Développement Objet en APL+Win 3.5

par Eric Lescasse

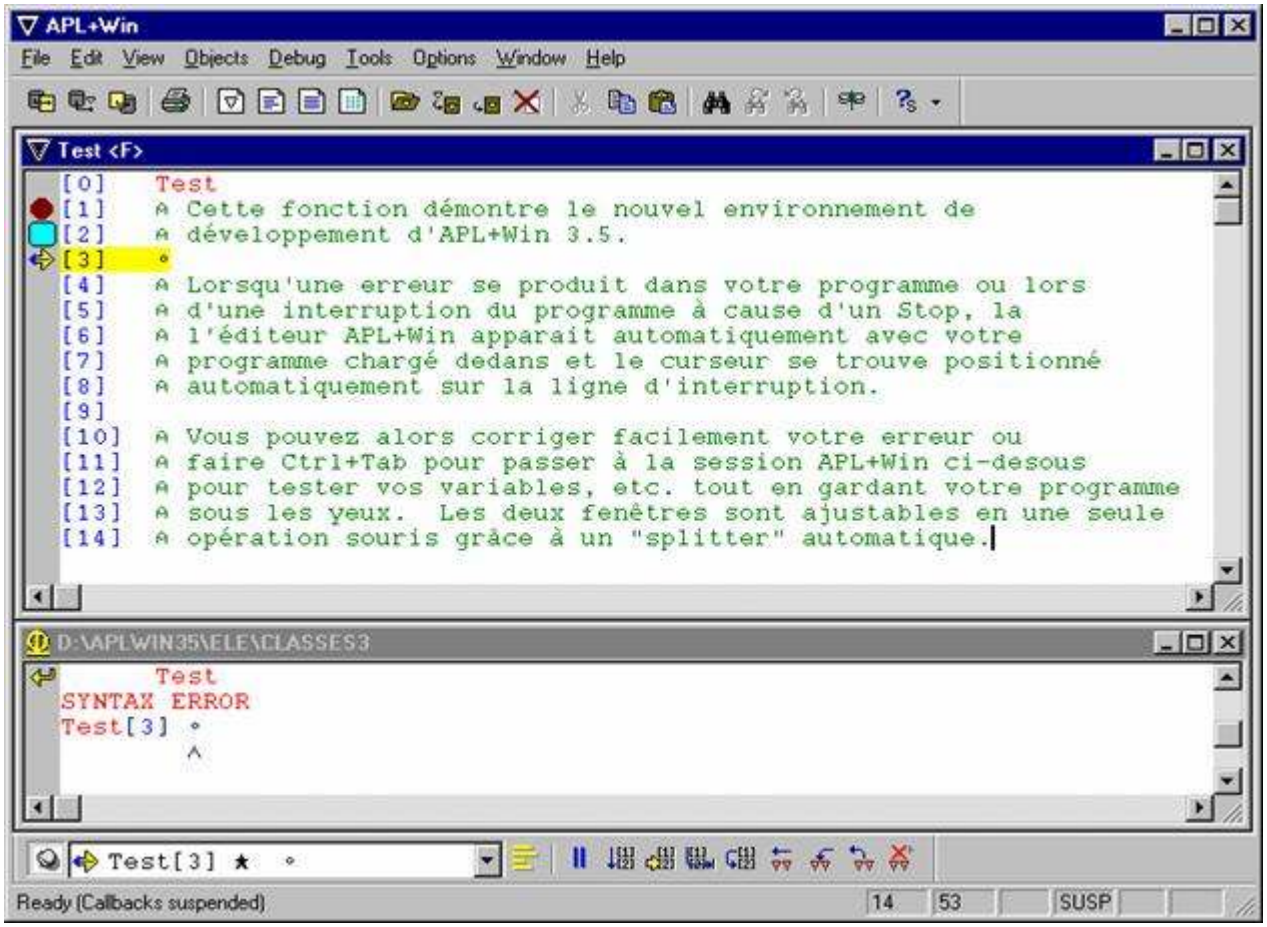
Introduction

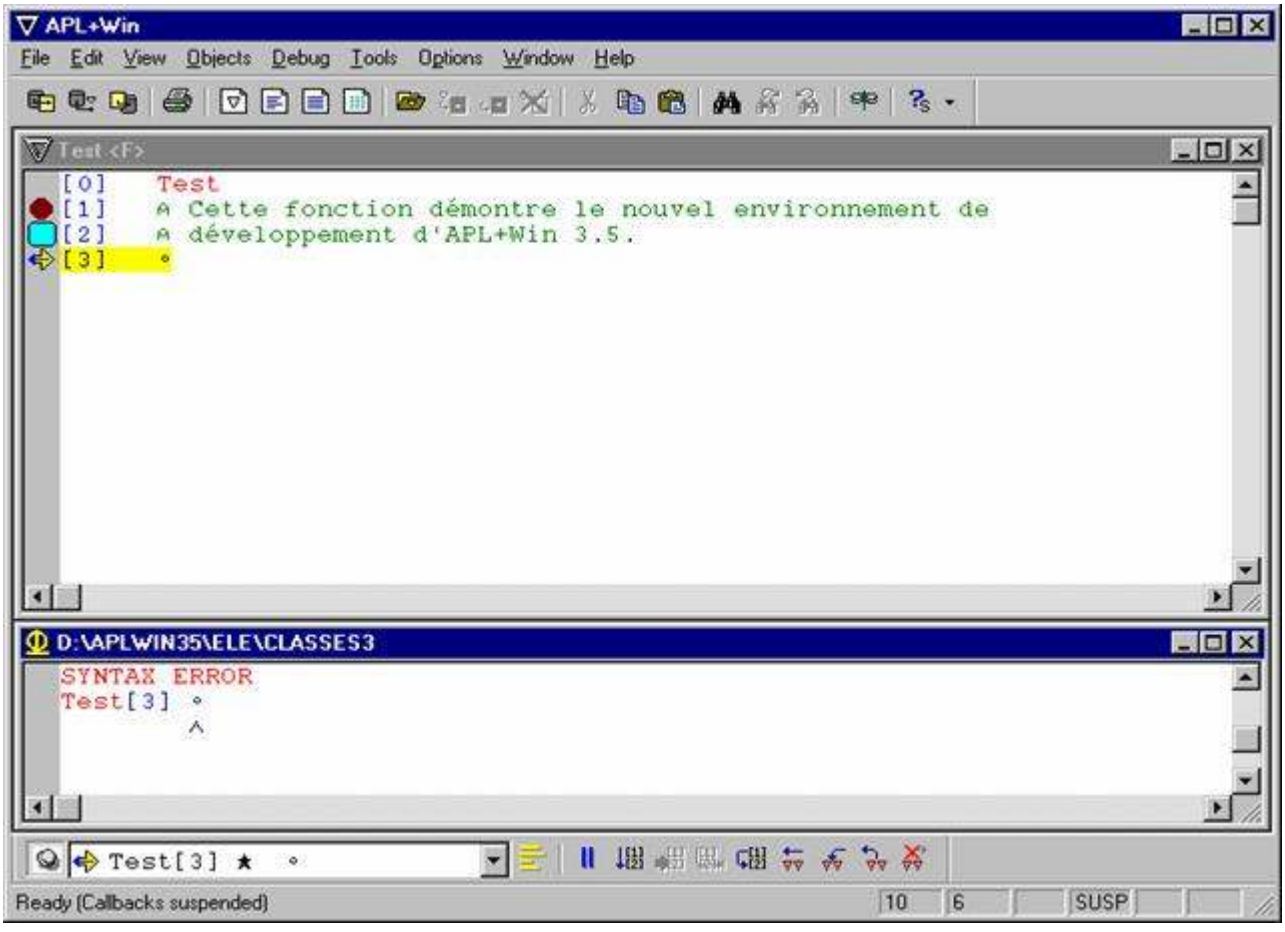
La version 3.5 d'**APL+Win** va sortir dans quelques semaines et pour la première fois, il sera possible de créer ses propres objets en **APL+Win**, au **niveau système**.

Le but de cet article est de décrire cette nouvelle fonctionnalité, l'une des plus importantes innovations **APL** depuis de nombreuses années.

APL+Win 3.5

Avant tout, décrivons succinctement quelques nouvelles caractéristiques d'APL+Win 3.5 **dont nous nous servirons dans cet article.**


APL+Win
File Edit View Objects Debug Tools Options Window Help
Test <F>
[0] Test
[1] ⍝ Cette fonction démontre le nouvel environnement de
[2] ⍝ développement d'APL+Win 3.5.
[3] ∘
[4] ⍝ Lorsqu'une erreur se produit dans votre programme ou lors
[5] ⍝ d'une interruption du programme à cause d'un Stop, la
[6] ⍝ l'éditeur APL+Win apparait automatiquement avec votre
[7] ⍝ programme chargé dedans et le curseur se trouve positionné
[8] ⍝ automatiquement sur la ligne d'interruption.
[9]
[10] ⍝ Vous pouvez alors corriger facilement votre erreur ou
[11] ⍝ faire Ctrl+Tab pour passer à la session APL+Win ci-desous
[12] ⍝ pour tester vos variables, etc. tout en gardant votre programme
[13] ⍝ sous les yeux. Les deux fenêtres sont ajustables en une seule
[14] ⍝ opération souris grâce à un "splitter" automatique.
D:\APLWIN35\ELE\CLASSES3
Test
SYNTAX ERROR
Test[3] ∘
^
Test[3] ★ ∘
Ready (Callbacks suspended)
14
53
SUSP


APL+Win
File Edit View Objects Debug Tools Options Window Help
Test <F>
[0] Test
[1] ⍝ Cette fonction démontre le nouvel environnement de
[2] ⍝ développement d'APL+Win 3.5.
[3] ∘
D:\APLWIN35\ELE\CLASSES3
SYNTAX ERROR
Test[3] ∘
^
Test[3] ★ ∘
Ready (Callbacks suspended)
10
6
SUSP

Cet image écran démontre le nouvel environnement de développement d'**APL+Win 3.5**.

Vous pouvez remarquer :

· Le nouveau « look » à base de boutons plats de type Word 97

· La présence d'une **barre d'outils** « **debogueur** » au-dessus de la barre de statut

· La présence d'une nouvelle « gouttière » le long de l'éditeur **APL+Win** et de la session **APL+Win**

· L'ouverture automatique de l'éditeur **APL+Win** avec positionnement du curseur sur la ligne fautive lors d'une interruption de programme (erreur ou « stop »)

Vous pouvez passer de l'éditeur à la session APL, par l'appui de **Ctrl+Tab**.

Cet environnement de développement est idéal pour mettre au point ses programmes.

La gouttière sert à placer/retirer visuellement des « stop » , des « trace » et des marqueurs (« **bookmarks** »)

Les marqueurs sont très utiles lorsque vous travaillez sur vos objets. En effet, comme nous allons le voir dans cet article, les objets sont souvent des fonctions APL de grande tailles, puisqu'ils doivent incorporer de nombreux comportements et propriétés. Une fois quelques marqueurs positionnés par l'appui de **Ctrl+F2** sur la ligne comportant le curseur, il vous suffit d'appuyer sur **F2** pour sauter d'un marqueur à l'autre. La navigation dans un très long programme devient très aisée et rapide.

Par ailleurs citons quelques autres améliorations de l'environnement de développement **APL+Win 3.5**:

· La limite maximum du nombre de lignes mémorisées dans la session **APL** est portée de 2000 à 8000 lignes· Des bulles explicative apparaissent automatiquement lorsque le curseur survole un nom de fonction ou de variable dans la session **APL** ou l'éditeur[1]· Le support des noms longs d'espace de travail· Des barres d'outils personnalisables et flottantes/détachables· F1 ouvre l'aide contextuelle correspondant au mot sous lequel se trouve le curseur

· □STOP et □TRACE positionnables visuellement par Ctrl+. Et Ctrl+, dans l▽éditeur

Outre les améliorations de l'environnement de développement **APL** (une partie négligée par tous les fournisseurs **APL** depuis de longues années), **APL+Win** 3.5 comporte un très grand nombre d'améliorations touchant tous les aspects de l'APL. Il serait trop long et hors de propos de les décrire ici.

Intéressons-nous maintenant à l'amélioration la plus importante : la possibilité de définir et d'utiliser ses propres objets.

Rappel sur la programmation objet en APL+Win

Avant d'analyser comment **APL+Win** vous permet de créer puis utiliser vos propres classes d'objets, il est utile de rappeler comment fonctionne la programmation objet en **APL+Win**. La programmation objet en **APL+Win** s'effectue à l'aide des fonctions systèmes ⎕WI et ⎕NI.

