

ProduitTest

***Diagnostic Visuel des Potentialités Economiques
d'un Produit Nouveau***

Version 1.0.

Dossier de Programmation à Usage Exclusivement Pédagogique

**J.M. CHOFFRAY
ESSEC
B.P. 105
95021 CERGY-PONTOISE**

**FNEGE
2 Avenue Hoche
75008 PARIS**

LIMITATION DE GARANTIE ET DE RESPONSABILITE

Le logiciel publié dans les pages qui suivent a été développé en collaboration avec la Fondation Nationale pour l'Enseignement de la Gestion. Bien qu'il ait fait l'objet de nombreux tests, l'auteur n'offre aucune garantie, implicite ou explicite, au sujet de sa qualité ou de son rendement, ou à l'effet qu'il soit propre à l'usage particulier auquel l'utilisateur pourrait le destiner.

Le programme est publié "tel quel". L'utilisateur assume tous les risques afférents à sa qualité et à son rendement. L'auteur ne pourra en aucune circonstance être tenu responsable de dommages directs ou indirects reliés à son utilisation, même au cas où il aurait été avisé de la possibilité de tels dommages.

AVIS DE L'EXISTENCE D'UN DROIT D'AUTEUR

Il est interdit de reproduire sur bande, disque magnétique ou tout autre support, toute partie du logiciel, à toute autre fin que l'usage personnel, sans l'autorisation préalable et écrite de l'auteur.

ProduiTest et Apple sont des marques déposées et Macintosh est une marque de Apple Computer.

EVOLUTION DU LOGICIEL

Ce logiciel est publié pour des raisons exclusivement pédagogiques, afin d'illustrer les possibilités de programmation interactive dans un environnement Macintosh. Il ne fera l'objet d'aucune révision future et n'est pas disponible dans sa version source Pascal sur disquette.

REMERCIEMENTS

Ce logiciel représente plus d'un an de travail ! Un tel investissement, même pour un enseignant, requiert une grande disponibilité, un peu d'acharnement et, surtout, beaucoup d'encouragements.

C'est à ce titre que je voudrais remercier le Groupe ESSEC, en la personne de ses principaux responsables - Maître J. Coudy, Messieurs J.C. Tournand, D. Xardel, et Mademoiselle F. Rey - pour avoir maintenu et développé, depuis de nombreuses années, un environnement favorable à la recherche.

Le rôle actif de la FNEGE, à toutes les étapes de ce projet, ne peut être sous-estimé. Sans l'aide constante et la confiance que m'ont témoignées J.M. Doublet et M. Bernard, ce projet n'aurait, plus que vraisemblablement, jamais abouti !

Enfin, et surtout, je ne peux m'empêcher de penser à ma famille, en particulier à Jean-Christophe et Alexandre, qui, plus que tout autre, m'ont assisté dans ce projet.

TABLE DES MATIERES

1. Avertissement	p. 1
1.1. Remerciements	p. 2
1. Introduction	p. 4
2. ProduiTest Version 1.0	p. 5
3. Environnement de Travail	p. 6
4. Références	p. 12
5. Globals.iPas	p. 13
6. ProduiTest.Pas	p. 18
7. Interfaces	
7.1. Unit1.iPas	p. 22
7.2. Unit2.iPas	p. 23
7.3. Unit3.iPas	p. 24
7.4. Unit4.iPas	p. 25
7.5. Unit5.iPas	p. 26
7.6. Unit6.iPas	p. 28
7.8. Unit7.iPas	p. 30
8. Sources	
8.1. Unit1.Pas	p. 32
8.2. Unit2.Pas	p. 39
8.3. Unit3.Pas	p. 49
8.4. Unit4.Pas	p. 70
8.5. Unit5.Pas	p. 76
8.6. Unit6.Pas	p. 97
8.7. Unit7.Pas	p. 141
9. ProduiTest.Res	p. 182
10. ProduiTest.Link	p. 196

Often I think of computing as an extraordinary, real-life adventure game beckoning the computer warrior to descend into a yawning cavern riddled with many dark and intriguing passageways, each begging to be explored. At the end of each corridor there lies a treasure, usually small and mostly ordinary, but sometimes great and wonderous. These treasures are jewels of knowledge and enlightenment about the marvels of logic.

John Bogan, Mactutor, Vol. I N° 6

1. INTRODUCTION

Comment mieux exprimer les joies de la programmation dans un environnement aussi novateur que celui du Macintosh, au fur et à mesure qu'on en maîtrise les multiples possibilités !

Le programme que vous avez dans les mains est le fruit d'une recherche récente dont l'objectif est double :

- . mettre à la disposition des étudiants en gestion et des cadres d'entreprises un outil de diagnostic des potentialités économiques d'un produit nouveau, intégrant les principaux concepts et modèles développés en théorie de la diffusion,
- . créer un interface utilisateur reposant sur une saisie graphique de l'information et l'interprétation graphique des résultats.

Pour de nombreuses raisons techniques sur lesquelles je ne m'éterniserai pas, le Macintosh me parut, il y a plus de seize mois être l'ordinateur sur lequel cette recherche était la plus susceptible d'aboutir. J'entrepris donc de développer mon programme dans cet environnement.

L'expérience, au départ douloureuse du fait de l'absence de langages de programmation puissants, s'est révélée passionnante. Après avoir goûté aux charmes de la programmation en assembleur, en UCSD Pascal (MacAdvantage) et en Macintosh Pascal, j'ai découvert et adopté TML Pascal, environnement dans lequel le programme qui suit vous est proposé. Depuis lors, de nombreux autres environnements de programmation sont disponibles, dont Turbo Pascal et LightSpeed Pascal.

