TOOLBOX mag

n°5

Juin - Juillet 1991

Prix: 590 F par abonnement.

Sommaire de la revue :

Présentation de la disquette ToolBox Mag Numéro 5

Pas beaucoup d'images dans la revue, et pas de couleurs. Et pourtant, ToolBox-Mag est en images, en couleurs, et en musique. Comment cela? Eh, la disquette, pardi... Voici son catalogue:

/COMP: la présentation de ce numéro 5. Les graphismes sont de *Nicolas Bergeret*, la musique de *Sébastien Mousseron*, l'animation de *Dominique Delay*, lequel fête ainsi en fanfare son entrée au Conseil de Rédaction de ToolBox-Mag (où il remplace Y. Kœnig). Bootez la disquette!
/ICONS: une icône pour le Finder, pour présenter l'application EdVersion de *Bernard Fournier*.
/ASM.ET.PAS.SRC: deux programmes, avec leurs sources, de *Bernard Fournier*, démontrant l'insertion de routines assembleur en TML Pascal et en Orca Pascal. Voir pages 16 à 19.
/EDITVERSION: l'éditeur des nouvelles ressources version de *Bernard Fournier*, avec ses sources. Voyez pages 32 et 33. Attention, malgré les apparences, le fichier *EdVers.R* ne doit pas être lancé. Ce ne sont que les ressources de la vraie application, qui s'appelle *EdVers*.
/GENESYS: les fichiers "New Application" générés, en divers langages, par l'éditeur de ressources *Genesys*. Voyez l'article de *François Uhrich* pages 4 à 7.
/LASER.SANS.MAC: une fonte PostScript et sa fonte-écran, à télécharger sur l'une de vos LaserWriter. Plus un fichier d'anti-démo d'Appleworks-8 avec la Laser. Voyez l'article de *Jean-Yves Bourdin* pages 8 à 15.
/LYNX: l'accessoire anti-plantages, avec ses fichiers sources, de *Patrick Desnoues*. Voyez page 31. N'empêche, qu'est-ce qu'il a de beaux yeux...
/RINCLUDE: un fichier d'includes à jour pour votre catalogue Includes pour Rez d'Orca/APW. Ce fichier est de *Patrick Desnoues et Bernard Fournier*, il est indispensable pour pouvoir assembler et compiler les programmes des mêmes.
/SELECT: le magnifique sélecteur graphique de *Jean Destelle*, avec ses sources complets. Le sélecteur lui-même est constitué par le fichier *SelectSys*. Voyez pages 22 à 29. Attention, le fichier *Select0.rsrc* est un source de ressources pour l'application *ResDoctor*, laquelle vous est fournie dans le numéro 6 de ToolBox-Mag.
PRODOS est un faux, comme d'habitude. Voyez l'explication page 30 du numéro 6 de ToolBox-Mag.

ToolBox-Mag

Directeur de la Publication
Eric Weyland

Rédacteur en Chef
Jean-Yves Bourdin

Mise en pages
Dominique Digoy

Secrétariat
Janine Loiseleux

Conseil de Rédaction
Jean-Pierre Charpentier
Dominique Delay
Patrick Desnoues
Bernard Fournier
Olivier Goguel
Stéphan Hadinger
François Hermellin
Hubert Loiseleux
Claude Pélisson
Jean-Luc Schmitt
François Uhrich

Rédaction, Siège Social
ToolBox-Mag
6, rue Henri-Barbusse
95100 Argenteuil - France
Téléphone: (1) 30 76 18 64
Télécopie: (1) 39 47 44 08

Impression
Imprimerie /Reprographie /Graphisme - Argenteuil

Les auteurs de ToolBox-Mag n° 5

Nicolas Bergeret
Jean-Yves Bourdin
Dominique Delay
Patrick Desnoues
Jean Destelle
Bernard Fournier
François Hermellin
Sébastien Mousseron
François Uhrich
Eric Weyland

Avertissements légaux

ToolBox-Mag est un magazine qui prend l'Apple II GS au sérieux. De ce fait même, il est indépendant d'Apple Computer et d'Apple Computer France, auxquels il n'est, grâce au ciel, pas rattaché.

ToolBox-Mag est disponible uniquement par abonnement et par correspondance. Le magazine est composé inséparablement d'une ou plusieurs disquettes magnétiques pour ordinateur Apple // GS et d'une revue sur papier.

Magazine indépendant, ToolBox-Mag permet à ses auteurs de s'exprimer sous leur propre responsabilité. Leurs propos n'engagent ni ToolBox-Mag ni la Société Toolbox.

ToolBox-Mag ne peut pas offrir une garantie supérieure à celle qu'offre le chapitre "Limitations de Garantie et de Responsabilité" des documentations officielles Apple, soit quasiment rien.
Ecrit par des amateurs du GS, il s'engage néanmoins à publier des programmes ne contenant pas plus de bugs que GS Write, et à maintenir un standard de compétence au moins égal au niveau moyen des concessionnaires Apple français en matière d'Apple II GS...

L'Editorial du Directeur :

HyperCard et le Père Noël

Mr René Coustal, de l'Aude, a gagné une prolongation de son abonnement: il a été le premier à découvrir que le numéro de la "Référence Produit" du soit-disant "bon pour HyperCard gratuit" de notre précédent numéro, traduit en codes Ascii hexadécimaux, donnait le mot... "*Poisson*"! Oui, l'encadré de la page 28 du numéro 4 (le numéro d'Avril !) de ToolBox-Mag était tout entier **un poisson d'Avril**.
Toutes nos excuses à ceux d'entre vous qui ont mordu à l'hameçon, mais... nous avons bien ri. Méfiez-vous pour l'année prochaine: depuis Wozniak, **l'Apple II a le sens de l'humour.** La vie est trop courte pour développer sous 4D ou compiler en DBase IV!

Apple-France a trouvé la parade à notre poisson: depuis le 18 Avril, selon le 3614 Apple, il vend HyperCard GS (en version US) comme **un produit commercial normal**, par ses concessionnaires *[espérons que, ce coup-ci, ça n'est pas un poisson!]*.
S'il n'a pas la compétence pour en faire une version française, peu importe: HyperCard, c'est surtout un langage (HyperTalk), qui ne doit pas être francisé. Quant aux documentations françaises, les livres sur HyperCard pullulent.

Notre canular contenait-il une critique déguisée d'Apple France? Oui et non. **Oui**, parce qu'on dessert sciemment le GS en mettant dans sa boîte l'inexcusable GS-Write, produit non-Apple, au lieu de ce magnifique produit Apple qu'est HyperCard couleur. **Non**, parce que, Mac ou GS, si vous voulez un HyperCard complet, il faut l'acheter, la version monochrome livrée avec les Macs étant réduite à une sorte de squelette pour sous-développés. **Non aussi**, parce qu'Apple n'a pas à jouer les Père Noël, mais à assurer le service normal du produit qu'il nous a vendu.
Certes, Apple n'est pas sans défaut: quand son service est défaillant, nous le relevons (voyez l'article sur "la Laser sans le Mac"); quand des concessionnaires Apple, pour vendre d'autres produits, qualifient d'"obsolète" la seule machine au monde capable de faire tourner Hypercard en couleurs, ils ne font que confirmer leur incompétence.
Mais dans le fond, Apple peut bien, de son côté, faire, dire et vendre ce qu'il veut: nous avons nos GS, nous en faisons ce que nous voulons. Nos GS, **c'est notre affaire**.

SynthLab, InWords, Genesys, les nouveaux Orca, vous avez tout ça? **"Gate"**, le nouveau jeu GS en français, vous êtes arrivé au bout? HyperCard GS, ses six disquettes et ses trois manuels, vous avez tout exploré?
Eh bien, pas moi. Je vais donc continuer à me servir de mon GS; en commençant par apprendre **un extraordinaire logiciel d'Apple, qui n'existe sur aucun autre ordinateur personnel: HyperCard couleur.**

ERIC WEYLAND

C facile 5 :

C facile... avec Genesys !

François Uhrich

Dans cette 5° édition de "C Facile", il ne sera pas question du C! C'est qu'il est temps pour ToolBox-Mag de parler d'un utilitaire indispensable pour tous ceux qui programment l'Apple II GS, en C ou dans un autre langage: Genesys.

Depuis novembre, la **version 1.2** de l'Editeur de Ressources **Genesys** est enfin disponible: quatre disquettes (deux pour Genesys et deux pour le système 5.04 complet) et une nouvelle édition complète de la documentation (près de 130 pages).

Les utilisateurs de la première version avaient pu se plaindre de pas mal de plantages dans l'édition des ressources. Ces problèmes ont aujourd'hui disparu, et cela est dû moins à des corrections dans Genesys qu'à la **nouvelle version du système** 5.04. En effet de graves problèmes imputables à la première version du Resource Manager ont été corrigés à Cupertino.

Genesys et les ressources permettent au programmeur, avant même que la moindre ligne de programmation n'ait été écrite, de créer des menus, des fenêtres, des contrôles, des icônes, des curseurs, des textes, etc. Bref tout l'environnement graphique que son œuvre, en Assembleur, en Pascal, en C ou en ce qu'il voudra, utilisera une fois achevée. A ce stade final, le programmeur (et dans une moindre mesure l'utilisateur) pourra toujours modifier les ressources du programme sans avoir nullement besoin de ré-assembler ou re-compiler des milliers de lignes de sources!

Je me rappelle aujourd'hui sans nostalgie aucune, le temps, la patience et la rage qu'il fallait pour créer quasiment en code hexadécimal une fenêtre avec quelques contrôles à l'intérieur! Sans parler des multiples re-compilations parce que le contrôle au fond à droite après le deuxième feu rouge était encore décalé d'un pixel sur la gauche... A l'époque, le programmeur devait aussi utiliser des ressources... mais nerveuses celles-là!

Avec Genesys, vous pouvez par exemple créer une fenêtre, la déplacer, modifier sa taille, choisir ou non la présence d'un titre, d'un ou des ascenseurs, d'y ajouter les contrôles que vous voulez parmi les douze de types étendus (Simple Button, Check Box, Icon Button, Line Edit, List, Picture, Pop-up Menu, Radio Control, Scroll Bar, Size Box, Static Text et enfin Text Edit). Tout cela est bien sûr visible immédiatement à l'écran. Pour changer de place un contrôle, il vous suffit de cliquer sur celui-ci et de le déplacer à la souris!

Le Shell Genesys

L'environnement Genesys est du type graphique, souris, menus déroulants, aujourd'hui classique. A côté des menus habituels *Apple*, *File*, *Edit*, on trouve d'abord le menu *Layout*: *Show Coordinates* pour faire afficher en continu les coordonnées de la souris, *Set Origin* pour modifier l'origine des coordonnées et enfin *Reset Origin*. Il est à remarquer qu'au moment de changer la taille d'une fenêtre ou de changer la position d'un contrôle, le repère relatif à la fenêtre considérée est immédiatement positionné. Il est donc inutile d'utiliser *Set Origin* dans ces cas courants.

Le menu suivant est consacré aux fenêtres: *Windows* (*Hide "filename"*, *Show "filename"*, *Show Next Window*, *Center Horizontally*, *Center Vertically*, *Center Both*).

Enfin le contenu du menu *Options* est fonction de la ressource en cours d'édition. C'est par exemple grâce à ce menu que l'on pourra ajouter un contrôle à une fenêtre (*Add Control*). De nombreuses ressources peuvent être testées immédiatement comme les fenêtres avec leurs contrôles, les menus, etc.

Dans la version actuelle, Genesys ne fonctionne qu'en mode 640. Il est toutefois possible de créer les ressources pour la résolution 320: pour cela utiliser uniquement la première moitié de l'écran! Visuellement, c'est tout de même assez gênant...

▶

Il est à remarquer que tout le travail effectué sur un fichier ressource est en fait réalisé sur disque (dossier Gen.Work) sur un fichier de travail que Genesys a la bonne idée de créer à partir de l'original. Les modifications ne sont donc réellement prises en compte qu'après une sauvegarde.

D'autre part dans le menu *File*, l'option *New Application* permet la création automatique des ressources standards de base: menus *Apple*, *File* et *Edit* complets, alerte *About*, et *Tool Table* pour l'initialisation des Outils.
L'option *Import* permet de recopier un fichier data dans une ressource quelconque (on pense immédiatement aux images, aux sons et même aux codes ressources...).

Design Master : un concurrent de Genesys ?

Design Master est un logiciel édité par Byte Works Inc. D'après le label de qualité habituel de cet éditeur (Orca/M, Orca/Pascal, Orca/C, Orca/Disassembler), on peut s'attendre à ce que ce logiciel soit à nouveau une totale réussite. C'est raté!

Design Master permet la création de menus, de fenêtres et de dialogues; de sauvegarder ces objets sous la forme de ressources (sauf les dialogues puisqu'il n'y a pas de ressources pour le Dialog Manager) ou de sources en C, Pascal et Assembleur.
J'ai bien utilisé le mot **création** et pas **édition**, car les objets (en fait un unique objet par fichier...) ne peuvent être sauvegardés que sous la forme d'un fichier au format propre à Design Master. Il est donc **impossible de charger et de modifier une ressource quelconque!** C'est dire l'intérêt plus que limité de ce logiciel.

Le mode de construction des objets est une succession de dialogues peu pratiques par rapport à la fluidité de Genesys. Un bon point pourtant: la possibilité de travailler en mode 320 ou 640.

En fait l'atout principal (faute d'autres atouts) du logiciel est son **manuel d'utilisation**: ses 192 pages sont un véritable cours d'utilisation des ressources ou des sources générés par Design Master.

On doit parfois supporter qu'un bon logiciel ait un mauvais manuel, mais certainement pas l'inverse...

F.U.

Les Editeurs de Ressources

Une autre excellente idée: chacun des éditeurs de ressource est donné sous la forme d'un fichier situé dans le catalogue */Gen.Edit*. Ainsi sans modifier le Shell de Genesys, de nouveaux éditeurs ou de nouvelles versions de ceux déjà existants pourront voir le jour! Les programmeurs pourront alors s'en donner à cœur joie, mais uniquement lorsque la structure de ces éditeurs aura été publiée, car à l'heure actuelle cette structure ne semble pas encore stabilisée.

Voici les éditeurs joints à Genesys 1.2 avec quelques remarques sur chacun d'eux:

✔ **Alert Window, Error Window:**
• Choix des Icônes aisée parmi *Stop, Note, Caution, Disk et Disk Swap*; il manque toutefois la possibilité de choisir une icône personnalisée.
• Les neuf tailles d'alerte sont possibles, mais là aussi il manque la possibilité de donner une taille quelconque.
• Le message de l'alerte peut être saisi directement, mais il manque les possibilités de justification, de changement de fonte, de couleur.
• Les boutons sont facilement créés (limitation normale à 3), modifiables et interchangeables.
✔ **CDev Flags**: Dialogue simple et complet.
✔ **Control**: les douze types de contrôles étendus sont facilement utilisables. A noter toutefois que les tables de couleurs sont presque totalement absentes; les listes demandent encore pas mal de travail de programmation: mais c'est normal, car les éléments d'une liste sont le plus souvent créés par le programme et ne sont donc pas des ressources.
✔ **HexAscii**: quand il n'y a pas d'éditeur disponible, la ressource concernée est visualisée sous la forme hexadécimale et Ascii. Toutefois il ne s'agit que de visualisation, pas de modification.
✔ **Icon, Cursor**: deux éditeurs analogues et très complets avec déplacement (shift) et remplissage de couleur (fill) pour les icônes.
✔ **Menu Bar, Menu, Menu Item**: à la création d'une barre de menus, les menus standards suivants sont créés (la barre sera vide si la touche Option est enfoncée):
• *Apple (About).*
• *File (New, Open, Close, Save, Save as, Revert to saved, Quit).*
• *Edit (Undo, Cut, Copy, Paste, Clear).*
✔ **Picture:** les images ne peuvent être créées que par l'intermédiaire du Presse-Papier (ClipBoard); ce n'est pas un problème, car cela permet l'utilisation de logiciels de dessin comme *Platinum Paint*, dont un éditeur dans Ge- ▶

nesys pourrait difficilement atteindre la qualité.
✔ **C String, C1 Input String, Pascal String, Text, Text Block, TextBox2 String, W String**: un unique éditeur gère ces différents types de chaînes de caractères. Le couper/coller est utilisable. Une particularité du type *TextBox2* est la présence de commandes de justifications, de changements de fontes et de changements de couleurs.
✔ **Tool Table**: dialogue simple d'emploi qui permet de créer la ressource utilisable par StartUpTools pour le chargement et l'initialisation des Outils.

Genesys et le langage REZ

Certains éditeurs actuels de Genesys n'offrent pas toutes les possibilités prévues dans la toolbox: l'absence de table de couleurs pour les contrôles et l'impossibilité de gérer la mise en page d'un texte d'une alerte sont deux exemples significatifs.
La seule solution est de générer vos ressources grâce à Genesys sous forme de source langage REZ; de modifier ce source et de le compiler pour retrouver vos ressources définitives. La version 2.0 de Genesys devrait supprimer cet inconvénient.

Je continuerai quand même de recommander la sauvegarde de la version finale de vos ressources sous REZ, car la gestion des ressources par le Resource Manager est des plus surprenante… En effet, quand vous modifiez une ressource et que celle-ci augmente en taille, le Resource Manager libèrera l'emplacement précédent pour écrire la nouvelle ressource en fin de fichier ressource. Il s'ensuit un processus de fragmentation interne du fichier comparable à celle d'un disque! Le fichier ressource peut voir ainsi augmenter sa taille de façon surprenante...

Une bonne habitude à prendre par les programmeurs est donc de sauvegarder en langage REZ leur fichier ressource puisque la compilation REZ produira un fichier ressource non fragmenté.

Si le programme doit être amené à modifier certaines de ses propres ressources, il est fortement recommandé de placer ces ressources modifiables *en fin de source REZ*, tout spécialement celles qui risquent d'augmenter en taille. Relisez aussi l'article de Bernard Fournier dans ToolBox-Mag 4.

F.U.

✔ **Two Rects**: dialogue permettant la saisie des coordonnées de deux rectangles. Ce type de ressource est souvent utilisé pour conserver la dimension d'une fenêtre à la fois en résolution 320 et 640. Un bon exemple est celui du NDA Tableau de Bord.
✔ **Window**: sans nul doute l'éditeur le plus utilisé avec celui des contrôles. Toutes les composantes d'une fenêtre sont ici modifiables: taille, position, zoom, titre, barre d'info, ascenseurs, etc. Sans oublier bien sûr la possibilité d'ajouter des contrôles et de les déplacer.
✔ **Window Color**: toutes les composantes de la table de couleurs d'une fenêtre peuvent ici être modifiées. Les couleurs du titre (texte, fond, pattern), la couleur du contour, des bords d'une alerte, etc.
[Attention toutefois avec la table de couleurs, car un bug encore présent dans le système, et répertorié par Apple, peut réserver quelques mauvaises surprises à votre programme dans le cas où la table de couleurs est une ressource liée à une fenêtre. Il est recommandé de se charger manuellement de la liaison par *SetFrameColor*.]

Tous ces éditeurs sont liés entre eux. Ainsi si j'ajoute un bouton icône à une fenêtre, je peux voir à l'écran à la fois les éditeurs de fenêtre, de contrôle et d'icône.

Les valeurs par défaut des ressources créées sont elles-mêmes des ressources de Genesys. Cela permet donc à un programmeur français d'éditer ces ressources (obligatoirement sur une copie de Genesys) pour les franciser! Ainsi la barre de menus créée par défaut pourra être: Apple (pardon! Pomme), Fichier et Edition...

La génération des sources

L'édition des ressources est déjà un plaisir mais la génération des sources est une véritable splendeur!

Avec Genesys, vous pouvez sauvegarder vos ressources non seulement sous forme de ressources (si, c'est possible!) mais aussi sous forme de **programmes-sources** dans les langages suivants: *APW/Orca ASM65816, APWC, APW Rez, Lisa 816, Merlin 16, Orca/C, Orca/Pascal, TML Pascal II* et même *MPW IIgs Pascal*! D'autres langages sont sélectionnables mais les sources produits ne font qu'indiquer la non implémentation actuelle de ces langages... il s'agit de Micol Basic, Micol Macro, MPW IIgs Assembler et MPW IIgs C.

Les fichiers en charge de la traduction des ressources dans ces langages sont situés dans ▶

le catalogue */Gen.Lang*, et sont écrits dans un langage interprété propre à Genesys: *SCGL* ou *Source Code Generation Language*. Ce langage, dont la syntaxe au premier abord paraît quelque peu touffue, est décrit en détail dans l'Appendice A de la documentation. Cela permet au programmeur d'écrire lui-même ses traducteurs et de modifier ceux déjà existants!

Si le langage choisi est un assembleur, les ressources seront traduites sous la forme de data utilisables par la suite avec la toolbox, non plus sous forme de ressources mais de pointeurs. Cela peut être le choix du programmeur voulant d'une part limiter au maximum les accès disques, et ne voulant d'autre part pas que ses fenêtres, menus et autres données soient modifiables.

Si le langage est de type Pascal ou C, alors le source généré est un véritable programme tenant compte des ressources présentes! Ainsi en choisissant *New Application* et en générant par exemple (et au hasard...) le source Orca/C, on obtient sans taper une seule ligne du source **un programme immédiatement compilable** et qui est équivalent au squelette en C présenté dans Toolbox Mag n°3 !!!
(Vous trouverez sur la disquette, en catalogue */Genesys*, les sources ainsi générés en divers langages, avec un DefaultFile francisé).

C'est à ma connaissance une première dans l'histoire de l'Apple II, depuis les générateurs de programmes Applesoft du bon temps: **un programme générateur de programmes!**

La documentation

Les 130 pages que constituent cette documentation traitent quasiment exclusivement de l'utilisation du shell et des différents éditeurs. Le langage SCGL est décrit dans l'Appendice A. Bref ce n'est pas avec cette documentation que l'utilisateur de Genesys apprendra à se servir des ressources... Normal car cette documentation existe déjà, c'est le volume 3 du manuel de référence de la Toolbox !

Reste que cette documentation, parfois simpliste, est tout de même indispensable pour découvrir tous les petits trucs d'utilisation: par exemple la touche Option et les flèches permettent le déplacement pixel par pixel des fenêtres et des contrôles...

Je tiens également à mentionner **une des aides les mieux réalisées qu'il m'ait été donné de voir**. En effet cette Aide (Help du menu ♁) réagit suivant le contexte: l'utilisateur est en train d'éditer un menu; il sélectionne Help et c'est le texte concernant les menus qui est immédiatement affiché! Un pop-up menu permet bien sûr d'avoir accès aux nombreux autres sujets traités par l'Aide.

En conclusion, **Genesys est un must pour tous les programmeurs de la Boîte à Outils de l'Apple II GS**. Sa structure bien conçue dès le départ (shell + modules) permet à Genesys toutes les évolutions et tous les enrichissements. La version 2.0 promet des merveilles...
Si le but est simplement de modifier quelques icônes et quelques chaînes de caractères, l'investissement me paraît excessif, car, malgré les ressources, les traductions exigent souvent une intervention supplémentaire dans le code. Mais, pour celui qui veut tirer toutes les possibilités de son Apple II GS, **Genesys est aujourd'hui totalement indissociable de l'environnement de programmation** (Apw/Orca - Merlin - TML Pascal 2).

SSSI (Simple Software Systems International, Inc):
4612 North Landing Drive NE
Marietta, GA 30066, USA

■

Entre Genesys et Rez

Pour un utilisateur de Genesys, sauvegarder ses ressources en format Rez est une bonne habitude à prendre, en tout état de cause (d'autant que Genesys permet cette sauvegarde directe). Rez est le langage fondamental des ressources, et Genesys ne peut pas tout.

Mais, s'il ne s'agit que d'avoir un Resource Fork bien rangé, le petit utilitaire en shareware **LLRE** (Low Level Resource Editor, déjà évoqué dans ToolBox Mag) y pourvoit impeccablement. Il suffit de créer un fichier avec LLRE, y "copier une par une" l'ensemble des ressources de son application, puis y copier ensuite le Data Fork. Renommer enfin le fichier et lui mettre le type et le type auxiliaire de l'original.
Je recommande également cette pratique après toute modification directe des ressources d'une application existante avec Genesys, sans quoi elle enfle.
D'autre part, il existe désormais un compilateur/décompilateur de ressources qui peut, dans la plupart des cas, se substituer avantageusement à Rez/Derez: c'est le **ResDoctor** de Jean Destelle...

JYB

La Laser sans le Mac

J.Y. BOURDIN

Oui, il y a un ordinateur Apple qui est au moins aussi bon, peut-être même meilleur, dans son genre, que le G.S.
Non, en aucun cas ce nouvel ordinateur Apple ne remplace le G.S. Au contraire, il s'y connecte directement, et étend de façon vertigineuse ses possibilités.
Voilà deux des leçons que je tire de mon expérience récente de la Personal LaserWriter NT Apple, et que j'aimerais vous faire partager. Il faudra bien aussi, hélas, évoquer une troisième leçon, une simple confirmation: c'est bien un produit Apple ! Quand je me suis décidé, je me doutais de ce qui m'attendait, car c'est ce que vérifient chaque jour tous les utilisateurs de G.S: la contradiction entre des produits (des machines) de première classe, enthousiasmants même, et un service en-dessous de tout. Eh bien, je n'ai pas été déçu, ni dans un sens, ni dans l'autre !
N.B: quand vous trouverez sans autre précision les mots "Laser" ou "LaserWriter" dans cet article, traduisez par "Personal LaserWriter NT Apple": le nom est un peu long.

Une machine enthousiasmante

Pour comprendre exactement ce qu'est la LaserWriter, et pourquoi elle est enthousiasmante, il faut tout de suite se retirer de la tête une idée fausse: **ce n'est pas une imprimante**. Le LaserWriter Reference le dit, au détour d'une phrase: **c'est un ordinateur**, un ordinateur complet et très puissant.
Il a tout d'un ordinateur: le microprocesseur (celui du Mac), la Ram (extensible sur certains modèles, dont la Personal), la Rom, AppleTalk, un port série, un port SCSI sur certains modèles: le LaserWriter Reference montre même un port ADB (le port de votre clavier et de votre souris), réservé pour une "extension future" sur la NTX! *[Rien ne vous empêche d'ailleurs de vous servir de la LaserWriter comme d'un pur et simple ordinateur: elle est très douée pour certains types de calcul. Nous verrons peut-être cela une autre fois.]*
Simplement, le terminal de visualisation, ▶

l'écran de cet ordinateur, si vous préférez (la comparaison n'est pas de moi: la partie de l'ordinateur qui contrôle cet "écran" s'appelle bel et bien un "contrôleur vidéo") est un peu spécial: c'est une mécanique de photocopieuse; c'est la feuille de papier qui est le terminal.
Que le mot "Laser" ne vous égare pas: la technique pour déposer des grains de toner (poudre de photocopieuse) sur un tambour grâce au rayon laser n'a rien de spécifique Apple, c'est une (excellente) mécanique de photocopieuse (Canon sur la Personal) qu'on retrouve ailleurs.
La vraie force de la LaserWriter est dans l'ordinateur proprement dit, dans l'unité centrale, à savoir:
① Le langage que cet ordinateur a en Rom, **PostScript**, le langage de description de page: ce langage n'est pas d'Apple, mais d'Adobe.
② L'interpréteur de ce langage, qui se charge de traduire ses commandes en 0 (point blanc) ou en 1 (point noir) pour le Laser de la photocopieuse. On appelle cela de la "rasterisation" en anglais (il paraît qu'on dit "pixellisation" en français: ce n'est guère mieux). Cette "rasterisation" est le travail d'Apple.
③ L'ensemble de la gestion des entrées-sorties (AppleTalk, disque SCSI éventuel, port série, etc), de la gestion de la mémoire, etc, tous travaux classiques sur un ordinateur. Tout cela est aussi le travail d'Apple.
C'est donc bel et bien un ordinateur: un ordinateur dédié, mais un ordinateur complet. Il est essentiel de comprendre cela pour trois raisons:

① Il en découle une **indépendance complète entre le langage PostScript et l'imprimante:** vous pouvez passer le même fichier PostScript sur des imprimantes d'une résolution bien supérieure à celle de la LaserWriter, les imprimer sur film et non sur papier, etc. PostScript étant un langage de description de page de type vectoriel, est indépendant du nombre de points par pouce du terminal. Quel que soit le terminal, PostScript donnera toujours le meilleur résultat possible.
[Entre autres, cela signifie que vous pouvez porter votre fichier GS chez ce qu'on appelle un "flasheur" pour qu'il l'imprime sur film. Il suffit d'imprimer votre fichier GS sur disque sous forme de fichier PostScript. Restera au flasheur à le passer sur disquette Mac avec Apple File Exchange, et à imprimer le fichier PostScript en veillant à employer les mêmes fontes PostScript que vous. S'il ne sait pas faire ne serait-ce qu'une seule de ces trois opérations, changez de flasheur!]
② Une fréquentation un peu assidue de la LaserWriter vous apprend très vite d'où vient sa qualité d'impression: elle vient essentiellement de PostScript et de l'interpréteur Apple, et pas d'abord de la technologie Laser.
③ Le modèle de la communication entre le GS et l'ImageWriter est totalement inadapté pour comprendre ce qui va se passer ensuite: le GS (ou le Mac, ou l'IBM) ne pilote en aucun cas l'imprimante. Ce qui se passe entre le GS et la LaserWriter, c'est **de la communication entre deux ordinateurs**: classiquement, cette communication se fait soit par le port série avec un logiciel de communication, soit par le Réseau (port AppleTalk).
On est même fondé à parler de l'ordinateur comme d'un simple **périphérique de la LaserWriter**: dans le cas de la communication par le port série, l'ordinateur fournit un écran et un clavier, et c'est tout, à l'ordinateur principal, la LaserWriter. Dans le cas général de la communication par AppleTalk, l'ordinateur n'est rien d'autre qu'un **éditeur de fichiers PostScript**: il fournit à la LaserWriter des fichiers dans son langage, qu'elle va interpréter et exécuter. C'est bien pourquoi il est possible de parler de "la Laser sans le Mac": n'importe quelle machine qui peut envoyer un fichier Ascii peut "piloter" la LaserWriter. Apple en est le premier conscient, puisque la Personal est conçue pour pouvoir se connecter non seulement à un GS, mais aussi à une brouette de chantier ou à un Macintosh.