Les objets que vous pouvez utiliser en **APL+Win** sont en standard:

· Tous les objets de Windows 95/98 et NT (**Button, Check, Combo, Edit, Form, Frame, Imagelist, Label, List, Listview, MDIForm, Media, Menu, Option, Page, Picture, Printer, Progress, RichEdit, Scroll, Selector, Spinner, Status, Timer, Toolbox, Trackbar, Tree** et l'objet # ou objet Système)

· Tous les objets **OCX** et **ActiveX** existants (des milliers d'objets, Freeware, Shareware et produits commerciaux)

· Tous les logiciels qui supportent la technologie **COM** de **Microsoft**, tels **Excel**, **Word**, etc.

· Les objets de communication **TCP/IP** (avec la fonction système ⎕NI)

Tous les objets que vous pouvez utiliser en APL+Win se caractérisent par:

· Des **propriétés** (=des états de ces objets)

· Des **méthodes** (=des actions que l'on fait effectuer à ces objets)

· Des **événements** (=des informations que ces objets transmettent lorsqu'ils subissent des actions)

Quelques exemples d'utilisation d'objets en APL+Win

Créons une fenêtre:

```
'ff' ⎕WI 'New' 'Form'

ff
```

Par cette instruction nous créons une fenêtre Windows. En fait nous créons une nouvelle (**New**) instance de la classe d'objet fenêtres (**Form**) et nous donnons le nom **ff** à cette instance. Nous avons utilisé un premier objet: l'objet **Form**.

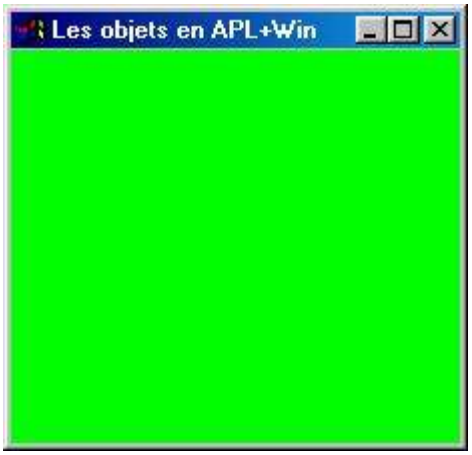
Changeons deux propriétés de cet objet:

```
'ff' ⎕WI 'caption' 'Les objets en APL+Win'
```

Par cette instruction nous indiquons que nous voulons modifier le titre (**caption**) de notre instance de fenêtre (que nous appellerons désormais abusivement "notre objet") **ff**.

Nous pouvons de même interroger la taille de la fenêtre (sa propriété **size**), puis modifier cette taille et modifier la couleur de la fenêtre (propriété **color**).

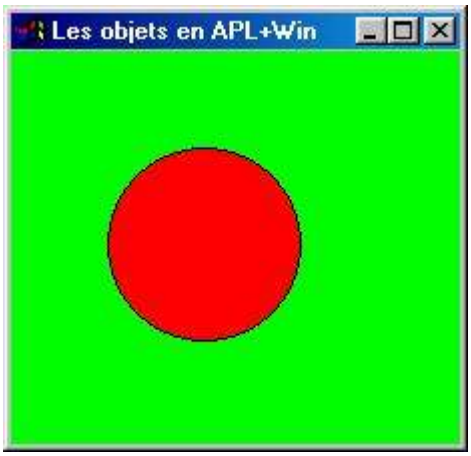
```
      'ff' ⎕WI 'size'

5.9375 42.375


      'ff' ⎕WI 'size' 12.25 28

      'ff' ⎕WI 'color' 0 255 0
```



Maintenant, rendons notre fenêtre "toujours visible" puis appliquons une méthode à notre fenêtre.

```
      'ff'⎕WI'style'16
```

```
'ff'⎕WI'Draw' 'Clear'('Brush'1 255)('Circle'6 12 6 .5)
```



Finalement, faisons en sorte de réagir à un événement se produisant sur notre fenêtre. Les événements possibles sont:

```
'ff' ⎕WI 'events'
```

```
Close DdeConnect DdeDisconnect DdeExecute Delete Destroy DragEnter

 DragLeave DragOver Drop DropDown ExitError Focus Hide KeyDown

 KeyPress KeyUp Modified MouseDouble MouseDown MouseDrag  MouseEnter
MouseLeave MouseMove MouseUp Move Open Paint Reopen Resize Send Show
Unfocus Wait
```

Indiquons à **APL+Win** que nous souhaitons être informé de tout **clic souris** se produisant dans notre fenêtre.

```
'ff' ⎕WI 'onMouseDown' '⎕WARG'
```

Cette instruction indique à l'interpréteur **APL+Win**, que dès qu'un clic (événement **MouseDown**) se produira dans le fenêtre, et à ce moment seulement, il devra exécuter la chaîne de caractères ŒWARG et donc afficher dans la session APL, le contenu de la variable système ŒWARG. Cette variable contient toujours les informations importantes relatives à l'événement qui vient de se produire.

Maintenant, voyons si nous sommes précis et essayons de cliquer au centre du cercle rouge. Je viens de le faire et voici ce qui s'est affiché dans la session APL:

6 12.125 1 1 0

Mon clic est tombé assez proche du centre du cercle dont les coordonnées étaient 6 12 (voir l'instruction **Draw** ... **Circle** ci-dessus)!

Pour l'instant nous n'avons utilisé qu'un objet de class **Form**.

Utilisation d'un objet ActiveX

Utilisons maintenant un objet **ActiveX**, c'est à dire un objet externe à **APL+Win**.

La grande nouvelle est que vous disposez, peut être sans le savoir, de dizaines, voire de centaines de ces objets, gratuitement, sur votre propre micro ordinateur. Et certains d'entre eux sont exceptionnels de fonctionnalités et d'utilité! Et vous pouvez vous en servir aussi facilement en **APL+Win** que s'il s'agissait d'un objet interne à **APL**, avec la même fonction ŒWI et la même syntaxe qui a déjà été exposée ci-dessus.

Comment connaître la liste de tous ces merveilleux objets?

C'est très simple: il suffit d'interroger la propriété '**xclasses**' de l'objet système:

```
      ⍴aaa←'#'⎕wi'xclasses'

259 4
```

Comme vous le voyez, je dispose personnellement de 259 objets **ActiveX** sur mon ordinateur. Je n'en liste ci-dessous que quelques uns au hasard. Ces objets sont très variés. Avec les quelques objets cités, je peux facilement réaliser des applications **APL+Win** qui:

- Gèrent des images
- Utilisent des objets grilles sophistiqués (**Formula One** et **Videosoft**)
- Incorporent **Internet Explorer 5** dans mes fenêtres APL!!!
- Font de superbes graphiques en 2D et 3D (**ChartFX**)
- Contrôlent l'orthographe de documents (**VisualSpeller**)
- Transfèrent des fichiers sur Internet (**Microsoft Internet Transfer Control**)
- Utilisent des champs texte avec masques de saisie dans mes fenêtres
- Et même créent un véritable éditeur **HTML WYSIWYG** de haute qualité grâce à l'objet **DHTML Edit Control** de **Microsoft**.

```
   aaa[;1 2]

Contr¶le d'¤dition d'images Kodak        {6D940280-9F11-11CE-83FD-02608C3EC08A}

Microsoft Forms 2.0 ToggleButton         {8BD21D60-EC42-11CE-9E0D-00AA006002F3}

Navigateur Web Microsoft                 {8856F961-340A-11D0-A96B-00C04FD705A2}

VC Formula One 5.0 Workbook              {13E51003-A52B-11D0-86DA-00608CB9FBFB}

ChartFX Control                          {8996B0A1-D7BE-101B-8650-00AA003A5593}

VisualSpeller Control                    {97F4CED0-9103-11CE-8385-524153480001}

:-) VideoSoft FlexGrid Control           {8099FCD0-0A81-11D2-BAA4-04F205C10000}

Microsoft Internet Transfer Control, version 6.0 {48E59293-9880-11CF-9754-00AA00C00908}

Microsoft Masked Edit Control,       version 6.0  {C932BA85-4374-101B-A56C-
00AA003668DC}

DHTML Edit Control for IE5                        {2D360200-FFF5-11d1-8D03-
00A0C959BC0A}

DHTML Edit Control Safe for Scripting for IE5     {2D360201-FFF5-11d1-8D03-
00A0C959BC0A}
```

Et je n'ai listé là qu'une dizaine d'objets parmi les 259 dont je dispose.

Un exemple:

Créons une fenêtre **APL** contenant **Internet Explorer 5**, connectons-nous à Internet et allons visiter mon site Web:

```
'int' □WI 'New' 'Form'

'int.msie5' □WI 'New' 'Navigateur Web Microsoft'

'int'□WI'onResize' "'int.msie5'□WI'where'0 0,'int'□WI'size'"
```

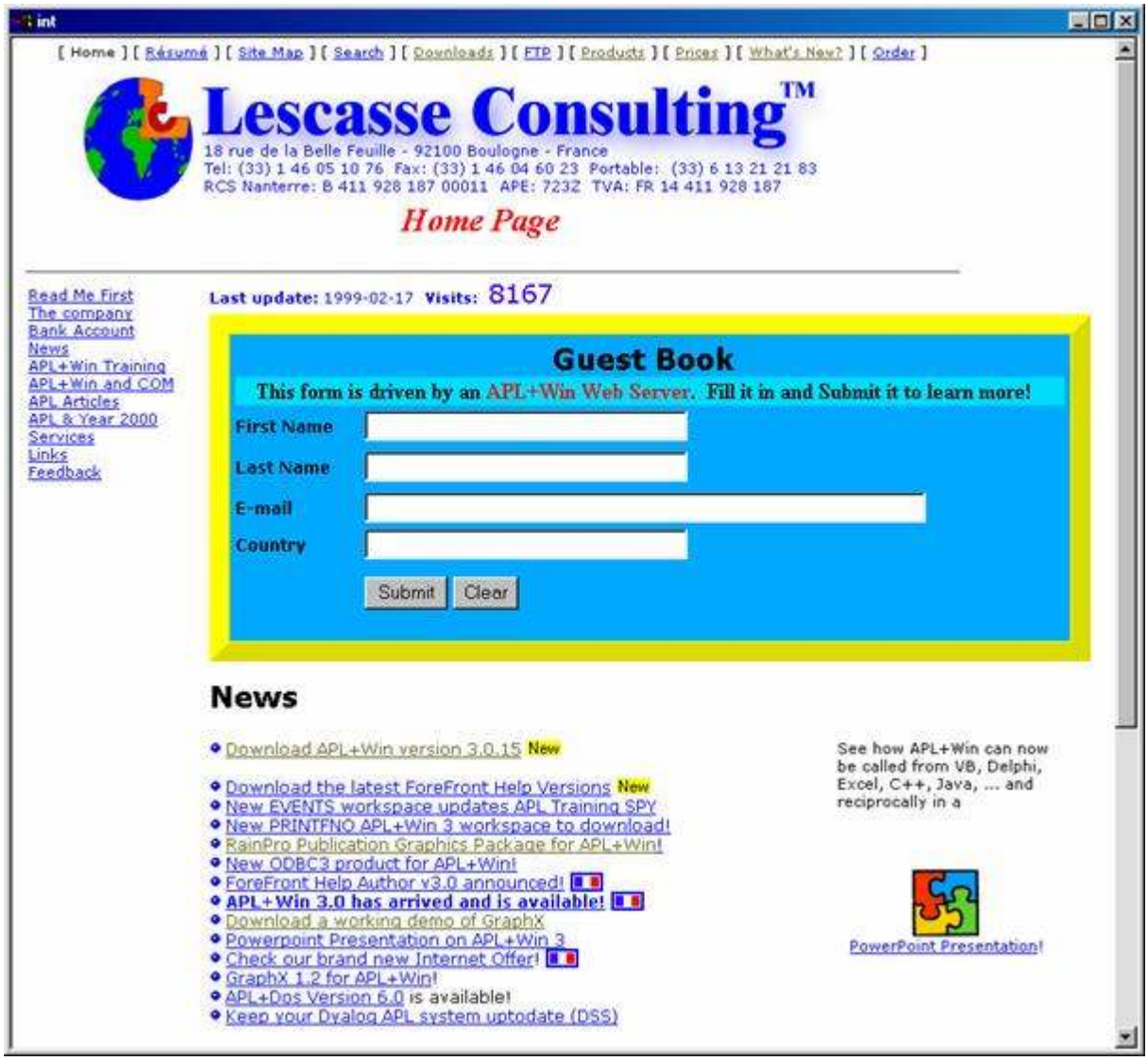
· Le première instruction crée une fenêtre **APL**.