La plus grande difficulté à laquelle on se trouve confronté lorsqu'on envisage un développement "sérieux" sur le Macintosh est l'isolement. Il y a, indépendamment des efforts considérables d'Apple dans ce domaine, peu, trop peu de contacts entre développeurs d'applications, souvent maladivement jaloux de leurs découvertes !

C'est la raison essentielle pour laquelle je publie aujourd'hui ce programme. S'il peut vous éviter quelques nuits blanches, j'aurai, personnellement, atteint mon objectif.

2. ProduitTest Version 1.0.

Il n'est peut-être pas très beau ! Il s'arrête fort heureusement rarement ! Mais, ce n'est pas un traitement de texte, et il est publié.

Programmer une machine aussi puissante et "intuitive" que le Macintosh requiert beaucoup de patience et une technique différente. La notion fondamentale autour de laquelle s'articule un programme de ce type est celle d'évènement qu'il faut saisir, interpréter, et traiter.

Cette programmation, par "gestion des interrupts" n'est en fait pas plus complexe que la programmation à laquelle nous sommes tous habitués sur une machine "séquentielle", indépendamment de tout ce qu'on a pu écrire et lire sur le sujet. Elle requiert simplement d'autres réflexes, qui une fois acquis, vous paraîtront naturels !

Le drame, toutefois, réside dans le manque d'applications réelles en code source. La documentation disponible est souvent trop technique et difficile à comprendre, hors la présence de cas concrets. Seuls, quelques exemples de programmation présentent tel ou tel concept, mais rares sont ceux d'entre eux qui nous en donnent une vue d'ensemble.

J'ai voulu réparer cette injustice faite au Macintosh et à ces concepteurs en publiant mon premier programme. Je le fais d'autant plus volontiers que ne revendique pas la paternité de certaines routines, pour lesquelles je me suis largement inspiré d'exemples découverts au hasard de mes lectures. Plusieurs ouvrages et revues que j'ai trouvé utiles vous sont donnés en annexe.

Bien sûr, comme toutes les histoires heureuses, celle-ci se termine bien. J'ai pu "confidentiellement" tester mon programme sur les nouveaux Macintosh SE et Macintosh II au cours du mois de mars. Devinez quoi ? Il tourne, semblerait-il, parfaitement !

3. ENVIRONNEMENT DE TRAVAIL

Ce programme a été développé sur un Macintosh 512K, équipé d'un disque dur HD20. Vous trouverez dans les Figures 1-3, le détail de mon environnement de travail.

Le programme comporte sept unités externes correspondant aux principales tâches de l'application. J'ai très largement documenté chacune des routines afin d'en faciliter la compréhension et l'éventuelle modification. Des références fréquentes sont faites à Inside Macintosh en respectant les abbréviations reprises dans la Figure 4 de même qu'aux deux manuels du système TML Pascal : le User Guide (UG) et le Reference Manual (RM). Il va sans dire que la précision de ces références ira en s'amenuisant, au fur et à mesure que de nouvelles versions de ces documents seront disponibles.

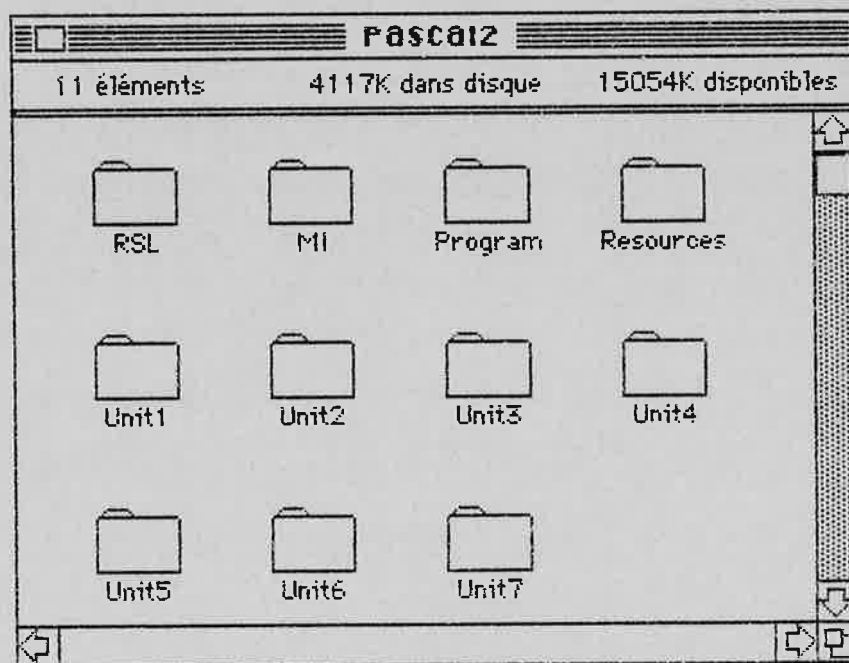
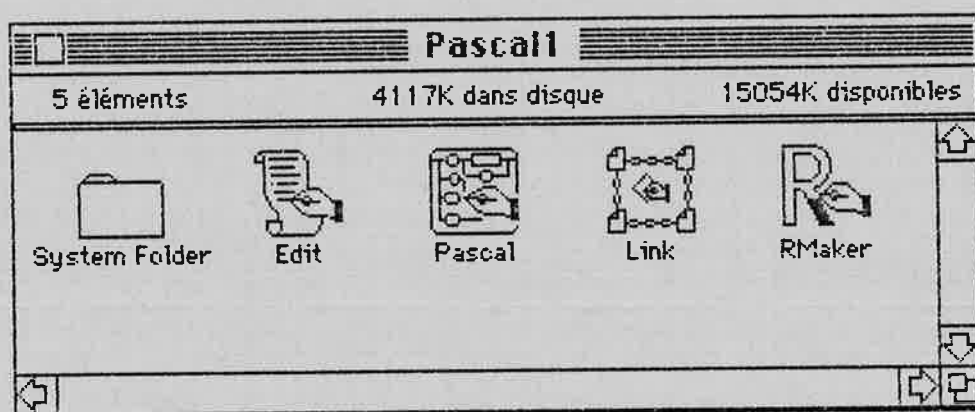
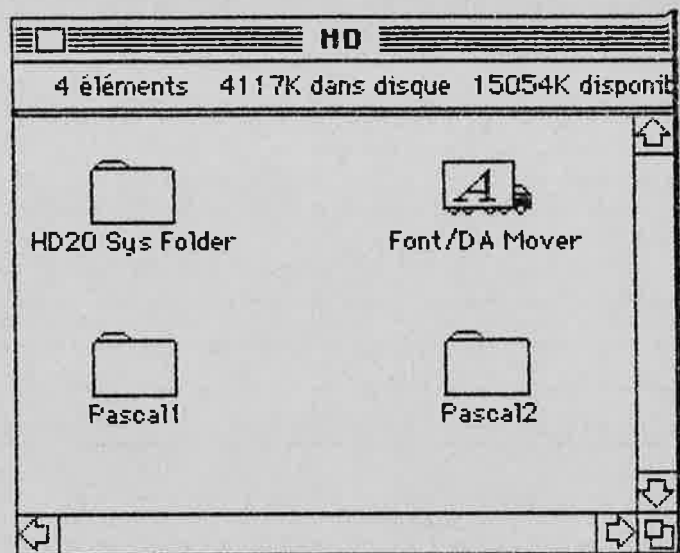