La qualité Laser

De ce qui précède découlent au moins trois conséquences:
① une machine sans PostScript ne correspond pas à la définition ci-dessus, et l'essentiel de la "qualité Laser" lui manquera, quoi qu'en disent les publicités qui se limitent à la question de la résolution. Car dans ce cas-là, c'est précisément l'ordinateur qui devra se charger de piloter vraiment l'imprimante, comme le GS le fait pour l'ImageWriter. Or, cet ordinateur a un langage graphique (QuickDraw) qui est conçu pour gérer son écran point par point. Ses résultats sur papier seront donc d'une qualité terriblement inférieure à celle de PostScript.
Une machine sans PostScript, comme la SC ou la LS, n'est pas vraiment une LaserWriter, même si elle écrit avec un laser et si la partie photocopieuse a la même mécanique: il suffit de chercher à y imprimer quelques cercles pour s'en apercevoir tout de suite! Autrement dit, je **déconseille très vivement toute imprimante Laser non PostScript**.
Que la SC et la LS d'Apple ne soient pas ▶

pour l'instant, faute de drivers, compatibles avec le GS, ne me fait donc pas pleurer. **Un ordinateur comme le GS mérite une bonne qualité d'impression**, laissons les qualités inférieures aux Macs bon marché.

② Inversement, s'il existait une ImageWriter II ou une StyleWriter avec PostScript et interpréteur Apple, la qualité de leurs impressions serait dans les mêmes eaux que celle de la LaserWriter. Si cela n'existe pas, ce n'est pas pour des raisons techniques, mais pour des raisons économiques (quitte à être cher à cause de PostScript, autant mettre une mécanique d'impression de qualité).

③ L'utilisateur doit **éviter absolument toutes les situations où on passerait par l'ordinateur et par QuickDraw au lieu de passer par l'imprimante et par PostScript:** par exemple, n'utilisez jamais les fontes de l'ImageWriter genre Bobigny ou Saint-Denis pour la Laser, mais utilisez les fontes correspondant à des fontes PostScript (Avant-Garde, etc); utilisez des dessins sous forme de fichiers PostScript plutôt que sous forme de dessins GS, etc.

Précisions

Quelques précisions encore sur la Personal NT proprement dite:

PostScript et TrueType.

La "technologie" vectorielle TrueType, d'Apple et de MicroSoft, condamne-t-elle PostScript?

Une seule réponse: **tracez des ronds, et imprimez-les**. Si vous êtes tenté par la StyleWriter, ses "360 points par pouce" et sa "technologie TrueType", faîtes des ronds, vous comprendrez pourquoi elle est si bon marché.

Apple n'a intégré jusqu'à présent TrueType dans aucune de ses imprimantes. Ses plus récentes imprimantes sont soit livrées avec PostScript en Rom (Personal NT), soit avec rien du tout en Rom (LS, StyleWriter, etc). PostScript est chez tous les flasheurs, TrueType n'est guère pour le moment qu'une Init sur certains Macs.

Pour faire un bon langage d'impression, il ne suffit pas de savoir programmer: il faut aussi être un excellent imprimeur. En matière d'imprimerie, c'est la machine PostScript qui est l'unité centrale, l'ordinateur n'est que le périphérique.

Je ne peux donc recommander que de **faire comme Apple a fait avec la Personal NT**: choisir PostScript.

JYB

• Les publicités parlent de 37 fontes au lieu de 35 sur la NT ou NTX non-Personal: ce sont deux fontes pour émuler celles de l'IBM, qui sont d'ailleurs inaccessibles en mode Apple. En fait, pour nous, ce sont les 35 fontes habituelles.

• Les mêmes publicités vous parlent d'une impression à 4 pages-minute au lieu de 8 pages-minutes pour les autres: c'est vrai, mais seulement de la partie photocopieuse! Or **l'essentiel du temps dit "d'impression" de chaque page est en fait le temps de traitement par l'ordinateur de votre fichier PostScript**. Et là, la Personal NT, ayant un meilleur microprocesseur, et une version plus récente de PostScript, est meilleure et **plus rapide** que la NT non-Personal. La NT ne rattrape la Personal NT que si l'on imprime plusieurs fois la même feuille.

• L'absence du port SCSI de la NTX n'est pas vraiment gênante: au lieu de lui mettre un disque dur, rajoutez-lui de la mémoire. Elle a 2 Mégas d'origine, mais elle accepte jusqu'à 8 Mégas de Ram, alors que la NT non-Personal est bloquée sur ses 2 Mégas.

• Autre point très fort de la Personal: étant une machine plus récente, elle contient aussi une version plus récente de PostScript. Et cela, nous le verrons peut-être une prochaine fois, n'apporte pas qu'un gain de vitesse.

• La partie mécanique/alimentation papier est excellente, supérieure à la NT: très facile pour les enveloppes, les transparents, les recto-verso, etc.

Faut-il encore vous convaincre de la qualité de la Personal NT? Eh, mais **ce que vous lisez en ce moment même, ce sont des pages qui en sortent**, imprimées sur papier parfaitement ordinaire, et fidèlement reproduites par notre vaillant imprimeur!

Sur l'ensemble, c'est bel et bien **une machine enthousiasmante, de la qualité Apple**.

Quel dommage que le service soit, lui aussi, de la qualité Apple habituelle...

Un service en-dessous de tout

Le service, ici, c'est ce qu'Apple livre pour le GS avec la Personal LaserWriter NT. D'abord, la documentation nous confirme clairement qu'il s'agit d'un matériel compatible Apple II GS, que c'est bien un périphérique du GS: une annexe lui est dédiée. Cette annexe est essentiellement consacrée à la configuration du GS, elle est claire et exacte. Je ne répéterai donc pas son contenu ici. Elle ne mentionne pas la possibilité de branchement sur le port série (avec un câble GS/ImageWriter I), mais passons: voyons les disquettes. ▶

Deux disquettes sont livrées, qui contiennent les fontes-écran nécessaires pour utiliser les 35 fontes en Rom de la LaserWriter, plus un programme pour cataloguer les fontes de l'imprimante, télécharger des fontes, etc. Bien, installons tout ça: le Finder ne reconnaît pas les disquettes: elles sont au format HFS du Macintosh. **Rien, strictement rien, n'est fourni comme logiciel pour le GS!**

Or, si vous n'avez pas les **fontes-écran** (QuickDraw) correspondant aux fontes en Rom (PostScript) de la Laser, **vous ne pourrez pas utiliser les fontes de la Laser avec le GS!** Retour à l'annexe GS de la documentation: une formule sybilline à souhait laisse entendre qu'on pourrait peut-être trouver (ou mettre?) ces polices sur la disquette système du GS; or les disques système d'Apple ne contiennent que les trois fontes de l'antique LaserWriter (Times, Helvetica, Courier). **Il manque les fontes-écran pour Avant-Garde, Bookman, Helvetica Narrow, New Century Schoolbook, Palatino, Symbol, Zapf Chancery et Zapf Dingbats**. Où sont-elles? Mais sur la disquette Mac, bien sûr! Seulement voilà: le Mac n'est pas fourni avec!

Qu'on ne me dise pas qu'il est impossible, ou simplement difficile, de passer ces fontes Mac en fontes GS: cela m'a pris 20 minutes, et sans Mac, vous allez le voir dans la troisième partie. Demandez à votre concessionnaire le prix de la Personal NT: eh bien, **Apple n'a pas daigné consacrer 5 F sur ce prix à une disquette de fontes-écran pour son ordinateur Apple II GS!** Et en plus, l'utilisateur moyen n'a aucun moyen, aucune indication pour s'en sortir. Heureusement qu'en France, nous avons un "Apple II Service Team": que serait-ce sans!

Aucun moyen non plus n'est fourni par Apple pour **télécharger une fonte PostScript à partir du GS**: là aussi, vous allez trouver comment faire dans la troisième partie, avec un exemple de fonte sur la disquette. Oui, c'est parfaitement possible. Seulement, là, pas question de passer sur GS le programme Mac des disquettes Apple: il n'y tourne pas.

L'Apple II GS étant, comme la Personal LaserWriter NT, un matériel vendu par Apple, je suis au regret de dire, comme utilisateur de matériel Apple, que **le service d'Apple est bel et bien en-dessous de tout**.

Bien entendu, une fois que vous avez vous-même appris, en y passant le nombre d'heures nécessaire, à importer sur GS les fontes QuickDraw du Mac, à télécharger les fontes PostScript, à contourner le bug des caractères Symbol (voir ToolBox-Mag 3), etc, eh bien, ma foi, ça marche, c'est tout. Apple a beau tout faire pour forcer l'achat d'un Mac: **il n'y a pas besoin d'un Mac pour faire marcher la Laser**. C'est ce que nous allons voir.

La Laser sans le Mac

Quel est l'usage principal d'un Mac avec une LaserWriter? C'est d'imprimer des textes mis en pages sur l'écran avec des fontes-écran correspondant à celles de la Laser, ou avec des fontes qu'on télécharge dans la Laser. Or, le GS est parfaitement capable de faire tout cela.

Rien, absolument rien, ne distingue une page sortie de la Laser via un Mac de la même page sortie de la Laser via un GS. C'est une page Laser, c'est tout. Bien entendu, les applications sont différentes, les drivers aussi: mais la qualité du résultat imprimé est la même. Vous imprimiez sur ImageWriter, vous imprimerez sur LaserWriter, avec la qualité Laser, de façon tout aussi simple et transparente.

En particulier, j'insiste sur ce point encore peu connu, **le driver de LaserWriter du 5.04 est spontanément capable de reconnaître les fontes Postscript téléchargées dans la Laser** quand elles correspondent au nom exact d'une fonte-écran. Apple sous-entend qu'il faudrait cependant un Mac pour télécharger ces fontes, et que le GS devrait donc être en réseau: c'est faux, nous allons le voir. **Des milliers de fontes PostScript sont dès aujourd'hui disponibles pour les utilisateurs d'Apple II GS.**

Pour le reste, le seul point à rappeler qui ne soit pas déjà dans les documentations Apple, c'est la nécessité d'utiliser des **logiciels vraiment GS, qui affichent avec QuickDraw et impriment avec les drivers Apple.** Ils sont légion, que ce soit AppleWorks-GS, Graphic Writer III, BeagleWrite/MultiScribe, Medley, etc.

Une chose à noter: en règle générale, dans ces applications GS, il faut choisir le mode dit "Macintosh" dans le menu Mise en Pages (Page Setup).

Si l'expression "Allô, Michel..." vous dit quelque chose, eh bien sachez que *la NT a été la machine qui a fait lâcher son joystick à Michel*. Si! Michel a d'ailleurs un conseil à vous donner en matière de logiciel pour la Laser: essayez *Designer Prints*, de *MECC*, vous ne le regretterez pas.

Appleworks 8 et la Laser

Pour imprimer sur Laser avec Appleworks-8, il faut passer par l'Emulateur ImageWriter. Vous trouverez sur la disquette, catalogue /LA- ▶

SER.SANS.MAC/, un fichier Appleworks-8 appelé "Laser.AWP". Imprimez-le sur Laser, et vous verrez le résultat:

• Les caractères non-proportionnels sont traduits en Courier, et le résultat est plutôt laid: pas meilleur qu'une machine à écrire! Proportionnel 1 donne du Times 10, un peu trop serré. Le meilleur mode d'impression est en Proportionnel 2, qui donne du Times 12.

• Hélas, dans ce mode, les caractères circonflexes donnent un résultat largement aussi atroce qu'en Proportionnel sur ImageWriter II (bug du Control-H).

• Le mode justification totale produit des résultats affreux, voire aberrants, en mode proportionnel/Times.

• Malheureusement, le mode Justification Gauche produit aussi n'importe quoi: en Proportionnel, la longueur totale de la ligne n'est pas la même qu'en justification totale, et, si on change de fonte, la longueur de ligne change aussi!

• Enfin, l'écran d'Appleworks ne sert plus à grand-chose dans les modes proportionnels: les lignes imprimées sont bien plus longues que les lignes affichées sur l'écran Appleworks, et Appleworks 8 interdit, dans le traitement de textes, le défilement horizontal de l'écran.

Je n'ai pas cherché ce qui venait, dans tous ces problèmes et bugs, de l'Emulateur ImageWriter, et ce qui venait d'Appleworks, mais le résultat est là: Appleworks 8 est sans doute encore utilisable avec la Laser dans ses parties Tableur et Base de Données, pour des listings en somme. Mais, pour le traitement de textes, **ses résultats papier sont aussi parfaitement inacceptables que ceux d'un Macintosh avec ImageWriter**. Bref, encore une fois, pour la rédaction, il est temps de jeter AppleWorks-8.

Plus exactement: rien ne nous interdit, bien sûr, d'utiliser Appleworks 8 pour gérer nos fichiers. Mais, si nous voulons une impression de qualité GS, que ce soit sur Laser ou sur ImageWriter, une seule solution: **passer le fichier dans Appleworks-GS pour l'imprimer**.

Conversion des fontes écran Mac

Puisque nous savons que nous n'aurons la qualité des fontes en Rom de la Laser qu'avec les fontes QuickDraw correspondantes dans notre système, il nous faut maintenant voir comment convertir les fontes écran de la disquette Mac sur une disquette GS. Même si vous n'avez pas de Laser, ces indications peuvent vous servir si vous voulez convertir des fontes-écran du Mac.

Excusez-moi, mais ce passage contiendra un peu de polémique: car non seulement Apple ne fait pas lui-même ce travail, mais il nous propose pour ce faire une mauvaise méthode. Le passage que je critique se situe dans le *"Guide de l'Apple II" édité par Apple France, 2e semestre 1989, pages 53 à 57.*

Rappelons d'abord que, si nous avons cette conversion à faire, c'est parce que ces fontes sont dans la Rom de la Laser qu'Apple nous a vendue, et que les fontes-écran correspondantes sont **livrées sur une disquette incompatible** avec l'ordinateur qu'Apple nous a également vendu. On appréciera donc la colossale niaiserie de ce passage où Apple introduit les motivations supposées de la conversion: *"Vous admirez certaines polices de caractères du Macintosh. Il vous est possible d'en disposer sur votre Apple II GS, voici comment"*. Quand on est **contraint** à faire cette conversion par la **déficience d'Apple lui-même**, la cuistrerie Mac, habituelle chez Apple, devient franchement insupportable… ▶

L'utilité du monopole

Vous le savez sans doute, dans un pays comme la France où règne la libre concurrence, où le "refus de vente" est un délit, le système des concessionnaires exclusifs Apple, nom pudique du *monopole*, ne peut que difficilement se concilier avec la loi. Apple s'est donné du mal pour faire reconnaître comme simplement *licite* (c'est-à-dire pas en contravention) cette pratique. Et cela n'a été admis qu'au nom de l'argument d'un *"meilleur service"*, d'un *"service supplémentaire"* offert par le réseau des concessionnaires.

La déficience du service Apple sur la Personal NT est l'occasion rêvée pour les concessionnaires Apple de montrer leur capacité à assurer ce "service supplémentaire", et de justifier ainsi le surcoût du monopole pour l'utilisateur: si vous en connaissez un qui rajoute au package Apple les fontes-écran pour le GS, et une version GS de l'application Mac d'utilitaires Laser, faîtes-nous savoir son nom, nous le publierons dans ToolBox Mag!

Sinon, qu'on ne s'étonne pas de voir les utilisateurs de GS en première ligne pour vouloir acheter leur matériel Apple au supermarché du coin, et se précipiter dès qu'ils le peuvent sur du matériel non-Apple pour compléter leur configuration: ils font chaque jour l'expérience que le monopole des concessionnaires Apple ne signifie rien d'autre, pour l'utilisateur d'Apple II GS, que la garantie du prix le plus haut pour le service le plus bas!

JYB

Typique aussi de la cuistrerie Mac, l'incapacité d'Apple à concevoir qu'on puisse faire certaines choses sans Mac, et mieux qu'avec un Mac: après avoir essayé de forcer l'usage du Mac pour avoir les fontes écran, Apple essaie de le forcer encore, par le Guide de l'Apple II, pour... faire la conversion pour le GS! Il préconise en effet l'utilisation d'*Apple File Exchange* du Mac et de *Font Munger* (application Apple pour Mac) pour faire la conversion.

Or non seulement le Mac est inutile, le GS suffit, mais **les applications Mac en question sont buguées**, et je déconseille vigoureusement leur utilisation.

• Pour *Apple File Exchange*: au moment où j'écris ces lignes, Apple a réalisé trois nouvelles versions du système Mac depuis que ToolBox Mag a signalé le bug d'AFE avec le Syquest. Or, ce bug est toujours là, et si vous avez à la fois un GS et un Mac, il ne serait pas étonnant que vous ayez un Syquest.
Essayez de faire passer sur Mac un fichier GS avec ressources, et vous verrez qu'AFE ne sait pas le faire. Faîtes-lui passer maintenant un "sparse file" de Prodos (comme le fichier "Joli.Ecran" de la disquette ToolBox Mag 4), et vous vous rendrez compte qu'AFE vous trompe (il n'émet pas de message d'erreur, mais le fichier Mac est faux, les blocs à 00 sont oubliés). Bref, **AFE n'est pas fiable**.

• Pour *Font Munger*: ce n'est pas seulement sur le SE que cette application plante, comme le signale le Guide, mais pratiquement sur tous les Macs vendus en ce moment par Apple. Non seulement il faudrait un Mac, mais il faudrait aller le chercher aux puces! De plus, Font Munger, quand il ne plante pas, refuse de convertir les fontes de plus de 32k (il a les limites du Mac 128/512, justement). Or les fontes du GS vont jusqu'à 64k.
En somme: **même si vous avez un Mac, faîtes la conversion sur le GS, c'est plus sûr!** Voici comment faire sans le Mac et ses bugs.

Il vous faut les deux applications suivantes: *A2FX, de Chan Wilson*, en freeware, pour rapatrier sur le GS les fichiers Macs. Et *Resource Spy, de Stephen Chick*, en shareware à 15$, pour convertir les fontes-ressources du Mac en fontes GS. Après avoir lu les documentations de ces deux logiciels, il suffit de faire ce qui suit:

① Passez sous A2FX. Introduisez la disquette Mac de la Personal NT sur laquelle est écrit "Font/DA Mover". Sélectionnez le fichier appelé "Personal LaserWriter NT Fonts", de type DMOV. Convertissez ce fichier sur un disque GS en choisissant la formule "Resource Fork as Data File".
② Quittez A2FX: vous devez avoir un fichier appelé quelque chose comme "X.PERSONAL.LASE", de type $E0. Changez ce type en $06 (BIN). *[Si vous ne savez pas comment faire pour changer un type de fichier, voyez ce même Guide de l'Apple II, pages 55-56: la seule chose non contestable de ce passage du Guide est un programme Applesoft changeur de type de fichier, signé d'Emile Schwarz, qui ne demande que quelques modifications pour changer les $E0 en $06. Bien entendu, une multitude d'applications et accessoires GS le permettent aussi.]*
③ Passez sous Resource Spy, et ouvrez ce fichier. Il vous présente trois types de ressources: FOND, FONT, et "vers" (ce dernier ne nous intéresse pas). Ouvrez d'abord la ressource FOND: une suite d'ID apparaît, suivis d'un numéro, suivis d'un nom de fonte (Avant-Garde, etc). Notez soigneusement le nom de chaque famille de fonte et le numéro de son ID: ce sera ensuite le numéro de famille de la fonte.
Ouvrez ensuite les ressources FONT. Une suite de familles de fontes apparaît. Dans chaque famille, la première fonte est vide, c'est le nom de la famille. Notez ce nom. Notez aussi que cette première fonte de chaque famille a un numéro qui sert de point de départ, de zéro pour les autres. Toutes les autres fontes porteront un numéro égal à ce nombre +10, +12, +14, +18, ou +24: cela indique la taille de chaque fonte. Ouvrez ensuite la deuxième fonte de cette famille: ce sera votre première fonte, elle sera de taille 10. Choisissez dans le menu de Resource Spy l'item de conversion en fonte GS: la fonte apparaît sur l'écran du GS.
Le programme vous demande alors le nom de la fonte (c'est-à-dire le nom de la famille): vous l'aviez noté, donnez-le lui exactement, en respectant majuscules et minuscules. Il vous demande ensuite le numéro de la fonte: c'est le numéro de la famille, qui correspond à l'ID que vous aviez noté au début. Il vous demande enfin la taille: c'est 10, vous l'aviez noté. Sauvez le fichier: vous avez converti une des fontes de taille 10. Passez à la suivante, elle sera de taille 12, et répétez la procédure: ainsi de suite, jusqu'à la fin. Copiez enfin, bien sûr, toutes ces fontes-écran GS dans le catalogue /SYSTEM/FONTS de votre disque système.
Voilà, la conversion est faite, vous pouvez utiliser la Laser sans le Mac. Les pessimistes diront que ce travail ne vaut guère que les 5 F de la disquette qu'Apple n'a pas mise. Les opti- ▶

mistes diront qu'ils ont gagné le prix d'un Mac.

Comment télécharger une fonte PostScript dans la Laser ?

Si le driver de LaserWriter de GS/OS est capable de reconnaître, c'est bien le moins, une fonte PostScript déjà téléchargée dans l'imprimante, il n'est en revanche pas capable de procéder lui-même à ce téléchargement. Est-ce d'ailleurs à lui de le faire, ou aux applications? En tout cas, pour le moment, c'est à nous de le faire.
L'application Apple livrée avec la Laser pour ce faire étant incompatible GS, il faut donc nous débrouiller nous-mêmes. Le principe est simple, et ToolBox Mag vous en a déjà parlé: le CDEV LaserWriter est capable de transmettre tout fichier PostScript. Or une fonte PostScript n'est rien d'autre qu'un fichier PostScript.
Plutôt qu'une formule générale, j'ai choisi de montrer, pas-à-pas, comment faire sur un exemple. Bien entendu, vous devez tourner sous 5.04 minimum, avoir installé les fichiers système nécessaires, allumé votre Laser, etc.

① Copiez la fonte ToolBox.24 depuis le catalogue /LASER.SANS.MAC/ de la disquette de ce numéro dans le catalogue /SYSTEM/FONTS/ de votre disque système.
② Renommez, dans le catalogue /SYSTEM/DRIVERS/ de votre disque système, le fichier IWEm en IWEm.Apple.
③ Copiez le fichier IWEm.ToolBox depuis le catalogue /LASER.SANS.MAC/ de la disquette de ce numéro dans le catalogue /SYSTEM/DRIVERS/ de votre disque système.
④ Renommez /SYSTEM/DRIVERS/IWEm.ToolBox en /SYSTEM/DRIVERS/IWEm.
⑤ Passez dans un traitement de textes GS quelconque, AppleWorks GS ou autre. Tapez quatre ou cinq lettres en majuscules (obligatoirement), et passez ces lettres dans la fonte ToolBox 24. Vous devez voir plusieurs fois un rectangle entourant le mot "OK": c'est ce que donne la fonte-écran.
⑥ Passez dans le Tableau de bord du menu , choisissez LaserWriter: cliquez sur la cartouche affichant "Emulateur ImageWriter". Le CDEV se charge de télécharger notre faux Emulateur ImageWriter, qui est en fait la fonte PostScript ToolBox.
⑦ Quittez le Tableau de bord, revenez dans le traitement de textes. Passez dans l'item de menu Mise en Pages (Page Setup), choisissez le mode "Macintosh". Enfin, imprimez. Regardez la feuille imprimée par la Laser: ce n'est pas "OK" qui est écrit, mais autre chose. C'est bien la preuve que vous avez utilisé une fonte téléchargée dans la Laser, et téléchargée par le GS. **C'était donc possible!**

Attention, ce que vous avez fait ici est ce qu'on appelle un *"téléchargement permanent"* dans l'imprimante, c'est-à-dire que cette fonte restera dans la mémoire de l'imprimante jusqu'au prochain Reset de la Laser. Pour un téléchargement "provisoire" (qui s'effacerait après usage), il faudrait un autre en-tête PostScript. Par ailleurs, si ça n'a pas marché, c'est sans doute qu'il y a un mot de passe dans votre Laser: remplacez le "Ø" de la première ligne de IWEm.ToolBox par ce mot de passe.

Au boulot, Apple!

Si Apple nous laisse nous débrouiller tout seuls pour télécharger des fontes PostScript dans la Laser, il pourrait au moins permettre aux applications GS de reconnaître ces fontes: le type PostScript ($B0, AuxType $0719) ne permet pas de distinguer entre fontes et autres fichiers, ni entre type 1 et type 3.
Allez, au boulot, Apple: et ne tarde pas trop...

Comment convertir les fontes PostScript pour le GS ?

Il n'y a absolument **aucune conversion à faire**: une fonte PostScript, qu'elle soit "de type 1" ou "de type 3", n'est rien d'autre qu'un **fichier Ascii absolument ordinaire**, et ces fontes sont dans le commerce (prenez une version d'origine en Ascii, ou une version pour GS, pas une version tripatouillée pour le Mac). Si vous voulez télécharger cette fonte de la façon décrite ci-dessus, il faut simplement, à l'aide de n'importe quel éditeur de texte, lui copier au tout début la première ligne du fichier IWEM.TOOLBOX.

Et le Wysiwyg, et Page Maker ?

- Quand même, tu n'exagères pas un peu? Il y a sur le Mac des applications bien plus puissantes que sur le GS, pour la PAO par exemple. Et il y a le Wysiwyg (le tel-écran/tel-écrit), les grands écrans...
- C'est vrai, le GS, comme toutes les machines dans sa gamme de prix, n'est tout simplement **pas assez cher** pour constituer une bonne station de travail PAO professionnelle. Pour les écrans A3 ou A4, les logiciels genre FrameMaker, il faut se tourner vers le Next, les IBM chers et les Macs chers.
Le Macintosh n'a aucun privilège parmi les machines incompatibles GS. La Personal NT est compatible IBM (compatible n'importe quoi, en fait: tout ce qui peut lui envoyer un fi- ▶

chier Ascii est compatible).
Si vous avez besoin d'aller au-delà des possibilités du GS en matière d'impression Laser, si vous avez un travail professionnel complexe de mise en pages à faire par exemple, suivez ce qui est la règle dans ce cas: ne partez jamais de la machine. Cherchez le **logiciel** qui fait exactement ce que vous voulez, et **achetez la machine sur laquelle ce logiciel tourne le mieux**, sans a priori. Dans le domaine professionnel, une seule chose compte: "to get the job done", que le travail soit fait. La machine sur laquelle on le fait est sans importance. Même le prix est chose négligeable: on le facture aux clients.
Mais s'il ne s'agit que de faire du traitement de textes ou de la mise en pages ordinaire, si le budget rentre en compte, bref s'il s'agit d'un **usage personnel**, vous n'avez **aucun besoin d'une machine incompatible GS**, quelle qu'elle soit, en plus de la Personal NT et de votre GS. Les logiciels GS sont parfaitement suffisants.

