dix fonctions `USR<n>` n de 0 à 9 ont été ajoutées pour implémenter cette fonctionnalité.

Forme simple pour les routines classiques en langage machine

Il existe un mode simple de définir les points d'entrée des routines en langage machine.

```
DEFUSR[n] = <parm1>
```

La routine est alors censée accepter un unique argument et la valeur spécifiée *parm1* représente l'adresse du point d'entrée qui est alors censée être > à 255. Si n n'est pas précisé, alors c'est le vecteur usuel en $0A..$0C de la fonction `USR` en Applesoft qui est mis à jour par cet ordre.

Lors de l'appel par la fonction `USR[n]`,

- La valeur de l'unique argument est disponible pour la routine dans l'accumulateur `FAC` de la page zéro (fonctionnement inchangé par rapport à Applesoft classique) ;
- Le type de l'unique argument reste disponible dans la cellule `VALTYP` en page zéro.

Forme plus évoluée pour les routines en langage machine acceptant deux arguments

Dès lors que la routine doit accepter deux arguments au lieu d'un, une forme alternative existe :

```
DEFUSR[n] = <parm1>,<parm2>
```

Où *parm1* désigne un mode de fonctionnement dont la valeur ici doit être de 64 (plus de détails dans un tableau ci-dessous) et *parm2* désigne l'adresse du point d'entrée de la fonction.

Lors de l'appel par la fonction `USR[n]`,

- La valeur du premier argument est disponible pour la routine dans l'accumulateur ARG de la page zéro;
- Le type du premier argument (`VALTYP`) se trouve dans la cellule en page zéro à l'adresse $BE ;
- La valeur du deuxième argument est disponible dans l'accumulateur principal FAC ;
- Le type du deuxième argument reste disponible dans la cellule `VALTYP` en page zéro.

Je vous invite à inspecter le source assembleur `TCUSRFNDEMO.S` qui fournit des exemples de routines en langage machine pour des exemples d'implémentation de fonctions simples en assembleur.

Forme évoluée pour appeler des routines utilisateur écrites en Applesoft

L'autre nouveauté de Peersoft relative à la définition et à l'appel de fonctions utilisateur tient dans ce que l'on appelle les « procedural functions », qui seraient l'équivalent dans d'autres dialectes de fonctions utilisateur définies en BASIC sur plusieurs lignes.

Selon le nombre d'arguments admissibles, les instructions peuvent adopter l'une des deux formes :

- `DEFUSR[n] = <parm1>,<nomVariableSortie>,<nomVariableEntrée>,<numéroLigne>`
  si la fonction n'accepte qu'un unique argument, *parm1* ayant comme valeur 128 ou 129 (cf. tableau ci-dessous).

- `DEFUSR[n] = <parm1>,<nomVariableSortie>,<nomVariableEntrée1>,<nomVariableEntree2>,<numéroLigne>`
  si la fonction accepte deux arguments, *parm1* ayant alors comme valeur 192 ou 193 (cf. tableau ci-dessous).

Dans cette version de Peersoft, les variables servant de passerelle comme les arguments des fonctions ne peuvent être que numériques (FP ou de type entier)[2].

Pour discriminer la déclaration d'une PF de la déclaration d'une routine utilisateur en langage machine, on joue sur la valeur de *parm1* tel que spécifié dans l'ordre `DEFUSR` et particulièrement de son bit 7 ce qui donne le tableau suivant :

# du bit	Signification si positionné à 1	Signification si positionné à zéro
7	Indique la déclaration d'une PF	Déclaration du point d'entrée d'une routine en langage machine
6	Indique que deux arguments sont attendus	Unique argument attendu
1	Dans le cas d'une PF, indique un segment de stockage des variables séparé lorsque le corps de la routine s'exécute. On parlera alors de PF « dynamique ».	Dans le cas d'une PF, indique que, lorsque la fonction est évaluée, elle l'est dans le même contexte de stockage des variables que l'appelant. On parlera alors de PF « statique ».

Certaines règles doivent cependant être respectées dans la gestion des PF au sujet des contextes où ces dernières peuvent ou non être appelées.

1. On ne peut appeler une PF dans le corps d'une PF ;
2. On se contentera d'appeler une PF dans une unique co routine (si le mode de fonctionnement par co routine est actif) ;
3. On évitera d'appeler des PF dynamiques dans des contextes de traitement des interruptions.

Il est possible que certaines de ces limitations tombent dans une version future de l'outil.

Nouveaux messages d'erreur liés à la gestion des PF

Erreur #	Message	Description
38	EMBEDDED PF NOT SPPORTED IN THIS RELEASE	Remonté si l'appel à une PF a lieu alors que une fonction PF est en cours d'exécution.
39	ILLEGAL OP WHILE PF IS ACTIVE	

2 Une version ultérieure permettra le passage de données chaines de caractères en entrée comme en sortie (i.e. comme résultat).

Table of content