· La seconde crée une instance du navigateur **Internet Explorer 5** et l'incorpore dans le fenêtre (la notation **int.msie5** indique que l'objet **msie5** est un enfant de la fenêtre **int**). Vous noterez que **Navigateur Web Microsoft** est le nom exact de l'un des objets **ActiveX** existants sur ma machine (voir ci-dessus)

· La troisième instruction indique à **APL+Win** qu'il devra redimensionner **Internet Explorer** en lui donnant les dimensions exactes de la fenêtre chaque fois que celle-ci changera de taille (c'est à dire lorsque je modifierai la taille de la fenêtre à l'aide de la souris)

Maintenant, supposons que j'active ma connexion Internet. Pour aller visiter mon site Web, avec ma mini application APL, il suffit que j'écrive:

```
'int.msie5' ⎕WI 'Navigate' 'http://www.lescasse.com'
```



Il n'est bien sûr pas nécessaire que **Internet Explorer** occupe la totalité de ma fenêtre **APL** et je pourrais aisément créer une application **APL** sophistiquée qui utilise **Internet Explorer** dans une partie de la fenêtre de mon application!

Bien sûr, la fenêtre ci-dessus n'est pas seulement un image mais bien une instance complète **d'Internet Explorer 5**, me permettant de naviguer librement sur mon site et sur **Internet**, en cliquant sur les liens hypertextes qui apparaissent dans la page.

Au passage, j'en profite pour vous indiquer qu'APL (aussi bien APL+Win que Dyalog APL) permet désormais de développer des applications Internet complètes et complexes. Il est en effet possible de développer un Serveur Web en APL (en moins de 200 lignes d'APL+Win).

Si vous vous connectez sur mon site **Web** à l'adresse **http://www.lescasse.com** (voir ci-dessus) entrez votre nom et votre adresse **E-mail** dans le formulaire **Guest Book**, qui se présente à vous. Vous recevrez alors une page de réponse qui vous expliquera qu'un **Serveur Web APL** vient de traiter votre requête et vous présentera quelques citations très intéressantes de développeurs américains concernant **APL+Win** et **Internet**.

Lorsque vous cliquerez sur le bouton **Submit**, ces informations seront automatiquement envoyées à mon serveur Internet dans le centre de Paris et traitées sur ce serveur par un **APL+Win**.

Ce **Serveur Web APL** est un "**Service NT APL+Win**" qui fonctionne de façon permanente et fiable depuis plusieurs mois maintenant.

Il analyse les réponses du formulaire, les enregistre dans une base de données (un simple fichier **APL** à composantes) et m'envoie automatiquement un **E-mail** pour me prévenir et m'informer du visiteur qui vient d'aller voir mon site.

Pour l'instant, mon Serveur Web **APL** ne gère que ce formulaire, mais il a été construit pour gérer autant de formulaires, compteurs Internet, bases de données, etc. qu'il est possible d'imaginer et ceci pour tous les visiteurs simultanés qui se connectent sur mon site.

Utilisation d'objets COM

Un autre type de développement objet consiste à exploiter la technologie **COM** de **Microsoft**.

De façon très simplifiée cette technologie a été créée pour permettre à des objets et logiciels disparates de communiquer et d'échanger des données entre eux.

A titre d'exemple, montrons comment il est possible en quelques instructions **APL** de complètement piloter **Excel** et d'exploiter **Excel** depuis une application **APL+Win**.

L'expression:

```
'ex' ⎕wi 'New' 'Excel.Application'
```

démarre **Excel** 97 sous la forme d'un objet **APL** nommé **ex** et le charge en mémoire (sans le montrer pour l'instant).

Vous pouvez immédiatement demander à voir la liste de propriétés, méthodes et événements disponibles, comme pour tout objet. Ces listes étant trop longues contentons-nous de demander le nombre de propriétés, méthodes et événements disponibles pour notre objet **Excel**:

```
      ⍴'ex'⎕wi'properties'
```

```
159
```

```
      ⍴'ex'⎕wi'methods'
```

```
60
```

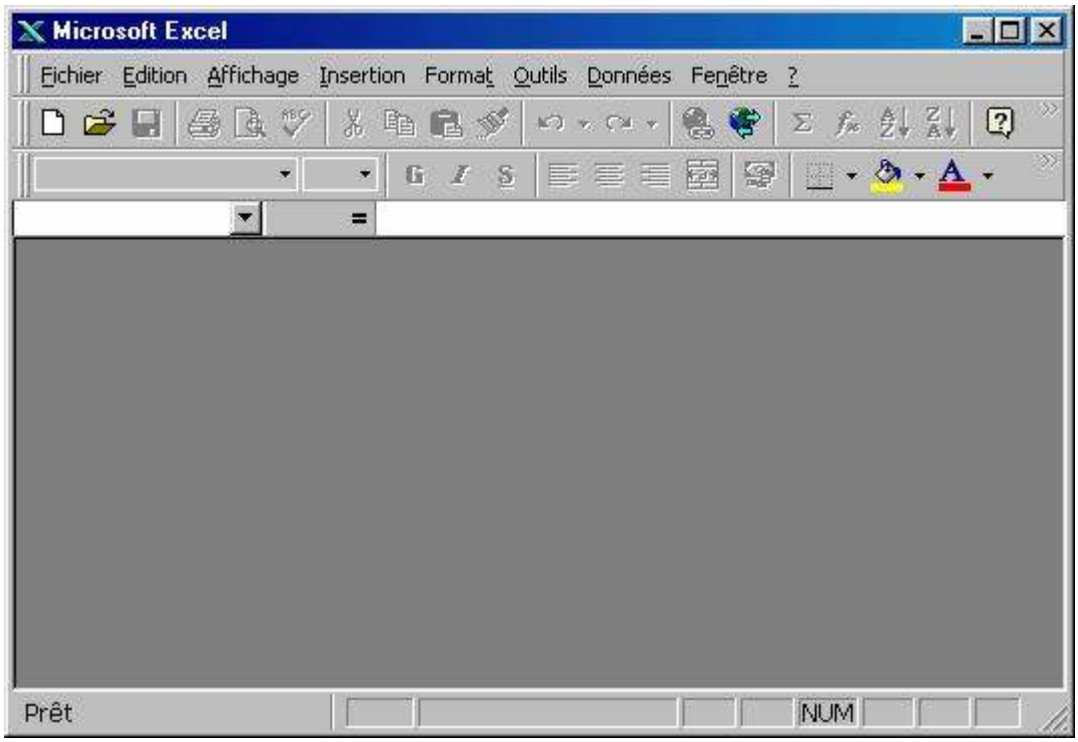
```
      ⍴'ex'⎕wi'events'
```

```
6
```

Les objets **COM** sont en réalité constitués d'une hiérarchie d'objets et sous-objets. L'accès aux sous-objets se fait par l'intermédiaire de certaines propriétés de l'objet **COM**, avec une syntaxe de "redirection" de cette propriété vers un sous objet.

Commençons par montrer notre objet **Excel**:

```
    'ex'⎕wi'visible'1
```



Vous remarquerez qu'aucun classeur n'est ouvert par défaut. En effet un classeur **Excel** est un objet (en fait un sous-objet **d'Excel**) et il nous faut le créer.

Il faut tout d'abord créer un objet "**Collection de Classeurs**". Appelons-le **ex.wkbks**:

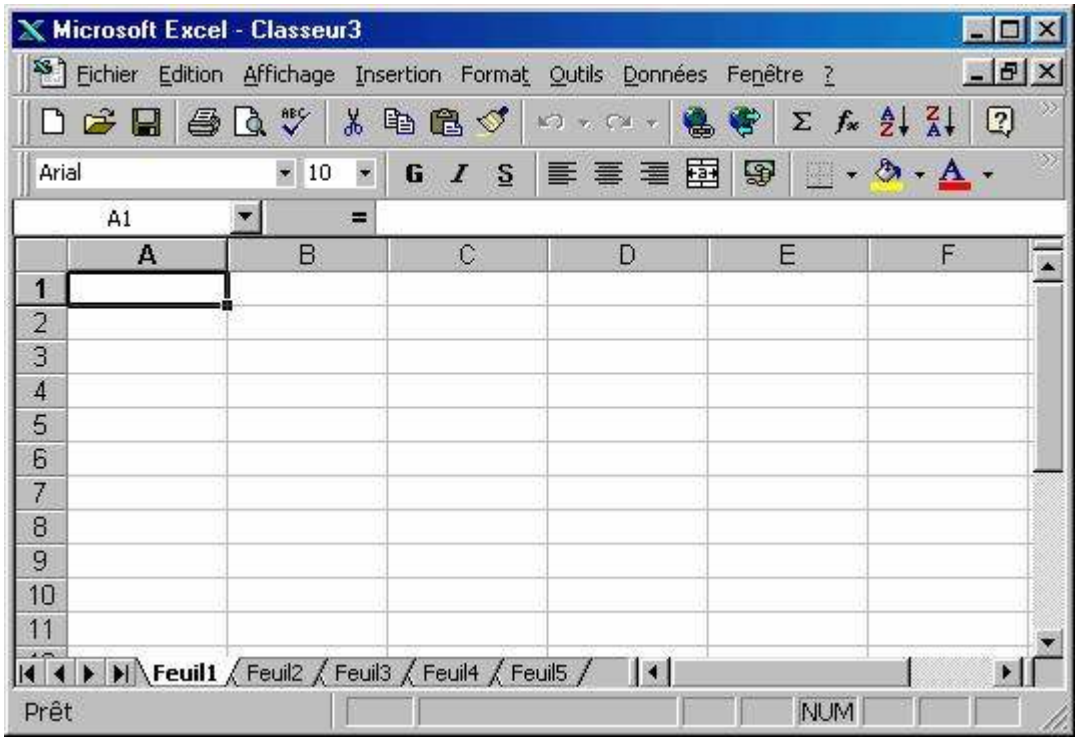
```
      'ex'⎕wi'Workbooks>ex.wkbks'
```

La syntaxe précédente, nouvelle pour Œwi, permet de rediriger la propriété **Workbooks** vers la création d'un objet relatif à cette propriété.

Maintenant, créons un classeur (=ajoutons un classeur **Excel** à notre collection), avec 5 feuilles de calcul:

```
'ex'□wi'SheetsInNewWorkbook'5

'ex.wkbks'□wi'XAdd>ex.wkbk'
```

Notre fenêtre **Excel** devient:



Il nous faut maintenant rendre actif le **Classeur** ainsi créé.

```
   'ex.wkbks'□wi'Item>ex.wkbk'1

'ex.wkbk'□wi'Activate'
```

La documentation du modèle objet **d'Excel** nous indique qu'il faut maintenant créer un objet "**collection de Feuilles**" ainsi qu'un objet "**Feuille**" et rendre actif cet objet feuille en passant son numéro à la propriété **Item**, puis en invoquant la méthode **Activate**:

```
 'ex.wkbk'Œwi'Worksheets>ex.wksht'

'ex.wksht'Œwi'Item>ex.wksht'2

 'ex.wksht'Œwi'Activate'
```

Ces 3 instructions sont similaires aux instructions ayant permis de créer les objets "**collection de Classeurs**" et "**Classeur**" ci-dessus.

Par exemple, créons un tableau généralisé en APL:

```
  report←?5 3⍴10000

      report←'APL+Win' 'APL+Dos' 'APL+Unix' 'APL+PC' 'APL+Link',report

      report←(AV2ANSI¨'' 'Jan' 'Fev' 'Mar'),report

      report

           Jan  Fev  Mar

APL+Win  1742 6851 5158

APL+Dos  9066 5967 4005

APL+Unix 7432 9805 3459

APL+PC   3804 9630 8291

APL+Link  996 1328 1045


      ⍴report

6 4
```

Pour installer ce tableau dans la feuille **Excel** active, il faut créer un objet "**plage de cellules**" ("**range**" en anglais), enfant de notre feuille:

```
    'ex.wksht'□wi'Range>ex.rng' 'B2:E7'

      'ex.rng'□wi'Value'(⊂report)
```

Notez qu'**Excel** traite ses tableaux en plaçant les colonnes avant les lignes, contrairement à **APL**, d'où la nécessité de transposer notre tableau généralisé **APL**.