Quant au Wysiwyg, alors là, pas d'illusions: QuickDraw et PostScript, 72 points par pouce et 300 points par pouce, un écran (Mac, GS ou IBM) et une feuille de papier, ce n'est pas du tout la même chose. De toute façon, **quelle que soit votre machine, il vous faudra imprimer plusieurs exemplaires de votre feuille de papier**, corriger et recorriger plusieurs fois **à partir du papier** avant d'obtenir le résultat désiré.
Mac et GS partagent sur ce point la même limite, qui s'appelle QuickDraw, et que les pixels écran soient carrés ou rectangulaires est sans importance. Il n'y a en fait que deux solutions pour avoir du vrai Wysiwyg: soit on aligne l'imprimante sur l'ordinateur, et on la fait imprimer avec QuickDraw. C'est la "solution SC". Dans ce cas, la qualité de l'impression papier est tout simplement inacceptable en fonction du prix de l'ensemble!
Ou bien, on aligne l'ordinateur sur l'imprimante, et on lui fait faire ses affichages écran avec PostScript. Dans ce cas, ce n'est plus un Apple (ni GS ni Mac), ce n'est pas non plus un IBM: c'est un Next. Ceci dit, les moniteurs vidéo n'étant pas à 300 points par pouce, le "Wysiwyg" n'y sera jamais parfait.

Alors, on se l'offre ?

Entre une machine enthousiasmante et un service consternant, comment trancher? Commercialement, Apple devra bien un jour réfléchir à cette question, et comprendre pourquoi une majorité d'utilisateurs d'ordinateurs personnels préfère galérer sur des machines affligeantes.

Mais vous, vous n'êtes pas comme ça: vous avez et vous gardez un Apple II GS, vous connaissez ainsi la valeur du service Apple. Vous êtes, par la force des choses, nécessairement habitués, rôdés même, à vous débrouiller avec une machine enthousiasmante sans rien demander d'autre au constructeur que de bien vouloir vous la vendre et vous la réparer éventuellement. Vous ne serez ainsi nullement dépaysés!
Je peux donc tranquillement l'écrire: si vous voulez la **vraie** qualité Laser, et n'envisagez pas de tirer plus de 150 000 feuilles avec votre Laser (c'est une quantité considérable), **la Personal LaserWriter NT d'Apple est une magnifique machine, dont je ne peux que recommander l'achat.**

C'est cher, dîtes-vous? Ma foi, Apple a fait un gros effort sur le prix avec la Personal, ce sont bien les utilisateurs personnels qui sont visés, et n'oubliez pas que le prix tarif (qui vient encore de baisser) n'est rien d'autre chez Apple que le point de départ de la négociation: faîtes jouer la concurrence.
Malgré tout ça, ça reste cher pour un individu, c'est vrai. Eh bien, secouez un peu le patron: puisque la Personal NT est compatible IBM, voilà un moyen de faire entrer enfin du matériel de qualité dans votre entreprise! Il y a les clubs, les amis: une Laser, ça se partage!
Comme souvent avec Apple, c'est cher, mais c'est bien. **C'est cher parce que c'est bien**. Ce qui **coûte** moins **vaut** vraiment moins: méfiez-vous de tous ces matériels bas de gamme et à bas prix, "Low Cost", qu'Apple veut nous vendre à toute force en ce moment.
Entre nous, là, franchement, ne mentez pas: **je suis sûr que vous en avez envie.** Vous pouvez vous la payer en douceur, à crédit ou en leasing. "Combien êtes-vous prêt à mettre pour vous faire plaisir?", voilà la vraie question. Car cette machine-là vous remboursera en plaisir les heures de travail supplémentaire que vous ferez pour vous l'offrir. **Et si ce n'est pas pour pouvoir vous offrir ce qui vous fait plaisir, pourquoi donc travaillez-vous?**

Vous qui avez envie de goûter de nouvelles expériences informatiques, d'aller au-delà des possibilités de votre GS actuel, oui, c'est possible, et toujours avec le GS: grâce à la Personal NT.

Adresses:

Stephen Chick (Resource Spy): 2-1417 Port Village, Port Island, Kobe, Japon 650.

Chan Wilson (A2FX): 3372 Middlefield Road, Palo Alto, CA 94306, USA. ■

Pascal et Assembleur : Le passage de paramètres

Bernard Fournier

Le Pascal est certes un langage évolué et dont l'approche est aisée, mais si on ne considère que le point de vue de la rapidité de l'exécution, l'Assembleur est imbattable. Il peut arriver que certaines routines (recherche/remplacement dans un traitement de texte, animations graphiques...) requièrent une vitesse d'exécution optimale: dans ces cas, le Pascal est inadapté.
Ne partez pas ! Inutile de vous affoler en imaginant être obligé de jeter votre compilateur Pascal à la poubelle! **Il est tout à fait possible de compiler une application écrite en Pascal qui fait appel à certaines routines écrites en assembleur**. Le problème à résoudre est alors le **passage des paramètres** entre le corps du programme et la routine (et réciproquement).

Avec TML Pascal

Avant toute chose, lisons attentivement la documentation: au chapitre 17, il est brièvement indiqué que l'on peut employer des modules externes compilés en un autre langage que TML, que l'on peut employer la directive *Inline* pour définir de courtes routines assembleur et enfin à l'appendice B sont décrites en détail les conventions d'appels pour les procédures et fonctions.
Avant d'aller plus loin, je lance un appel... **toute personne sachant comment linker un module assembleur avec un programme TML est priée de se faire connaître.** En effet, bien qu'au chapitre 17 les auteurs indiquent (en 6 lignes) qu'une procédure peut être déclarée *EXTERNAL* et donc qu'on peut linker le code Pascal avec un code non-Pascal, ils n'indiquent pas la démarche à suivre... et malgré mes recherches, je n'ai pas réussi à effectuer le lien entre Assembleur et TML II.

Ceci étant, il reste la méthode des *Inline*. Lorsqu'on veut définir un module assembleur dans un programme Pascal, on peut employer des procédures déclarées *INLINE*. Cette directive accepte un argument: la valeur hexadécimale du code assembleur.

Exemple: on veut définir une procédure *Break* permettant d'interrompre l'exécution d'un programme et de passer sous moniteur (afin de débugguer une routine par exemple).
En assembleur l'instruction à employer est *BRK* dont le code hexadécimal est $00.
Voici comment définir la procédure *Break* en Pascal:

```
procedure p00; inline $00;
procedure Break;
begin
   p00;p00;
end;
```

et l'appel se fera désormais simplement par:

```
Break;
```

Note: dans la procédure *Break* on fait appel deux fois à *BRK* car l'instruction *BRK* a besoin d'un paramètre, ici on lui passe la valeur $00.

Notre première tentative d'inclure un code assembleur dans un programme Pascal ayant été couronnée de succès, passons à un exercice plus difficile: appelons une routine de la carte accélératrice TransWarp (cette routine ne fonctionnera que si la TransWarp est en ligne bien sûr).
En assembleur, le code à employer est *JSL BC/FF4C* pour l'appel à *DisableDataCache*; ce qui en hexadécimal donne: *22 4C FF BC*. En Pascal on va obtenir:

```
procedure p22; inline $22;
procedure pBC; inline $BC;
procedure pFF; inline $FF;
procedure p4C; inline $4C;

procedure DisableDataCache;
begin
   p22;p4C;pFF;pBC;
end;
```

Là, on se rend compte que l'emploi des *Inline* est assez lourd, et que ce sera réservé aux routines Assembleur très courtes. D'où mon appel pressant: si quelqu'un sait comment mixer des sources Assembleur avec TML II, qu'il nous le fasse savoir, cela profitera à toute la communauté du GS.

Maintenant, un exercice encore plus difficile: **le passage d'un paramètre à la routine as-** ▶

sembleur et la récupération en sortie d'un résultat. Ceci fait typiquement appel à une fonction. Par exemple, on va faire appel à la routine Transwarp *SetCurSpeed* qui demande la vitesse souhaitée en entrée et retourne la vitesse réelle en sortie.
Avant d'aller plus loin, consultons la documentation dans l'appendice D. Il nous y est expliqué que TML passe sur la pile les paramètres déclarés dans une fonction ou une procédure, et ce dans l'ordre de leur déclaration. C'est à dire que pour une procédure du type *MaProcedure(param1,param2,...,paramN)*, TML va générer un code du type:

```
lda param1
pha
...
lda paramN
pha
JSL MaProcedure
```

Si l'on a déclaré une fonction, TML va prévoir l'emplacement de la valeur de retour sur la pile, et ce sera au programmeur de dépiler la valeur de retour:
Exemple avec *MaFonction(param1,...,paramN): paramRetour*, on aura le code suivant:

```
pha
lda param1
pha
...
lda param2
pha
JSL MaFonction
pla
```

A noter que les valeurs passées par leurs adresses seront empilées dans l'ordre suivant:

```
pea Adresse_Haute
pea Adresse_Basse
```

Maintenant, analysons comment TML code l'accès et la sortie de la routine:

```
phd   ; sauver le précédent pointeur de pile
tsc
sec
sbc  #xx
tcd   ; créer un nouveau pointeur de pile
clc
adc  #yy
tcs   ; allouer espace pour stockage local
.....  ; routine personnelle
tdc
clc
adc  #xx
tcs   ; désallouer le stockage local
pld   ; récupérer le pointeur de pile
lda 2,s
sta mm,s
lda 1,s      ; adresse de retour mise
sta mm-1,s   ; sous les paramètres
tsc
clc
adc #mm-2
tcs
rtl
```

Bien évidemment, ce code est modifié selon la présence ou nom de paramètres de sortie (fonction ou procédure) et selon la longueur de stockage des paramètres d'entrée et de sortie.
Avant de passer aux exemples, une dernière indication: nos routines assembleur devront pouvoir récupérer et modifier les paramètres donnés en entrée. Pour ce faire, il va falloir utiliser certaines conventions:
• pour un paramètre de 4 octets au plus à modifier, le résultat obtenu par une fonction est suffisant (on peut également avoir deux entiers: valeurs basses et hautes du *LongInt*).
• pour modifier des tableaux de valeurs (tris par exemple), on passera l'adresse d'implantation des variables et on ira modifier ensuite ces mêmes variables par notre routine.
• certaines procédures (celles de la TransWarp par exemple) utilisent A et X comme valeurs en entrée et donnent des résultats dans A et X: pour ce faire, ce sera à nous de positionner les deux registres **avant** l'appel et de sauver ces mêmes registres **après** l'appel.
Note: une valeur *LongInt* est empilée par le compilateur valeur haute d'abord, valeur basse ensuite.

lecture d'une variable: *LDA adr1,S*
transfert de registre: *TAX* ou *TXA*
sauver un registre: *STA adr2,S*

Les valeurs de *adr1* et *adr2* seront déterminées comme suit:
pile lors de l'appel pour la fonction *MyFunction(var1: integer; var2: LongInt; var3: integer): LongInt;*

```
----------------------   $26
-                    -
-    résultat        -   $24
-                    -
==========           ==  $22
-      var5          -
----------------------   $20
-                    -
-      var4          -   $1E
-                    -
----------------------   $1C
-      var3          -
----------------------   $1A
```

selon nos besoins, on pourra récupérer dans A et/ou X n'importe quelle valeur de la pile et stocker le résultat dans la même pile:

- charger *var3* dans A:

```
LDA $1A,s
```

- charger *var2* dans A et X:

```
LDA $1E,s  ; valeur haute
TAX
LDA $1C,s  ; valeur basse
```

- charger *var1* dans X:

```
LDA $20,s
TAX
```

- stocker le résultat dans la pile:

```
STA $22,s   ; valeur basse
TXA
STA $24,S   ; valeur haute
```

Je pense que ce croquis avec l'exemple sera 'parlant' et permettra à tout un chacun de lire/sauver n'importe quelle variable sur la pile lors des appels à une routine assembleur.

Exemple n°1: un entier en entrée, un entier en sortie

```
procedure p22; inline $22;
procedure p24; inline $24;
procedure pFF; inline $FF;
procedure pBC; inline $BC;
procedure pA3; inline $A3;
procedure p83; inline $83;
procedure p1A; inline $1A;
procedure p1C; inline $1C;
function SetCurSpeed(var1: integer): integer;
begin
    pA3;p1A;    { LDA $1A,s : passer VAR1 dans
                  l'accumulateur}
    p22;p24;pFF;pBC;{ appel à ma routine (ici
                  SetCurSpeed de la TWGS) }
    p83;p1C;    { STA $1C,s : sauver le résultat
                  de la fonction }
end;
```

dans le corps du programme on aura:

```
Entree:= 2000;     { vitesse lente TWGS souhaitée }
Sortie:=PandASM1(Entree);
```

Exemple 2: deux entiers en entrée, deux entiers en sortie

```
{ définir les inline dont on a besoin }
function Pand ASM(var1,var2: integer): LongInt;
begin
    pA3;p1C;    { LDA $1C,s }
    pAA;        { TAX: mettre var1 dans X }
    pA3;p1A;    { LDA $1A,s : mettre var2 dans
                  A}
    p22;.....   { JSL ma fonction }
    p83;p1E;    { STA $1E,s  : sauver A dans la
                  pile: valeur basse du résultat}
    p8A;        { TXA }
    p83;p20;    { STA $20,s  : sauver A dans la
                  pile: valeur haute du résultat }
end;
```

dans le corps du programme on aura:

```
Resultat:=PandASM(valeur1,valeur2);
Res1:=LoWord(Resultat);
Res2:=HiWord(Resultat);
```

Note: on peut sur le même modèle avoir la fonction *MaFonction(var1: LongInt): LongInt*; cette fonction acceptant un entier long en entrée fonctionne de la même manière que la précédente avec ses deux entiers (on a 4 octets empilés dans les deux cas). Dans le cas du *LongInt* en entrée, la valeur basse sera dans A et la valeur haute dans X (conventions ISO de passage des paramètres).

Sur la disquette de ce numéro, vous trouverez **un listing TML permettant d'écrire n'importe quelle routine assembleur** (les procédures Inline sont définies avec la convention suivante: p suivi de la représentation hexadécimale du code. Elles se trouvent dans l'Unit *PandASMU.p*).

A titre d'exemple est fourni l'interfaçage avec la Transwarp GS: cet exemple a été écrit en collaboration avec mon ami Francis Wolkowitch pour qui j'ai écrit cet article afin qu'il ne fasse plus jamais des *BlockMove* de deux octets... Etudiez attentivement le source, de nombreux exemples illustrent l'emploi de fonctions et procédures avec/sans paramètres d'entrée et/ou de retour pour des appels à des routines asssembleur. Et plus particulièrement, vous remarquerez l'emploi d'une **routine Assembleur pour scruter le clavier:**

```
function KeyPress(a:integer): integer;
begin
    pA9;p00;p00; { LDA #$0000 }
    pE2;p20;    { SHORT M }
    pAF;p00;pC0;p00;{ LDA >Kbd }
    pAA;        { TAX on sauve la valeur}
    p29;p80;    { AND #$80 }
    pF0;pF7;    { BEQ lda>Kbd a-t-on tapé une
                  touche ?}
    p8F;p10;pC0;p00; { STA >KbdStrobe si oui
                  on efface KbdStrobe}
    p8A;        { TXA on récupère la valeur}
    p29;p7F;    { AND #$7F }
    pC2;p20;    { LONG M }
    p83;p1C;    { STA $1C,s conserver le résultat
                  sur la pile}
end;
```

et on récupère la valeur ASCII de la touche pressée par *Key:=KeyPress(0)*. Facile, non ? ▶

Avec ORCA/Pascal

Le passage de paramètres entre Orca/Pascal et l'assembleur est décrit au chapitre 5 du manuel de référence Orca/Pascal.
A la différence de TML, ORCA accepte très facilement des routines complètes écrites en assembleur et qui ne seront ajoutées à l'application qu'au moment du linkage. Chaque procédure ou fonction définie dans un module non Pascal (C, Assembleur) sera déclarée EXTERN dans le corps du programme. Le passage de paramètres se fait bien entendu par la pile et dans l'ordre des déclarations:
avec *resultat:=MaFonction(parm1, ..., parmN);*
on aura les paramètres empilés comme suit: *parm1, parm2,...,parmN,* et avant de quitter la routine, le programmeur devra dépiler le contenu de la pile afin de transmettre le résultat dans A (et X éventuellement).
Note: Il est impératif de se créer une page directe locale immédiatement en entrant dans la routine Assembleur (et ne pas oublier de restaurer le pointeur de pile en quittant !)

pile avant appel:	*pile après appel:*
anciennes valeurs	anciennes valeurs
	parm1
	parm2
	....
	parmN
	adresse de retour

La modification de plusieurs variables peut se faire facilement en passant comme paramètres les adresses d'implantations des variables.

En résumé, pour le programmeur Pascal, l'appel de fonctions et procédures assembleur se résume à:
• déclarer lesdites procédures et fonctions EXTERN.
• appeler les procédures et fonctions.
L'écriture du module assembleur obéira aux règles suivantes:
• les paramètres passés dans la fonction sont empilés par le compilateur dans l'ordre de leurs déclarations.
• la valeur de retour doit être dans A (valeur basse) et X (valeur haute).
• si on passe les adresses des variables en tant que paramètres, on peut les modifier directement en mémoire.
• une fonction assembleur peut appeler une procédure ou fonction Pascal.
La documentation ORCA étant suffisamment explicite sur le sujet, je ne m'étendrai pas plus, si ce n'est pour signaler la **supériorité de ORCA par rapport à TML**: on peut véritablement écrire des routines assembleur.

Exemple: module Pascal à sauver sous le nom SOURCE.PAS:

```
program Function_Asm(output);
uses common;
var resultat, entree: integer;
function MyFunction(a: integer): integer; extern;
                { on déclare la fonction EXTERN }
begin
        entree:=123;
        writeln('valeur en entree ',entree);
        resultat:=MyFunction(entree); { appel à la
                fonction }
        writeln('valeur en sortie ',resultat);
end.
{append 'Routine.asm'}
```

module Assembleur à sauver sous le nom ROUTINE.ASM:

```
MyFunctionstart
parm    equ 4   paramètre Integer empile
ret     equ 1   adresse de retour
        tsc     récupérer pointeur de pile
        phd     sauver l'ancien registre de page directe
        tcd     nouvelle page directe = pointeur de pile
        .....   faire le travail voulu et mettre ré-
                sultat dans X
        lda ret+1   positionner l'adresse de retour
        sta parm
        lda ret-1
        sta ret+1
        pld     récupérer l'ancien registre de page di-
                recte
        pla     pile sur l'adresse de retour
        txa     résultat dans A
        rtl
        end
```

compiler ces modules comme suit:

```
compile Source.Pas keep=$
compile Routine.Asm keep=$
link source routine keep=$
```

Dans cet article j'ai détaillé le passage de paramètres à partir du Pascal vers des routines Assembleur, le but étant de montrer la simplicité de la chose. Il est évident que ORCA est plus souple quant à l'emploi de routines externes, mais il n'était pas inintéressant de montrer que TML peut faire la même chose, au prix d'une programmation un peu lourde.
On ne peut souhaiter que la sortie rapide d'une mise à jour de Complete Pascal II avec un compilateur pour APW comme c'était le cas pour les versions 1.0 et 1.5 de TML I, ce qui rendra alors TML tout aussi performant que Orca. ■

Nouvelles du monde extérieur :

Exotica

François Hermellin

Apple: encore des produits nouveaux

Apple continue à sortir des produits plus vite que son ombre:

• En Juillet, deux nouvelles **imprimantes Laser** sont prévues. Celles-ci sont basées sur les moteurs laser Canon SX, et proposent 8 pages par minute. La plus perfectionnée devrait comporter un connecteur Ethernet.

• En Août, un nouveau **scanner** remplacera le modèle 16 niveaux de gris.

• En Octobre, le **Mac Tower 040**, dit Mac Ti, sera commercialisé. Basé sur le 68040, il remplacera le FX dans la catégorie "haut-de-gamme". Il serait fortement question d'un **Mac Classic 030**, destiné à remplacer l'actuel SE 30.

• En Novembre, et c'est certainement l'annonce la plus attendue, Apple se déciderait à sortir des "**portables**"... portables !

Le plus petit, réalisé grâce à l'aide de Sony, comporterait un lecteur, un écran LCD rétro-éclairé, et devrait être cadencé à 16 MHz. Il possèdera également un disque dur interne de 20 Mo et 2 Mo de RAM. Son autonomie serait de 2 à 3 heures. Il serait proposé dans la zone des 2500 $.

Le second "note-book", d'une taille plus importante, sera basé sur un 68030 et devrait coûter 3200 $.

Le dernier modèle correspondra au CI, avec un 68030 à 25 MHz. Il sera doté d'un écran à matrice active, comme les portables actuels, d'un disque dur de 40 Mo et d'un superdrive.

Enfin !

Alléluia! Le système 7.0 arrive! Le 13 mai, la version US sera disponible, nous devrons attendre jusqu'en Juin pour disposer de la version française. Enfin sur Mac des sous-catalogues dans le dossier Système, et des accessoires en-dehors du fichier System.

Next prend ses aises

Next s'installe plus confortablement. La société de Steve Jobs transporterait son siège européen de Genève à **Sophia-Antipolis**, avec, à terme, un centre de 300 personnes. Steve Jobs est venu en personne en Europe présenter les nouvelles machines et orientations de la société. Beaucoup de monde présent mais pas de grandes nouveautés présentées.

Il semblerait que Next annonce d'ici 6 mois une **nouvelle machine à base d'un processeur RISC**. Next compterait bien ainsi garder l'image d'une société qui a toujours un métro technologique d'avance.

En revanche, il semble que la société C-Cube ait quelques problèmes à mettre au point sa puce de compression-décompression d'images pour Next. La carte Next Dimension comportera peut-être une puce provenant d'une entreprise concurrente, créée par des anciens de C-Cube.

Inmos et l'architecture parallèle

Inmos a annoncé le **T9000**, dernier-né des microprocesseurs à architecture parallèle (connus également sous le nom de **Transputers**). La filiale de SGS-Thompson a implanté sur la même puce tout ce qui compose normalement un ordinateur (microprocesseur RISC proprement dit, mémoire, canaux de communication...).

La puissance dégagée est considérable: cadencé à 50 MHz, le T9000 atteint **70 Mips** (et 200 Mips en crête). Le débit des ▶

canaux de communications permettra à plusieurs T9000 de communiquer entre eux et de joindre leur puissance.

General Magic

Apple est-il à la recherche de son génie créatif? De plus en plus mal à l'aise entre Next, qui symbolise l'innovation, et les alliances nécessaires avec des partenaires japonais (notamment dans le domaine des portables), la société à la a décidé de créer une petite équipe autonome, disposant de moyens financiers considérables, et regroupant les individus les plus créatifs d'Apple pour mettre au point les machines de demain.
Cette petite cellule de recherche, baptisée General Magic Inc et située à Mountain View, travaille actuellement sur un ordinateur "**palmtop**" sans clavier, disposant de liaisons radios pour ses communications avec le reste du monde.

Ici Radio Apple?

Apple s'intéresse d'ailleurs de très près aux réseaux communiquant par ondes radios. Plus simples d'installlation, ils ne nécessiteraient plus les habituels kilomètres de câbles. Apple a ainsi déposé une demande auprès du gouvernement US afin de disposer d'une fréquence libre.

Encore une réorganisation chez Apple

Bouleversement au sein de la division R&D d'Apple, le groupe Recherche et développement est désormais scindé en trois divisions distinctes: matériels, logiciels et systèmes complexes. Le service marketing sera également divisé en trois parties.
Le but de cette réorganisation serait de permettre une meilleure adéquation entre ingénierie et marketing en vue de l'introduction de nouveaux modèles.

L'informatique au Japon

Pour qu'**Exotica** mérite son titre, voici quelques notes sur l'état du marché informatique au Japon, pour préparer vos prochaines vacances: car le Japon est un des grands de l'informatique.

Traditionnellement, le Japon était considéré comme possédant une avance certaine dans le domaine du hardware, mais avec une faiblesse pour tout ce qui concerne le logiciel.

Les choses sont en train de changer. Avec une progression de plus de 60% entre 87 et 88, le marché du logiciel au Japon dépasse la moyenne mondiale. Les développements sont cependant encore en grande partie réalisés en interne dans les entreprises, échappant ainsi aux enquêtes.
La faiblesse, jusqu'ici, de l'industrie du logiciel peut être expliquée par l'absence de politique dirigiste de la part du MITI à l'égard du logiciel, contrairement à ce qui a été le cas pour le matériel informatique.
Le développement de logiciels au Pays du Soleil Levant est dominé par les constructeurs, qui contrôlent étroitement les éditeurs de softs. Les solutions "clés-en-mains" (ordinateurs + applications) sont très courantes.
Les six constructeurs les plus importants (Fujitsu, Nec, Hitachi, Toshiba, Mitsubishi et OKI) ont développé jusqu'à présent plus de 9000 logiciels. La part des logiciels importés est forte, avec un peu moins du tiers des logiciels professionnels utilisés. Les logiciels qui se vendent bien aux USA connaissent également de bonnes ventes au Japon dans la plupart des cas. Aussi, il arrive que des sociétés éditrices de logiciels ne réalisent en fait que la localisation de programmes US.

Au plan du hard, si Apple pénètre bien (voir les Exotica de ToolBox-Mag 4), IBM a créé en Mars l'OADG ou Groupe des Développeurs de l'Architecture Ouverte. Celui-ci a pour objectif de promouvoir au Japon les compatibles PC. En effet, ces derniers sont peu présents dans ce pays.

En ce qui concerne les constructeurs japonais, la société Nec représente à elle seule plus de 50% des ventes de matériel informatique. Les machines les plus vendues pour une utilisation domestique/professionnelle sont le FM Tower de Fujitsu, le X68000 de Sharp et le PC 89 de Nec.
La plupart offrent la compatibilité PC en option: il suffit d'acquérir le système MS-Dos. Ces machines sont généralement largement plus évoluées que les "brouettes" PC (lecteur de CD-Rom en version de base, coprocesseur graphique...)
Enfin, l'avance japonaise dans le domaine de l'image vidéo tend à engendrer un quasi-monopole mondial des Japonais sur les consoles de jeu: Taïto travaille sur console Nintendo, vous le savez...

■

Enfin un sélecteur qui ne fait rien d'autre !

Sélect

Documentation pour l'utilisateur

Jean Destelle

Présentation de Sélect

Sélect est un sélecteur de programmes, et rien que cela. Il permet de constituer un nombre illimité d'écrans différents pouvant comporter chacun les noms de 40 programmes ou applications, et lance directement tout programme SYS, S16, ou Applesoft (Prodos 8).
Sélect assure la fonction principale de lancement remplie par le Finder, ou d'autres sélecteurs, mais **ne fait volontairement que cela**. Vous pourrez continuer à utiliser les autres fonctions de Prosel, Wings, ou le Finder, en les lançant à partir de ***Sélect***. De ce fait, l'écran d'entrée de ***Sélect*** peut ne comporter que quelques applications, et son emploi par un utilisateur novice est absolument sans danger.
Les écrans de ***Sélect***, de couleurs claires et aux textes bien lisibles, sont esthétiques, et ne comportent **pas de données inutiles**. Ayant créé une case pour le lancement d'une application à un endroit de l'écran, vous la retrouverez toujours au même emplacement, tant que vous le souhaitez du moins. Une fonction complémentaire permet le lancement rapide de tout programme non connu de ***Sélect***.
Il est facile de circuler d'un écran à l'autre de ***Sélect***, en cliquant simplement à chaque fois dans une case. Chaque écran sera de la couleur choisie et les cases de commandes peuvent être créées, modifiées, colorées, déplacées, ou supprimées très facilement.