Figure 1 : Contenu des Dossiers HD, Pascal 1 et Pascal 2

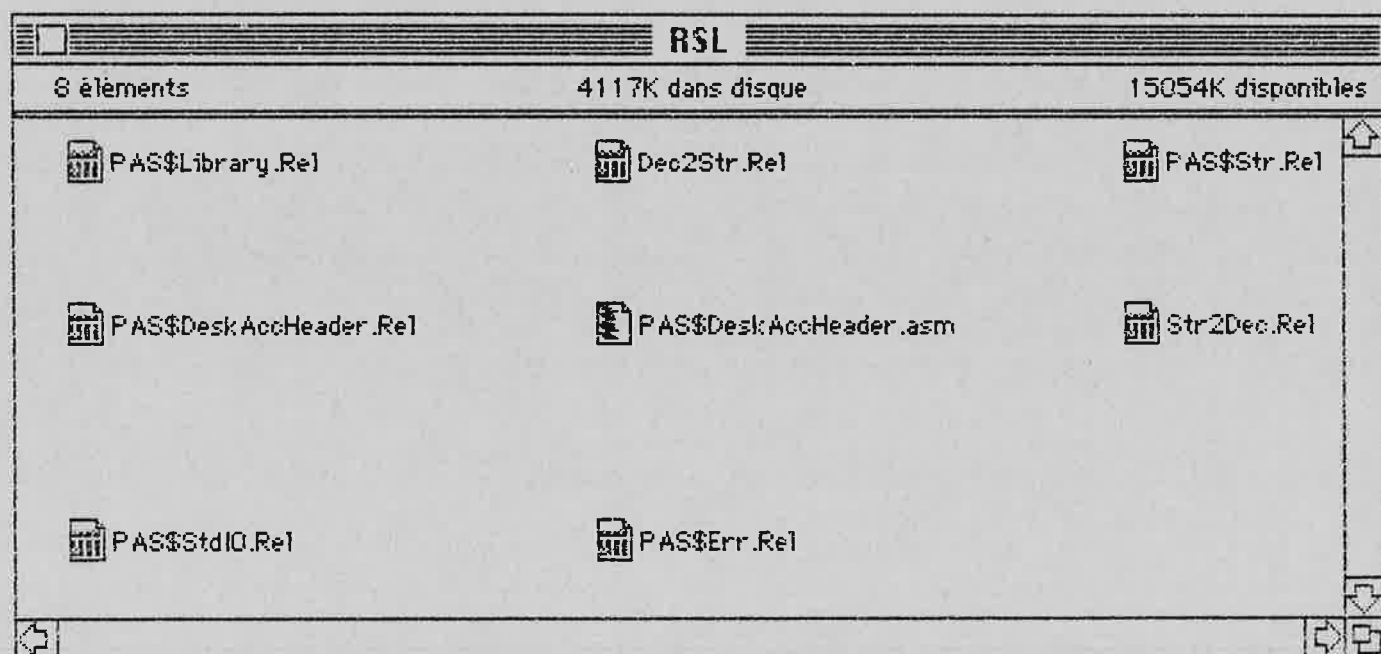
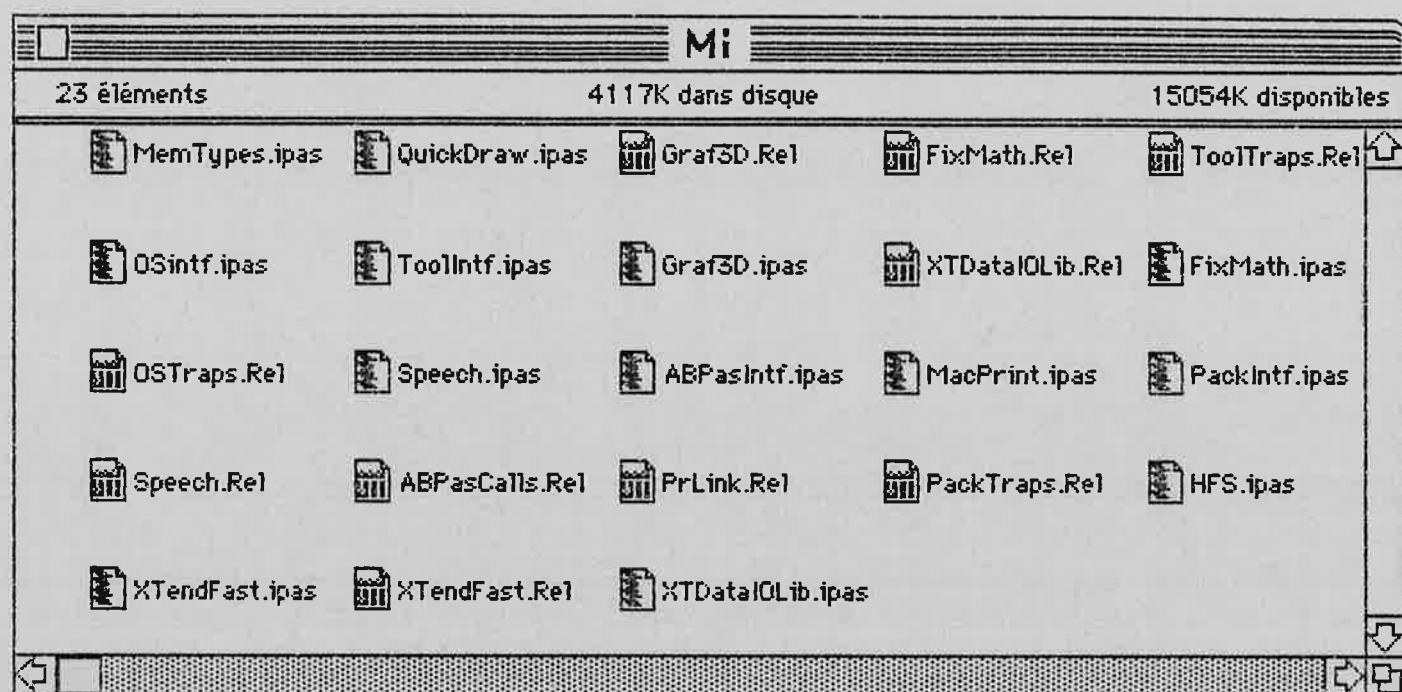