Ajoutons maintenant une série de formules en bas du tableau pour effectuer les sommes des colonnes:

```
    'ex.wksht'□wi'Range>ex.rng' 'C8:E8'
      'ex.rng'□wi'Formula' '=sum(C3:C7)'
```

Notez que nous n'avons indiqué qu'une formule pour 3 cellules et qu'Excel s'est chargé tout seul, d'une part de reproduire la formule dans chacune des 3 cellules **C8**, **D8** et **E8** et d'autre part de la traduire en =**SOMME(C3:C7)** =**SOMME(D3:D7)** et =**SOMME(E3:E7)**.

Maintenant encadrons les diverses plages de titre:

```
'ex.wksht'□wi'Range>ex.rng' 'B3:B7'
  'ex.rng'□wi'BorderAround'1 3
'ex.wksht'□wi'Range>ex.rng' 'B2:E2'
  'ex.rng'□wi'BorderAround'1 3
'ex.wksht'□wi'Range>ex.rng' 'B8:E8'
  'ex.rng'□wi'BorderAround'1 3
'ex.wksht'□wi'Range>ex.rng' 'B2:E8'
  'ex.rng'□wi'BorderAround'1 4
```

Puis alignons à droite le contenu des colonnes **C D** et **E**:

```
    'ex.wksht'□wi'Range>ex.rng' 'C2:E8'
      'ex.rng'□wi'HorizontalAlignment' ¯4152
```

Changeons maintenant la police des titres de ligne et de colonnes de notre tableau. Pour cela il faut créer un objet "**police**" qui soit un enfant de l'objet "**plage**":

```
    'ex.wksht'□wi'Range>ex.rng' 'B3:B7;C2:E2;B8:E8'
      'ex.rng'□wi'Font>ex.rng.fnt'
  'ex.rng.fnt'□wi'xName' 'Arial'
  'ex.rng.fnt'□wi'size'12
  'ex.rng.fnt'□wi'Bold'1
```

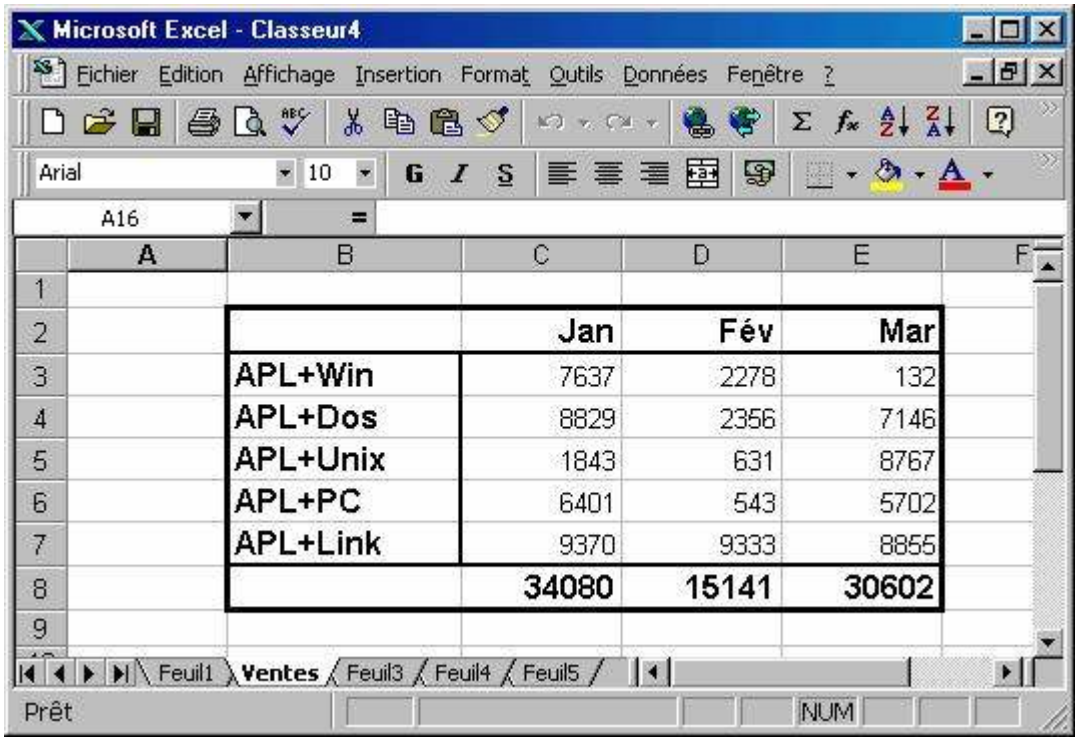
Puis élargissons la colonne **B** pour tenir compte de la police de plus grande taille:

```
    'ex.wksht'□wi'Range>ex.rng' 'B1'
      'ex.rng'□wi'ColumnWidth'16
```

Pour terminer donnons le nom "**Ventes**" à l'onglet de la feuille active:

```
    'ex.wksht'□wi'xName' 'Ventes'
```

Finalement notre objet **Excel** et notre tableau ont l'aspect suivant:



	Jan	Fév	Mar
APL+Win	7637	2278	132
APL+Dos	8829	2356	7146
APL+Unix	1843	631	8767
APL+PC	6401	543	5702
APL+Link	9370	9333	8855
	34080	15141	30602

Changeons le titre de notre fenêtre **Excel**:

```
'ex'□wi'caption'(AV2ANSI'Démonstration APL+Win Objet à AFAPL')
```

Notez l'utilisation de la fonction utilitaire `AV2ANSI` (livrée avec **APL+Win**) pour convertir les caractères du ŒAV en caractères **ANSI**.

Il faut bien que grâce à la technologie **COM**, **Excel** est devenu un objet **qu'APL** peut manipuler à loisir, exactement comme nous le souhaitons.

Enfin, si nous souhaitions sauvegarder notre **workbook**, rien ne serait plus simple:

```
      'ex.wkbk'□wi'SaveAs' 'c:\temp\afapl.xls'
```

et pour le recharger plus tard:

```
     'ex.wkbks'□wi'XOpen' 'c:\temp\temp.xls'
```

Qu'avons nous appris à l'étude de cet exemple?

1. Des produits tels **qu'Excel**, **Word**, etc. sont des objets **ActiveX** qu'il est possible de manipuler entièrement depuis APL+Win

2. La syntaxe à utiliser ne fait appel qu'à une seule fonction système **APL+Win** (`Œwi`) et est très simple

3. Un produit comme **Excel** est une hiérarchie d'objets que l'on peut représenter ainsi:

Résumons toutes les instructions APL relatives à notre exemple Excel dans une fonction APL:

```
    ∇ ExcelExample;report
[1]    A∇ ExcelExample -- Démontre le pilotage d'Excel depuis APL
[2]    A∇ avec la technologie COM
[3]
[4]    report←?5 3⍴10000
[5]    report←'APL+Win' 'APL+Dos' 'APL+Unix' 'APL+PC' 'APL+Link',report
[6]    report←(AV2ANSI¨'' 'Jan' 'Fév' 'Mar'),report
[7]
[8]    ⎕wself← 'ex'⎕wi'Create' 'Excel.Application'('visible'1)
[9]             'ex'⎕wi'Workbooks>ex.wkbks'   A crée un nouveau classeur
[10]            'ex'⎕wi'SheetsInNewWorkbook'5 A nombre de feuilles à
ajouter
[11]     'ex.wkbks'⎕wi'XAdd>ex.wkbk'
[12]     'ex.wkbks'⎕wi'Item>ex.wkbk'1
[13]      'ex.wkbk'⎕wi'Activate'
[14]      'ex.wkbk'⎕wi'Worksheets>ex.wksht s'
[15]    'ex.wkshts'⎕wi'Item>ex.wksht'2
```

```
[16]     'ex.wksht'⎕wi'Activate'

[17]

[18]     'ex.wksht'⎕wi'Range>ex.rng' 'B2:E7' ⍝ envoi d'un tableau APL ds
Excel

[19]       'ex.rng'⎕wi'Value'(⍉report)

[20]

[21]     'ex.wksht'⎕wi'Range>ex.rng' 'C8:E8' ⍝ création de formules ds
Excel

[22]       'ex.rng'⎕wi'Formula' '=sum(C3:C7)'

[23]

[24]     'ex.wksht'⎕wi'Range>ex.rng' 'C2:E8' ⍝ changement alignement
horizontal

[25]       'ex.rng'⎕wi'HorizontalAlignment' ¯4152

[26]

[27]     'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7' ⍝ encadrement de plages de
cellules

[28]       'ex.rng'⎕wi'BorderAround'1 3

[29]     'ex.wksht'⎕wi'Range>ex.rng' 'B2:E2'

[30]       'ex.rng'⎕wi'BorderAround'1 3

[31]     'ex.wksht'⎕wi'Range>ex.rng' 'B8:E8'

[32]       'ex.rng'⎕wi'BorderAround'1 3
```

```
[33]     'ex.wksht'⎕wi'Range>ex.rng' 'B2:E8'

[34]       'ex.rng'⎕wi'BorderAround'1 4

[35]

[36]     'ex.wksht'⎕wi'Range>ex.rng' 'B3:B7;C2:E2;B8:E8' ⍝ changement de
police

[37]       'ex.rng'⎕wi'Font>ex.rng.fnt'

[38]   'ex.rng.fnt'⎕wi'xName' 'Arial'

[39]   'ex.rng.fnt'⎕wi'size'12

[40]   'ex.rng.fnt'⎕wi'Bold'1

[41]

[42]     'ex.wksht'⎕wi'Range>ex.rng' 'B1'  ⍝ changement de largeur de
colonne

[43]       'ex.rng'⎕wi'ColumnWidth'16

[44]

[45]     'ex.wksht'⎕wi'xName' 'Ventes'    ⍝ changement du nom de la
feuille

[46]

[47]           'ex'⎕wi'caption'(AV2ANSI'Démonstration APL+WinObjet à
AFAPL')

[48]

[49]      'ex.wkbk'⎕wi'SaveAs' 'c:\temp\afapl.xls'  ⍝ sauvegarde du
classeur
```

```
[50]     'ex.wkbks'⎕wi'XOpen' 'c:\temp\temp.xls' ⍝ chargement d'un autre
classeur
```

∇

Créer vos propres objets en APL+Win

Avec la nouvelle version 3.5, **APL+Win** va plus loin et vous permet de créer vos propres objets, au niveau système.

C'est à dire que vous allez pouvoir définir leurs **propriétés**, **méthodes** et **événements**, puis vous servir de la fonction `⎕WI` pour utiliser vos objets comme s'ils s'agissaient d'objets Windows standard.

Ce qui est remarquable, c'est la **pureté** et la **simplicité** du système conçu par APL2000 pour vous permettre de créer vos objets.

Les concepts de base de ce nouveau système sont les suivants: il suffit d'ajouter à APL+Win deux nouveaux événements:

· Un événement `onNew` qui se produit chaque fois que l'on se sert de `⎕WI 'New'`. En interceptant la demande de création d'un objet, avant qu'il ne soit créé par le système, il est ainsi possible de le créer soi-même avec toutes les caractéristiques souhaitées, en s'appuyant sur un ou plusieurs objets existants dans APL+Win

· Un événement `onAction` qui se produit chaque fois que vous tentez d'utiliser une propriété ou une méthode sur votre propre objet, avant que le système APL+Win ne tente d'exécuter cette propriété ou méthode

Grâce à ces 2 idées de génie, APL2000 a pu implanter un système qui vous permette de créer vos propres objets, quelque soit leur nature et leur complexité, sans alourdir son interface Windows actuelle, sans modifier vos habitudes de programmation, sans ajouter la moindre fonction ou variable système à APL+Win.

Avec le système qu'ils ont créé, vous pouvez commencer à bâtir une hiérarchie d'objets aussi complète et sophistiquée que celle de **Delphi** par exemple.

Vous pourrez utiliser l'héritage, le polymorphisme, etc...

Vous pourrez enfin réellement réutiliser vos objets d'une application à l'autre, sans avoir à "réinventer la roue" à chaque fois.

Après avoir expliqué la création d'objets personnalisés en APL+Win, je vous montrerai quelques exemples d'objets extraits de ma propre bibliothèque d'objets.

Comment créer un objet personnalisé en APL+Win 3.5

La meilleure façon de démontrer est (comme toujours) de prendre un exemple.

Tous les langages de développement Windows possèdent un objet "**Edit**" ou "zone de texte" pour permettre la saisie à l'écran. Tous possèdent également un objet "**Label**" ou "zone de texte statique" pour permettre l'affichage de texte dans une fenêtre.