Sélect stocke ses données dans la partie ressource du fichier *SelectSys*. Il n'a donc besoin d'**aucun fichier de data**: le Resource Manager se charge de tous les accès disque.
Sélect éteint son écran après deux minutes de non-emploi, afin de l'économiser, et permet aussi d'éteindre le système.

Sélect nécessite un disque système avec GS/OS 5.04 ou ultérieur, contenant, dans son catalogue principal, le fichier *Basic.Launcher*. On installe normalement ***Sélect*** en copiant le fichier *SelectSys* de la disquette ToolBox-Mag à la place du programme *Start* (habituellement le Finder), dans le sous-catalogue */System* du disque de démarrage. Voyez dans ToolBox-Mag 6 l'article d'Eric Weyland sur GS/OS 5.04 pour l'installation de ***Sélect*** comme *Start*.
Rien n'empêche toutefois de recopier *SelectSys* dans un autre dossier et de le lancer au moyen du Finder ou d'un autre sélecteur. Veillez à ce que le disque contenant ***Sélect*** ne soit pas protégé en écriture.

Premiers contacts avec Sélect

Lorsque vous avez installé ***Sélect***, les données de sélection sont pratiquement vides, et ne comportent que quelques cases remplies à titre d'exemples. Vous pouvez les essayer pour bien comprendre l'emploi de l'application. Les cases de lancement se réfèrent à quelques mini-programmes de démonstration ou des programmes se trouvant sur la disquette ToolBox-Mag. Vous pourrez supprimer ces petits programmes par la suite.
Au départ vous disposez de trois écrans, cela pour vous montrer comment circuler de l'un à l'autre. Le premier à apparaître est l'***écran principal.***
Il comporte: en haut, une ***case de titre*** longue: c'est le nom de l'écran. Juste en-dessous, une ligne de quatre cases de titres. Notez que les ***cases de titres*** sont entourées d'un cadre.
Les autres cases présentes sont des ***cases de lancement*** et comportent toutes la désignation d'un programme, ou d'une destination vers un autre écran.
Par exemple l'une des cases comprend la mention **Ecran Auxiliaire**. Si vous cliquez dans cette case, vous vous retrouvez dans un autre écran. Pour revenir au premier écran, cliquez dans la case **Ecran Principal**.

Pour lancer un programme connu du sélecteur, il suffit de cliquer dans la case corres- ▶

pondante. Lancez l'un ou l'autre des mini-programmes de démonstration. Vous reviendrez toujours directement à *Sélect* si l'application lancée comporte une sortie "propre".

Pour lancer un programme inconnu du sélecteur, déroulez le menu **Fichier** et actionnez le choix **Lancer autre**. Une fenêtre de choix apparaît, et tous les programmes lançables par *Sélect* sont à votre disposition.

Ajouter des programmes

Actionnez, dans l'écran principal, la case **Editer Sélect**. Sur la barre de menus qui apparaît, le menu **Editeur** est en grisé, donc actuellement inaccessible. Pour l'utiliser, vous devrez actionner le choix **Editer l'écran**, dans le menu **Fichier**.
Sélect se place alors en *mode éditeur*. Ce mode est reconnaissable à *quatre signes*:
• toutes les cases sont entourées de rectangles blancs.
• dans le menu à droite, la marque **Sélect** est remplacée par **Edit Sélect**.
• le menu **Editeur** est accessible.
• le numéro de l'écran apparaît en haut sur la droite.

Pour ajouter une case de lancement de programme, vous devez d'abord choisir un emplacement, n'importe lequel pourvu qu'il soit vide, vous pourrez ensuite déplacer la case.
Donc, cliquez dans une case vide. Elle s'inverse.
Choisissez **Ajouter...** dans le menu **Editeur.** Vous pouvez choisir dans un dossier ou dans un disque quelconques le programme à lancer. Quand il est choisi, vous cliquez pour l'ouvrir.
Une fenêtre de dialogue de *Sélect* apparaît alors, vous précisant les données d'accès de ce programme. N'y changez rien, sauf en ce qui concerne le **nom de sélection** qui sera reproduit dans la case de lancement de votre écran. Souvent le nom du programme dans le catalogue n'est pas très élégant. *Sélect* vous permet donc de réécrire un nom en bon français. Vous avez droit à 20 caractères au maximum.
Quand c'est bon, cliquez pour sortir. Les données sont alors enregistrées. C'est fini. Vous retrouverez votre écran avec, à la place choisie, le nom de votre programme, dans une case de couleur blanche.
Pour colorer ce nom, cliquez sur sa case, choisissez **Couleur**, cliquez sur la couleur choisie parmi les six qui vous sont offertes.
Pour déplacer une case: cliquez sur la case. Ensuite enfoncez la touche Ó, et cliquez sur la case de destination (qui doit être vide).

Vous pouvez ainsi ajouter tous les programmes que vous souhaitez. Pour supprimer une case, il suffit de cliquer dessus, et de choisir la fonction **Supprimer**. Vous pouvez aussi ajouter des cases de titres, les colorer et les déplacer, de la même façon que des cases de lancement. Vous pouvez corriger un nom de sélection ou un chemin d'accès en utilisant le choix **Modifier...** de l'éditeur.
Pour sortir de l'éditeur, utilisez le choix **Quitter l'Editeur** du menu **Fichier** ou bien actionnez Ó-S.
Si vous désirez éditer un autre écran existant, il faut quitter l'éditeur, puis vous placer dans l'écran choisi en cliquant dans sa case. Ensuite, appelez l'éditeur et faites les modifications souhaitées.

Création d'un nouvel écran

Il ne vous est pas proposé de choix pour supprimer directement un écran: il suffit de supprimer les cases pour obtenir un écran vierge.
Lorsqu'on crée un nouvel écran, *Sélect* se charge de dessiner la grande case de son titre, et fabrique deux autres cases: l'une dans l'écran de départ pour aller vers le nouvel écran, et l'autre dans l'écran nouveau pour revenir à la case de départ. Vous pourrez rajouter dans un écran donné d'autres cases d'accès vers d'autres écrans, par exemple pour retourner directement à l'écran principal.
Placez-vous dans l'écran de départ, c'est-à-dire celui à partir duquel vous souhaitez généralement appeler votre nouvel écran. Cliquez dans l'emplacement de la case choisie pour accéder à cet écran. Actionnez le choix **Ajouter un écran** du menu **Editeur**.
Dans la fenêtre de dialogue qui apparaît, vous devrez choisir le nom de l'écran, et la couleur. Pour la couleur, indiquez un nombre de 1 à 15 dans la case correspondante. En cliquant sur le bouton **Test Couleur**, vous verrez apparaître un rectangle de la couleur choisie. Ce sera la couleur du fond de l'écran. Vous pourrez toujours facilement modifier cette couleur, comme le nom de l'écran, en utilisant le choix **Modifier écran** du menu **Editeur.**
Lorsque l'opération est effectuée, vous revenez dans l'écran de départ où vous trouvez la case blanche vers le nouvel écran. Pour voir ce nouvel écran, vous devez quitter l'éditeur, puis cliquer sur sa case.
Vous devez alors voir, sur la gauche du nouvel écran, la case de retour. Vous pouvez maintenant éditer ce nouvel écran. Vous pouvez ajouter d'autres cases de circulation vers d'autres écrans par l'item **Ajouter accès écran** du menu.

▶

Quelques conseils

Conservez soigneusement votre disquette ToolBox-Mag.
Vous aurez besoin d'y revenir si vous désirez refaire complètement la distribution des écrans de *Sélect*. Si vous voulez tout changer, le plus simple est en effet de recopier le fichier original *SélectSys* à la place du vôtre.

Utilisez au mieux les titres et les couleurs.
• Il est pratique de grouper dans une même colonne, sous un titre, les programmes d'une même famille.
• Regroupez, dans l'écran principal, toutes les applications dont vous vous servez souvent, réservant les écrans auxiliaires aux programmes les moins utilisés.
• N'hésitez pas à créer un deuxième accès dans l'écran principal vers un programme situé logiquement dans un autre écran, si vous devez vous en servir souvent.
• Pour une utilisation professionnelle ou par des béotiens: créez des écrans différents pour les principaux utilisateurs, ne comportant que ce qui leur est utile.

Lancement de programmes binaires et exec
Il est théoriquement possible de lancer ces fichiers par le Standard File. Toutefois, comme absolument rien ne distingue les fichiers de programmes lançables d'autres fichiers textes ou binaires, le choix ne serait guère facile. Solution: créez un mini-programme de lancement en Applesoft, que vous lancerez lui-même par *Sélect*.

Cohabitation avec d'autres sélecteurs
Un sélecteur (comme *Sélect*) est une application comme une autre. Il doit donc pouvoir se lancer et se quitter comme toute application. Si ce n'est pas le cas de votre sélecteur: jetez-le. S'il s'agit du Finder ou de Wings, ToolBox-Mag a publié des patches pour leur donner un Quit normal.

Emploi des accessoires de bureau
Quand vous utilisez un NDA, sa fenêtre s'ouvre au dessus de l'un des écrans de *Sélect* (qui est une fenêtre ordinaire, mais fixe).
Si vous cliquez dans votre écran sans avoir fermé le NDA, ce dernier va aller se cacher derrière la fenêtre de votre écran, mais sera toujours là. Pour le revoir à nouveau, vous devez actionner, dans le menu **Fichier**, la fonction **Fermer** qui va fermer la fenêtre de votre écran. Pour revenir à l'écran de *Sélect*, il suffira d'actionner **Ouvrir l'écran**.

■

Platinum Paint: l'abondance

Eric Weyland a dit dans nos colonnes beaucoup de bien de Platinum Paint, de Beagle Bros. Il n'est pas le seul de son avis: je ne connais pas moins de **trois moyens** de se procurer cet excellent logiciel.
1. Le mien: étant un acquéreur de son ancêtre, **MiniPaint** de Jem Software, quand j'ai été averti de l'offre de Platinum à bon prix en échange d'une page de la documentation de GS Paint (elle commence à manquer de pages de garde, celle-là), j'ai acquis la version US outre-Atlantique. Acheter directement au producteur américain facilite parfois les mises à jour, mais ne donne pas de documentation française.
2. La formule Toolbox, qui comprend la version US avec sa documentation, plus une documentation française, plus une version francisée du logiciel, qui se présente comme un complément de la documentation française (elle est d'ailleurs partiellement bilingue). La version US est estampillée Beagle, la documentation française, incluant la version francisée, est estampillée Toolbox.
3. La formule Bréjoux: une version française de Platinum, avec documentation française. Cette version est entièrement estampillée Beagle, mais n'inclut pas la version originale US.
Quelle formule choisir, où acheter, à quel prix? ToolBox-Mag, magazine sans publicité, ne s'intéresse vraiment pas à ce genre de questions. Vous êtes bien assez grand pour choisir votre formule et votre commerçant vous-même: dans les trois cas, vous aurez un Platinum Paint complet.

Dans le fond, que les menus de Platinum Paint soient en serbo-croate, en anglais, en français, francisés, etc, importe en fait fort peu: il s'agit d'un **logiciel de dessin**, après tout, pas d'un traitement de textes.
La difficulté de Platinum est de **comprendre ce que font ses fonctions**, ce qui n'est pas une question de nom des menus, mais de *lecture de la documentation* (si vous n'êtes pas à l'aise en anglais, ma formule n'est pas pour vous).

Si Platinum Paint est bien le logiciel de dessin d'aujourd'hui pour le GS, il ne suffit pas de l'acheter pour savoir ce qu'est une courbe de Bézier: il faut surtout **lire de près sa documentation,** indispensable pour exploiter toutes les possibilités de cet outil très puissant.

JYB

Sélect, Tml Pascal et les ressources

Jean Destelle

Sélect a été écrit à titre d'exercice d'emploi des ressources, et pour rendre plus agréable la circulation entre les divers programmes répartis dans tous les coins de notre disque dur.
Vous trouverez sur le disque dans le catalogue */Select*:
• *SelectSys*: l'application complète, comportant le sélecteur, son éditeur, et dans sa partie ressources, les données de lancement et des écrans.
• les sources, écrits en TML Pascal II, de *SelectSys*; le fichier texte *Plan.Src.Select* est un plan global des fichiers sources.
• le fichier de ressources nécessaire à la compilation.
• le listing des ressources (dans leur version minimale de départ) établi avec l'analyseur de ResDoctor.

Nous supposons que vous avez fait une copie-papier des fichiers-sources concernés. Attention: TML II peut compiler, mais ne peut pas imprimer directement ces sources sur ImageWriter. En effet, à la demande de notre Rédacteur en Chef, les commentaires comportent les caractères français accentués, que TML ne sait pas imprimer. Pour imprimer, utilisez un traitement de textes GS comme AppleWorks-GS, le NDA Write-It, etc (le driver d'ImageWriter de GS/OS 5.04 permet une impression rapide de listings en mode graphique).

SelectSys ne peut pas être compilé en mémoire, mais seulement sur disque, à cause de la référence interne au nom du fichier.

Première partie: *Sélect*, TML Pascal et la Toolbox

Le cahier des charges

Sélect doit assurer deux fonctions bien distinctes: la création et la gestion des données nécessaires au lancement des programmes, d'une part: c'est le rôle de l'éditeur; l'exploitation de ces données et le lancement lui-même, d'autre part: c'est la fonction de lancement du Sélecteur.
Nous voulions obtenir, dans un environnement Desktop classique, un nombre illimité d'écrans différents comportant chacun des titres, des cases de couleurs (agissant comme des boutons de contrôle) portant les noms des programmes à lancer, des cases de circulation pour aller et venir d'un écran à l'autre, tout cela très facile à utiliser, à créer, à modifier, à colorer, à déplacer. On s'imposait 40 cases au maximum par écran. De plus, l'application devrait comporter une fonction de lancement pour tout programme non connu du sélecteur.

Le Control Manager étant trop lent si les contrôles sont nombreux, nous avons renoncé à ses boutons, et la gestion des cases de lancement est faite par l'application, en utilisant des ressources.

Subdivisions du programme

Sélect est divisé en **trois parties**:
1. L'Unit *AuxSelect1* (source *AuxSelect1.pB*), qui comporte un nombre restreint d'outils généraux pouvant servir de base de départ à tout programme en desktop:
• démarrage et fermeture des outils,
• gestion des erreurs,
• création des menus et des fenêtres,
• gestion des nouveaux contrôles de type 1.
2. L'Unit *SelUnit* (source *SelUnit.pB*), contenant plusieurs procédures réutilisables, montrant en particulier comment on peut résoudre, en Pascal, certaines difficultés de recours à de petites routines en mode émulation. Cette unit assure entre autres la fonction d'extinction du GS, avec éjection des disquettes, et d'économiseur d'écran.
3. Le programme complet *SelectSys* (source *Select.sys.pB*), contenant tout ce qui est spécifique à l'application. Il utilise, bien entendu, d'une part les deux units ci-dessus, d'autre part, un grand nombre d'units d'interface de TML Pascal II, et enfin, un fichier de ressources *Select0.rsrc* que vous trouverez sous forme compilée dans *Select0.r*.

L'Unit polyvalente AuxSelect1

Cette unit comporte deux parties: d'abord les utilitaires les plus généraux, puis quelques outils destinés à faciliter l'emploi des nouvelles fonctions de type 1 de ControlManager.
Vous pouvez l'employer, telle quelle, comme début de tout programme en desktop. Il suffit de lui adjoindre un fichier de ressources réduit, comportant seulement les éléments nécessaires au chargement des outils et au menu de base. Le programme peut alors se réduire à quelques lignes, et pourra être utilisé comme point de départ d'un nouveau source.

Les fonctions de base de *AuxSelect1* n'ont rien d'original, et de nombreux exemples de ce genre ont déjà été publiés. Toutefois, vous trouverez quelques outils particuliers:
• la fonction *BreakAlert*, qui emploie *AlertWindow* de WindowMgr, et peut être placée à un endroit quelconque de votre programme, pour provoquer un arrêt et vous donner l'information souhaitée. Vous en rencontrerez quelques applications volontairement conservées, dans le source *SelectSys.p*. ▶

• la procédure *Marque*, destinée à placer une marque distinctive, le nom de votre programme par exemple, dans la barre de menus, tout à droite. Toutes les applications desktop ont tendance à se ressembler, et cela permet de savoir où on se trouve sans passer par le "A propos" du menu ♂.
• la procédure *SetFrenchFont* permettra de mettre en place votre police personnelle de caractères.
• les fonctions destinées à la gestion des contrôles constituent une interface pratique. Car les nouveaux contrôles du type 1, introduits avec GS/OS et le système 5.0, ne sont pas aussi faciles d'emploi que ceux des dialogues du DialogMgr.
Nous avons pris la décision d'employer les nouveaux types de fonctions, et les ressources chaque fois que nous le pouvions. Vous trouverez donc là six fonctions qui devraient entrer dans votre arsenal.
Le Pascal permet, avec une grande facilité, de constituer ainsi sa toolbox personnelle. En général, on crée ces procédures ou ces fonctions à l'occasion d'un besoin particulier, dans le cours de l'écriture d'un programme. Ensuite, on peut reclasser dans des units particulières tout ce qui est intéressant pour d'autres applications. Vous trouverez quelques exemples de "futurs outils" dans *SelUnit*.

Le Pascal et le mode émulation

Le mode émulation permet de se servir des routines du "firmware", absolument comme si nous étions sur notre vieil Apple II. Pour passer d'un mode à l'autre, ce que fait assez souvent le système, on utilise en général l'Assembleur. Les habitués du Pascal n'auraient pas beaucoup de moyens, a priori, pour se sortir de cette difficulté. Dans l'unit *SelUnit*, vous trouverez quelques exemples de procédures employant le GS en mode émulation.
La procédure *Ejecter* montre comment on peut placer une petite routine en langage machine à un endroit donné de la mémoire, puis faire appel à elle. Nous avons adapté une petite routine classique d'éjection SmartPort, en l'assemblant "sur le pouce", pour la loger en $00/300, et pour accepter un paramètre: le numéro de device.
Ensuite, il suffit de transformer le code hexa en une chaîne Pascal baptisée *s*, et d'utiliser la procédure Pascal *StuffHex*, qui place une chaîne hexa directement en mémoire à l'adresse *a* (comme l'aurait fait une série de poke): *stuffHex(pointer(a),s)*.
L'appel se fera par *FWentry* (MisTools). Cette fonction admet 4 paramètres: le premier paramètre à transmettre (accumulateur A), le deuxième paramètre à transmettre (registre X), le troisième paramètre à transmettre (registre Y), enfin l'adresse de la routine à appeler.
FWEntry passe en mode émulation, exécute la routine, et repasse en mode natif. Après exécution de la routine, le résultat éventuel se retrouvera dans la valeur de la fonction. En sortie, on retrouve, dans un record de type *FWRec*, les valeurs des trois registres après exécution de la routine, plus le statut du processeur.
Nous n'utiliserons qu'un paramètre à l'entrée: *NoDevice*, le numéro du périphérique, et nous n'avons pas besoin de résultat en sortie. Notez que pour une commande du smartport, le numéro de device n'est pas le même que le numéro de GS/OS. L'instruction s'écrit:
LeFWRec := FWEntry(NoDevice, 0,0,$0300);
Les procédures *LancerProgramme* et *LancerAutre* de *SelectSys* utilisent également *FWEntry* pour appeler "Cout" (FDF0).

[NDLR: si l'appel à FWEntry est orthodoxe et réutilisable, il n'en va pas de même de la pratique qui consiste à occuper par des pokes un bout de mémoire en banc 0, sans rien demander au Memory Manager, et sans s'occuper de ce qu'on remplace. A moins de faire comme ***Sélect*** *et d'éteindre le GS aussitôt après, cette pratique ne nous semble pas réutilisable.]*

Lancement d'un programme par GS/OS

Le lancement d'un programme se fait au moyen de la fonction *Quit* de GS/OS, qui admet deux paramètres regroupés par le Pascal en un record: le chemin d'accès de l'application à lancer (pointeur vers une *GSstring*), et le paramètre *Flags* (ou *quitflag*) qui comporte deux éléments:
• bit 15: si 1, GS/OS reviendra au programme que l'on quitte après exécution de celui qui va être lancé;
• bit 14: si 1, GS/OS essaiera de conserver en mémoire le programme que l'on quitte, pour le relancer plus vite ensuite.
Dans notre cas, nous rendons obligatoire le retour à ***Sélect***. Mais nous ne nous laisserons pas tenter par sa conservation en mémoire, car nous lançons aussi des programmes Prodos 8. Nous donnerons donc au *quitflag* la valeur $8000 dans tous les cas.
Pour lancer un programme en Prodos 8, nous passerons par *Basic.Launcher*, qui a été conçu à cet effet: on lance *Basic.Launcher* après avoir posté un message en poste restante au *Message Center*, indiquant le chemin d'accès du programme P8 à lancer (pour l'utilisation de MessageCenter, voir Toolbox-Mag 4, pages 5 et suivantes).

Ici, deux difficultés, pour l'utilisateur, inhérentes à Prodos 8:
• Rien de commode ne distingue un programme binaire exécutable et un simple fichier binaire de données. Même chose pour les fichiers exec et les fichiers texte. Nous les avons donc exclus de la sélection par StandardFile, mais rien n'empêche d'en programmer le lancement par l'éditeur de ***Sélect****.*
• GS/OS reconnaît soit '/' , soit ':' comme séparateur dans les chemins d'accès, ainsi que des préfixes numérotés. Prodos 8 ne connaît que '/' comme séparateur, et ne connaît pas les préfixes numérotés. Vérifiez que les chemins d'accès sont bien libellés.
Nous avons par ailleurs limité la longueur des chemins d'accès complets à 127 octets, et les noms de fichiers à 32 octets, ce qui semble raisonnable. Par ▶

précaution, nous mettons en préfixe 0 (le préfixe courant) le chemin d'accès de l'application avant de la lancer.

Deuxième partie: *Sélect* et les ressources

Sélect emploie presque exclusivement les nouveaux outils introduits par le système 5.0. Pour tous les menus, les fenêtres, contrôles des dialogues et les alertes de "A propos..." et "Instructions", vous trouverez donc des ressources de types standard. Chaque "écran" de ***Sélect*** est en fait une fenêtre ordinaire, mais sans titre ni barre de défilement, afin de ne pas perdre de place. Ces ressources standard ont été créées par l'éditeur de ressources de TML Pascal II. Malheureusement, celui-ci a des bugs et ne laisse pas de trace écrite de ce qu'il a fait. Les ressources ont donc été ensuite listées, complétées, modifiées et retouchées avec le programme "ResDoctor", qui a été écrit pour cette fonction. Pour ***Sélect***, il a également fallu créer des ressources spéciales, non standard.

L'emploi de ressources spéciales

Pour nos ressources, nous avions retenu l'organisation suivante:

- *Pour chaque écran*, une ressource comportant:
  - un numéro d'écran pour le retour à l'application,
  - la couleur de l'écran,
  - son nom,
  - une liste des cases de l'écran, repérée chacune par 2 données:
    - l'ID de la ressource de la case,
    - le numéro de position de cette case sur l'écran (de 1 à 40, 10 lignes de 4 cases).
- *Pour chaque case* (de lancement ou de titre), une "ressource d'item" comportant (au moins):
  - le nom écrit sur la case (20 caractères),
  - la couleur de la case,
  - une indication de la fonction de la case:
    - lancement d'un programme SYS ou S16,
    - lancement d'un programme Basic,
    - titre ne lançant rien du tout
    - changement d'écran.

Il fallait donc écrire, en Pascal, les structures de données correspondant à chacune des ressources dont nous avons besoin, sous forme de déclarations de types: les *records*. Vous trouverez ces déclarations au début du source de *SelectSys*. Sont définis trois types de records: le *ScreenRec* (ressource type $1002, décrit un écran); l'*ItemPositionRecord* (pas de ressource utilisée), comportant deux données, l'ID de l'Item, et le numéro de position utilisés dans la liste du *ScreenRec;* et l'*ItemRec* (ressource type $1004, décrit la case).

La difficulté est de trouver **comment passer des records du Pascal aux ressources de GS/OS**. Chaque ressource doit contenir très exactement tous les éléments du record, sous le format exact défini par la déclaration du type du record. Quand nous aurons besoin, par exemple pour un lancement, des données contenues dans la ressource de type $1004 concernant une case, nous demanderons à ResMgr de charger cette ressource dans un bloc de mémoire préparé pour recevoir le record. Puis nous exploiterons ce record comme une variable record ordinaire du Pascal.

Nous pouvons faire cela de plusieurs manières. Par exemple, voici comment on prend les données pour dessiner chaque case au moment de l'ouverture d'une fenêtre écran (voir tout à fait à la fin du source, dans la procédure *FaireRectangles*):
D'abord dans la déclaration des variables, on trouve une variable locale *CetItem* qui aura le type *SelectItemRec:*

```
var CetItem: SelectItemRec; b: longint ;
```

La variable b sera l'ID de la ressource.
Dans le corps de la procédure, à l'intérieur de la boucle de dessin des cases, nous extrayons quatre lignes (N est le numéro de position déterminé un peu avant):

```
CetItem:=SelectItemRecHndl
(LoadResource($1004,b)^^;
        if CetItem.flag1 = 2 then titrage:= true;
        Rectangle(N, CetItem.nom, CetItem.Couleur);
        (* dessiner la case*)
        ...
```

Première ligne: la fonction *LoadResource* de ResMgr charge la ressource si elle n'est pas déjà en mémoire, et retourne un handle vers cette ressource. On "déréférence" ce handle pour obtenir le record lui-même. Les deux lignes suivantes indiquent ce qu'on fait du contenu de ce record: si le paramètre *flag1* du record vaut 2, alors c'est un titre. On appelle ensuite la procédure de dessin d'une case *Rectangle(...*, en lui passant 3 paramètres: le numéro de la position du rectangle, le nom à écrire et la couleur du rectangle, pris dans le record.

Les ressources en mémoire

Le handle *CetItem* fourni par la fonction *LoadResource* appartient au ResMgr. Vous avez le droit de vous en servir, mais il n'est pas à vous: c'est un handle de ressource. Il est répertorié dans la 'ResMap', ce carnet d'adresses que ResMgr a chargé tout au début du lancement de l'application.
Les conditions de résidence en mémoire de la ressource dépendent de "l'attribut" de la ressource elle-même mais il y a une petite ambiguïté: quand on utilise *LoadResource*, ResMgr crée un handle pour recevoir la ressource, tout au moins la première fois. Les attributs de ce handle seraient alors $C31C (locked, fixed, Purge 3, NoCross, page et NotSpecMemory), cela quel que soit l'attribut de la ressource. En effet, *l'attribut du handle et celui de la ressource sont deux choses différentes*. L'un est géré par Memory Manager et l'autre par Resource Manager.
Vous aurez donc la possibilité de déverrouiller et de verrouiller ce handle comme tout autre. Vous pourrez en modifier le contenu, l'allonger par exemple. Mais attention, il faudra prévenir le propriétaire ▶

par *MarkResourceChange*. Vous ne pouvez pas en disposer pour libérer la mémoire, à moins d'en prévenir auparavant ResMgr par *DetachResource*.

Un peu de syntaxe

Pour pouvoir considérer le contenu du bloc désigné par le handle comme un record de type donné, il faut faire un *cast*, c'est-à-dire un changement de type qui précise au compilateur Pascal que nous regarderons ce handle comme si c'était un handle sur le record en question. C'est ce que fait *SelectItemRecHndl(...*. Lorsque le compilateur aura lu cela, il traitera ce qui est entre les parenthèses comme un handle de *SelectItemRec*.
Les deux circonflexes de la fin de cette même ligne d'instructions indiquent que l'on prend le contenu du contenu, c'est à dire le record lui-même. En effet, un accent circonflexe placé à la suite d'une variable de type pointeur indique que l'on prend la donnée qui est à l'adresse du pointeur, et un handle n'est autre qu'un pointeur vers un pointeur.
Le point qui sépare les deux mots *CetItem.flag1* est très important. Il indique que le *flag1* en question est le paramètre du record *CetItem*. De même *CetItem.nom* désigne le paramètre *nom* du record *CetItem*, et *CetItem.Couleur* le paramètre *couleur*.