Figure 2 : Contenu des Dossiers MI et RSL

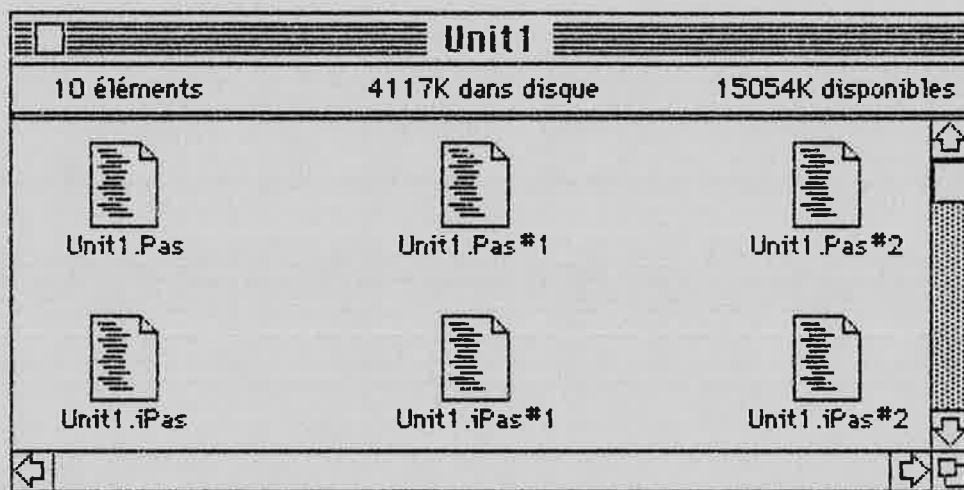


Figure 3 : Exemple du Contenu d'un Dossier Unit

Avant de vous livrer les codes source du programme, je voudrais enfin attirer votre attention sur trois tâches qui, si elles devaient être reprogrammées aujourd'hui, pourraient être considérablement simplifiées :

- . la première a trait au stockage de l'information sur disquette, qui "artificiellement" condense l'ensemble des données utilisées sous la forme d'une chaîne STRING,
- . la deuxième concerne les écrans de saisie "product" et "environnement" qui pourraient être assez facilement remplacés par des fenêtres de dialogues. Mais que voulez-vous, je souhaitais apprendre à manipuler des TextEdit Records !
- . la troisième a trait à la gestion des menus, qui pourrait être simplifiée en regroupant l'ensemble des routines qui la concerne.