Lorsque vous voulez créer une zone de saisie dans une fenêtre, vous devez la plupart du temps associer une zone "**Edit**" et un "**Label**".

Exemple:

```
     □wself←'ff'□wi'Create' 'Form'('size' 5 35)('scale'5)('caption'
'Démo Label/Edit')

      □wself←'ff.lab'□wi'Create' 'Label'('caption' 'Nom')('where'10 10
20 30)

   □wself←'ff.edi'□wi'Create' 'Edit'('where'10 43 20 100)
```

Comme vous le constatez, il n'est pas très difficile de créer une fenêtre avec un **Label** et un **Edit**.

Malgré tout, il faut 2 instructions puisqu'il y a 2 objets à créer (en dehors de la fenêtre). Par ailleurs nous constatons que le texte du **Label** n'est pas à la même hauteur que le texte saisi. Enfin, si nous devions déplacer la zone de saisie, en mode "design" ou par programme, il faudrait effectuer un second déplacement similaire pour le **Label**, avec le risque de ne pas le repositionner tout à fait correctement par rapport à la zone **Edit**.

Egalement, si nous souhaitions changer la police des éléments de notre fenêtre, il faudrait effectuer deux opérations: un changement de police pour le **Label** et un changement de police pour la zone **Edit**.

Toutes ces remarques nous amènent à imaginer un nouveau type d'objet qui serait la combinaison d'un **Label** et d'un **Edit**, toujours parfaitement solidaires l'un de l'autre et parfaitement positionnés l'un par rapport à l'autre.

Tentons de créer un tel objet.

Choisissons **TLabelEdit** comme nom de classe pour ce nouveau type d'objet.

La première chose à faire est de déclarer ce nouvel objet de façon à ce qu'il soit connu d'APL+Win.

Ceci est réalisé en ajoutant ce nom de classe d'objet à la propriété **newclasses** de l'objet système d'APL+Win.

```
'#'⎕wi'newclasses' 'TLabelEdit'
```

Maintenant définissons une gestion d'événement pour l'événement **OnNew**:

```
'#'⎕wi'onNew' 'TLabelEdit"New"'
```

Ceci signifie que lorsque nous tenterons de créer une instance de notre objet **TLabelEdit** en écrivant:

```
'ff.le'⎕wi'New' 'TLabelEdit'
```

l'événement **onNew** se déclenchera et exécutera la fonction **TLabelEdit** avec un argument droit égal à '**New'.**

Vous remarquerez que nous prenons grand soin de nommer la fonction qui gèrera notre nouvel objet, du même nom exact que la classe de cet objet. En effet, lorsque nous aurons créé des dizaines ou centaines d'objets personnalisés, cela facilitera la maintenance. Par ailleurs, en **APL+Win**, lorsque l'on appuie sur **Ctrl+Shift+O** lorsque le curseur est sur un nom de fonction, cela ouvre la fonction dans l'éditeur: il sera donc très pratique de simplement mettre le curseur sur le mot **TLabelEdit** à l'écran et de faire **Ctrl+Shift+O** pour éditer la fonction qui gère cet objet.

Création d'un nouvel objet

Nous allons donc développer une fonction **TLabelEdit** correspondant à notre objet, et nous allons nous efforcer de placer la totalité de notre objet dans cette fonction (propriétés, méthodes, gestion d'événements, ...) sans appeler de sous-programmes.

De cette façon notre objet sera facilement réutilisable. Il suffira de copier sa fonction dans une autre application.

Le squelette que je recommande pour les fonctions d'objets personnalisés est le suivant:

```
   ∇ A TClass B;⎕io;⎕wself
[1]   ⍝∇ A TClass B -- TClass Template Object
[2]   ⍝∇ A ↔ object name
[3]   ⍝∇ B ↔ 'property'
[4]   ⍝∇   or 'property' value
```

```
[5]   ⍝▽   or 'Method'
[6]   ⍝▽   or 'Method' argument1 ... argumentN
[7]
[8]   ⎕io←1                                    ⍝ environnement
[9]   :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end           ⍝ objet par défaut
[10]  :select B
[11]  :case'New'                               ⍝ constructeur
[12]      ⎕wself←A ⎕wi'⋆Create' 'Class'        ⍝ création objet
[13]          ⎕wi'⋆onAction' 'TClass"Action"'  ⍝ onAction handler
[14]  :case'Action'
[15]      :select↑⎕warg
[16]      :case'class'
[17]          ⎕wres←'TClass'
[18]      :case'methods'
[19]          ⎕wres←(⎕wi'⋆methods')
[20]      :case'properties'
[21]          ⎕wres←(⎕wi'⋆properties')
[22]      :else
[23]          ⎕wres←⎕wi'⋆'
[24]      :end
[25]  :else
[26]      ⎕error'Unkown TClass command: ',B
[27]  :end
    ▽
```

L'argument droit de nos fonctions d'objets seront des noms de **propriétés** (suivies de valeurs ou non), des noms de **méthodes** (suivies d'arguments ou non), des noms **d'événements**.

L'argument gauche de nos fonctions d'objet sera toujours le nom de l'**instance** d'objet que nous souhaitons créer.

En ligne 9, s'il n'y avait pas d'argument gauche passé à la fonction d'objet, nous récupérerions le nom de l'objet dans la variable système ŒWSELF. Ensuite, le reste de la fonction est une clause :**select** dont les différents :**case** comprendront toujours:

· Le :**case'New'** ou "constructeur" qui contiendra les lignes de code servant à créer une instance de notre objet

· Le :**case'Action'** qui recevra et gèrera tous les appels à des propriétés ou méthodes de l'objet

· Le :**else** qui renverra une erreur en cas de non reconnaissance du nom de la propriété ou méthode invoquée.

Que mettre donc dans le constructeur de notre objet? Eh bien, il nous faut créer deux objets, un **Label** et un **Edit**.

Mais nous souhaitons nous assurer qu'ils possèdent tous les deux une échelle "**pixels"**, ce qui se fait en fixant leurs propriété '**scale'** à 5. Voici la fonction **TLabelEdit**, obtenue en remplaçant **TClass** par **TLabelEdit** partout dans la fonction **TClass**, puis en ajoutant les lignes 12 et 13:

```
   ▽ A TLabelEdit B;⎕io;⎕wself
[1]   ⍝▽ A TLabelEdit B -- TLabelEdit Template Object
[2]   ⍝▽ A ←→ object name
[3]   ⍝▽ B ←→ 'property'
[4]   ⍝▽   or 'property' value
[5]   ⍝▽   or 'Method'
[6]   ⍝▽   or 'Method' argument1 ... argumentN
[7]
[8]   ⎕io←1                                    ⍝ environment
[9]   :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end           ⍝ default object
[10]  :select B
[11]  :case'New'                               ⍝ constructor
```

```
[12]      ⎕wself←A ⎕wi'⋆Create' 'Edit'('scale'5) ⍝ create an Edit object

[13]      ⎕wself←(A,'Lab')⎕wi'⋆Create' 'Label'('scale'5) ⍝ now a Label
[14]          ⎕wi'⋆onAction' 'TLabelEdit"Action"' ⍝ onAction handler

[15]  :case'Action'

[16]      :select↑⎕warg

[17]      :case'class'

[18]          ⎕wres←'TLabelEdit'

[19]      :case'methods'

[20]          ⎕wres←(⎕wi'⋆methods')

[21]      :case'properties'

[22]          ⎕wres←(⎕wi'⋆properties')

[23]      :else

[24]          ⎕wres←⎕wi'⋆'

[25]      :end

[26]  :else

[27]      ⎕error'Unkown TLabelEdit command: ',B

[28]  :end
```

▽

Nous pouvons désormais créer une instance de notre objet:

```
'ff.le'⎕wi'New' 'TLabelEdit'
```

Toutefois les 2 sous-objets qui le composent se trouvent positionnés au centre de l'écran et sont superposés.

Nous souhaitons que notre objet **TLabelEdit** dispose d'une propriété **where** qui nous permettent de les positionner correctement sur notre fenêtre.

Création d'une propriété

Il nous faut d'abord pouvoir préciser quel est le texte du **Label** associé au contrôle **Edit**. Il faut pour cela que nous puissions changer la propriété **caption** du **Label**. Mais n'oublions pas que notre objet est un ensemble **Label + Edit** et qu'il ne connait pas lui-même de propriété **caption**.

Il faut donc en définir une.

La définition d'une propriété d'un objet personnel est très simple. Il suffit d'ajouter un :**case'nom_de_propriété'** au :**select** du :**case'Action'** de l'objet.

Toutefois pour une propriété il faut traiter 2 cas:

- L'interrogation de la valeur actuelle de la propriété
- Le changement de valeur de la propriété

Ceci sera très simplement réalisé à l'aide d'une structure :**if ... :else ... :end**, comme suit:

```
:case'caption'
    :if 1=⍴⎕warg
        ⎕wres←(A,'Lab')⎕wi'caption'
    :else
        (A,'Lab')⎕wi'caption'(2↓⎕warg)
    :end
```

Vous noterez 2 points essentiels.

Lors de l'utilisation d'une propriété ou d'une méthode, la variable système `⎕warg` contient toujours en premier élément le nom de la propriété ou méthode invoquée, et dans les éléments suivants, la nouvelle valeur de la propriété ou les arguments de la méthode.

Par exemple, si nous exécutions:

```
      'ff.le'□wi'caption' 'Essai'
```

Œwarg contiendrait un vecteur généralisé de 2 chaines de caractères, comme le montre l'instruction suivante:

```
      ]display □warg
.→---------------.
|.→------..→----.|
||caption||Essai||
|'-------''-----'|
'∈---------------'
```

Par ailleurs, il faut placer dans la variable système Œwres ce que nous désirons recevoir comme résultat.

La clause :**if** correspond à l'interrogation de la valeur actuelle de la propriété **caption** et la clause :**else** à sa modification.

Toutes les propriétés d'un objet doivent être implémentées de cette façon.

Après avoir ajouté le :**case'caption'** à notre objet **TLabelEdit**, nous pouvons utiliser la propriété **caption** de notre objet:

```
      'ff.le'□wi'caption'
leLab
```

En **APL+Win**, la **caption** par défaut d'un **Label** est égale à son nom.

```
      'ff.le'⎕wi'caption' 'Nom du salarié'
```

Le changement de valeur d'une propriété ne rend pas de résultat.

Comme vous le constatez, l'implémentation d'une propriété d'un objet est une tâche relativement simple.

Création d'une méthode

Pour calculer la position exacte du contrôle **Edit**, il nous faudra connaître la longueur exacte en pixels du **Label**.

Pour ce faire nous pouvons écrire une fonction utilitaire **LabelSize** dont voici le code:

```
    ∇ R←LabelSize L;F;C;D;E;F;⎕io
[1]  ⍝∇ R←LabelSize L- Calcule les dimensions en pixels du Label
[2]  ⍝∇ L ←→ nom de l'objet Label
[3]  ⍝∇ R ←→ longueur du label en pixels
[4]
[5]  ⎕io←1
[6]  C←L ⎕wi'caption'              ⍝ texte du Label
[7]  D←(¯1+L⍳'.')↑L                ⍝ nom de la fenêtre
[8]  E←D ⎕wi'scale'                ⍝ échelle de la fenêtre
[9]  D ⎕wi'scale'5                 ⍝ force échelle "pixels"
[10] :if 0∈⍴F←L ⎕wi'font'          ⍝ si pas de police définie
[11]     F←'MS Sans Serif'8 0      ⍝ police par défaut
[12] :end                          ⍝ fin
[13] R←∈D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C) ⍝ longueur Label
[14] D ⎕wi'scale'E                 ⍝ rétablie échelle fenêtre
    ∇
```

Exemple:

```
      LabelSize'ff.leLab'
13 22
```

Toutefois, de façon à éviter tout sous programme pour notre objet **TLabelEdit**, il est souhaitable d'implémenter cette fonction utilitaire en tant que **méthode** de l'objet.

Comme pour l'ajout d'une propriété, l'ajout d'une méthode à notre objet consiste à ajouter un :**case'nom_de_méthode'** au :**select** du :**case'Action'** de l'objet.