Création de nos ressources

Pour pouvoir utiliser nos ressources, il faut, auparavant, avoir créé au moins un exemplaire de chacun des nouveaux types de ressources. Nous l'avons fait en rajoutant, avec l'éditeur de ResDoctor, les ressources nouvelles au listing des ressources standard.
On part du fichier de ressources créé par TML II (qui est en code), on le décompile avec la fonction "Créer Fichier-Source" de ResDoctor, puis, avec le "Grand Editeur", on rajoute les deux ressources nouvelles dans le source.
Evidemment, il faut écrire celles-ci "à la main", puisque il s'agit de types non standard. ResDoctor donne un moyen simple de le faire, à partir de la définition des records. Voici ce qu'il suffit d'ajouter [les textes entre (* et *) sont des commentaires]:

```
Resource not Standard{($1002, 1, attr.=$0000, (* le 1er écran *)
L 1,                   (* longint =numéro de l'écran d'origine *)
w 4,                   (* integer =la couleur *)
b 17,                  (* octet = longueur du titre *)
t" Ecran principal  ", (*occuper les 20 caractères permis *)
w 1,                   (* integer =1 , 1 seule case utilisée *)
L 1,                   (* longint =ID de la rsrc de la case *)
w 5,                   (* integer = n° de position de la case. *) ))
```

```
Resource not Standard{($1004, 1, attr.=$0000, (* la 1re case *)
w 14,                  (* integer = couleur de la case *)
w 0,                   (* integer :flag1 = programme SYS *)
$8000,                 (* integer: quitflag = onStack *)
b 5,                   (* octet: longueur du nom *)
t"Basic       ",       (* nom : occuper les 20 carac permis *)
b 14,                  (* octet : longueur du chemin d'accès *)
t":BASIC.SYSTEM",      (* chemin d'accès *) ))
```

Remarque: les chaînes des noms, définies comme ayant 20 caractères au maximum dans les records, ont été placées dans les ressources de façon que les 20 caractères du texte soient occupés, qu'ils appartiennent ou non à la chaîne effective du nom. Pour repasser du record à la ressource, il faut tenir compte de la façon dont le Pascal range le record en mémoire, suite à notre déclaration de type.

Si vous faites une analyse du fichier ressources après compilation par ResDoctor, vous verrez que les ressources ont été enregistrées sous la forme ci-dessous.

```
(*  RESOURCE Type $1002   Not standard type    Count: 1   *)
(*================================================*)
Resource not Standard{($1002,  1, attr.=$0000,  (* Unknown format *)
h"01 00 00 00 04 00 11 20 45 63 72 61 6E 20 70 72 "  (* ### Ecran pr *)
h"69 6E 63 69 70 61 6C 20 20 20 20 01 00 01 00 00 ",(* incipal ### *)
h"00 05 00 ",                  )}
```

```
(*  RESOURCE Type $1004   Not standard type    Count: 1   *)
(*================================================*)
Resource not Standard{($1004, 1, attr.=$0000, (* Unknown format *)
h"0E 00 00 00 00 C0 05 42 61 73 69 63 20 20 20 20 ",(* #à#Basic    *)
h"20 20 20 20 20 20 20 20 20 20 20 0E 2A 3A 42 41 ",(*        #*:BA *)
h"53 49 43 2E 53 59 53 54 45 4D ",                  (* SIC.SYSTEM *) )}
```

Cette analyse ne fournit pas de renseignements complètement décodés, puisque nos ressources ne sont pas d'un type connu de ResDoctor. Les deux ressources, structures de longueurs variables, sont représentées en hexa. Sous cette forme, il sera possible de les éditer ultérieurement.

Gestion de nos ressources: création d'une ressource

La méthode utilisée pour créer des ressources dans le fonctionnement de ***Sélect*** est la suivante:
• on crée une variable du type du record correspondant à la ressource, en le définissant par un handle (variable locale et non globale, c'est très important) qui appartient à l'application;
• on assigne aux divers paramètres du record les valeurs voulues;
• on utilise *AddResource* pour ajouter cette ressource à la liste de celles gérées par ResMgr, qui se l'attribue alors (le handle n'appartient plus à l'application): il commence par la marquer de son tampon "changed".
• on peut appeler *WriteResource* pour que ResMgr écrive cette ressource en mémoire; ResMgr la marque alors "not changed ".
• dès que possible, on appellera *UpDateResourceFile* pour mettre à l'abri les nouvelle données dans le disque. Cette fonction fait appel à *WriteResource* à propos de chacune des ressources portant la marque "changed", et remet de l'ordre dans son "carnet d'adresses " (la resmap) et dans le fichier (res-fork).

Prenons comme exemple la procédure *CreerLecran*: on utilise une variable-record correspondant au type de la ressource. Nous définissons cette variable locale par son handle:

var "NouvEcranHndl": screenRecHndl;
Nous devons ensuite créer le handle lui-même, au moyen de la fonction *NewHandle*:
NouvEcranHndl:= ScreenRecHndl(newhandle(35, gmymemoryID +$300, attrLocked, nil);
La longueur du handle est 35, car l'écran, tout neuf, n'a pour l'instant aucune case attachée à sa liste. Par la suite, la longueur de la ressource changera, au fur et à mesure qu'on ajoutera ou supprimera des cases. *gmymemoryID+$300* est notre *userID*, c'est-à-dire un numéro d'identification particulier pour ce bloc de mémoire. Le handle est créé verrouillé (*attrLocked*), par prudence.
On assigne ensuite aux divers champs de ce record les valeurs souhaitées, en utilisant l'instruction Pascal *with NouvEcranHndl^ ^ do....* Quand cette opération est terminée, le contenu du record doit être exactement conforme à celui de la ressource à créer.
Nous pouvons maintenant appeler ResMgr pour la suite des opérations. Nous devons connaître le numéro d'ID de la ressource à créer. Nous l'avons déjà demandé à ResMgr, au cours de la procédure de dialogue qui a précédé. Dans la procédure *AjouterEcran*, vous trouvez:
IDNouvelEcran:= UniqueResourceID(0,$1002)
(1er paramètre = 0, car nous commençons à numéroter à partir de 1.)
Nous pouvons alors confier la nouvelle ressource à ResMgr qui la répertorie dans son carnet d'adresses (la ResMap, en mémoire):
AddResource(Handle(NouvelEcranHndl, $0000, $1002, IDnouvelEcran);
Le premier paramètre est l'attribut: mis à $0000, il laissera possible toute modification ultérieure. Le second est le type de la ressource. Le troisième est l'ID. C'est fini, ce handle ne nous appartient plus.
Nous pourrions maintenant recopier la ressource en mémoire avec:
WriteResource($1002, IDnouvelEcran);
Mais nous préférons utiliser *UpDateResource:*
UpDateResourceFile(gResFileID);
gResFileID est l'identifiant du fichier de ressources de l'application, obtenu dès sa mise en route.
Nous ne libérons pas notre handle de 35 octets: il appartient au ResMgr. Au moment où l'application quittera, ResMgr libérera, s'il ne l'a pas déjà fait, tous les blocs en mémoire dont il a la gestion, après avoir vérifié que tout est bien écrit dans le disque.

Modification d'une ressource existante

Le processus sera similaire. Reportez-vous un peu plus loin dans le source, à la procédure *ModifierLEcran*. Nous allons cette fois "emprunter" la ressource au ResMgr pour la modifier, puis nous la tamponnerons nous-mêmes avant de la lui rendre.
Nous déclarons d'abord la variable locale *lEcranHndl*; comme la ressource existe, soit en mémoire, soit dans le disque, nous n'avons pas besoin de préparer nous-même un bloc de mémoire avec *NewHandle*. Nous prendrons la ressource là où ResMgr nous la déposera:
lecranhndl:= screenrechndl(loadresource($1002,numEcran);
Notez comment, par un *cast* habile, nous avons déguisé ce banal bloc de mémoire en une structure de données bien précise.
Nous pouvons alors modifier le record, en assignant des valeurs aux paramètres dans le record *lecranhndl^ ^:* on va changer les deux éléments qui peuvent être modifiés, le titre et la couleur de l'écran, en prenant les valeurs telles qu'elles ressortent du dialogue précédent.
On indique le changement par la fonction *MarkResourceChange*, avec comme premier paramètre *true*, signifiant "il y a eu changement". On fait enfin appel, comme précédemment, à *UpdateResourceFile*. ResMgr récupère alors son bien.
Notre "coup de tampon" consiste en fait en une modification de l'un des bits de l'"attribut" de la ressource, qui ne se trouve pas dans la ressource elle-même, mais dans le descriptif appelé *ResourceReferenceRecord (ResRefRec)* qui résume tout ce qu'il faut savoir sur cette ressource: type, ID, adresse, attribut, longueur et adresse du handle si la ressource existe en mémoire. Tous les *ResRefRec* sont classés dans le carnet d'adresses (la ResMap), et constamment remis dans le bon ordre (par type, puis par ID) par le ResMgr.
La fonction *DetachResource* a pour effet de mettre un *nil* à la place du handle dans le *ResRefRec*. ResMgr se désintéresse alors complètement de ce handle, ce qui permet à l'application de s'en occuper pour ses propres besoins. Si vous désirez faire rentrer une ressource ainsi "détachée" au bercail, il suffit d'appeler *AddResource*, si toutefois vous avez pris la précaution de veiller à ce que la ressource ne porte pas la mention "changed". Sinon, ResMgr n'en voudra plus !

En parcourant le source de *SelectSys*, vous trouverez d'autres méthodes de création ou de modification de ressources.
Les curieux pourront également comparer la méthode que j'ai trouvée pour que l'application puisse modifier ses propres ressources, avec celle de Bernard Fournier dans ToolBox-Mag 4, pages 48-49. Nous arrivons à un résultat analogue, après des souffrances communes...

[NDLR: voir aussi la note technique Apple 83, et la contribution de Philippe Manet dans ce numéro].

Voilà. Je n'ai fait qu'effleurer certains sujets, et laissé dans l'ombre des parties des sources de ***Sélect***. Si certains points retiennent votre attention ou vous posent problème, n'hésitez pas à demander des explications. Ce sera avec plaisir.

■

Revue soft :

Bataille de chars sur Apple II GS

Bernard Fournier

La société **Stratégic Simulation Group** (SSG) vient de publier son dernier logiciel de simulation de guerre: **Panzer Battles**. Le scénario se déroule sur le Front Russe et oppose les Alliés aux puissances de l'Axe dans un affrontement de blindés.
Cette société australienne dont ToolBox-Mag vous a déjà parlé, spécialiste des wargames depuis une décennie, fort connue sur Apple II, avait déjà publié deux logiciels sur GS: *Reach for the Stars* et *Halls of Montezuma*; sont également en préparation *Battles of Africa* contant l'épopée de Rommel dans les sables du désert, ainsi que *Golds of the Americas* qui retrace les aventures des conquistadores. Ces deux logiciels étant annoncés pour le printemps, il ne devraient plus tarder à être disponibles.

Revenons à **Panzer Battles**. Le jeu se compose de deux disquettes: une contenant le programme, et l'autre six scénarios prêts à l'emploi. En lançant le jeu, on arrive sur un dialogue permettant de choisir le mode de gestion: ordinateur ou joueur humain (on peut faire s'affronter deux joueurs, un joueur contre la machine, voire la machine contre elle-même). Une fois ces préliminaires accomplis, on entre dans le jeu proprement dit.

Première surprise: **on peut créer ses propres scénarios**. C'est là à mon avis le point fort de ce jeu: la possibilité pour les joueurs de créer leur terrain de manœuvre, les forces en présence avec leurs caractéristiques, les conditions climatiques (on peut également utiliser un des six scénarios fournis sur la deuxième disquette).
Cette élaboration se fait tout simplement en disposant des icônes sur une carte (dont on peut régler la dimension, afin d'avoir une surface de jeu plus ou moins grande). Chaque icône représente de façon très réaliste une unité combattante, un engin mécanisé, une position statégique, un élément de paysage, une voie de communication ou un obstacle naturel. En double-cliquant sur l'icône, on peut même la redessiner afin de se constituer un terrain de bataille original.
Une fois la carte dessinée, on va définir les caractéristiques de chaque unité: potentiel de déplacement, force de combat, conditions climatiques,etc. Bien entendu, on sauvegarde le scénario à tout moment.

Les phases de jeu sont très simples à jouer: les règles sont claires, et on ne se perd pas dans une documentation touffue et architechnique.

Les concepteurs du jeu ont réussi l'exploit de produire **un jeu simple d'emploi, avec un scénario évolutif et personnalisable**; mais attention, sous cette apparente facilité se cache un programme complexe régissant **des milliers de paramètres** donnant au jeu un réalisme saisissant.

Puis vient la phase de jeu: celle-ci se déroule très classiquement, chacun jouant à son tour (vivement un wargame en temps réel !). On déplace ses unités, on leur affecte des ordres et on observe les résultats.
Toutes les commandes se font à l'aide d'icônes permettant de définir des opérations de ravitaillement, de manœuvre, de repli, d'attaque, de parachutages, de bombardements, etc.

Ce jeu est d'une **conception tout à fait nouvelle pour le GS**, et la simplicité de sa mise en œuvre, ainsi que la richesse de ses possibilités, devraient en passionner plus d'un.

Stategic Simulation Group:

P.O. Box 261
Drummoyne, NSW. 2047
Australie

■

Lynx : Un accessoire anti-plantages

Patrick Desnoues

Enfin un Nda qui ne sert à rien... d'autre qu'à vous éviter les plantages causés par la fermeture d'un NDA, lorsque plusieurs accessoires sont actifs et que vous ne respectez pas leur ordre de fermeture. (Voir ToolBox Mag n°1 page 17).

Particularités : La fenêtre de Lynx disparaît chaque fois qu'un autre NDA est actif, vous évitant de le fermer intempestivement. Fermez les autres accessoires, et elle réapparaît aussitôt.

Condition d'utilisation : Si vous avez l'intention d'ouvrir plusieurs accessoires, il est impératif d'ouvrir Lynx avant tous les autres, ensuite vous n'aurez plus à vous préoccuper de leur ordre de fermeture, Lynx ne pouvant se fermer qu'après tous les autres.

Plan du programme Lynx v0.1

Note : Les instructions commençant par *_xxxx* concernent des appels à la boîte à outils de l'Apple II GS. Les instructions commençant par ➢ concernent des appels à des sous-programmes situés en général dans le fichier SP.

Module Main

OpenLynxDA
- Initialisation de la mémoire (*_MMStartup*)
- Réservation pagezéro (*_NewHandle*)
- Initialisation des outils (➢ InitTools) ➢ Gestion erreur
- Récupère le nom du Nda (*_LGetPathname2*)
- Ouverture du fichier Ressource (*_OpenResourceFile*) ➢ Gestion erreur
- Création de la fenêtre (*_OpenWindow [_NewWindow2]*)
- Crée la fenêtre de type Nda (*_SetSysWindow*)
- Charge les contrôles de la fenêtre (*_NewControl2*)
- Initialise le curseur (*_InitCursor*)

Fin

CloseLynxDA
- Ferme la fenêtre Nda (*_CloseWindow*)
- Ferme le fichier ressource (*_CloseResourceFile*)
- Fermeture des outils (➢ ShutDownToolAp)
- Libère la mémoire (*_DisposeAll*)
- Efface le numéro du Nda (*_DeleteID*)
- Ferme le gestionnaire de mémoire (*_MMShutDown*)

Fin

ActionLynxDA

TraiteEvent
- Si à l'intérieur de la fenêtre: présentation du programme (➢ APropos)

ActionEvent
- Gestion des événements concernant le Nda (*_TaskMasterDA*)

DoUpdate
- Redessine la fenêtre
- Dessin des contrôles (*_DrawControls*)

Fin

APropos
- Présentation du programme, ainsi qu'une petite aide.
- Affiche la fenêtre (*_NewWindow2*)
- Affiche les contrôles (*_NewControl2*)
- Gestion des événements (*_TaskMaster*)
- Si clic en-dehors de la barre de défilement ou appui sur une touche: fermeture de la fenêtre (*_CloseWindow*)

ActionRun
- Examine régulièrement s'il y a une fenêtre de type NDA (*_FrontWindow / GetWKind*).
  - Si oui: cache la fenêtre (*_HideWindow*)
  - Si non: boucle jusqu'à la dernière fenêtre
- Affiche la fenêtre (*_ShowWindow*)

InitLynxDA
- Ne fait rien

OpenWindow
- Sous-programme pour la création des deux fenêtres

Les variables du programme

Voir le source complet sur la disquette... ■

Versions de ressources et ressources de versions

Bernard Fournier

Dans mes précédents articles, j'avais insisté sur la nécessité de posséder un certain nombre d'ouvrages lorsqu'on veut aborder sérieusement la programmation. Aujourd'hui, nous allons voir que pour indispensable qu'elle soit, cette documentation est insuffisante: il est impératif de se procurer les Notes Techniques éditées par le support technique Apple (et disponibles par exemple sur les disquettes A2 Central).
Que trouve-t-on dans ces fameuses notes techniques? Tout simplement ce que l'on ne trouve nulle part ailleurs ! Trève de plaisanterie, ces notes techniques sont faites pour préciser certains points obscurs de programmation, corriger quelques bugs des outils ou du système, attirer l'attention sur des points précis, mettre en garde contre de mauvaises habitudes de programmation, informer de la disponibilité de nouvelles fonctions, etc. Elles corrigent aussi les erreurs des documentations existantes, dont elles constituent une mise à jour permanente.
En ce qui concerne les ressources, deux points ont attiré mon attention: l'emploi de ressources dans des accessoires de bureau (type NDA) et l'implémentation de deux ressources systèmes: rVersion et rComment.
Vous trouverez dans le numéro 6 mes commentaires sur accessoires et ressources. Parlons aujourd'hui des versions en ressources, plutôt d'une nouvelle version des ressources, les ressources-versions... enfin bon, vous allez comprendre.

C'est par les notes techniques que j'ai découvert que le Resource Manager avait deux nouvelles ressources implémentées et non documentées à ce jour: les *ressources $8029 et $802A* qui sont respectivement nommées *rVersion et rComment*.
Ces deux ressources permettent de stocker le numéro de version de l'application, son nom et le copyright, ainsi que quelques lignes de commentaires.
Ces informations seront affichables par l'option Informations d'une prochaine version du Finder (c'est écrit tel quel en toutes lettres dans la note technique 76 datée de Janvier 91: donc patience, un nouveau système est en gestation...).

La ressource rVersion ($8029)

Il est impératif de donner à cette ressource un identifiant égal à 1. Elle est constituée comme suit:

```
rVersion (1) {
LongInt;    /* n° de version
word;       /* octets réservés
pString;    /* nom de l'application
pString     /* copyright
}
```

La ressource rComment ($802A)

Il est également impératif de donner à cette ressource un identifiant égal à 1.
Elle est constituée comme suit:

```
rComment (1) {
string;
}
```

Ces deux ressources ont été implémentées par mes soins dans le **fichier Types.Rez.Aux** à placer dans le sous-catalogue /R.Libraries si vous utilisez l'éditeur/compilateur APW-REZ (le fichier Types.Rez.Aux est sur la disquette de ce numéro). Il suffit alors d'ajouter une ligne *#Include "R.Libraries/Types.rez.aux"* dans vos sources de ressources pour disposer de *rVersion* et *rComment* avec Rez.

Un éditeur de versions

Ces deux ressources m'ont semblé être un bon exemple pour se créer un éditeur personnel: j'ai donc écrit un petit utilitaire (vous le ▶

trouverez sur la disquette de ce numéro) qui permet non seulement de lire les ressources $8029 et $802A, mais aussi de les créer et de les modifier. Le source de cet utilitaire est en TML Pascal et ne présente pas de difficulté majeure.
Le seul point délicat est sûrement le **décodage/codage du numéro de version.**

Le numéro de version est stocké dans un entier long: soit 4 octets. Ces 4 octets (32 bits) sont constitués comme suit, selon les termes du Bureau Politique:

bits 32-25 : MajorVersion (2 chiffres codés en BCD)
24-21 : MinorVersion (1 chiffre codé en BCD)
19-16 : BugVersion (1 chiffre codé en BCD)
15-13 : Stage (1 chiffre codé en binaire)
12- 8 : toujours à 0
7- 0 : ReleaseVersion (2 chiffres codés en BCD)

les valeurs autorisées de *Stage* sont:
001 : développement
010 : alpha
011 : bèta
100 : final
110 : release
(*la valeur 101 est interdite*)

Qu'est ce qu'une valeur codée BCD ? Ces initiales signifient *Binaire Codé Décimal*. Concrètement, cela signifie que bien qu'exprimé en valeur hexadécimale, un nombre codé BCD ne sera composé que de chiffres décimaux (0...9). Ainsi on aura par exemple $29 ou $4 ou $12 mais jamais $1E ou $B.
Dans le numéro de version, on pourra par exemple avoir la valeur suivante: $25048006. Cette valeur se décompose ainsi:
$25 : MajorVersion
$0 : MinorVersion
$4 : BugVersion
$80 : Final stage
$06 : ReleaseVersion
soit en clair: version 25.0.4f.6

Le décodage de cette valeur va se faire comme suit: on va isoler chaque composant de cet entier long en lui faisant subir des décalages de bits vers la droite, de manière à récupérer une valeur apurée entière.
Par exemple, pour récupérer *MajorVersion* il faut éliminer les 24 bits de droite et ensuite décaler 24 fois vers la droite les bits 25 à 32 de manière à disposer d'une valeur entière représentant *MajorVersion* (ou plus simplement travailler sur la partie haute et ne décaler que 8 fois). On procède de même pour les autres valeurs (seul *Stage* est analysé différemment puisque la valeur est binaire).
Ensuite on procède à des tests de validité: d'abord si la version est de niveau *Développement*, il ne peut y avoir de valeur *Release* (consignes du BP); ensuite tenons compte du fait que des petits malins peuvent avoir traficoté le numéro de version avec un éditeur, sans se soucier de la validité de leurs œuvres; en définitive on a extrait chaque composant du numéro de version.

Note: les *BAnd* servent à purger la valeur des bits indésirables. Par exemple, pour *BugVersion*, seuls les bits 19 à 16 nous intéressent. On va donc éliminer les autres par un *BAnd $F* sur la valeur haute du *LongInt*.

Pour le recodage, on procède à l'envers. On décale à gauche la valeur de manière à positionner les bits en place, puis on fait un *BOr* avec la valeur *LongInt* stockant le résultat. En consultant le listing, les choses seront plus claires.

Remarque: la ressource *rComment* est une string de longueur quelconque (bien qu'il soit conseillé de n'employer qu'une centaine de caractères). La ressource qui la stocke ne peut pas être vide (*une ressource ne doit jamais être vide*), il faut donc tester si l'utilisateur a tapé au moins un caractère et si ce n'est pas le cas, forcer le contenu avec au minimum un espace.
Tout cela se fait avec des *SetHandleSize* appropriés et des déplacements d'octets depuis la variable de stockage vers le handle de la ressource à sauver. Ces déplacements sont la solution la plus simple que j'ai trouvée, bien qu'elle ne soit guère élégante pour récupérer un seul octet par exemple...

Dernière heure:
Au moment où je conclus cet article, je reçois la **mise à jour 1.2.4 de Genesys**. O stupeur, ils ont implémenté *rVersion* et *rComment* ! Mon utilitaire serait-il déjà dépassé?
On dirait que non, car la solution adoptée par Genesys ne me satisfait guère. Pour l'édition de *rVersion*, ils ont employé des LineEdit peu pratiques pour entrer 255 caractères; je préfère nettement mes TextEdit plus commodes.
Et pour finir, ils ont purement et simplement *oublié d'implémenter l'édition de rComment*. La ressource est créée, mais vide et sans possibilité de l'éditer !
SSSI aurait-il eu vent de la sortie imminente de l'utilitaire *EdVers* dans ToolBox-Mag et, pressé par le temps pour nous griller au poteau, oublié du coup la moitié du source? Hum...

■

Introduction à la programmation de l'Apple II GS:

Présentation

Jean Destelle

Présentation de la Rédaction

Faut-il ou non faire, dans ToolBox-Mag, une introduction systématique à la programmation de l'Apple II GS? Il y a des livres pour cela.
Oui, mais ils sont en anglais, et Apple-France a fermé son service développeurs GS. L'excellent Curcio n'est pas à jour: or, les innovations d'Apple (ressources, etc) mènent à changer considérablement (en les simplifiant) les habitudes de programmation du GS.
On peut faire 36.000 choses avec un ordinateur sans jamais le programmer. Mais on peut **aussi** le programmer. Or, il se trouve qu'au point où sont parvenus aujourd'hui les outils Apple et autres, **l'Apple II GS se trouve être sans doute la meilleure machine sur le marché pour apprendre la programmation**.
Beaucoup d'entre vous demandent, à juste titre, à ToolBox-Mag de fournir lui-même les outils permettant de comprendre ce qu'ils ne comprennent pas actuellement dans ses colonnes. L'initiation à la **programmation** du GS est bien la suite logique de l'initiation à l'**utilisation** du GS.
Donc, c'est décidé: **ToolBox-Mag démarre avec ce numéro une série d'introduction à la programmation de l'Apple II GS**, qui durera sur plusieurs numéros (vous êtes tous abonnés, ça tombe bien).

Mais il faut appeler un chat un chat: Mr Destelle se propose de vous faire parcourir, sans les détours inutiles, les chemins qu'il a parcourus. Quelqu'un qui se propose ainsi de **partager son savoir en le présentant méthodiquement**, cela s'appelle un **prof**. Et ce qu'il écrit, cela s'appelle des **cours**. Eh oui.
Quand l'ami Uhrich nous dit que "C Facile", je sais ce que pensent la plupart d'entre vous: "c'est facile... pour toi!".
Ils n'ont pas vraiment tort: quand on veut apprendre à programmer, c'est qu'on veut faire **plus** que cliquer la souris. C'est qu'on veut savoir et maîtriser ce qu'il y a **derrière** cette facilité. Ne disons donc pas trop que c'est facile: **ce qui est facile n'est généralement pas intéressant**. Or **programmer le GS, c'est intéressant**.
Ceci dit, il y a une vérité dans cette idée que c'est facile: à savoir que **maintenant, c'est devenu facile**. Avec le 5.02, les ressources, TML Pascal II, Genesys, ResDoctor, StartUpTools, TaskMaster, etc, **maintenant**, ça va. Mais ne cachons pas qu'**avant**, ça a été très dur. Autrement dit: tous ceux qui ont attendu un bon moment avant de s'initier à la programmation du GS... ont eu raison: **c'est le moment où jamais de s'y mettre**.

L'Apple II GS est un Apple II: c'est donc **une machine ouverte, conçue pour être programmée par ses utilisateurs eux-mêmes**, et pas seulement par des bureaucraties gigantesques genre MicroSoft ou Claris. Il s'agit bien de continuer sur le GS la tradition de Wozniak, l'homme qui avait mis dans le domaine public le source du moniteur de l'Apple II.
Mais c'est **un Apple II d'aujourd'hui**, 16 bits, Graphiques/Sons, etc: plus question de le programmer sérieusement en Applesoft. Mr Destelle nous a donc concocté une transition en douceur du Basic au Pascal, aussi bien qu'une initiation au noyau de base de la Toolbox du GS.
Dans les prochains mois, ToolBox-Mag vous présentera également une autre manière, encore plus facile mais moins polyvalente, pour programmer le GS: HyperCard GS.

Vous qui nous écrivez en commençant par **"moi qui ne programmerai jamais en C..."**, nous vous répondons qu'il ne faut jamais dire "Fontaine, je ne boirai pas de ton eau". De même que c'est avec des civils qu'on fait des militaires, c'est avec des non-programmeurs qu'on fait des programmeurs, avec des lecteurs de ToolBox-Mag qu'on fait des auteurs.
Vous n'avez rien à perdre à essayer: écrire un programme GS, c'est moins difficile et beaucoup moins dangereux (mais pas moins long, c'est vrai) que d'optimiser son disque dur. Mais **c'est beaucoup, beaucoup plus intéressant!**

JYB

Présentation de Jean Destelle.

Ne cherchez pas dans mes articles un exposé dogmatique et définitif sur ce que doit être la programmation du GS. Je me propose seulement de vous raconter, à ma façon, ce que j'ai fait et "qui marche", sans prétendre du tout que c'est ainsi qu'on doit faire obligatoirement.