*

* *

AL	Programming Macintosh Applications in Assembly Language
AM	The AppleTalk Manager :
CM	The Control Manager :
DD	The Disk Driver :
DL	The Dialog Manager :
DS	The Desk Manager :
DV	The Device Manager :
EH	The System Error Handler :
EM	The Toolbox Event Manager :
FL	The File Manager :
FM	The Font Manager :
MI	Macintosh Memory Management : An Introduction
MM	The Memory Manager :
MN	The Menu Manager :
OE	The Operating System Event Manager :
OU	The Operating System Utilities :
PK	Macintosh Packages :
PR	Printing from Macintosh Applications
PT	Putting Together a Macintosh Application
QD	QuickDraw :
RD	Inside Macintosh : A Road Map
RM	The Ressource Manager :
SL	The Segment Loader :
SM	The Scrap Manager :
SN	The Sound Driver :
SR	The Serial Drivers :
ST	The Structure of a Macintosh Application
TE	TextEdit :
TU	The Toolbox Utilities :
UI	Macintosh User Interface Guidelines
VR	The Vertical Retrace Manager :
WM	The Window Manager :

Figure 4 : Abréviations Utilisées dans les Commentaires de
Programmation

4. REFERENCES

Andrieux A. et C. Droulers (1985), Programmez Votre Macintosh (Paris : MacGraw Hill).

Apple Computer Inc. (1985), Inside Macintosh, Promotional Edition (Cupertino : Apple Computer Inc.).

Chernicoff S. (1985), Macintosh Revealed Volume I and II (New Jersey : Hayden Book Co.).

Commander J. (1985), Macintosh Assembly Language Programming (Blue Ridge Summit PA : Tab Books Inc.).

TML Systems (1985), MacLanguage Series Pascal : User's Guide and Reference Manual (Melbourne Florida : TML Systems).

5. Globals.iPas

```
if ILLE Globais.ifPas: VAR COMMON to Unit1.Pas/Code... Unit7.Pas/Code, and ProduitEst.Pas/Code }
```

```
{ *****
#
#      Programme conçu et développé
#      par Jean-Marie Choffray,
#      avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
# ***** }
```

```
CONST                                     {Identifiers are 16 characters significant}

screenWidth    = 512;                    {Width of screen in pixels QD-12}
screenHeight   = 342;                    {Height of screen in pixels QD-12}
menuBarHeight  = 20;                     {Height of menu bar in pixels MN-4}
titleBarHeight = 18;                     {Height of window title bar in pixels WM-7}

textMargin     = 2;                      {Inset from view to destination Rect TE-6}

dlgTop         = 100;                    {Top edge of dialog box for Get and Put dialogs}
dlgLeft        = 85;                     {Left edge of dialog box for Get and Put dialogs}

appleID        = 2001;                   {Menu ID for Apple menu}
  aboutItem     = 1;                      {Item number for About... command}

fileID         = 2002;                   {Menu ID for File menu}
  newItem      = 1;                       {Item number for New command}
  openItem     = 2;                       {Item number for Open command}
  {-----}                               {Disabled item consisting of a line MN-13}
  closeItem    = 4;                       {Item number for Close command}
  saveItem     = 5;                       {Item number for Save command}
  saveAsItem   = 6;                       {Item number for Save As... command}
  {-----}                               {Disabled item consisting of a line MN-13}
  printItem    = 8;                       {Item number for Print command}
  {-----}                               {Disabled item consisting of a line MN-13}
  quitItem     = 10;                      {Item number for Quit command}

editID         = 2003;                   {Menu ID for Edit menu}
  undoItem     = 1;                       {Item number for Undo command}
  {-----}                               {Disabled item consisting of a line MN-13}
  cutItem      = 3;                       {Item number for Cut command}
  copyItem     = 4;                       {Item number for Copy command}
  pasteItem    = 5;                       {Item number for Paste command}
  clearItem    = 6;                       {Item number for Clear command}

simulationID   = 2004;                   {Menu ID for Simulation menu}
  productParItem = 1;                     {Item number for Product Par command}
  {-----}                               {Disabled item consisting of a line MN-13}
  innovationParItem = 3;                   {Item number for Innovation Par command}
  imitationParItem = 4;                   {Item number for Imitation Par command}
  influenceParItem = 5;                   {Item number for Influence Par command}
  potentialParItem = 6;                   {Item number for Potential Par command}
  {-----}                               {Disabled item consisting of a line MN-13}
  replaceParItem = 8;                     {Item number for Replace Par command}
  {-----}                               {Disabled item consisting of a line MN-13}
  externalVCParItem = 10;                 {Item number for ExternalVC Par command}
```

internalVCParItem = 11;	{Item number for InternalVC Par command}
learningParItem = 12;	{Item number for Learning Par command}
{-----}	{Disabled item consisting of a line MN-13}
environmtParItem = 14;	{Item number for Environmt Par command}

analyzeID = 2005;	{Menu ID for Analyze menu}
summaryItem = 1;	{Item number for Summary data command}
{-----}	{Disabled item consisting of a line MN-13}
innovationCurItem = 3;	{Item number for Innovation Curv command}
imitationCurItem = 4;	{Item number for Imitation Curv command}
penetrationCurItem = 5;	{Item number for Penetration Curv command}
{-----}	{Disabled item consisting of a line MN-13}
replaceCurItem = 7;	{Item number for Replacmt Curv command}
volumeCurItem = 8;	{Item number for Volume Curv command}
{-----}	{Disabled item consisting of a line MN-13}
internalVCCurItem = 10;	{Item number for InternalVC Curv command}
totalCCurItem = 11;	{Item number for TotalCost Curv command}
{-----}	{Disabled item consisting of a line MN-13}
profitCurItem = 13;	{Item number for Profit Curv command}