Ajoutons donc le code de la fonction **LabelSize** à notre objet **TLabelEdit**. Notre fonction **TLabelEdit** devient:

```
    ▽ A TLabelEdit B;C;D;E;F;L;⎕io;⎕wself
[1]   ⍝▽ A TLabelEdit B -- TLabelEdit Template Object
[2]   ⍝▽ A ←→ object name
[3]   ⍝▽ B ←→ 'property'
[4]   ⍝▽   or 'property' value
[5]   ⍝▽   or 'Method'
[6]   ⍝▽   or 'Method' argument1 ... argumentN
[7]
[8]   ⎕io←1                                  ⍝ environnement
[9]   :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end         ⍝ objet par défaut
[10]  :select B
[11]  :case'New'                             ⍝ constructeurr
[12]      ⎕wself←A ⎕wi'⋆Create' 'Edit'('scale'5)
[13]      ⎕wself←(A,'Lab')⎕wi'⋆Create' 'Label'('scale'5)
[14]      A ⎕wi'⋆onAction' 'TLabelEdit"Action"'
[15]  :case'Action'
[16]      :select↑⎕warg
[17]      :case'class'
```

```
[18]          ⎕wres←'TLabelEdit'
[19]      :case'methods'
[20]          ⎕wres←(⎕wi'⋆methods')
[21]      :case'properties'
[22]          ⎕wres←(⎕wi'⋆properties')
[23]      :case'caption'                    ⍝ propriété caption
[24]          :if 1=⍴⎕warg
[25]              ⎕wres←(A,'Lab')⎕wi'caption'
[26]          :else
[27]              (A,'Lab')⎕wi'caption'(2⊃⎕warg)
[28]          :end
[29]      :case'LabelSize'                  ⍝ méthode LabelSize
[30]          L←A,'Lab'                     ⍝ nom du Label
[31]          C←L ⎕wi'caption'              ⍝ texte du Label
[32]          D←(¯1+L⍳'.')↑L                ⍝ nom de la fenêtre
[33]          E←D ⎕wi'scale'                ⍝ échelle de la fenêtre
[34]          D ⎕wi'scale'5                 ⍝ force échelle pixels
[35]          :if 0∊⍴F←L ⎕wi'font'          ⍝ si police non définie
[36]              F←'MS Sans Serif'8 0      ⍝ police par défaut
[37]          :end                          ⍝ fin
[38]          ⎕wres←∊D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C)
[39]          D ⎕wi'scale'E                 ⍝ rétablit échelle fen.
[40]      :else
[41]          ⎕wres←⎕wi'⋆'
[42]      :end
[43]  :else
[44]      ⎕error'Unkown TLabelEdit command: ',B
[45]  :end
```

```
    ∇
```

Utilisons notre nouvelle méthode:

```
      'ff.le'⎕wi'LabelSize'
13 69
```

Tout marche bien: notre **Label** a une longueur de 69 pixels et une hauteur de 13 pixels.

Il nous faut maintenant implémenter une propriété **where** pour pouvoir positionner parfaitement le **Label** et son Contrôle **Edit** associé.

Création de la propriété where

Créons donc la propriété **where** pour notre objet.

Lorsque nous définirons une position dans l'écran pour **TLabelEdit**, ce sera celle du coin supérieur gauche du **Label**. La position du **Edit** devra se déduire automatiquement de celle du **Label** en respectant:

- Le fait que le texte du **Label** devra être à la même hauteur que le texte du contrôle **Edit**
- Le fait que le contrôle **Edit** devra commencer 5 pixels après le dernier caractère du **Label**

Voici comment implanter la propriété **where**:

```
[47]      :case'where'                        ⍝ propriété where
[48]          L←A,'Lab'                       ⍝ nom du Label
[49]          D←C←∈1↓⎕warg                    ⍝ argument de where
[50]          :if 4≠⍴C                        ⍝ contrôle argument
[51]              ⎕wres←'La propriété where doit contenir 4 éléments'
[52]              :return                     ⍝ sortie si erreur
[53]          :end
[54]          (E F)←⎕wi'LabelSize'L            ⍝ dimensions du label
[55]          D[1]←C[1]-3                     ⍝ position vert. du Edit
[56]          D[2]←C[2]+F+5                   ⍝ position horz. du Edit
[57]          D[3]←E+7                        ⍝ ajuste hauteur du Edit
[58]          D[4]←C[4]                       ⍝ longueur du Edit
```

```
[59]          C[3 4]←E F                    ⍝ dimensions du Label

[60]          L ⎕wi'⋆where'C                ⍝ positionne le Label

[61]          A ⎕wi'⋆where'D                ⍝ positionne le Edit
```

Le principe consiste à récupérer la valeur de la propriété **where** de notre objet **TLabelEdit** (ligne 49), à affecter cette propriété au **Label** (ligne 60), à récupérer la longueur exacte en pixels de notre **Label** (ligne 54 qui appelle notre méthode **LabelSize**), puis à ajuster précisément la propriété **where** du contrôle **Edit** associé au **Label**, en fonction de la propriété **where** du **Label** (lignes 55 à 58) et enfin de forcer les dimensions du **Label** à être précisément celles de sa **caption** (ligne 59).

Vous noterez la présence des étoiles devant la propriété **where** en lignes 42 et 43.

Ceci est en effet nécessaire pour la raison suivante: nous avons implémenté une propriété **where** pour notre objet **TLabelEdit** qui porte le même nom qu'une propriété existante dans APL+Win (**where**) pour les contrôles **Label** et **Edit**.

La présence d'une étoile devant **where** en **ligne 42 et 43** indique à APL+Win d'exécuter la propriété **where** standard des **Label** et **Edit** et non d'appeler à nouveau la propriété **where** que nous venons de définir dans notre objet **TLabelEdit**.

Si nous n'avions pas mis d'étoile (*) devant **where** en **ligne 43**, notre programme entrerait dans une boucle sans fin!

Essayons notre propriété **where**:

```
      'ff.le'⎕wi'where'10 10 15 100
```

```
    'ff.le'⎕wi'where'40 50 15 200
```

Vous noterez que la zone **Edit** est parfaitement positionnée dans les 2 cas par rapport au **Label**.

Le texte de la zone **Edit** est exactement à la même hauteur que le texte du **Label**.

La zone **Edit** commence exactement à la distance souhaitée après le **Label**.

Enfin, le dernier élément de notre propriété **where** fixe la longueur du contrôle **Edit**. Il était en effet inutile de l'appliquer à la longueur du **Label** puisque celle-ci est calculée et modifiée dans le code de la propriété **where** de **TLabelEdit**.
Maintenant, que se passe-t-il si nous modifions la propriété **caption** de notre **Label**:

```
'ff.le'⎕wi'caption' 'Nom'
```

Comme vous le constatez, la position de la zone **Edit** n'est pas ajustée automatiquement pour être placée juste après le **Label**.

Comment pouvons-nous remédier à ce petit problème de notre objet **TLabelEdit**.

Il suffit d'ajouter l'instruction suivante à la propriété **caption** de notre objet **TLabelEdit**:

```
A ⎕wi'where'((3↑(A,'Lab')⎕wi'⋆where'),4⊃A ⎕wi'⋆where')
```

En effet, le fait de changer la propriété **where** de notre objet **TLabelEdit** permet de recalculer parfaitement la position du **Label** et du **Edit.** Comme nous ne voulons pas changer la position du **Label,** nous lui réaffectons la position actuelle (soit: (3↑(A,'Lab')⎕wi'⋆where')). Il nous faut seulement récupérer la largeur de notre contrôle **Edit** (soit: 4⊃A ⎕wi'⋆where'). Vous noterez à nouveau l'utilisation des étoiles pour indiquer que nous voulons utiliser la propriété **where** standard d'**APL+Win** et non la propriété **where** de notre objet **TLabelEdit.**

Réessayons maintenant de modifier la **caption** de notre objet:

```
'ff.le'□wi'caption' 'Nom'
```

Notre objet **Edit** se repositionne maintenant parfaitement, juste après le **Label** en tenant compte de la nouvelle longueur de ce dernier.

A vrai dire, nous commençons à disposer d'un nouvel objet personnalisé (**TLabelEdit**) qui correspond exactement à nos désirs. Il nous permettra d'aller beaucoup plus vite pour définir des fenêtres de saisie et de plus nous permettra d'obtenir une interface parfaite au pixel près.

Bien d'autres propriétés et méthodes pourraient être ajoutées à cet objet, mais je souhaitais simplement vous montrer les principes du développement d'objet personnalisés en APL+Win 3.5.

Ajout du Destructeur

Il nous reste toutefois une fonctionnalité indispensable à implémenter dans notre objet **TLabelEdit**: la destruction de l'objet.

En effet, de même que tout objet personnalisé doit comporter un **constructeur** (méthode '**New'**), tout objet doit en principe comporter un **destructeur**.

Si nous n'ajoutions pas de destructeur à notre objet, nous n'effectuerions pas une destruction propre de l'objet en écrivant:

```
'ff.le'□wi'⋆Delete'
```

En effet cette instruction détruit le contrôle **Edit** (qui porte le même nom que notre objet) mais pas le **Label**, qui s'appelle **ff.leLab**.

La preuve en est le résultat de cette destruction:

Pour résoudre ce problème il nous faut gérer l'événement **onDelete** sur l'objet **Edit** et, dans la fonction de gestion de cet événement, il nous faut détruire l'objet **Label**.

Voici le code nécessaire:

```
:case'New'                                  ⍝ constructeur
    ⎕wself←A ⎕wi'⋆Create' 'Edit'('scale'5)
        ⎕wi'onDelete' "TLabelEdit'onDelete'"
    ⎕wself←(A,'Lab')⎕wi'⋆Create' 'Label'('scale'5)
    A ⎕wi'⋆onAction' 'TLabelEdit"Action"'
:case'onDelete'                             ⍝ destructeur
    (A,'Lab')⎕wi'⋆Delete'
```

Ainsi, la destruction de l'objet **TLabelEdit** est parfaite.

Cette même technique de destruction, utilisant l'événement **onDelete** du sous-objet principal (ici le contrôle **Edit**) d'un objet personnalisé doit être appliquée systématiquement pour détruire les sous-objets associés (ici le **Label**).

Essayons notre nouveau **destructeur**. Mais d'abord recréons, en une seule instruction, un objet **TLabelEdit** dans notre fenêtre: `'ff.le'⎕wi'⋆Create' 'TLabelEdit'('caption' 'Département')('where'60 10 15 150)`

Et maintenant détruisons le:

```
'ff.le'⎕wi'⋆Delete'
```

Cette fois-ci la destruction est parfaite.

Création d'un événement

Supposons que dans une application utilisant l'objet **TLabelEdit**, nous désirions pouvoir réagir à toute modification de la propriété **police** de notre objet **TLabelEdit**.

Tout d'abord, il nous faut créer une propriété police pour notre objet **TLabelEdit**, qui modifie à la fois la police du **Label** et la police du **Edit**.

En voici le code:

```
[36] :case'font'

[37]     :if 1=ρ⎕warg

[38]         ⎕wres←(A,'Lab')⎕wi'⋆font'

[39]     :else

[40]         C←1↓⎕warg

[41]         L←A,'Lab'

[42]         L ⎕wi'⋆font'C

[43]         A ⎕wi'⋆font'C

[44]         A ⎕wi'where'((3↑L ⎕wi'⋆where'),4⊃A ⎕wi'⋆where')

[45]         A ⎕wi'⋆Event' "TLabelEdit'onFontChange'"
'onFontChange'(1↓⎕warg)

[46]     :end
```

En ligne 45 nous avons défini un nouvel événement qui se produira donc à chaque fois que l'utilisateur ou l'application modifiera la propriété **font** de l'objet **TLabelEdit**.

Cette instruction utilise la nouvelle méthode **Event** (disponible pour tout objet d'**APL+Win 3.5**) pour indiquer au système de générer immédiatement un événement **onFontChange** qui sera géré par l'expression

TLabelEdit'onFontChange'.

Ajoutons donc un :**case'onFontChange'** à notre objet pour gérer cet événement.

```
:case'onFontChange'                    ⍝ événement personnalisé

   □←"La police de l'objet ",□wself," a changé et est devenue ",□warg
```

Essayons maintenant de modifier la police de notre objet:

```
'ff.le'□wi'⋆Create' 'TLabelEdit'('caption' 'Nom de famille')('where'30
10 10 200)
```

```
          'ff.le'□wi'font' 'Arial'20 1

La police de l'objet ff.le a changé et est devenue  Arial 20 1
```

Vous remarquerez par ailleurs que la propriété **where** de notre objet a été parfaitement conçue puisqu'elle a permis de réadapter automatiquement la hauteur du **Label** et du **Edit**, ainsi que leurs positions relatives, sans que nous n'ayions eu a ajouter le moindre code pour ce faire.