Mon expérience de programmeur Apple II m'a enseigné une leçon majeure: **quand ça marche, c'est bon**! N'y revenir que le moins possible (le mieux est l'ennemi du bien), mettre soigneusement de côté pour le réutiliser plus tard tout ce qui a donné satisfaction.

Deuxième fruit de l'expérience: **écrivons des programmes faciles à lire**. Il m'est arrivé de devoir modifier l'un ou l'autre de mes gros programmes de gestion en Basic qui tournent allègrement depuis presque dix ans. Par exemple quand le législateur oblige à des modifications démentielles des données dans la paye, c'est arrivé il y a deux ans. Et ça recommence !
Alors, on s'aperçoit que le plus important, c'est la clarté, à la lecture, d'un programme. Qu'il soit plus long, éventuellement plus compliqué, ce n'est pas grave, pourvu que ce soit logique et clair, le plus clair possible.

Avec le GS, il m'a fallu quitter l'Applesoft pour entrer dans le monde du 16 bits. Je suis donc "entré en Pascal".
Dans ce langage, j'ai retrouvé le pas-à-pas détaillé et logique auquel le basic nous avait habitués. Mais j'ai aussi découvert, par la pratique, progressivement, la création des "procédures", des "fonctions" avec leurs "paramètres", tout en m'exerçant à employer les outils Apple.

Car le GS tout nu, sans ses outils, ce n'est pas grand chose. Apple nous a livré, avec cette machine, un véritable trésor (pas très accessible, comme tout trésor): la ToolBox. Il y a des choses précieuses dans tous les coins. A nous de les trouver et de les mettre en valeur.

Cette conversion au Pascal a été le début d'une exploration passionnante. Vite, il m'est apparu nécessaire de constituer une "trousse d'outils" personnelle. D'abord quelques outils simples, puis ensuite, des "units" plus copieuses...
Que celà reste enfoui dans mes disquettes, sans autre emploi que mon propre GS, me faisait un peu de peine... ToolBox Mag est venu à point. Pour **partager le savoir**, sans doute: mais surtout pour **partager le plaisir.**

On ne s'attardera pas, dans cette série, à étudier de façon exhaustive l'ensemble des fonctions de tel ou tel outil de la ToolBox du GS.
Il y en a plus de 1200 je crois, dont beaucoup sont en fait à usage interne de la ToolBox elle-même. Pour le programmeur, le merveilleux serviteur "TaskMaster" en a rendu inutile le plus grand nombre.
Il y a peut-être à peine une trentaine qui servent quotidiennement, et sur ceux-là, nous reviendrons. Tous les autres, on ne les rencontre que beaucoup plus rarement.

Ne considérez surtout pas que programmer le GS avec la ToolBox, ce soit difficile! **Ce qui est difficile, c'est de programmer le GS sans la ToolBox**. Alors là, c'est difficile, c'est sûr.

Depuis le système 5.0, avec les ressources, TaskMaster qui fait tout à notre place, un bon Pascal, comme le TML II, accompagné d'un bon éditeur de ressources "étudié pour", comme ResDoctor, programmer le GS, c'est devenu "trivial", comme disent les pédants de l'informatique. A la condition d'accepter de jouer le jeu. C'est-à-dire d'utiliser ce qui nous est offert, en respectant les règles.

Vous vous simplifierez la vie si vous construisez, petit à petit, vos outils, vos units, vos ressources de base. Une fois que ça marche, on les met en conserve et on s'en ressert longtemps. Ensuite, on copie, on colle, on compile, et ça va très, très vite.
ToolBox-Mag est fait pour que nous puissions nous faire tous profiter, les uns les autres, de ces gagne-temps. Ne laissons pas passer l'occasion.

Vous lirez, me dit-on, ma prose sur une plage, en vacances, entre la baignade et le pastis.
Rendez-vous compte: d'autres galèrent comme des forçats à "compiler en DBase IV" pour s'offrir enfin des vacances sans programmation. Vous et moi, avec nos GS, nous nous intéressons à la programmation... pour le plaisir! Allez, amusez-vous bien, et à bientôt.

JD

Introduction à la programmation de l'Apple II GS
Du Basic au Pascal : premiers pas

Jean Destelle

Dans ce qui suit, nous supposerons que vous connaissez à peu près convenablement l'Applesoft de l'Apple II, et que vous êtes capable, à la lecture d'un programme Applesoft, d'en comprendre les instructions et d'en suivre le déroulement avec ses branchements, ses boucles, ses assignations de variables.
Cela suffira pour apprendre à programmer l'Apple II GS, même si vous ne vous sentez pas capable de composer vous-même un programme un peu compliqué: car le Pascal est un langage plus humain, plus clair que le Basic, vous vous y mettrez aisément.
Nous apprendrons d'abord à comprendre le Pascal à la lecture. Pour bien suivre notre progression, il sera souhaitable que vous disposiez de **TML Pascal II**, avec lequel les exemples seront traités. Avec cette application, qui porte le nom maintenant de **Complete Pascal**, est livré un manuel comportant une initiation au langage Pascal très succincte mais fort bien faite, écrite en anglais, mais très accessible. Néanmoins, si vous préférez l'**Orca Pascal** de ByteWorks, vous n'aurez que peu de modifications à faire.
Dès que nous parlerons de ressources, l'utilitaire **ResDoctor**, créé spécialement pour compléter TML Pascal II, vous rendra les plus grands services, en vous permettant d'aller visiter les fichiers de ressources, et de les éditer.

Pourquoi le Pascal ?

Le GS a visiblement été développé, comme le Mac, dans l'esprit du langage Pascal. Les fonctions, les procédures des outils de la Toolbox ont été définies pour utiliser les protocoles Pascal. Il y a donc une **unité indiscutable entre le langage Pascal et le système**.
Malheureusement, la mode du langage C étant intervenue entre temps, Apple nous a doté de très peu de documentation technique en Pascal, alors que tous les outils du Mac sont décrits dans les livres de référence en employant ce langage ! Il y a toutefois un très bon ouvrage édité par Apple, qui traite de la programmation du GS à travers trois langages en parallèle : l'Assembleur, le C et le Pascal. Il s'appelle **Programmer's Introduction to the Apple II GS**. Ce livre est fourni avec un disque comportant les sources des exemples développés. Je ne saurais trop vous le recommander, car il contient à peu près tout ce qu'il n'est pas facile d'approcher dans le GS.

Programmer le GS en *Assembleur* est une approche intéressante mais difficile. L'emploi des outils oblige soit à employer un grand nombre d'instructions à chaque appel, soit à utiliser des "macros" de plus en plus importantes, ce qui finit par créer un langage évolué qui n'a plus de l'assembleur que le nom! On doit reconstituer, artificiellement une façon d'écrire des sources par petits morceaux qui se calque sur le système des procédures du Pascal. Les deux seuls gros avantages en faveur de l'assembleur, c'est d'abord qu'utilisé abondamment par les professionnels, il a été doté par ceux-ci de perfectionnements pas encore tous accessibles au Pascal; et ensuite qu'il permet d'optimiser la place occupée en mémoire par le programme, et sa rapidité d'exécution.

Pourquoi pas le *Basic*? La réponse est facile. Nous pouvons continuer de programmer en Basic Applesoft pour utiliser notre GS en mode Apple II. Associé en cas de besoin à de brèves routies en binaire, il permet de faire des programmes de tous genres extrêmement performants. Mais l'Applesoft ne peut pas utiliser les outils de la Toolbox qui fonctionne, elle en mode "natif", spécifiquement GS.
De plus, il y a des limitations de principe dans le Basic, telles que toute modification finit, en fin de compte, par transformer le Basic en un genre de Pascal déguisé qui fonctionnerait en mode interprété, ce qui n'est pas forcément fameux. Il existe plusieurs Basic transformés pour le GS qui fonctionnent correctement. Mais si vous désirez les utiliser, vous aurez autant de travail d'apprentissage que pour vous mettre au Pascal.

Passons maintenant à l'examen des **principales différences entre Basic et Pascal**. Elles tiennent plus à des questions de principes qu'à des questions de langage (au sens grammatical du terme).

Compilateur et interpréteur

L'Applesoft est un langage *interprété*, c'est-à-dire que le programme une fois écrit est conservé sous cette forme. Il est, lorsqu'on le lance, relu, mot par mot, et traduit par l'interpréteur pour être transformé en code machine. C'est souvent assez lent. D'autant que certaines opérations comme le traitement des chaînes de caractères n'ont pas toujours été très bien conçues, et demandent énormément d'instructions.

Le Pascal est généralement un langage *compilé*. Vous ne pouvez pas directement utiliser le programme que vous avez écrit et qu'on appelle *le source* (au masculin, je ne m'y ferais jamais, il s'agit du "programme source"). Il doit être compilé, c'est-à-dire transformé en code par le "compilateur" du Pascal. Le compilateur fournit alors *le programme objet*, qui est le code directement utilisable par la machine.
L'exécution du programme compilé est évidemment beaucoup plus rapide que celle d'un langage interprété, et souvent à peine moins rapide que s'il avait été écrit en Assembleur.
Comme tout compilateur, les diverses versions du Pascal associent au moment de la compilation, au code de votre programme, un morceau plus ou moins important de code, le "runtime", qui comporte des outils de compilation. Cela fait que le code obtenu est souvent nettement plus long que s'il avait été écrit en Assembleur.
Une fois compilé, le programme peut être lancé, comme vous l'auriez fait pour un programme Basic. La compilation elle-même ne prend pas ▶

beaucoup de temps: de quelques secondes, pour un programme très bref, à quelques minutes pour un source d'une centaine de Ko. En moyenne: une trentaine de secondes.
Un autre avantage de la compilation est qu'elle est accompagnée d'un contrôle grammatical de ce que vous avez écrit. Le compilateur vous signale les erreurs de langage et vous pouvez les corriger.
Signalons encore une différence d'ordre pratique: en Applesoft, votre ordinateur se trouve en mode "interpréteur" et vous pouvez lui donner des ordres directs, sans passer par un programme, ce qui vous permettait parfois de vérifier l'écriture d'une instruction. Lorsque vous travaillez sur un programme Pascal, vous vous trouvez dans un "Editeur" de textes, tout à fait identique à celui d'un traitement de textes. Vous n'avez donc plus accès directement aux commandes de l'ordinateur. Il vous faudra passer par le moniteur si vous désirez actionner directement votre GS, ou bien employer un accessoire de bureau.

Structure des programmes

Le Basic nous oblige à entrer nos instructions sous formes de lignes numérotées à la queue-leu-leu. Les branchements par *GOTO* , *GOSUB* ou *ON X GOTO* se font obligatoirement à une ligne donnée, c'est-à-dire à une adresse précise du programme. On ne s'en rend pas bien compte quand on est très habitué au Basic, mais c'est un carcan très raide, qui finit par alourdir l'écriture et ne permet pas bien l'emploi de sous-programmes.
Le Pascal n'est pas corseté de cette façon. Il n'a presque pas d'obligation de structure, et permet beaucoup de fantaisies d'emploi. La structure est le fait de la **décision du programmeur** et non une **obligation dûe au langage**. Il permet la création de routines (procédures ou fonctions) qu'on peut appeler par leur nom, en leur transmettant tous les paramètres souhaitables. De ce fait, la programmation en Pascal s'oriente naturellement vers une **structure modulaire**, et vous en verrez très vite l'immense intérêt pratique.

La manipulation des données

Une différence importante se trouve dans la façon dont on peut définir des variables, des constantes, ou d'autres types de données. Le Basic ne connait que très peu de types de données: essentiellement les nombres entiers, les nombres réels, les caractères et les chaînes de caractères, et quelques formes de tableaux utilisant ces types. De plus, les noms de variables ne sont reconnus que sur deux caractères, ce qui limite considérablement leur emploi.

Dans le Pascal, il est possible de créer à chaque instant n'importe quel type de donnée, logique, numérique, ou autre. Il suffit d'en définir le "type". Les données, variables, ou constantes, sont de plus identifiables par des noms d'une longueur quelconque (jusqu'à 255 caractères malgré tout).
Il existe dans le langage de base un grand nombre de types déjà définis, mais on peut en ajouter autant qu'on veut. C'est du reste ce qui se passe avec la Toolbox, qui nécessite la création de structures de données nombreuses et différentes, chacune adaptée à un problème particulier.
Le Pascal reconnaît sous le nom de *Record*, une structure très souple dans laquelle on peut inclure des tas d'éléments différents, de types, de longueurs variées, que l'on pourra ensuite manipuler sous le simple nom d'une variable. Cette faculté du Pascal lui permet de s'adapter instantanément à n'importe quel problème, qu'il soit mathématique, statistique, sociologique, administratif, comptable, etc. Il devient en quelque sorte un langage universellement utilisable. C'est pourquoi on l'enseigne beaucoup à ceux qui se destinent à l'informatique.
En contrepartie, comme Pascal peut tout faire, il devient nécessaire de lui préciser à chaque fois ce qu'on veut. Et cela peut paraître souvent un peu lourd. On ne peut plus parachuter une variable au milieu d'une instruction comme on le fait en Basic. Il faut toujours avoir "déclaré" auparavant son existence et son type. C'est la rançon de cette universalité.

Différences pratiques

Entre un programme en Basic et un programme en Pascal, il y a en fait assez peu de différences à la lecture. On n'est pas perdu quand on passe de l'un à l'autre. Beaucoup de noms d'instructions sont les mêmes.
La disposition des lignes est un peu différente: on utilise les grandes facilités d'édition que l'on a actuellement pour donner une allure aussi parlante que possible aux programmes Pascal, de façon à mieux voir les boucles, les branchements. On rencontre dans le Pascal beaucoup plus de subdivisions pour définir des blocs à l'intérieur du programme:
• la séparation des routines entre elles et leur dénomination: *procédures* et *fonctions*;
• l'emploi des mots *begin* et *end* en début et fin de chaque bloc d'instructions;
• certains caractères de ponctuation sont employés différemment.
On trouve aussi beaucoup plus d'informations annexes, de commentaires. En effet, comme ces programmes seront compilés, cela n'agrandira pas les fichiers des programmes, puisque le compilateur sautera les commentaires. On peut donc transformer un programme Pascal en véritable roman-fleuve. C'est un peu ce que nous ferons dans les sources écrits à votre intention.

Passons maintenant rapidement en revue **les différences de syntaxe**, nous réservant d'y revenir à l'occasion de l'explication de certaines formes d'instructions.

Les différences typographiques

Elles ont trait essentiellement à la "ponctuation" et à l'utilisation de quelques caractères particuliers. Citons rapidement, en ordre dispersé:
• Plus de numéros en tête de ligne. Exceptionnellement, on pourra le faire, mais ce n'est pas l'habitude. Si on met un numéro de ligne, cela s'appellera un *label*, et on devra le déclarer au préalable. La "ligne", en Pascal, n'a plus de sens réel. Le compilateur ne tiendra pas compte d'un retour à la ligne, même à l'intérieur d'une instruction. Cela permet d'insérer de longs commentaires, et de diviser certaines instructions complexes en morceaux expliqués séparément.
• Comme séparateur entre les instructions, le point-virgule remplace les deux points du Basic. Contrairement au Basic, le Pascal demande presque ▶

toujours un point-virgule en fin de chaque instruction. Il y a quelques rares cas où on ne doit pas mettre de point-virgule: notamment avant le mot *else* dans l'instruction *if then ... else*. Le signe : (deux points) a un tout autre sens en Pascal. On s'en sert pour définir les types des variables que l'on déclare. Exemple:

var maVariable: integer;

• Comme en Basic, on peut mettre des espaces un peu partout (sauf dans les mots réservés).

• La virgule a un sens précis, et sert de séparateur, pour séparer par exemple deux paramètres dans l'écriture d'une fonction ou d'une procédure [*Moveto (10,20)* est analogue au Basic *Htab 10: Vtab 20*], ou bien pour séparer plusieurs des éléments constitutifs d'un ensemble (*set*), d'une énumération ou d'une déclaration: *var a,b,c,d : integer;* est une déclaration de plusieurs variables d'un même type.

• La parenthèse utilisée en opérations mathématiques ou logiques s'emploie absolument comme en Basic: *y := ((3 *x)+2) / 14;* s'écrit de la même façon dans les deux langages. Elle s'emploie aussi pour contenir des paramètres, voir exemple ci-dessus. Mais elle ne s'emploie pas pour définir un tableau. On utilise alors les crochets: *Tableau [i,j]* en Pascal correspond à *TABLEAU (I,J)* en Basic.

• Les signes mathématiques s'emploient de la même façon, à l'exception du signe = (égale). Pour le signe = le Pascal lève l'ambigüité du Basic. Quand il s'agit d'une égalité logique ou mathématique, on emploie le signe = seul. On emploie aussi = seul pour donner une valeur à une constante: *Const kResourceID = 3;*
Mais quand il s'agit d'une "assignation", c'est-à-dire d'une instruction qui fixe une valeur à une variable, Pascal emploie les deux signes :=, sans espace entre les deux: *if x = 3 then y := x + 4;* attention, cela sera le sujet de fréquentes erreurs, mais le compilateur saura généralement les reconnaître.

• Le point d'interrogation n'a aucun sens particulier en Pascal. Il ne veut pas dire *Print*.

• Pour désigner une chaîne, Pascal emploie l'apostrophe au lieu des guillemets du Basic. Il s'agit de l'apostrophe "ordinaire", celle que JYB appelle l'apostrophe "dactylographique". *Ch := 'Le Texte';* équivaut au Basic *CH$ = "Le Texte"*:
Remarque: On peut donc mettre une virgule dans une chaîne. En revanche, si on veut mettre une apostrophe, il faudra la doubler. *Ch := 'C''est ainsi !'* donnera *C'est ainsi !*

• Pour inclure des remarques, Basic ne disposait que de l'instruction *REM*. En Pascal, on dispose de deux moyens différents. Le plus courant, et le plus facile à utiliser dans les éditeurs, est la cause de beaucoup d'ennuis car il utilise, en caractères américains, des accolades { et }, qui se traduisent, en clavier français, par é et è. Cette difficulté est du même ordre que celle que nous rencontrons avec les crochets [et], qui sont aussi des caractères américains spéciaux, et se traduisent en français par les signes ° et §. TML Pascal II attend un GS en clavier américain avec une ImageWriter configurée en américain, sinon il ne parviendra pas à imprimer (voir ToolBox-Mag 3, page 30).
Le deuxième moyen passe aussi bien en américain qu'en français: il emploie les deux signes (* pour indiquer le début d'un commentaire , et *) pour en indiquer la fin. Comme il est également beaucoup plus facile à distinguer, c'est ce que nous recommanderons. Exemple: *(*ceci est une remarque*)* .

• Le point est utilisé pour indiquer l'appartenance d'un élément à un record donné. Nous y reviendrons en temps utile. Par exemple, *monRecord.Flag* désigne l'élément *Flag* dans le record *monRecord*. Le point peut aussi signaler la fin du programme, s'il est placé après *end*.

• Quelques caractères un peu particuliers comme l'arobas @ et le circonflexe ^ sont utilisés par le Pascal pour définir des emplois de pointeurs: *a := lePointeur^;* signifie "a égale le contenu de l'adresse *lePointeur*". Dans *cePointeur := @LaChaine;*, *cePointeur* prend la valeur de l'adresse de la variable *laChaine*.

• Pascal ne distingue pas les majuscules et les minuscules dans ses mots réservés et dans les noms d'identificateurs que vous créez.

Les différences de construction

Globalement, un programme Pascal ne s'écrit pas comme un programme Applesoft. Ces différences sont d'ordres très variés. Nous ne parlerons que de celles qui intéressent le déroulement général d'un programme normal pour le GS. Prenez, afin de suivre ce qui suit, un des sources Pascal à votre disposition, pour reconnaître les divers éléments.

❶ Le Nom du programme doit figurer en tête du programme, sous la forme du mot *program* suivi du nom. Exemple: *program NomProgramme;*
Le nom est limité à 12 caractères, sans point ni signe. On peut utiliser des chiffres, mais pas au début.

❷ On doit trouver ensuite la directive *Uses* destinée à indiquer les Units dont le programme va avoir besoin. Ensuite, la liste des Units, séparées par des virgules, et terminées par un point.
Les *Units* sont en fait des modules particuliers de programme, préétablis, que vous pouvez réutiliser en fonction des besoins. TML Pascal II fournit toute une série d'Units standard qui servent d'intermédiaires avec la Toolbox du GS.

❸ Le programme doit comporter ensuite, si il en existe, les déclarations des types, des constantes et des variables "globales".
Nous y reviendrons, mais sachez dès maintenant qu'une *variable globale* peut être utilisée n'importe où dans le programme. Par opposition, on peut créer des *variables locales* à l'intérieur d'une procédure donnée, qui ne servent qu'à l'intérieur de cette procédure. Par exemple, la variable locale utilisée comme index pour compter les tours effectués par une boucle. Quand la boucle a terminé, on n'en a plus besoin. En Applesoft, toutes les variables sont "globales".

❹ Ensuite commence l'écriture du programme proprement dit, qui peut comporter divers modules qu'on appelle des *procédures* ou des *fonctions*. Vous trouverez donc:

• Obligatoirement un "main program", c'est-à-dire le programme principal. Vous le trouverez toujours à la fin du source, parce qu'il se termine par "*end.*", avec un point et non un point-virgule, qui est le signal de fin de compilation.
Ce "main program" n'est pas précédé d'un titre: procédure, ou fonction. Il débute simplement ▶

par un *begin* et se termine par *end*. Il comporte les instructions principales du déroulement du programme, et les appels aux diverses procédures qui sont détaillées dans le corps du programme.
NB: Il ne paraît pas logique de placer à la fin ce qui est primordial, mais *c'est* logique: en Pascal, il n'est possible d'appeler une procédure que si elle a été précédemment déclarée, il est donc justifié de placer à la fin le programme principal, qui utilise ces procédures.
Avant le *main program*, dans le corps du programme, vous trouverez les diverses *procédures* et *fonctions*, déclarées dans un ordre permettant l'appel de procédures antérieures. Donc toutes les procédures secondaires seront en général placées avant les procédures les plus importantes.

Déclarations de procédures et fonctions

Une *procédure* est en fait une "routine" c'est-à-dire un sous-programme que l'on peut "appeler" par son nom. Au lieu de faire un *goto* ou un *gosub* comme en Basic, il suffit de citer le nom de la procédure pour en provoquer son déroulement. On peut transmettre à une procédure des données appelées alors *paramètres d'entrée*. La procédure peut mettre à votre disposition, après son exécution, des *paramètres de sortie*. Vous commencez à sentir là un net progrès par rapport au Basic.
Mais ce n'est pas tout: une *fonction* est une procédure dont on peut ensuite utiliser le nom à la place d'une simple variable, qui prend alors comme valeur la valeur du résultat de la fonction. Comme en mathématiques, somme toute, mais avec beaucoup plus de possibilités.

Chaque procédure ou fonction comporte d'abord une "déclaration" de son nom, suivi des paramètres qu'elle utilise, en entrée et en sortie.
Premier exemple, une procédure simple, sans paramètre: *procedure MainEventLoop;* cette procédure n'a pas de paramètre (pas de parenthèse après le nom). Elle sera donc exécutée comme telle, et ne fournit aucun paramètre à la sortie.
Deuxième exemple, une fonction avec paramètre. Supposons que notre programme comporte une fonction que nous avons créée pour mettre au format une chaîne de caractères, par exemple pour lui donner une longueur constante de 30 caractères. Nous l'avons déclarée ainsi:
function FormatString(lastring: str32): str32;
Notez que *function*, mot anglais, s'écrit avec un 'u' et non un 'o'. Cette fonction admet en entrée un paramètre *lastring* qui est de type *str32* (c'est-à-dire une chaîne limitée à 32 caractères), et en sortie elle fournira également une autre chaîne, qui portera, comme nom de variable, le nom de cette fonction.
Dans notre programme, nous pourrons utiliser le nom de cette fonction à la place d'une variable de même type. Par exemple, nous pourrons écrire:
DrawString (FormatString('Vive le GS !'));

DrawString est une *procédure* (de l'outil GS QuickDraw) qui admet un *paramètre*, une chaîne de 255 caractères maximum, et va dessiner le texte de cette chaîne sur l'écran. Elle a été déclarée quelque part dans l'unit "QuickDraw" sous la forme:
procedure DrawString(str: str255);
On peut donc l'utiliser, directement, comme ceci:
DrawString ('Vive le GS !');
mais si nous voulons utiliser notre fonction de modification de format *FormatString*, nous écrirons:
DrawString (FormatString('Vive le GS !'));
Dans la parenthèse qui suit le nom de la procédure, nous avons utilisé directement, comme paramètre, la fonction que nous avons créée.

Remarque 1: si nous voulons pouvoir utiliser la ligne d'instructions ci-dessus, il faut:
• avoir placé l'unit "QuickDraw" dans la liste des units après *uses* en début de programme.
• avoir également mis dans cette liste l'unit "Types", dans laquelle est défini le type *str32*.
• avoir décrit la fonction *FormatString* auparavant dans le programme.
Remarque 2: vous constatez, dans cet exemple, que rien ne distingue la façon dont nous avons utilisé une procédure de la toolbox, *DrawString*, d'une fonction que nous avons créée nous-mêmes. **C'est l'un des grands intérêts du Pascal pour programmer le GS.** Il n'y a aucune différence entre le traitement des outils et celui des procédures créées par nos programmes. Seul le nom change. Attention donc à ne pas créer de procédures dont les noms seraient déjà employés par Apple! Si nous donnons des noms français, nous ne risquons pas grand-chose.

La directive forward

Il est malgré tout possible de se référer à une procédure ou une fonction écrite ultérieurement dans le programme, en faisant une déclaration préliminaire avant la procédure qui l'appelle. Il suffit d'en prévenir le compilateur au moyen de ce qu'on appelle une *directive*. Cette "directive" particulière s'écrira *Forward;* on la placera à la suite d'une recopie exacte de la déclaration de la procédure ou fonction concernée. Par exemple,
function FormatString(lastring: str32): str32; Forward;
signifie que cette fonction existe dans le programme. Le compilateur saura ainsi qu'il pourra la trouver plus loin.
Il y a aussi d'autres directives possibles pour le compilateur. Certaines se signalent simplement par le signe $ suivant immédiatement une accolade ouvrante. Elles se rapportent à des instructions particulières pour la façon de compiler, ou d'établir les liens entre segments de programme après la compilation.

A l'intérieur des procédures et fonctions

Nous retrouvons à l'intérieur de chaque procédure ou fonction une structure du même genre que celle de notre programme entier.
• Déclaration du nom: *Procedure MaProcedure;*
• Déclaration des constantes locales si elles existent: *Const …*
• Déclaration des types, s'il y en a: *Type …*
• Déclaration des variables: *Var …*
• Le mot *begin*, début de l'écriture des instructions;
• Les instructions, qui peuvent se décomposer en divers "blocs", commençant le plus souvent chacun par *begin* et finissant par *end;*. Il y a d'autres "mots-clés" que *begin* et *end* pour délimiter un bloc, par exemple *repeat* et *until*.
• Enfin *end;* marque la fin des instructions de cette procédure ou fonction. ▶

Les divers blocs peuvent s'imbriquer les uns dans les autres, et on facilite la lecture en décalant vers la droite l'écriture des instructions chaque fois qu'on entre dans un nouveau "bloc".
Voici un exemple très simple: une fonction qui calcule, arrondie à un entier, la surface d'un cercle.

```
function SurfaceDuCercle(lerayon: integer): longint;
const pi = 3.14156;
var Surface: longint;
begin
    Surface:= round(lerayon * lerayon * pi);
    SurfaceDuCercle:= Surface;
end;
```

Voici, en deuxième exemple, comment nous écrirons la fonction *FormatString* citée plus haut.

```
function FormatString(lastring: str32): str32;
const Espaces = '                              '; (*30 espaces*)
var s1: str255; (*chaîne de 255 caractères maximum*)
begin
  s1:= concat(lastring, Espaces); (*ajouter des espaces*)
  s1:= copy(s1, 1, 30); (*raccourcir à 30 caractères*)
  FormatString:= s1;  (*donner sa valeur à la fonction*)
end;
```

Nous avons utilisé deux fonctions standard du Pascal: la fonction *concat*, qui réalise la même chose que les additions de chaînes avec le signe + en Basic (ce qui n'existe pas en Pascal); et la fonction *copy*, qui s'emploie comme *MID$* en Basic.
Attention, dans un cas comme celui-ci, à ne pas choisir une variable locale *s1* trop petite pour recevoir la chaîne, qui pourra faire plus de 32 caractères, car un tel sans-gêne risquerait de perturber les données dans la pile, et il n'y a pas grand-chose de plus grave. *Str255* est le type standard de chaîne le plus utilisé.