aboutID = 3001;	{Resource ID for About... Stop alert}
cantPrintID = 3002;	{Resource ID for Can't Print Stop alert}
wrongTypeID = 3003;	{Resource ID for Wrong Type Stop alert}
saveID = 3004;	{Resource ID for Save Caution alert}
IOErrID = 3005;	{Resource ID for I/O Error Stop alert}
emptyTextID = 3006;	{Resource ID for Empty Text Stop alert}
estimatorHighID = 3007;	{Resource ID for Too High Caution alert}
estimatorLowID = 3008;	{Resource ID for Too Low Caution alert}
lossID = 3009;	{Resource ID for Cost >= Price Caution alert}
nullPenetID = 3010;	{Resource ID for Zero Diffusion Stop alert}
nullCostID = 3011;	{Resource ID for Zero Cost Stop alert}
nullPotentialID = 3012;	{Resource ID for Zero Potential Stop alert}
badModelID = 3013;	{Resource ID for Bad Model Stop alert}

TYPE

screenType	= (product, innovation, imitation, influence, potential, replace, externalVC, internalVC, learning, environmt, summary, innovationPlot, imitationPlot, penetrationPlot, replacePlot, volumePlot, internalVCPlot, totalCPlot, profitPlot);	{Enumerated set of screens}
textType	= (none, division, brand, market, evaluator, adopter, fCost, production, price);	{Enumerated set of Text records}
apiHandle	= ^apiPtr;	{Handle to application data}
apiPtr	= ^applicationData;	{Pointer to application data}
applicationData =		
RECORD		
divisionEditRecd	: TEHandle;	{Handle to Division Edit record TE-5}
brandEditRecd	: TEHandle;	{Handle to Brand Edit record TE-5}
marketEditRecd	: TEHandle;	{Handle to Market Edit record TE-5}
evaluatorEditRecd	: TEHandle;	{Handle to Evaluator Edit record TE-5}
innovationPar	: LongInt;	{Models parameters TML-RM-C-1}
imitationPar	: LongInt;	

```

influencePar      : LongInt;
potentialPar      : LongInt;
replacePar        : LongInt;
externalVCPPar    : LongInt;
internalVCPPar    : LongInt;
learningPar       : LongInt;

adopterEditRecd   : TEHandle;      {Handle to Adopter Edit record TE-5}
fCostEditRecd     : TEHandle;      {Handle to Fixed Cost Edit record TE-5}
productionEditRecd : TEHandle;      {Handle to Production Edit record TE-5}
priceEditRecd     : TEHandle;      {Handle to Price Edit record TE-5}

dirty             : BOOLEAN;        {Data change flag TML-RM-C-1}
padding           : Byte;           {Extra byte for padding as BOOLEAN = 8 bits}

fileNumber        : INTEGER;        {File reference number}
volNumber         : INTEGER;        {Volume reference number}
END; {RECORD}

```

```

emptyCircle =
RECORD
  circle          : RECT;           {Cursor activation on curves QD-9}
  percentResponse : REAL;           {Percent ordinate of curves TML-RM-C-2}
  nominalResponse : REAL;           {Nominal ordinate of curves OR minimum}
  predictor       : INTEGER;        {Abscissa of response curve TML-RM-C-1}
END; {RECORD}

```

VAR

```

theEvent      : EventRecord;      {Current event EM-11}

theWindow     : WindowPtr;        {Pointer to application window WM-11}
wdRecord      : WindowRecord;     {Space to store window record WM-10}

apiData       : applicationData;  {Application data record}

theText       : TEHandle;         {Handle to currently active edit record TE-5}

cursCircles   : ARRAY[1..15] OF emptyCircle; {Array of activation circles TML-RM-3-5}

appleMenu     : MenuHandle;       {Handle to Apple menu MN-8}
fileMenu      : MenuHandle;       {Handle to File menu MN-8}
editMenu      : MenuHandle;       {Handle to Edit menu MN-8}
simulationMenu : MenuHandle;       {Handle to Simulation menu MN-8}
analyzeMenu   : MenuHandle;       {Handle to Analyse menu MN-8}

iBeam         : CursHandle;       {Handle to I-Beam cursor TU-11}
cross         : CursHandle;       {Handle to Cross cursor TU-11}
watch         : CursHandle;       {Handle to Wristwatch cursor TU-11}

theScreen     : screenType;       {Enumerated set of screens}
textRecd      : textType;         {Enumerated set of text records}

divisionRect  : Rect;             {ViewRect of Division TextEdit record QD-9}
brandRect     : Rect;             {ViewRect of Brand TextEdit record QD-9}
marketRect    : Rect;             {ViewRect of Market TextEdit record QD-9}
evaluatorRect : Rect;             {ViewRect of Evaluator TextEdit record QD-9}