Le texte du **Label** et du **Edit** restent à la même hauteur, au pixel près.

Nous nous somme contentés d'afficher dans la session **APL** une information sur l'événement de changement de police, mais bien sûr, nous aurions pu développer tout autre code **APL** pour gérer l'événement de changement de police.

Nous pourrions aller plus loin. Par exemple, lorsque nous changeons la police de la fenêtre, ce qui change par héritage la police des contrôles de la fenêtre, donc du **Label** et du contrôle **Edit**, nous pourrions définir un événement **onFontChange** sur la fenêtre qui permette de redimensionner le **Label** et le contrôle **Edit**.

Je vous laisse le soin de mettre en place cette fonctionnalité, à titre d'exercice.

La page suivante nous reproduit le code complet de la fonction **TlabelEdit.**

Code de l'objet TLabelEdit

```
    ∇ A TLabelEdit B;C;D;E;F;L;⎕io;⎕wself;G;H
[1]   ⍝∇ A TLabelEdit B -- TLabelEdit Template Object
[2]   ⍝∇ A ←→ object name
[3]   ⍝∇ B ←→ 'property'
[4]   ⍝∇   or 'property' value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method' argument1 ... argumentN
[7]
[8]   ⎕io←1                                ⍝ environnement
[9]   :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end       ⍝ objet par défaut
[10]  :select B
[11]  :case'New'                           ⍝ constructeur
[12]      ⎕wself←A ⎕wi'⋆Create' 'Edit'('scale'5)
[13]          ⎕wi'onDelete' "TLabelEdit'onDelete'"
[14]      ⎕wself←(A,'Lab')⎕wi'⋆Create' 'Label'('scale'5)
[15]      A ⎕wi'⋆onAction' 'TLabelEdit"Action"'
[16]  :case'onFontChange'                  ⍝ événement personnalisé
[17]      ⎕←"La police de l'objet ",⎕wself," a changé et est devenue
",⎕warg
[18]  :case'onDelete'                      ⍝ destructeur
[19]      (A,'Lab')⎕wi'⋆Delete'
```

```
[20]  :case'Action'
[21]      :select↑⎕warg
[22]      :case'class'
[23]          ⎕wres←'TLabelEdit'
[24]      :case'methods'
[25]          ⎕wres←(⎕wi'⋆methods')
[26]      :case'properties'
[27]          ⎕wres←(⎕wi'⋆properties')
[28]      :case'caption'                ⍝ propriété caption
[29]          :if 1=⍴⎕warg
[30]              ⎕wres←(A,'Lab')⎕wi'caption'
[31]          :else
[32]              L←A,'Lab'
[33]              L ⎕wi'caption'(2⊃⎕warg)
[34]              A ⎕wi'where'((3↑L ⎕wi'⋆where'),4⊃A ⎕wi'⋆where')
[35]          :end
[36]      :case'font'
[37]          :if 1=⍴⎕warg
[38]              ⎕wres←(A,'Lab')⎕wi'⋆font'
[39]          :else
[40]              C←1↓⎕warg
[41]              L←A,'Lab'
[42]              L ⎕wi'⋆font'C
[43]              A ⎕wi'⋆font'C
[44]              A ⎕wi'where'((3↑L ⎕wi'⋆where'),4⊃A ⎕wi'⋆where')
[45]              A ⎕wi'⋆Event' "TLabelEdit'onFontChange'" ''(1↓⎕warg)
[46]          :end
[47]      :case'where'                  ⍝ propriété where
```

```
[48]          L←A,'Lab'                    ⍝ nom du Label
[49]          D←C←∈1↓⎕warg                 ⍝ argument de where
[50]          :if 4≠⍴C                     ⍝ contrôle argument
[51]              ⎕wres←'La propriété where doit contenir 4 éléments'
[52]              :return                  ⍝ sortie si erreur
[53]          :end
[54]          (E F)←⎕wi'LabelSize'L         ⍝ dimensions du label
[55]          D[1]←C[1]-3                  ⍝ position vert. du Edit
[56]          D[2]←C[2]+F+5                ⍝ position horz. du Edit
[57]          D[3]←E+7                     ⍝ ajuste hauteur du Edit
[58]          D[4]←C[4]                    ⍝ longueur du Edit
[59]          C[3 4]←E F                   ⍝ dimensions du Label
[60]          L ⎕wi'⋆where'C               ⍝ positionne le Label
[61]          A ⎕wi'⋆where'D               ⍝ positionne le Edit
[62]      :case'Destroy'                   ⍝ méthode Destroy
[63]          (A,'Lab')⎕wi'⋆Delete'        ⍝ destruction du Label
[64]          A ⎕wi'⋆Delete'               ⍝ destruction du Edit
[65]      :case'LabelSize'                 ⍝ méthode LabelSize
[66]          L←A,'Lab'                    ⍝ nom du Label
[67]          C←L ⎕wi'caption'             ⍝ texte du Label
[68]          D←(¯1+L⍳'.')↑L               ⍝ nom de la fenêtre
[69]          E←D ⎕wi'scale'               ⍝ échelle de la fenêtre
[70]          D ⎕wi'scale'5                ⍝ force échelle "pixels"
[71]          :if 0∈⍴F←L ⎕wi'font'         ⍝ si police non définie
[72]              F←'MS Sans Serif'8 0     ⍝ police par défaut
[73]          :end                         ⍝ fin
[74]          ⎕wres←∈D ⎕wi'Draw'((⊂'Font'),F)((⊂'?Text'),C)
[75]          D ⎕wi'scale'E                ⍝ rétablit échelle fenê.
```

```
[76]      :else
[77]          ⎕wres←⎕wi'⋆'
[78]      :end
[79]  :else
[80]      ⎕error'Unkown TLabelEdit command: ',B
[81]  :end
    ∇
```

Développement d'objets non visuels

Il est parfaitement possible de créer des objets non visuels en APL+Win 3.5.

Par exemple un objet "**Base de Données Relationnelle**" ou un objet "**Chronomètre**" par exemple.

Sans entrer dans les détails, sachez qu'il suffit de s'appuyer sur un objet visuel tout en laissant cet objet visuel caché.

L'objet visuel qui demande le moins de ressources système lors de sa création est un objet **Menu popup**.

La création d'un tel objet est très simple:

```
'mnu'⎕wi'New' 'Menu'
```

d'où le modèle général d'objet non visuel en **APL+Win 3.5**:

```
    ∇ A TNonVisualClass B;C;⎕io;⎕wself
[1]   ⍝∇ A TNonVisualClass B -- TNonVisualClass Template Object
[2]   ⍝∇ A ←→ object name
[3]   ⍝∇ B ←→ 'property'
[4]   ⍝∇   or 'property' value
[5]   ⍝∇   or 'Method'
[6]   ⍝∇   or 'Method' argument1 ... argumentN
[7]
[8]   ⎕io←1                                  ⍝ environnement
[9]   :if 2≠⎕nc'A' ⋄ A←⎕wself ⋄ :end         ⍝ objet par défaut
```

```
[10]  :select B
[11]  :case'New'                                  ⍝ constructeur
[12]      ⎕wself←A ⎕wi'⋆Create' 'Menu'            ⍝ création objet non visuel
[13]         ⎕wi'⋆onAction' 'TNonVisualClass"Action"' ⍝ onAction handler
[14]  :case'Action'
[15]      :select↑⎕warg
[16]      :case'class'
[17]          ⎕wres←'TNonVisualClass'
[18]      :case'methods'
[19]          ⎕wres←(⎕wi'⋆methods')
[20]      :case'properties'
[21]          ⎕wres←(⎕wi'⋆properties'),'attach' 'wherelc'
[22]      :else
[23]          ⎕wres←⎕wi'⋆'
[24]      :end
[25]  :else
[26]      ⎕error'Unkown TNonVisualClass command: ',B
[27]  :end
    ▿
```

Héritage

La possibilité de créer ses propres objets devient vraiment passionnante lorsque l'on peut faire intervenir la notion **d'héritage**.

Je réserve le développement détaillé de cette notion d'héritage à un prochain article. En attendant, les passionnés pourront réfléchir à la façon d'implémenter **l'héritage** dans les modèles d'objets décrits précédemment.

Le principe de **l'héritage** est le suivant.

Nous avons défini un objet générique **TLabelEdit** avec quelques **propriétés** et **méthodes**.

Supposons dans une application que nous ayons besoin d'un objet disposant de toutes les fonctionnalités de **TLabelEdit**, mais qui de plus n'accepte que de la **saisie numérique**.

Deux solutions s'offrent à nous:

· Soit copier la fonction **TLabelEdit** sous le nom **TLabelEditNum** par exemple, puis lui rajouter la fonctionnalité nécessaire

· Soit créer un nouvel objet de taille minimum, qui hérite de l'objet **TLabelEdit** et de toutes ses fonctionnalités, mais qui en plus sache rejeter toute saisie non numérique

La première solution est à rejeter définitivement et catégoriquement: c'est celle que nous avons tous employé pendant des années, jusqu'à l'avènement de la programmation Objet.

Pourquoi la rejeter?

C'est très simple: si vous êtes amené à modifier la fonction **TLabelEdit**, vous devrez répéter les mêmes modifications dans son clone **TLabelEditNum**. Et si vous avez créé 10 fonctions clones de **TLabelEdit**, vous devrez répéter les modifications 10 fois. Si vous ne le faites pas, vos différentes versions de **TLabelEdit** vont commencer à diverger avec tous les problèmes de maintenance que cela entraine:

· Quelle est la bonne version?

· Comment réunifier toutes les versions de la même fonction?

· Etc.

La bonne solution est donc d'utiliser **l'héritage**.

L'héritage est une fonctionnalité, lorsqu'on la maitrise, qui est absolument extraordinaire, presque magique.

Lorsque vous savez faire hériter un objet d'un autre objet, vous pouvez alors songer, enfin, à construire une hiérarchie complète d'objets, descendant d'un objet générique, par exemple appelé **TObject**. Vous pouvez ainsi construire un arbre d'objets tous les objets héritant automatiquement des fonctionnalités de leurs parents, grand-parents, etc.

Le développement d'un tel ensemble d'objet est certes un très gros travail, mais les bénéfices pour le développeur sont immenses:

· gains de temps de développements phénoménaux

· sécurité des applications considérablement accrue

· taille du code de l'application sensiblement réduite

· facilité de développement grandement améliorée

Quelques exemples d'objet

Pour terminer permettez moi de vous présenter quelques exemples d'objets développés en APL+Win 3.5, extraits de ma propre librairie d'objets APL.

Par manque de place, je ne ferais pas ou peu de commentaires sur ces objets, mais vous montrerai simplement leur utilisation.

L'objet TAgent

```
'ag'⎕wi'⋆Create' 'TAgent'

'ag'⎕wi'Merlin' 'New'

'ag'⎕wi'Merlin' 'Show'
```

Merlin est un personnage animé, capable de très nombreux comportements, capable de parler dans plusieurs langues, de comprendre et de réagir à la parole, d'afficher ce qu'il prononce dans des bulles, de jouer de la musique, etc... Voici un extrait de ses comportements

```
'ag'⎕wi'Merlin' 'PlayMethods'
Acknowledge Alert Announce Blink Confused Congratulate Congratulate_2
      Decline DoMagic1 DoMagic2 DontRecognize Explain GestureDown Ges
     tureLeft GestureRight GestureUp GetAttention⋆ GetAttention⋆Conti
     nued Greet Hearing_1 Hearing_2 Hearing_3 Hearing_4 Hide Idle1_1
     Idle1_2 Idle1_3 Idle1_4 Idle2_1 Idle2_2 Idle3_1 Idle3_2 LookDown
     ⋆ LookDown⋆Blink LookLeft⋆ LookLeft⋆Blink LookRight⋆ LookRight⋆B
     link LookUp⋆ LookUp⋆Blink MoveDown MoveLeft MoveRight MoveUp Ple
     ased Process Processing Read⋆ Read⋆Continued Reading RestPose Sa
```

```
d Search Searching Show StartListening StopListening Suggest Sur
prised Think Thinking Uncertain Wave Write⋆ Write⋆Continued Writ
ing
```

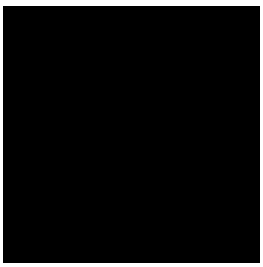
Essayons quelques uns de ces comportements:

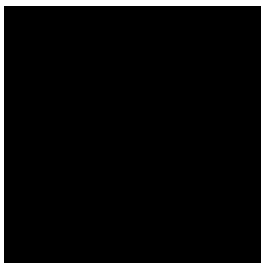
```
'ag'□wi'Merlin' 'Play' 'Announce'
```

```
'ag'□wi'Merlin' 'Speak' 'I love A.P.L. I want to program in A.P.L all
the time!'
```

Merlin a 3 frères et soeurs. Faisons les apparaitre:

```
'ag'⎕wi'Peedy' 'New' ⋄ 'ag'⎕wi'Peedy' 'Show'

'ag'⎕wi'Robby' 'New' ⋄ 'ag'⎕wi'Robby' 'Show'

'ag'⎕wi'Genie' 'New' ⋄ 'ag'⎕wi'Genie' 'Show'


'ag'⎕wi'Peedy' 'MoveTo' 600 300


'ag'⎕wi'Peedy' 'Play' 'Idle3_3'
```





Pour ma fille de 8 ans, interessée par APL, langage qui permet d'animer de si drôles créatures, j'ai réalisé une petite interface en APL+Win pour lui permettre de s'amuser un peu. La voici:

```
   )load 54 merlin
54 MERLIN SAVED 04/10/1999 22:53:09
```

La fenêtre montre les 4 personnages en train d'écrire (exécution de la méthode **Writing**).