Autre exemple: une procédure comportant des "imbrications". Cette procédure doit signaler certaines conditions par plusieurs "bips" du système, séparés par des silences, en fonction d'un nombre donné.

```
procedure MultiBip(lenombre: integer);
const normal = 1; grave = 2; tresgrave = 3;
var a,b,c, duree: integer;
begin
    duree:= 1000; (*délai d'attente = nombre de boucles*)
    if lenombre < 0 then a:= tresgrave;
    if lenombre in [0 .. 300] then a:= normal;
    if lenombre >= 300 then a = grave;
    for b:= 1 to a do
            begin
              SysBeep;          (*déclencher un Bip*)
              c:=0;
              repeat
                 a:= a;         (*on passe du temps à rien*)
                 c:= c +1;
              until c >= duree;
            end;
end;
```

Vous pouvez admirer l'inutilité pratique indiscutable de cette procédure, dont le seul but est de vous montrer comment on visualise, par des décalages, les diverses imbrications de blocs d'instructions à l'intérieur d'une procédure, et de vous faire faire connaissance avec des boucles obtenues de façon différentes. Vous ne retrouvez pas, dans la première boucle, l'habituel *Next* du Basic. Pascal ne connaît pas *Next*. Pas besoin, puisque *begin* et *end* encadrent bien le bloc à reboucler. En revanche, on a ajouté, vous l'avez remarqué, le mot *do*, qui indique le début de la boucle. Si la boucle n'avait comporté qu'une seule instruction, nous n'aurions pas en besoin de *begin ... end*.
La deuxième méthode *repeat ... until* n'existe pas en Basic. Elle est très pratique, car elle économise les instructions *begin* et *end*. Sa structure ressemble en fait beaucoup aux boucles que nous faisions avec un *goto*. Nous l'aurions écrite ainsi:

```
20  DU = 1000
.....
45  C = 0
50  A = A: C = C+1 : IF C < DU THEN GOTO 50
```

Remarquons à cette occasion combien le retour à une ligne de numéro donné est astreignant. On ne pourrait pas recopier la ligne 50 ailleurs sans la modifier. Dans le Pascal, avec des routines repérées par leurs noms, tous les morceaux de nos programmes sont "relogeables", recopiables, sans aucune adresse fixe.
Remarquez la façon nouvelle dont le Pascal permet de vérifier qu'un nombre est inclus dans un "intervalle" donné: *if leNombre in [0 .. 300]* signifie "si le nombre est dans l'intervalle 0 (inclus) à 300 (inclus)". Nous aurions pu écrire aussi: *if (lenombre >= 0) and (lenombre < 300) then a:= normal;*
Il y aurait bien d'autres façons d'écrire la procédure ci-dessus. Nous vous laissons le soin d'y réfléchir. Le Pascal propose en effet plusieurs façons différentes de résoudre le problème des boucles, et celui des comparaisons.
Remarquez simplement aujourd'hui que cette procédure comporte des déclarations de constantes et de variables qui sont des données "locales". Un paramètre *lenombre* qui est un integer (nombre entier) sera transmis à la procédure lors de son appel. Cela évite de créer une variable globale qui ne servirait qu'à cette occasion.

Intérêt des variables locales

Les constantes et les variables locales n'occuperont pas de place gênante en mémoire. Le compilateur utilisera pour les stocker momentanément la *pile*, zone de mémoire réservée pour cela. Sitôt son travail effectué, la "pile" sera vidée de ces données.
Voyez en passant l'intérêt de ce procédé: quand nous créons une variable temporaire en Basic, elle est traitée comme les autres, et va encombrer la mémoire, surtout s'il s'agit d'une chaîne de caractères, de façon quasi-définitive. On a très vite une mémoire embarbouillée de vieilleries.
Il faudra cependant veiller, en Pascal, à ce que les variables locales ne nécessitent pas un emplacement de mémoire supérieur à la dimension de cette "pile". Sans quoi les pires ennuis guettent!

Nous nous en tiendrons là pour ce premier article sur les bases du Pascal. Il contient déjà quelques éléments vous permettant de déchiffrer un peu les sources. N'hésitez pas à vous lancer: faites l'effort d'essayer de les comprendre. Vous verrez que ça n'est pas si difficile. Dans les articles qui suivront, nous examinerons certaines façons de procéder propres au Pascal, en commençant par la plus épineuse, la question des déclarations. ■

Introduction à la programmation de l'Apple II GS : La gestion de la mémoire

Jean Destelle

[Dans cet article, comme dans ceux de cette série, la programmation du GS est expliquée à l'aide d'exemples en Pascal. Toutefois, comme le vocabulaire de la ToolBox est universel, et utilisé de la même façon par les autres langages comme l'Assembleur et le C, il intéressera tout lecteur désireux de mieux comprendre la programmation du GS.]

Le gestionnaire de mémoire

Contrairement à l'Apple II, qui dispose d'une mémoire figée, le GS gère tout l'espace mémoire de façon **dynamique**, avec l'aide d'un gestionnaire très performant: **MemoryManager**. Nous n'avons pas besoin de savoir de quelle façon la mémoire est utilisée. C'est MemoryMgr qui s'en charge. Et il le fait très bien.

Toutes les données traitées par l'application (ou le système) sont rangées dans des "Blocs", espaces de mémoire. Ces blocs peuvent avoir n'importe quelle dimension, pourvu que la mémoire puisse les contenir.

Si nous désirons écrire quelque chose en mémoire, nous devons appeler MemoryMgr (par *NewHandle* ou par une autre fonction de la ToolBox) qui nous allouera un bloc déterminé en fonction de nos besoins.

Pendant le déroulement de l'application, MemoryMgr est souvent obligé de déplacer les blocs de données dans la mémoire en les répartissant au mieux, notamment si la mémoire commence à être encombrée. Il le fait soit lorsque l'application a besoin d'un autre espace, soit tout simplement quand l'un des outils de la ToolBox en demande.

Handles et pointeurs

Chaque bloc (même de quelques octets) est repéré par un "handle". Tous les handles sont regroupés dans un endroit fixe bien précis de la mémoire. Disons, pour simplifier, qu'un handle est un "pointeur" vers un autre "pointeur" qui indique lui-même où se trouve le bloc en mémoire.

Un "pointeur" est une adresse dans la mémoire, écrite de façon particulière, sur 4 octets, comme un entier long (le Memory Manager est prêt pour le 65832, il est "32-bits clean", comme on dit sur Mac, dès le départ).

Un "handle" est d'abord un pointeur, c'est-à-dire une adresse écrite en mémoire à un endroit connu. Cette adresse est celle d'une structure de données appelée "*Memory Block Record*" dont certains auteurs nous disent que nous n'avons pas besoin de la connaître. Raison de plus pour y fourrer notre nez !

Le "Memory Block Record"

C'est en fait un record de 20 octets de long, qui comporte six paramètres, et résume ce qui concerne le bloc dans la mémoire.

- Le premier paramètre, le plus important, est tout simplement l'adresse du début du bloc (le pointeur).
- Le second est un "flag" (drapeau) appelé "attributs", qui résume les conditions de déplacement et de conservation du bloc en mémoire: suivant la valeur de ce flag, le bloc sera fixe ou mobile, on pourra le "purger" (effacer le contenu) ou non, il résidera (ou non) dans un banc donné de mémoire, etc.
- Le troisième est un "ID", un numéro d'identification, qui indique à qui appartient ce bloc, c'est-à-dire le programme qui a créé le handle.
- Le quatrième est la longueur, en octets, du bloc.
- Les deux suivants sont respectivement les adresses du handle suivant et du handle précédent.

Du handle au pointeur

Quand un programmeur parle d'un handle, il désigne en fait l'adresse où le *Memory Block Record* se trouve en mémoire. Le premier paramètre de cette structure n'est autre que l'adresse actuelle du bloc concerné par ce handle. Voilà pourquoi on peut considérer, en simplifiant, qu'**un handle est un pointeur vers le pointeur du bloc de mémoire**.

Création des pointeurs

Le Pascal permet de créer des pointeurs avec la plus grande facilité. On utilise l'opérateur "@" qui se place juste devant le nom de la variable pointée.

Par exemple, définissons deux variables:

```
var   LaChaine: str255;
      LePointeur: ptr;
```

Pour obtenir que la variable *LePointeur* désigne effectivement l'adresse de *LaChaine*, il suffit d'écrire:

```
LePointeur:= @LaChaine;
```

Nous pourrions utiliser tout simplement *@lachaine* à l'intérieur d'une instruction pour désigner le pointeur lui-même. Cela revient au même. C'est ce que nous ferons lorsqu'on nous demandera un pointeur comme paramètre dans l'une des fonctions de la ToolBox.

Remarque: l'adresse fournie par *LePointeur* ne sera pas celle du premier caractère de la chaîne, mais celle du premier octet de la variable "chaîne Pascal", c'est-à-dire l'octet donnant le nombre de caractères. Si on veut pointer directement sur le premier caractère, on devra utiliser:

```
LePointeur:= @LaChaine[1]
```

Le chiffre 1, entre crochets, désigne le caractère numéro 1 de la chaîne.

▶

Création des handles

Les handles sont créés, dans les programmes, soit directement par l'un des outils de la ToolBox, soit par le programmeur.

1er exemple:

var LaResHndl : Handle;

...

LaResHndle := loadResource($8006, 2);

La fonction *LoadResource* de Resource Manager crée directement un handle, et place la ressource demandée dans le bloc créé. La ressource (de type $8006), et de numéro ID 2, sera chargée du disque dans ce bloc, à moins qu'elle n'y soit déjà, auquel cas ResMgr retourne le handle existant. Le programmeur n'a donc pas à créer auparavant un handle.

2e exemple:

Nous voulons créer nous-mêmes un Handle. Il faut d'abord le déclarer parmi les variables:

var lehandle: handle;

Il faut ensuite créer le handle au moyen de la fonction *NewHandle:*

lehandle:= Newhandle(la longueur, MemoryID, Attributs , Nil);

[Apple a vraiment la plus haute considération pour les facultés intellectuelles des programmeurs GS. Dans le Mac, la fonction NewHandle ne demande qu'un seul paramètre: la longueur du bloc, et MemoryMgr se débrouille tout seul. A nous, on nous en demande quatre, et pas des plus simples…]

Nous devons donc déterminer les quatre paramètres de la fonction NewHandle:

1: La longueur du bloc demandé (nombre d'octets);

2: Un numéro d'identification pour ce bloc, indiquant à qui il appartient, qui sera formé à partir du numéro ID de notre application, tel que MemoryMgr nous l'a donné au moment de l'ouverture des outils. Ce pourra être aussi tout simplement cet ID.

3: Un "Bit-Flag", "Attributs", définissant les conditions de résidence du bloc en mémoire. *[Voir l'article sur la Gestion des Evènements pour l'explication de ce qu'est un Bit-Flag.]*

Nous ne rentrons pas dans le détail de ces attributs maintenant. Retenons seulement qu'il suffit le plus souvent d'indiquer *attrLocked* si on veut que le bloc soit verrouillé. Ou bien tout simplement "$00" (zéro) si on veut laisser MemoryMgr se débrouiller tout seul, ce qu'il sait parfaitement faire.

4: Là où vous lisez *Nil*, on peut mettre l'adresse du bloc, si nous l'imposons à MemoryMgr, ce qui ne sera que très rarement le cas. Nous écrivons donc *Nil*. *Nil*, en TML Pascal II, désigne un pointeur zéro, de type universel. C'est-à-dire qu'il peut être employé à la place de toute variable ou paramètre d'un type pointeur quelconque.

A la suite de l'instruction *NewHandle*, si nous n'avons pas demandé la lune à MemoryMgr, il aura fait le nécessaire, et créé le bloc de mémoire en fonction de nos besoins. Ce bloc sera pour l'instant vide. Nous pourrons y faire rentrer des données au moyen de l'une ou l'autre des fonctions permettant de recopier des blocs de données en mémoire. Leur emploi est d'une grande simplicité.

Nous disposons de *HandToHand* pour recopier tout le bloc d'un handle dans un autre handle, et *PtrToHand* pour recopier un bloc de mémoire repéré par son pointeur et sa longueur.

Il sera aussi possible d'entrer des données au moyen d'instructions particulières du Pascal, permettant d'assigner des valeurs aux divers éléments d'une structure de données si le bloc est prévu pour les recevoir.

Ensuite, pour utiliser le contenu du Handle, nous pourrons employer *HandToPtr* pour recopier le contenu d'un bloc désigné par son handle vers une adresse précise (pointeur) de la mémoire.

Sans nous servir du Handle, nous pourrions recopier d'un endroit à l'autre de la mémoire, des données au moyen de *BlockMove* qui demande comme paramètres, le pointeur de départ, celui de destination, et la longueur du bloc à déplacer. Mais, compte tenu que les blocs se déplacent souvent dans la mémoire du GS, l'emploi de cette instruction ne sera pas fréquent et devra se faire avec précaution.

Il existe une autre moyen pour accéder aux données contenues dans un bloc de mémoire: *le déréférencement du handle.*

Déréférencer un handle

Quand on a besoin de lire une donnée dans un bloc en mémoire, il faut de préférence se servir du handle du bloc. Il est facile pour le programmeur de prendre l'adresse du bloc recherché en lisant ce qui est écrit à l'adresse du handle. Cela s'appelle "déréférencer" le handle.

En Pascal on écrira:

*lePointeur := leHandle^*

ce qui se lit: "lePointeur égale le contenu de leHandle". De même:

maDonnee := lePointeur^;

ce qui se lit: "maDonnée = ce qui est à l'adresse lePointeur".

Remarquez qu'on peut aussi écrire, directement:

maDonnee := leHandle^ ^;

Nous avons ainsi écrit une instruction qui permettra d'assigner à la variable *maDonnee*, la valeur de ce qui est écrit à l'adresse du bloc. C'est-à-dire, en langage plus simple, de "lire la donnée".

Attention: comme nous travaillons en Pascal, il faudra bien veiller à ce que le bloc de mémoire concerné par le handle contienne une donnée d'un "type" (au sens du Pascal) compatible avec celui de la variable *maDonnee*.

Attention encore: MemoryManager passe son temps à déplacer les blocs de mémoire pour les mettre à un endroit meilleur, et cela n'est pas sans inconvénient pour les programmeurs. Dès que nous voulons lire ou écrire dans un bloc de mémoire, il faut être sûr qu'il ne va pas bouger pendant l'opération. Nous ne pouvons travailler en toute sécurité, après avoir déréférencé un handle, que si le bloc est fixe ou si nous sommes sûrs que MemoryMgr ne le touchera pas. Comment cela? En le verrouillant.

Verrouiller et déverrouiller

Chaque bloc de mémoire peut être rendu (temporairement ou définitivement) fixe en mémoire, si on le "verrouille". On peut le faire lors de la création du handle en choisissant pour l'attribut l'option *attrlocked*. ▶

On peut aussi verrouiller et déverrouiller un handle quand on le désire pendant le déroulement du programme, en utilisant les fonctions *HLock* et *HUnLock*). Avant de déréférencer un handle, pour être sûr qu'il ne bougera pas inopinément, on verrouille le bloc (par *HLock*) avant de déréférencer, et on le déverrouille (par *HUnLock*) quand on a fini (attention à ne rien oublier).

Agrandir un bloc de mémoire

Débutants, attention: voici une source de "plantages" graves.
Au moment de son allocation, par MemoryMgr, tout bloc de mémoire, repéré par son handle, a une dimension donnée. Si notre programme écrit à l'intérieur, sans déborder: pas de problème. Mais **si, par inadvertance, on écrit quelque chose qui déborde la dimension du bloc, c'est la catastrophe!** Il y a eu des heures et des journées perdues à causes de ces distractions, car le clash ne se produira pas forcément tout de suite, mais au moment le plus inattendu. Le plantage pourra se produire par exemple si vous écrivez plus d'éléments que prévus dans un tableau de variables.

Une fonction de MemoryMgr vous permet d'ajuster la taille de votre bloc à vos besoins: c'est *SetHandleSize: SetHandleSize (la nouvelle longueur, le handle);*
N'oubliez pas de l'utiliser chaque fois que ce sera nécessaire, et si vous devez agrandir le bloc, pensez à le déverrouiller auparavant: sinon, retour à la case départ...

Gérez bien la mémoire

Soyez très vigilants, car il peut y avoir des réveils en fanfare. Un peu plus tard, ou longtemps plus tard, votre application ou un outil du système peut avoir besoin de mémoire. Aussitôt, MemoryMgr se met en branle et remue tout son stock de blocs. Si, dans l'intervalle, une erreur de programmation s'est glissée, il est possible que ça se termine très mal.
On voit ainsi des plantages qui se manifestent longtemps après l'erreur. Plusieurs minutes, ou plus. On s'imagine alors que l'erreur provient de l'une des dernières instructions avant le plantage. Rien de moins sûr !
Alors soyons prudents, **verrouillons et déverrouillons chaque fois que cela paraît utile**, mais pas sans réfléchir. Et traitons avec respect la place dont nous disposons en mémoire. N'en abusons pas ! Du temps des 48 k de l'Apple II+, nous y faisions attention. Avec plusieurs mégas, les problèmes restent entiers, car les données manipulées, les outils du systèmes, et nos propres programmes ont pris du ventre. **La plupart des défauts graves de fonctionnement des applications proviennent d'une mauvaise utilisation de la mémoire.**

[A titre anecdotique, pensez que l'on conseille aux développeurs sur Mac d'employer un utilitaire qui se charge d'intercaler, automatiquement, entre deux instructions, tout au long du source d'un programme, un appel-bidon à MemMgr. Cela le force à ranger-déranger sans arrêt. De cette façon, s'il y a un défaut, le programme plante immédiatement, et on sait à quel endroit chercher l'erreur. Ce système s'appelle un "Heap Scrambler", parce qu'il "secoue" la tranche libre de mémoire dans tous les sens. Sur GS, Apple se contente de fournir un CDA "Memory Scrambler"...]

Libérer la mémoire

Quand on n'a plus besoin d'un emplacement de mémoire, il est poli de prévenir le responsable en l'informant qu'il peut récupérer l'emplacement.
Nous n'entrerons pas ici dans le détail de l'emploi des "purges". Sachez seulement pour l'instant que nous pouvons libérer l'espace occupé par un bloc au moyen de la fonction *DisposeHandle* de MemoryMgr, en lui donnant comme paramètre le handle à supprimer.
Lorsqu'en terminant l'application, nous n'aurons plus besoin de MemoryMgr, nous le renverrons à ses autres occupations (c'est fait par la fermeture des outils), et celui-ci libérera automatiquement tous les emplacements qu'il nous avait loués.

Visiter la mémoire du GS

Pour "visiter" à tout moment, pendant le déroulement d'un programme, la mémoire du GS, il existe un outil très pratique : le CDA *Nifty-List* qui est distribué en ShareWare. *Nifty-List* vous fournit une liste détaillée de tous les handles existant actuellement en mémoire. Vous pouvez ensuite allez voir leurs contenus en utilisant le moniteur.
Vous recherchez le handle du bloc qui vous intéresse, puis à l'aide du Moniteur auquel Nifty-List vous donne accès, vous pouvez aller voir sur place dans votre bloc.
Pour repérer "votre" handle, il faut utiliser le numéro d'identification qui l'accompagne. C'est le numéro que vous lui aurez donné au moment de sa création par *NewHandle*: soit l'identifiant général de votre application, ce que nous appelons *gmemoryID* dans nos sources, soit un numéro qui en est dérivé pour précisément le retrouver plus facilement. Ce numéro pourra être: *gmemoryID* + $100, ou *gmemoryID* + $200,... et ainsi de suite.
Sous Nifty-list vous tapez "Øi" puis Return et vous obtenez un tableau des handles et des paramètres du Memory Block Record. Nifty-list vous indique, à côté de l'identifiant de l'application à laquelle appartient chaque handle, le nom de celle-ci.
Vous trouvez donc vite le numéro de votre application. Ce sera probablement quelquechose comme $1002, ou $1003,... Si vous avez donné au handle qui vous intéresse le numéro *gmemoryID* + $300, cherchez le handle dont l'identifiant porte ce numéro: 1302 (ou 1303,...).
Ensuite, vous lisez le pointeur, c'est à dire l'adresse en mémoire de votre bloc. Et vous passez dans le moniteur par "*" puis return. Et vous allez lire dans la mémoire ce qui s'y trouve. Vous pouvez imprimer au passage les données qui vous intéressent, sans perturber l'application.

Le stockage des données en mémoire: les variables numériques

Si vous voulez profiter de ces renseignements, il faut savoir comment les données des divers types sont stockées dans la mémoire.
Pour le stockage des variables numériques, il y a trois cas principaux:

▶

• **Un octet (byte):**
C'est l'élément de base. Exemple: $1F. Il représente en général un nombre de 0 à 255, mais il peut représenter, par convention, autre chose: par exemple un entier signé compris entre -128 et +127. Dans ce cas, le bit 7 (de poids le plus élevé) est à 1 si le nombre est négatif. Ou bien il peut être le code d'un caractère ASCII. Le moniteur vous donne la valeur en hexa et la représentation Ascii.

• **Un entier (word, integer):**
C'est l'élément de stockage en mémoire le plus utilisé par le GS. Il est stocké sur deux octets (16 bits). Il représente en général un nombre entier signé, compris entre -32768 et +32767.
Par exemple, la valeur hexadécimale $0215 se décompose en deux octets: un octet de poids fort ($02 qui vaut 2 x 256 = 512), et un octet de poids faible ($15 qui vaut 16 + 5 = 21). La valeur de ce nombre est donc 512 + 21 = 533. Comme pour l'octet, c'est le bit de poids le plus élevé qui sert à déterminer le signe.
Attention, Apple II: *en mémoire, les deux octets d'un entier sont stockés dans l'ordre inverse de l'ordre naturel; on trouve l'octet de poids faible d'abord, l'octet de poids fort ensuite.*

• **Un entier long (Long, ou LongInteger):**
Stocké sur 4 octets, il représente un nombre entier signé compris entre - 2 147 483 648 et + 2 147 483 647.
Le processeur du GS est très bien équipé pour manipuler ces nombres de 4 octets. C'est une nécessité, car l'adressage d'une mémoire importante nécessite des pointeurs (les adresses) sur plus de deux octets.
En fait chaque pointeur utilise trois octets. Mais on rajoute un quatrième octet de valeur nulle, pour manipuler cette donnée de la même façon qu'un entier long.
Dans ce cas, encore, vous retrouverez, dans la mémoire, les octets dans l'ordre inverse de leur ordre naturel. Un pointeur représentant l'adresse **E1/32F4** (adresse de l'octet numéro $32F4 dans le banc de mémoire E1) sera écrit en mémoire sous la forme: **F4 32 E1 00** .

Le stockage des données structurées en mémoire

Une fois qu'a été définie leur structure (soit par la ToolBox, soit par la définition d'un type en Pascal, soit par la réservation des emplacements en mémoire par l'Assembleur), les données structurées sont stockées en mémoire comme une suite d'éléments dans l'ordre déterminé par leur définition.
L'ordre des éléments n'est pas inversé. L'inversion d'ordre citée plus haut pour les entiers et les entiers longs ne concerne que les données numériques élémentaires.

Quelques exemples:
• une **Chaîne Pascal**: chaîne de caractères. Elle débute par un octet: le nombre des caractères qui suivent. Ensuite figure chacun des octets représentant un caractère (sa longueur est de ce fait limitée à 255 caractères).
• une **Chaîne C**: chaîne de caractères. Pas de nombre de caractères au début, mais un octet de valeur 0 à la fin. Pas de longueur limite.
• une **Chaîne GS**: utilisée par GS/OS pour manipuler des chemins d'accès très longs. Au début: un entier sur deux octets indiquant le nombre de caractères, puis les caractères eux-mêmes. Dans ce cas, les deux premiers octets sont stockés en mémoire dans l'ordre inverse, puisqu'ils font partie d'un même nombre entier.
• un **tableau de nombres entiers**: les nombres entiers, chacun sur deux octets, sont stockés à la suite l'un de l'autre, dans l'ordre de leurs index, à la position relative correspondant à l'index. Chaque nombre occupe deux octets, stockés en mémoire dans l'ordre inversé.

Il en est de même pour toute structure ("record" ou "template", ou fichier plus important obéissant à une organisation interne précise). Il en est donc ainsi des ressources, qui ne sont que des données structurées particulières. Et cela ne rend pas toujours facile leur interprétation directe en mémoire.

L'interprétation de données dans la mémoire n'est pas toujours facile par simple lecture sur l'écran. Si rien de structuré n'apparaît, vous êtes peut-être dans un morceau de "code" c'est-à-dire des instructions en langage machine. Vous pouvez alors "désassembler" pour les examiner, soit avec le Moniteur, soit avec Nifty-List. De toutes façons, n'hésitez pas à imprimer le bloc qui vous intéresse. C'est infiniment plus facile de contrôler un bloc de mémoire sur le papier, un crayon à la main, que sur l'écran.

Nous en resterons là pour cette fois. Ce qu'il faut surtout retenir, pour ne pas se perdre ensuite dans l'interprétation des sources, c'est qu'*un nombre écrit dans un fichier source ne peut pas être retrouvé sous la même forme quand il aura été écrit dans la mémoire.* On le retrouvera avec l'ordre de ses octets inversés.

Et surtout, c'est qu'il faut *traiter l'emploi de la mémoire du GS avec beaucoup d'égards et de respect pour son gestionnaire, le MemoryMgr*: sinon, gare au coup de pied de la Mule du Pape! ■

Travail d'artiste

Suite à une erreur de manipulation de fichier dont la responsabilité incombe uniquement au Rédacteur en Chef de ToolBox-Mag, le nom de l'auteur du graphisme de l'animation de la disquette ToolBox-Mag 4 est passé à la trappe.
Dominique Delay est l'auteur de l'animation, de la partie **programmation** de la présentation. Dessiner, c'est autre chose: c'est un **travail d'artiste**.
L'artiste en question s'appelle **Nicolas Bergeret**, et vous pouvez admirer encore une fois sur la disquette de ce numéro 5 avec quel talent il contribue à la partie Graphique/Son de ToolBox-Mag.

JYB

GS News

Eric Weyland

Ce qu'il faut savoir

CinemaWare: ce n'est peut être pas terminé

Tout le monde a en mémoire les merveilles que nous a offert Cinemaware sur nos GS: *Rocket Ranger, Sinbad,Three Stooges*, etc. Tout le monde se souvient aussi que Cinemaware a cessé toute activité de distribution, et qu'elle abandonnait ses différents programmes de jeux à la société Electronic Arts pour que celle-ci en assure la diffusion.

Coup de théâtre ! Au moment de signer l'accord commercial, les purs et durs de Cinemaware ont fait volte face devant Electronic Arts, en déclarant qu'ils refusaient de vendre leurs merveilles à des "têtes de ...". Cette délicieuse soirée s'est terminée au bar du coin, d'où les joyeux drilles ont été expulsés... Espérons qu'ils vont reprendre leurs esprits rapidement !

6.0 = 7.0 ?

Après *SynthLab, HyperCard GS* et le *système 5.04*, les programmeurs d'Apple s'attaquent maintenant au système suivant. Andy Nicholas (l'auteur de la série des compacteurs 8 bits et 16 bits *Shrinkit*) est désormais un employé de la firme de Cupertino. Son rôle est de maintenir et surtout d'améliorer le Finder.

A cet effet deux formats viennent d'être définis par Apple: *Version Resources* et *Comment Resources*, cela nous donnera bientôt la possibilité d'associer une ligne d'informations aux différentes icônes que manipule le Finder. C'est en fait une fenêtre supplémentaire qui s'affichera lorsque l'on demandera un *"Get Info"*; l'utilisateur pourra y écrire ce que bon lui semble: cela fera très Macintosh. Bien entendu, ToolBox-Mag, toujours à l'avant-garde, vous aura déjà permis de parsemer tous vos programmes de ces ressources.