```

screenRect	: Rect;	{Overall screen rectangle QD-9}
questionRect	: Rect;	{Question label rectangle QD-9}
subjectiveRect	: Rect;	{Subjective estimation rectangle QD-9}
activeRect	: Rect;	{Active part of subjective rectangle QD-9}
estimatorRect	: Rect;	{Percent answer rectangle QD-9}
okRect1	: Rect;	{OK rect, product and environmt screens QD-9}
okRect2	: Rect;	{OK rect, estimation screens QD-9}
adopterRect	: Rect;	{ViewRect of Adopter TextEdit record QD-9}
fCostRect	: Rect;	{ViewRect of Fixed Cost TextEdit record QD-9}
productionRect	: Rect;	{ViewRect of Production TextEdit record QD-9}
priceRect	: Rect;	{ViewRect of Price TextEdit record QD-9}
scrapCompare	: INTEGER;	{Previous scrap count TML-RM-C-1}
scrapDirty	: BOOLEAN;	{Scrap change flag TML-RM-C-1}
deskOn	: BOOLEAN;	{Desk accessory flag TML-RM-C-1}
windowOn	: BOOLEAN;	{Application window flag TML-RM-C-1}
simulation	: BOOLEAN;	{Simulation flag TML-RM-C-1}
errorFlag	: BOOLEAN;	{I/O error flag TML-RM-C-1}
quitting	: BOOLEAN;	{Close application flag TML-RM-C-1}
finished	: BOOLEAN;	{Main loop control flag TML-RM-C-1}

6. ProduiTest.Pas

```

PROGRAM ProduiTest;                                {FILE: ProduiTest.Pas}

{ *****
#
#      Programme conçu et développé
#      par Jean-Marie Choffray,
#      avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
***** }

{$i HD:Pascal2:M1:MemTypes.ipas }                  {TML interface to General Purpose TYPES}
{$i HD:Pascal2:M1:QuickDraw.ipas }                 {TML interface to Macintosh Graphics Package}
{$i HD:Pascal2:M1:OSIntf.ipas }                     {TML interface to the Operating System}
{$i HD:Pascal2:M1:ToolIntf.ipas }                   {TML interface to Macintosh User Interface}
{$i HD:Pascal2:M1:PackIntf.ipas }                   {TML interface to Macintosh Packages}
{$i HD:Pascal2:M1:MacPrint.ipas }                   {TML interface to Macintosh Printing Mgr}

{$i HD:Pascal2:Program:Globals.iPas }               {Declarations COMMON to UnitA, UnitB, and}
                                                    {ProduiTest}
{$i HD:Pascal2:Unit1:Unit1.iPas }                   {Interface to Unit1 routines}
{$i HD:Pascal2:Unit2:Unit2.iPas }                   {Interface to Unit2 routines}
{$i HD:Pascal2:Unit3:Unit3.iPas }                   {Interface to Unit3 routines}
{$i HD:Pascal2:Unit4:Unit4.iPas }                   {Interface to Unit4 routines}
{$i HD:Pascal2:Unit5:Unit5.iPas }                   {Interface to Unit5 routines}
{$i HD:Pascal2:Unit6:Unit6.iPas }                   {Interface to Unit6 routines}
{$i HD:Pascal2:Unit7:Unit7.iPas }                   {Interface to Unit7 routines}

{$L HD:Pascal2:Resources:ProduiTest/Rsrc }          {Include in .LINK cont. file, the appl. resources}
{$T APPL FGPR }                                     {Set file type and creator name}
{$B+}                                                {Set application's bundle bit}

{-----}

{$S Seg1}
{$U< HD:Pascal2:Unit1:Unit1/Code }                  {Link library Unit1 to NEW segment Seg1}

{$S Seg2}
{$U< HD:Pascal2:Unit2:Unit2/Code }                  {Link library Unit2 to NEW segment Seg2}

{$S Seg3}
{$U< HD:Pascal2:Unit3:Unit3/Code }                  {Link library Unit3 to NEW segment Seg3}

{$S Seg4}
{$U< HD:Pascal2:Unit4:Unit4/Code }                  {Link library Unit4 to NEW segment Seg4}

{$S Seg5}
{$U< HD:Pascal2:Unit5:Unit5/Code }                  {Link library Unit5 to NEW segment Seg5}

{$S Seg6}
{$U< HD:Pascal2:Unit6:Unit6/Code }                  {Link library Unit6 to NEW segment Seg6}

{$S Seg7}
{$U< HD:Pascal2:Unit7:Unit7/Code }                  {Link library Unit7 to NEW segment Seg7}

{-----}

```

BEGIN {Main program}

initialize;

{Do one-time-only initialization}

REPEAT

UnloadSeg (@initialize);

{Unload the segment 1 SL-08}

UnloadSeg (@mainLoop);

{Unload the segment 2 SL-08}

UnloadSeg (@doFileChoice);

{Unload the segment 3 SL-08}

UnloadSeg (@doEditChoice);

{Unload the segment 4 SL-08}

UnloadSeg (@doSimulationChoice);

{Unload the segment 5 SL-08}

UnloadSeg (@doAnalyzeChoice);

{Unload the segment 6 SL-08}

UnloadSeg (@doContent);

{Unload the segment 7 SL-08}

mainLoop;

{One pass of main loop}

UNTIL finished;