Maintenant voyons si ces personnages peuvent aider une petite fille de 8 ans à apprendre APL:

```
'ag'⎕wi'APLTutor'
```

Je me suis servi de ces personnages dans des applications pour certains de mes clients. Ils sont utilisés pour informer les utilisateurs des nouveautés de l'application au chargement de celle-ci.

L'objet TODBC

Tout aussi sérieux, l'objet **TODBC** est implanté entièrement en APL+Win et permet de travailler avec toute base de données relationnelle.

Il utilise les **API ODBC 3** de Windows. C'est un objet non visuel complexe, de 1722 lignes, mais à lui seul il donne accès à toutes les base de données relationnelles depuis APL+Win 3.5.

```
'odbc'⎕wi'⋆Create' 'TODBC'

'odbc'⎕wi'Open' 'TBM Access Database'

80084012 80084512
```

```
'odbc'□wi'Tables'
```

Qualifier	Owner	Name	Type	Remarks
D:\tbm\Database\TBM		MSysACEs	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysColumns	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysIMEXColumns	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysIMEXSpecs	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysIndexes	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysMacros	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysModules	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysModules2	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysObjects	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysQueries	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysRelationships	SYSTEM TABLE	
D:\tbm\Database\TBM		MSysToolbars	SYSTEM TABLE	
D:\tbm\Database\TBM		agence	TABLE	
D:\tbm\Database\TBM		Chapitre	TABLE	
D:\tbm\Database\TBM		compteurs	TABLE	
D:\tbm\Database\TBM		creation	TABLE	
D:\tbm\Database\TBM		données	TABLE	
D:\tbm\Database\TBM		ELASTIC	TABLE	
D:\tbm\Database\TBM		heure contr	TABLE	
D:\tbm\Database\TBM		QDates	TABLE	
D:\tbm\Database\TBM		QNum	TABLE	
D:\tbm\Database\TBM		société	TABLE	
D:\tbm\Database\TBM		source	TABLE	
D:\tbm\Database\TBM		Type	TABLE	
D:\tbm\Database\TBM		ag	VIEW	

```
D:\tbm\Database\TBM       compteur                          VIEW
D:\tbm\Database\TBM       compteurs_Analyse1                VIEW
D:\tbm\Database\TBM       compteurs_Analyse2                VIEW
D:\tbm\Database\TBM       compteurs_Analyse4                VIEW
D:\tbm\Database\TBM       Consultation                      VIEW
D:\tbm\Database\TBM       donnée                            VIEW
D:\tbm\Database\TBM       Donnée cumulée agence             VIEW
D:\tbm\Database\TBM       Donnée cumulée société            VIEW
D:\tbm\Database\TBM       extrait                           VIEW
D:\tbm\Database\TBM       Requête1                          VIEW
D:\tbm\Database\TBM       Requête2                          VIEW
D:\tbm\Database\TBM       Requête3                          VIEW
D:\tbm\Database\TBM       Requête4                          VIEW
D:\tbm\Database\TBM       retours clients                   VIEW
D:\tbm\Database\TBM       SelCAextrait                      VIEW

     'odbc'□wi'methods'
Click Close Create Defer Delete Event Exec Modify New Open
     Ref Send Set SetLinks Close Columns Database Exec Init
      Open Query Tables

     'odbc'□wi'Columns' 'Chapitre'
Catalog             Scheme Table    Column  ODBC Type
D:\tbm\Database\TBM        Chapitre Numero  5
D:\tbm\Database\TBM        Chapitre libelle 12

     Type Name Display Size Buffer Size Digits Radix
     SHORT     5            2           0      10
```

```
      TEXT      50           50


      Nullable Remarks

      1        Numéro du chapitre  0

      1        Libellé du chapitre 1
```

```
      'odbc'□wi'?Exec'
```

```
Syntax:  'object'□wi'Exec' sql

where  sql  is most any SQL statement (use the Query method for Select
statements)

Example:
'odbc'□wi'Exec' 'insert into chapitre(numero,libelle)
values(199,''BRAVO'')'

'odbc'□wi'Exec' 'delete from chapitre where numero = 199'

'odbc'□wi'Exec' 'Update chapitre set libelle=''Provisions'' where numero
= 199'
```

```
      'odbc'□wi'Exec' 'select ⋆ from chapitre'

  1 Ventes

  2 Activité

  3 Credit clients

  4 Marges

  5 Effectifs

  6 Stocks

  7 Exploitation

  8 Alliance

  9

 10 Quotidien

 99 Chiffres Clés
```

```
      'odbc'□wi'Exec' 'insert into chapitre(numero,libelle)
values(199,''BRAVO'')'

0


      'odbc'□wi'Exec' 'select ⋆ from chapitre'

   1 Ventes

   2 Activité

   3 Credit clients

   4 Marges

   5 Effectifs

   6 Stocks

   7 Exploitation

   8 Alliance

   9

  10 Quotidien

  99 Chiffres Clés

 199 BRAVO

      'odbc'□wi'Exec' 'Update chapitre set libelle=''Provisions'' where
numero = 199'

0


   'odbc'□wi'Exec' 'select ⋆ from chapitre'

   1 Ventes

   2 Activité

   3 Credit clients

   4 Marges
```

```
   5 Effectifs
   6 Stocks
   7 Exploitation
   8 Alliance
   9
  10 Quotidien
  99 Chiffres Clés
 199 Provisions

      'odbc'□wi'Exec' 'delete from chapitre where numero = 199'
0
   'odbc'□wi'Exec' 'select * from chapitre'

1 Ventes
  2 Activité
  3 Credit clients
  4 Marges
  5 Effectifs
  6 Stocks
  7 Exploitation
  8 Alliance
  9
 10 Quotidien
 99 Chiffres Clés

      'odbc'□wi'Close'
```

L'objet TflexGrid

L'objet **TFlexGrid** fait appel à l'objet ActiveX **FlexGrid** de **VideoSoft**.

Il permet d'avoir une grille de saisie hierarchique avec cumuls à plusieurs niveaux et saisie dans les cellules par combo box multi-champs.

Exemple:

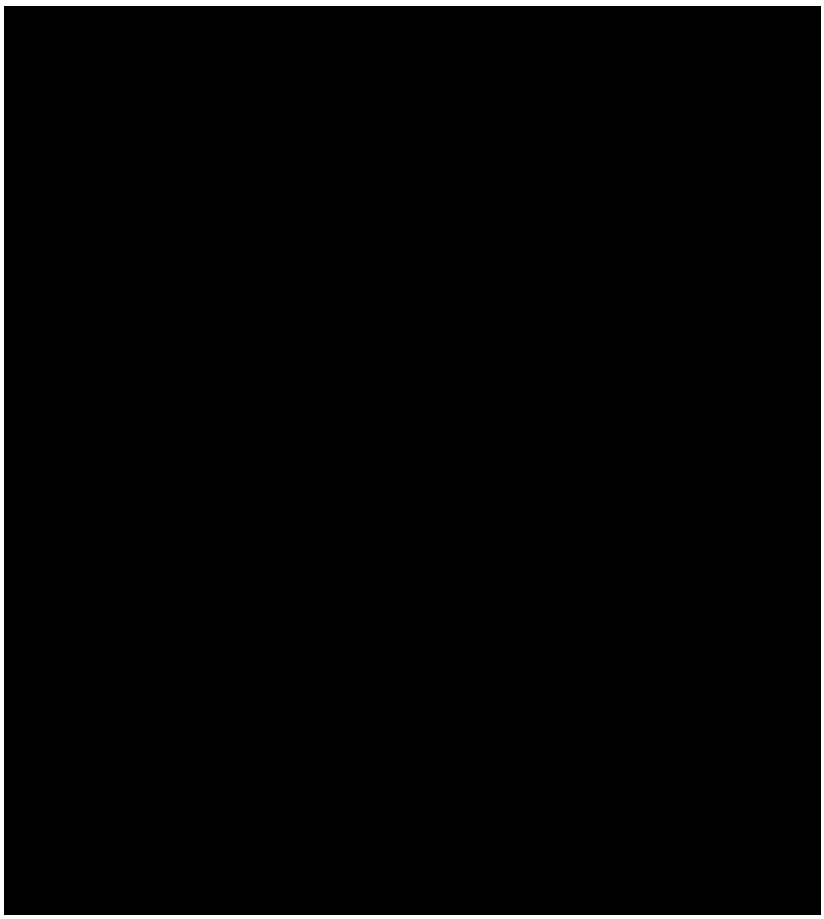
FlexGridExemple

```
    ▽ FlexGridExemple;data
[1]   ▲▽ FlexGridExemple -- Exemple montrant l'objet TFlexGrid
[2]   ▲▽ Nécessite l'objet ActiveX VSFLEX6.OCX de Videosoft installé
```

```
[3]
[4]   data←(4/'France' 'Germany' 'UK'),(12⍴2/'Gold' 'Silver')
[5]   data←data,(12⍴'I' 'E'),[1.5]⍤¨?12⍴200
[6]   data←'Region' 'Product' 'Type' 'Sales',⍪data
[7]   ⎕wself←'fmFG'⎕wi'⋆Create' 'TForm'('caption' 'TFlexGrid Example')
[8]   ⎕wself←'fmFG.ss'⎕wi'⋆Create' 'TFlexGrid'
[9]       ⎕wi'wherelc'0 0 ¯300 ¯400
[10]      ⎕wi'attach'1 2 3 4
[11]      ⎕wi'allowediting'1
[12]      ⎕wi'allowsortcols'1
[13]      ⎕wi'allowmovecols'1
[14]      ⎕wi'borderstyle'1
[15]      ⎕wi'Rows'1
[16]      ⎕wi'Cols'1
[17]      ⎕wi'FixedRows'1
[18]      ⎕wi'FixedCols'1
[19]      ⎕wi'ColWidth'0 300
[20]      ⎕wi'Add'(0 1)data
[21]      ⎕wi'colcombo'1'France|Germany|UK'
[22]      ⎕wi'colcombo'2'Gold|Silver'
[23]      ⎕wi'colcombo'3'I;Import|E;Export'
[24]  ⍝ Argument de OutlineBar:
[25]  ⍝ 0=sans boutons 1=complète 2=sans boutons du haut
[26]  ⍝ 3=sasn boutons du haut et sans lignes de connexion
[27]      ⎕wi'OutlineBar'1
[28]  ⍝ SubTotalPosition: 0=totaux en bas  1=totaux en haut
[29]      ⎕wi'SubTotalPosition'1
[30]      ⎕wi'SubTotal' 'Sum'4(¯1 1 2)
```

```
[31]  'fmFG'⎕wi'⋆size'300 400

[32]  'fmFG'⎕wi'DemoShow'

[33]  ⎕wself←'fmFG.ss'

      ▽
```

La fonction **FlexGridExemple** montre l'utilisation de l'objet **TFlexGrid** ainsi que de certaines de ses propriétés et méthodes.



L'affichage précédent montre la saisie en cours dans le champ **Type** par utilisation d'une Combo multi-champs.

[1] Par exemple, lorsque le curseur survole le nom de la variable `aaa` définie comme `aaa←⍳10`, la bulle fait apparaître : `aaa ← 10 ⍴ 1 2 3 4 5 6 7 8 9 10`