En travaillant (faut-il dire en galérant?) sur Mac, je n'ai pu trouver qu'une supériorité et une seule de Mac/OS, qu'il serait intelligent de porter sur GS/OS: le Standard File du Mac est capable de formater une disquette au vol; celui du GS se contente de voir que la disquette n'est pas formatée et la recrache séance tenante. Le système 6 intègrera-t-il cette possibilité?

En tout cas, ce nouveau système (on peut dès maintenant parler de 6.0) que prépare Apple gérera aussi ce que l'on appelle les **menus hiérarchiques**: il va y avoir maintenant des menus déroulants à l'intérieur des menus de la barre principale de menu. Là aussi ce n'est pas très original: cela existe sur Macintosh.

D'après nos agents à Cupertino, ce système 6.0 aurait **un FST pour lire les disques au format Macintosh.** Les utilisateurs de GS ayant accès à un Mac vont être comblés. En effet, il sera possible de partager les données se trouvant sur un disque dur connecté sur les deux ordinateurs, même sans AppleShare.

Cela confirme une fois de plus que c'est du côté logiciel qu'Apple fera des efforts sur l'Apple II GS. Côté hard, pas de surprise à venir: il n'existe pas de micro-processeur 65816 rapide et fiable, donc pas de GS rapide pour le moment. On parle cependant d'une **carte d'interface pour les lecteurs 3,5 1,4 Mégas** Apple, histoire d'unifier le matériel et de permettre les conversions.

Comme parallèlement le système 7 du Mac est en train de copier GS/OS, dans quelque temps, quand nous en serons au système 7 sur Mac et au système 6 sur GS, il faudra certains jours de bons yeux pour savoir quelle machine est derrière notre clavier ADB...

Offre Apple: profitez-en !

"Pourquoi donc Apple France nous propose-t-il de reprendre 5.000 F une unité centrale d'Apple II en cas d'achat d'un Mac LC? Serait-ce qu'Apple veut laisser tomber le GS?", telle est la question que se posent quelques inquiets parmi nos lecteurs.

En fait, la réponse est ▶

toute simple: **les gens d'Apple gagnent leur vie en vendant des ordinateurs**. Ils doivent vendre les produits qu'ils ont, pas ceux qu'ils n'ont pas, et ils essaient naturellement de les vendre d'abord à ceux qui sont déjà leurs clients.
Or, Apple rencontre avec ses clients existants un classique casse-tête commercial: comme votre ordinateur (le GS) marche, que vous en êtes content, **vous n'avez plus rien du tout à acheter à Apple** (sauf ses nouveaux systèmes, mais ça n'est pas rentable)! D'autant que pour la plupart des extensions possibles de cet ordinateur (cartes mémoire, disques durs, accélérateurs, scanners, etc), Apple est parfaitement dépassé par les produits non-Apple. Pour vendre ce qu'on a à des clients qui n'en ont pas besoin, il faut essayer de semer un peu d'inquiétude, de déstabiliser un peu ces clients pour les faire bouger: mais sans aller jusqu'à les envoyer chez les concurrents en disant trop de mal de ses propres produits. *Dur, dur, le commerce!*

La solution de ce casse-tête commercial classique est également classique: c'est de faire une offre commerciale particulière à ces clients. **Si Apple vendait des saucissons, il ferait des offres spéciales aux propriétaires d'Apple II de reprise contre des saucissons!** Pour cela, il essaierait de laisser planer un peu d'inquiétude, de laisser entendre que l'Apple II serait "technologiquement dépassé" par... le saucisson. **Les saucissons qu'Apple veut vendre, en ce moment, ce sont des Macs LC, voilà tout.**

Bien sûr, me dîtes-vous, vous préféreriez les vrais saucissons: **un saucisson, au moins, est compatible Apple II GS** (sauf en slot 3). De fait, le marché normal visé par les Macs bas de gamme, c'est celui des gens qui n'ont pas encore d'ordinateur, qui ne peuvent donc pas comparer avec le GS. Sacrifier un Apple II haut de gamme et bien plus de 10.000 F, pour n'obtenir qu'une machine de bas de gamme (LC, Low Cost, veut dire "à bas prix"), **incompatible GS, incompatible avec HyperCard Couleur**, cela ne ressemble guère à une bonne affaire, je vous l'accorde.
Je vous accorde aussi qu'il y a anguille sous roche: **pourquoi l'offre d'Apple est-elle valable seulement pour le LC**, et pas pour toute la gamme Mac à partir du LC, **pour le SI** par exemple? La seule réponse possible, c'est qu'**il y a des saucissons plus difficiles à vendre que d'autres...**
Mais ce n'est pas une raison pour être inquiet, ou pour s'énerver contre Apple. A une proposition commerciale, il faut **réagir commercialement**: cherchons ce que nous pouvons y gagner, et **négocions**.

Certains l'ont déjà compris, qui font la chasse, aux Puces de Montreuil, aux Unités Centrales de II Plus. **Pour tout acheteur potentiel de Mac LC comme pour tout concessionnaire Apple, une UC d'Apple II 8 bits vaut 5.000 F. Intéressant, non?**
D'autant que ce concessionnaire n'a pas que des produits incompatibles GS à vendre, il a aussi par exemple d'excellentes LaserWriter PostScript, parfaitement compatibles GS, des disques durs, etc. Comment votre concessionnaire va se débrouiller pour revendre à Apple votre antiquité en panne, ça n'est pas votre affaire: puisqu'il s'agit d'une offre commerciale, négocions...
Et si votre concessionnaire ne marche pas dans cette négociation, vous pouvez lui en proposer une autre: **pour lui, toutes les unités centrales d'Apple II se valent**. Il n'a donc pas de raison, surtout si vous lui achetez un petit quelque chose, un moniteur couleur GS par exemple, de refuser de vous faire un échange gratuit de votre UC de II Plus contre l'UC d'Apple II GS que lui a apportée, juste avant, un acheteur de LC pas dans le coup.
Attention, cependant, il est très probable que celui qui a apporté l'UC de GS n'était précisément pas un ignorant, puisqu'il avait un GS: **le GS en question risque fort d'être, disons... extrêmement dépouillé** (plus d'alimentation, VGC et Roms antédiluviens ou disparus, carte-mère en panne...). Je connais des malins qui ont refilé à Apple leur UC de GS en panne, revendu immédiatement leur Mac LC non déballé, et racheté aussitôt avec la différence une UC de GS d'occasion, gagnant une réparation de GS et un ou deux mégas de Ram gratuits!
Ils ont tout-à-fait raison, c'est cela le commerce! Apple savait bien ce qu'il susciterait avec cette offre, et il a décidé de l'accepter pour vendre ses Macs bas de gamme. Ceci dit, si vous tombez sur ce genre d'UC de GS, sachez qu'il vous en coûtera le tarif d'un échange de carte-mère pour avoir un vrai GS en échange de votre UC de II Plus: **c'est encore une très très bonne affaire!**

Donc c'est promis, juré, ce n'est pas un poisson: pour tous ceux qui ont un minimum le sens du commerce et le goût de la négociation, **l'offre d'Apple signifie en fait que la mise à jour de tout Apple II en Apple II GS est gratuite en France jusqu'au 30 Juin. Profitez-en, et faites-en profiter vos amis!**

Convention Apple II à Beauvais

Le 15 juin 1991, Mémoire Vive organise à Beauvais la **"Convention nationale de l'Apple II"**. Pour tout renseignement concernant cette manifestation:
Mémoire Vive - c/o Monsieur J. C. Andrieux - 65, Résidence Jeanne Hachette - 60000 Beauvais.

Captain Blood recherche compatibilité

Captain Blood, un jeu d'aventure spatiale édité par Mind-Scape ne fonctionne pas sur Apple II GS Rom 03. L'éditeur ne prévoit pas de mise à jour. Si un de nos lecteurs à réussi faire fonctionner ce jeu sur Rom 03, qu'il nous contacte.

Alimentations fragiles

Apple USA a reconnu qu'il y a parfois des problèmes d'alimentation avec les Apple II GS équipés de carte mère Rom 03. Certains blocs d'alimentation ne tiennent pas le choc, on s'en aperçoit seulement quand on remplit les slots avec des cartes gourmandes.
Les alimentations fautives sont celles fabriquées par la société Dyna Comp; on peut facilement les identifier car leur numéro de série débute par "I...". Heureusement, Dyna Comp a réglé le problème. Les alimentations adaptées ont sur le dessus une petite pastille rouge collée.
Bien que la garantie normale Apple ne soit que d'une année, Apple a étendu le remplacement d'une alimentation de ce genre jusqu'au 31 janvier 1992, car il s'agit du cas typique du "vice caché".
Même si votre GS Roms 03 marche, jetez donc un coup d'œil sur son alimentation: en février, il sera trop tard...

L'autre 65C816

La nouvelle console de jeu 16 bits de chez Nintendo, le "*Super-Famicom*", est équipé d'un micro-processeur 65C816. Je connais une excellente machine pour le programmer.
N'est-ce pas John Brooks?

Le hard

Zip Chip GS plus rapide

Le modèle 1600 de la Zip Chip GS de Zip Technologies (Zip Chip GSX: le plus cher) aura tout prochainement la possibilité d'être modifié et poussé à 10 Mhz: Zip Technologies vient d'en faire l'annonce aux Etats-Unis; en revanche, rien au sujet de l'incompatibilité AppleTalk.

Coprocesseur GS Æ

Outre le développement futur d'une carte d'interface pour disque dur Vulcan plus rapide, Applied Engineering travaille sur une série de coprocesseurs: la bête aurait pour nom "The Borg" et d'après un responsable de chez Æ "make Macintosh owners rip their spleens out with envy !". A suivre...

Pour les hackers: ProDev

Le debugger de ProDev est une carte d'interruption et de débugage pour Apple II GS. Enfin! Un véritable rêve pour les programmeurs fanatiques du langage machine et de l'hexadécimal, comme pour les déplombeurs et autres "hackers".
Le programme livré avec la carte permet d'examiner vos programmes et vos variables à n'importe quel moment. Vous pouvez suivre en mode pas à pas l'exécution des programmes, examiner le contenu de la pile et de tous les registres internes du 65C816. Seule une carte hard peut permettre de faire cela absolument à tout moment, à n'importe quel niveau d'interruption.

Le soft: les jeux

Mc GEE grandit

Voici de nouvelles aventures pour Mc Gee qui n'est plus un bébé. Retrouvez le dans *Mc Gee at the fun fair*. Toujours l'excellente qualité des programmes de Lawrence Productions.

Le soft: les utilitaires

Vitesse: Contours

Dans une récente interview qu'il a accordée à la revue GS+, le Président de la société Vitesse, Jim Carson, a dévoilé ce que serait le prochain produit de Vitesse.
Il s'agit d'un programme qui permettra d'utiliser des fontes vectorielles sur divers types d'imprimantes dont celles reconnues par Harmonie. D'après Jim Carson, les résultats devraient être sensiblement comparables à ceux donnés par une imprimante PostScript. Le but est donner des capacité proches de PostScript aux imprimantes qui ne sont pas PostScript; un pari qui semble difficile à relever...
Contours, c'est le nom de ce futur programme, existera en deux versions: une version utilisateur (Contours Basic) comprenant entre 20 et 25 polices de caractères vectorielles, et une version "power user" (Contours Editor) pour éditer ce type de polices.
L'expérience des fontes True Type d'Apple pour le Mac nous permet de préciser les choses: ce genre de logiciel, qui se limite aux fontes, n'est pas un langage complet de description de page, ce n'est donc pas vraiment un concurrent de PostScript. Ce qu'il peut rem- ▶

placer, c'est un programme comme Adobe Type Manager du Mac: il permet de donner une qualité proche des fontes Adobe à l'impression et à l'écran sur des machines sans PostScript. Mais pour les caractères seulement: la gestion des fontes, ce n'est que 20% de PostScript. Pour le reste, PostScript reste le standard.

Dans cette même interview, le Président de Vitesse a annoncé que les logiciels édités par sa compagnie seraient mis à jour dans le courant de l'année. Ce n'est pas un mal lorsque l'on pense aux problèmes de fiabilité des utilitaires disque de chez Vitesse... Le logiciel du scanner à main Quickie (déjà excellent) devrait lui aussi être amélioré avant la fin de l'année.

Les projets de Seven Hills

Dans un autre numéro de GS+, un autre défenseur de l'Apple II GS, Dave Hecker, Président de Seven Hills Sofware, a dévoilé quelques informations. La *version 1.1 de Graphic Writer III* est en cours de développement: elle devrait corriger les trop nombreux bugs de la version précédente.
Le développement de ce logiciel est assez compliqué. En effet, la programmation n'est pas faite par Seven Hills Software, mais par Datapak Software, pour qui l'Apple II GS n'est pas une grande priorité. Après la réalisation de la version 1.1, c'est Seven Hills Software qui devrait reprendre le flambeau du codage, et faire de Graphic Writer III un programme de PAO fiable et performant. Une nouveauté importante dans la version 1.1: la possibilité de tracer des traits fins.
Disk Access, *Font Factory*, *Independance* et *Super-Convert* sont des produits suivis et qui seront mis à jour par Seven Hills Software.

DeskPack Volume 2 (SSSI)

SSSI est l'éditeur de l'indispensable Genesys, l'éditeur de ressources. Leur prochain produit sera une série d'accessoires de bureau de grande qualité. Il s'agit de NDAs: un traitement de texte, une calculatrice, une base de données, un utilitaire de macro-commandes, un utilitaire de gestion de fichiers, un super presse-papier, un calendrier avec calepin, et un accessoire permettant d'installer d'autres accessoires.
Les accessoires vraiment intéressants sont bien sûr le NDA base de données, le NDA utilitaire de macro-commandes et le NDA DA Mover (pour charger d'autres accessoires), qui pour l'instant n'existent qu'en différentes versions non commerciales. De toute façon cette série d'accessoires va vite devenir indispensable.

Orca: nouvelles versions

Tous les langages de programmation de la série des Orca viennent d'être mis à jour par ByteWorks.
Dans *Orca/M* (l'assembleur), le Shell et les Utilitaires ont été modifiés, MacGen a été réécrit, il est maintenant plus rapide.
Orca/Disassembler est maintenant capable de désassembler des fichiers ressources (version 1.2).
Orca/Pascal en est à la version 1.3; *Orca/C* à la version 1.2; Prizm, l'environnement graphique d'Orca, à la version 1.1.
Entre autres choses, *la gymnastique de François Uhrich et Bernard Fournier pour écrire des Inits en Orca/C et en Orca/Pascal dans notre numéro 4 est devenue inutile.* ByteWorks a entendu nos gémissements: un **pragma** RTL tout simple permet désormais d'écrire des Inits sans aucun problème.
Ces corrections devraient contenter les utilisateurs avertis de ces différents langages qui rencontraient des problèmes dans les utilisations avancées.
Cela devrait aussi rassurer les personnes ayant des inquiétudes sur l'avenir du développement sur Apple II GS. Cet ordinateur reste de loin la meilleure machine pour apprendre à programmer dans une diversité de langages.

Encore un nouveau Prosel

Prosel 16, ce must de l'Apple II GS de Glen Bredon, est en perpétuelle évolution, et va changer de mode de diffusion. La dernière mouture contient un éditeur de texte (c'est celui de Merlin 16), l'utilitaire de back-up permet l'utilisation des lecteurs de disquettes haute densité 1.6 Mo (ceux de chez Æ - voir le numéro précédent de ToolBox Mag).
Glen Bredon vous donne maintenant la possibilité de personnaliser votre ProSel 16. En effet, un utilitaire permet d'éliminer certains modules de l'énorme fichier Start qui commençait à avoir un peu trop d'embonpoint: on peut supprimer la calculatrice "polonaise", le calepin électronique, l'éditeur de texte et diverses options qui ne sont pas vitales. Ce "nettoyage" ramène ainsi le fichier Start à une taille décente.
La prochaine version de ProSel 16 sera accompagnée d'une documentation américaine sur papier. Le fichier ASCII de documentation est donc supprimé; ce n'est pas vraiment un mal, car il devenait vraiment difficile de s'y retrouver en raison de toutes les mises à jour qu'il contenait...

HyperMusiques

The HyperStuff Collection, de Triad Venture, est un ensemble de plus de 30 musiques ▶

pour SynthLab. Plusieurs genres musicaux sont représentés: des chants patriotiques, des marches militaires, des chants italiens, des chants irlandais, des chansons enfantines et des chants de Noël, etc.
Le package est livré avec des *commandes externes pour HyperCard et HyperStudio* permettant de jouer les musiques SynthLab à partir de vos piles en quelques lignes de programme (HyperTalk pour HyperCard). Notez qu'**HyperStuff Collection est livré avec le Tool 035**, ce qui vous permettra de faire tourner pour pas cher les programmes de Jean-Pierre Charpentier dans ToolBox-Mag.

Rose 16

L'ami Fournier s'étant chargé des deux autres, je me dois de vous toucher un mot de l'éditeur de textes Rose 16, écrit par Randy Brandt et publié par Jem Software. Lui aussi est conçu comme un éditeur de texte pouvant remplacer celui d'APW.
Il peut avoir deux fichiers en mémoire; les scrollings sont rapides; on peut fabriquer des macro-commandes; attention il est moins complet qu'Edit 16 et MaxEd. Il se présente sous forme de commandes EXE pour APW et Orca.
Randy Brandt étant un des auteurs d'Appleworks-8, on se doute que Rose ressemble au traitement de textes d'Appleworks...

Clip Vision: des images

L'Hypermédia se déchaîne sur le GS. ClipVision, de... ClipVision, c'est une série d'images à utiliser dans des programmes de PAO ou hypermédia.

Clip Art Volume 2

Encore des dessins, mais de chez Roger Wagner cette fois. Le package est livré avec de nombreux fonds et d'idées de boutons...

Clip Sounds

Clip Sounds, de ClipVision, est une série de sons à utiliser dans vos piles. Les thèmes suivants sont abordés: machines, musiques, voix, bruits de la nature... Le package est composé de 10 disquettes 3.5"! Le son, c'est encombrant.

Le soft: freeware et shareware

FKT Graphics

Vous êtes amateur de mathématiques? Ce programme européen (en anglais avec une documentation en allemand) est pour vous. Ce shareware permet de tracer toute fonction mathématique au format QuickDraw ou BitMap. Les graphes obtenus peuvent être sauvegardés au format PNT. Plusieurs fonctions utilisant chacune une couleur peuvent être tracées à l'écran. La précision après la virgule est paramétrable, les coordonnées peuvent être cartésiennes ou polaires, l'impression peut se faire sur ImageWriter ou table traçante HP. Le programme a été écrit en Orca/Pascal par Dirk Fröhling.

1000 Bornes

Milestones 2000 est un jeu de Mille Bornes en version américaine écrit en Orca/Pascal par Ken Franklin. Comme mode de rémunération, l'auteur a choisi le "ReliefWare". Cela signifie que l'argent touché par l'auteur sera versé à des œuvres de charité; le prix est de $15.
Le jeu respecte entièrement les règles du Mille Bornes. Il est graphique et agrémenté de sons en stéréo. Attention aux "coups fourrés".

A lire

HyperCard couleur: le vrai manuel

Voici le livre de référence sur le langage HyperTalk, le langage de programmation d'HyperCard couleur (j'ai nommé HyperCard GS, les versions Mac étant en noir et blanc).
Il s'appelle *HyperCard II GS Script Language Guide, the HyperTalk Language*, et est édité par Apple Computer chez Addison Wesley. Je l'apperai l'*HyperTalk Guide*, pour trouver un nom plus court.
Ce livre décrit en grand détail les objets, les boutons, les cartes, les fonds et les piles. Tout ce qui concerne les scripts est analysé en profondeur; les fonctions de calcul sont détaillées.
Il explique aussi comment écrire les fameuses XCmd et XFcn, les modules externes pour HyperCard GS. C'est ce manuel dont vous allez avoir besoin pour écrire la commande d'interface avec le Tool219 pour jouer les musiques SoundSmith.
Excellent complément aux documentations fournies avec HyperCard. Disons que pour un bidouilleur un peu expérimenté, ou simplement quelqu'un qui connait un peu la version noir et blanc d'HyperCard, c'est le seul ouvrage vraiment à lire, tout le reste étant suffisamment expliqué dans les piles d'aide des six disquettes.
Il n'y a pas de doute: **HyperTalk est un des langages majeurs du GS**, un langage orienté objet avec sa hiérarchie, un langage modulaire et ouvert, le seul langage **interprété** qui soit au niveau du GS.
Si vous avez le Toolbox Reference et le GS/OS Reference, il vous en faut encore un: *l'HyperTalk Guide.*

■

Courrier des lecteurs

M.Imbert: Le Macintosh peut fort bien effacer une disquette sans procéder à son initialisation, et sans recours à Mac Eraser. Introduisez la disquette à effacer tout en appuyant sur Command-Option-Tab: 61 secondes pour effacer une disquette 1,44 Mo au lieu de 95 secondes pour l'initialisation. Est-il bien nécessaire de "débiner" une machine pour en faire "mousser" une autre?

ToolBox-Mag : Il ne faut ni 95 ni 61 secondes pour effacer une disquette, mais **2 secondes**: c'est le temps mis par Prosel-16 ou par Mac Eraser, car effacer une disquette n'est rien d'autre que **la réinitialiser sans la reformater.**

Le **formatage** consiste à préparer la disquette pour qu'elle soit lisible par un **lecteur de disquettes**. L'**initialisation** consiste à écrire sur un disque déjà formaté les informations nécessaires pour un **système d'exploitation** d'un ordinateur.

Une disquette peut être formatée sans être initialisée, par exemple si elle a été initialisée sur le même lecteur de disques, mais avec un autre système d'exploitation: c'est précisément le cas pour les disquettes 800k formatées sur lecteur Apple 3,5 (Mac ou GS), qui peuvent être ensuite indifféremment initialisées, **sans aucun reformatage**, soit en Prodos, soit en HFS-Mac. Ce que le GS sait faire, mais pas le Mac.

Si effacer une disquette ne prend que deux secondes, c'est que cette opération se limite à écrire sur les deux premières pistes de la disquette les blocs de boot et un catalogue vierge, **sans rien faire d'autre**. La commande que vous mentionnez du Finder du Mac est bel et bien un **reformatage** (mais sans vérification), suivi d'une **réinitialisation** (l'effacement proprement dit). Vous pouvez le vérifier en remplissant avec un éditeur de blocs un bloc inutilisé du disque: après le Command-Option-Tab du Finder, ce bloc est remis à zéro, prouvant que la disquette a été reformatée, et pas seulement réinitialisée.

Les faits ont la peau dure, et le Mac mérite mieux que les dénégations ("ce n'est pas vrai") et les jérémiades (on "débine" ma machine) par lesquelles trop d'employés d'Apple essaient si souvent de déguiser, quand on leur signale des manques ou des bugs, leur incompétence ou leur allergie au travail.

Apple lui-même est conscient des insuffisances très sérieuses du système du Mac, particulièrement si on le compare à celui du GS: c'est pourquoi il essaie de transporter certaines des caractéristiques de GS/OS sur le Mac, avec le Système 7.

Car, quand un ordinateur, quel qu'il soit, ne sait pas faire une chose que l'utilisateur veut lui faire faire, il y a une solution et une seule: retrousser ses manches, et **programmer**. C'est ce que fait Toolbox Mag pour l'Apple II GS. Mais c'est à d'autres qu'il appartient d'aller chercher dans le chapitre d'*Inside Mac* sur le *Disk Initialisation Package* pourquoi le Mac ne sait pas effacer en deux secondes une disquette formatée, et d'**écrire le programme qui le force à le faire**.

Michel Vivarelli: Mon établissement a un parc de 7 GS, équipés de PC Transporter. Nous avions projeté d'y ajouter des cartes Duet. Que devient cette carte, et plus généralement quel est l'avenir du GS dans les prochains mois?

ToolBox-Mag : Dans les prochains mois, vos élèves vont faire **de la programmation orientée objet**, en couleurs et en musique, avec HyperCard GS, quand tant d'autres vont encore galérer sur des Turbo-Pascal préhistoriques. Ils utiliseront aussi de **nouveaux logiciels GS en français** qui, chut...

Ces GS vont se connecter en **réseau**, grâce à un nouveau périphérique de l'Apple II GS qui a battu commercialement, à la régulière, la carte Duet: le Macintosh Plus d'occasion à 3.000 F, qui fait parfaitement tourner **AppleShare**; avouez qu'il serait dommage de consacrer à ces tâches serviles la seule machine qui fait tourner HyperCard couleur !

Ce réseau comprendra également une **LaserWriter Personal NT**, toujours sous AppleShare, et les stencils à alcool iront enfin à la poubelle.

Bon sang, pourquoi n'y avait-il rien de tout ça quand nous, nous étions élèves ? Ça n'est pas juste !

■

TOOLBOX MAG

ToolBox Mag : le magazine anti-galère

Nouveau dans sa formule, **ToolBox Mag** se compose d'une revue et d'une disquette intégrée : pas de listings interminables dans la revue ; seulement les parties pertinentes des programmes. Les sources intégraux et les programmes sont sur la disquette encartée dans la revue. **ToolBox Mag**, le magazine à lire devant son **GS** !

Nouveau dans son mode de diffusion, **ToolBox Mag** n'est pas vendu au numéro et est disponible uniquement par correspondance : on s'abonne pour six revues et six disquettes.

Nouveau dans son style et son ton, **ToolBox Mag**, le magazine informatique sans cravate "interdit aux galériens", vous rappellera pourquoi vous avez choisi le **GS** : pour le plaisir, bien sûr... Sa mission : permettre à tous les utilisateurs d'**Apple IIGS**, quel que soit leur niveau de départ, de mieux exploiter les qualités de cette magnifique machine.

Directeur de la Publication

Eric Weyland

Rédacteur en Chef

Jean-Yves Bourdin

Conseil de Rédaction

Jean-Pierre Charpentier
Patrick Desnoues
Bernard Fournier
Olivier Goguel
Stéphan Hadinger
François Hermellin
Yvan Kœnig
Hubert Loiseleux
Claude Pélisson
Jean-Luc Schmitt
François Uhrich

Au sommaire des trois premiers numéros, vous trouverez entre autre, une initiation au langage C sur GS, par François Uhrich; un guide d'utilisation des fontes GS, par Jean-Yves Bourdin; une initiation aux ressources du GS, par Bernard Fournier; un jeu de puzzle, par Bruno Boissière; PicMaster GS, un programme de Bernard Fournier pour convertir les images GS; un outil avec son source, d'Olivier Goguel, pour jouer les musiques SoundSmith; GS News, par Eric Weyland; un accessoire espion (NDA) de Patrick Desnoues; une étude sur l'Apple IIGS et la vidéo, par Jacques Liautard; Mastermind GS, un jeu en TML Pascal, par Raoul Barbieux. Et bien d'autres choses encore...

Abonnez vous dès maintenant. **ToolBox Mag** n'a ni le désir, ni les moyens de faire de la publicité dans des revues qui méprisent le **GS**, ni d'envoyer des courriers de rappel, et n'est pas disponible au numéro ou en kiosque. Vous risquez de manquer la lecture du seul magazine français pour l'**Apple IIGS** !

Lancer un magazine **Apple IIGS** en France en 1990, c'est un pari : nous l'avons osé. A vous de le gagner, et de montrer que notre devise est aussi la vôtre : "**Apple II fort et vert !**"

Bulletin d'abonnement à ToolBox Mag *(Le présent bulletin est valable jusqu'au 30 avril 1991)*

Je m'abonne pour 6 numéros du magazine **ToolBox Mag** (6 revues et 6 disquettes) au prix de 590 Francs (690 Francs hors C.E.E.). Pour le même prix, je recevrai gratuitement le catalogue **Toolbox**, et je bénéficierai d'une remise automatique sur tous les produits **Toolbox**.

Nom : Prénom :

❒ **Je m'abonne pour 6 numéros à partir du n°1**

Adresse :

❒ **Je prolonge mon abonnement actuel de 6 numéros**

Code Postal : Ville : Pays :

Profession : Age :

Ci-joint mon réglement par ❒ Chèque bancaire ❒ CCP ❒ Mandat

Le .... / .... / 1991

Toolbox - 6, Rue Henri Barbusse - 95100 Argenteuil - France - ✆ (1) 30 76 18 64 - Télécopie (1) 39 47 44 08