END. {Main Program}

7. Interfaces : Unit1.iPas...Unit7.iPas

```
{FILE Unit1.iPas: Interface Part for Unit1.code}
```

```
{ *****
#
#      Programme conçu et développé      #
#      par Jean-Marie Choffray,          #
#      avec l'aide de la FNEGE           #
#                                         #
# Copyright 1987      Tous droits réservés #
#                                         #
***** }
```

```
{-----}
```

```
PROCEDURE initialize; EXTERNAL;
    {One-time-only initialization}
PROCEDURE crash; EXTERNAL;
    {Return to Finder if major crash}
PROCEDURE setUpCursors; EXTERNAL;
    {Set up and lock cursors}
PROCEDURE setUpMenus; EXTERNAL;
    {Create program menus}
PROCEDURE menusFirst; EXTERNAL;
    {DeskTop menu bar}
PROCEDURE doStartup; EXTERNAL;
    {Process Finder startup information}
PROCEDURE readDeskScrap; EXTERNAL;
    {Desk Scrap -> TextEdit Scrap, adapt PasteItem}
PROCEDURE writeDeskScrap; EXTERNAL;
    {TextEdit Scrap -> Desk Scrap}
```

```
{-----}
```

{FILE Unit2.iPas: Interface Part for Unit2.code}

```
{ *****
#
#   Programme conçu et développé
#   par Jean-Marie Choffray ,
#   avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
# ***** }
```

{-----}

```
PROCEDURE mainLoop; EXTERNAL;
{One pass of main program loop}
PROCEDURE fixCursor; EXTERNAL;
{Adjust cursor for region of screen}
PROCEDURE doEvent; EXTERNAL;
{Get and process one event}
PROCEDURE doMouseDown; EXTERNAL;
{Handle mouse down events}
PROCEDURE doMenuClick; EXTERNAL;
{Handle click in menu bar}
PROCEDURE doMenuChoice (menuChoice : Longint); EXTERNAL;
{Handle user's menu choice}
PROCEDURE doAppleChoice (theItem : INTEGER); EXTERNAL;
{Handle choice from Apple menu}
PROCEDURE doAbout; EXTERNAL;
{Handle About ProduitTest... command}
PROCEDURE menusDesk; EXTERNAL;
{Activates desk accessory menu items}
```

{-----}

{FILE Unit3.iPas: Interface Part for Unit3.code}

```
{ *****
#
#      Programme conçu et développé      #
#      par Jean-Marie Choffray,         #
#      avec l'aide de la FNEGE          #
#
# Copyright 1987      Tous droits réservés #
#
# ***** }
```

{-----}

PROCEDURE doFileChoice (theItem : INTEGER); EXTERNAL;
 {Handle choice from File menu}

PROCEDURE doNew; EXTERNAL;

{Handle New command}

PROCEDURE menusEdit; EXTERNAL;

{Activates only Edit menu items}

PROCEDURE initAppData; EXTERNAL;

{Initialize application data record}

PROCEDURE doOpen; EXTERNAL;

{Handle Open... command}

PROCEDURE openFile (fileName : Str255; volRefNum : INTEGER); EXTERNAL;

{Open and read selected data file}

PROCEDURE ioCheck (resultCode : OSErr); EXTERNAL;

{Check for I/O error}

PROCEDURE readFile (filePathNum : INTEGER); EXTERNAL;

{Read application data file}

PROCEDURE decipherFile (theString : Str255); EXTERNAL;

{Decipher application data file STRING}

PROCEDURE doClose; EXTERNAL;

{Handle Close command}

PROCEDURE closeAppWindow; EXTERNAL;

{Close application window}

PROCEDURE closeSysWindow; EXTERNAL;

{Close system window}

PROCEDURE doSave; EXTERNAL;

{Handle Save command}

PROCEDURE doSaveAs; EXTERNAL;

{Handle SaveAs command}

PROCEDURE writeFile (theFile : INTEGER; volNum : INTEGER); EXTERNAL;

{Write application data to file}

PROCEDURE doPrint; EXTERNAL;

{Handle Print command}

PROCEDURE doQuit; EXTERNAL;

{Handle Quit command}

{-----}

{FILE Unit4.iPas: Interface Part for Unit4.code}

```
{ *****
 *
 *      Programme conçu et développé      *
 *      par Jean-Marie Choffray,          *
 *      avec l'aide de la FNEGE           *
 *
 * Copyright 1987      Tous droits réservés *
 *
 ***** }
```

{-----}

```
PROCEDURE doEditChoice (theItem : INTEGER); EXTERNAL;
  {Handle choice from Edit menu}
PROCEDURE doUndo; EXTERNAL;
  {Handle Undo command}
PROCEDURE doCut; EXTERNAL;
  {Handle Cut command}
PROCEDURE windowDirty (isDirty : BOOLEAN); EXTERNAL;
  {Mark window as dirty or clean}
PROCEDURE doCopy; EXTERNAL;
  {Handle Copy command}
PROCEDURE doPaste; EXTERNAL;
  {Handle Paste command}
PROCEDURE doClear; EXTERNAL;
  {Handle Clear command}
```

{-----}

{FILE Unit5.iPas: Interface Part for Unit5.code}

```
{ *****
#
#      Programme conçu et développé      #
#      par Jean-Marie Choffray,         #
#      avec l'aide de la FNEGE          #
#                                       #
# Copyright 1987      Tous droits réservés #
#                                       #
***** }
```

```
PROCEDURE dosimulationChoice (theItem : INTEGER); EXTERNAL;
    {Handle choice from Simulation menu}
PROCEDURE miniClean; EXTERNAL;
    {Clean inside of application frame}
PROCEDURE doProduct; EXTERNAL;
    {Handle Product command}
PROCEDURE setScreenProduct; EXTERNAL;
    {Screen layout for product parameters}
PROCEDURE blackCircle ( h, v : INTEGER); EXTERNAL;
    {Draw a little black circle at h, v-6, h+6, v}
PROCEDURE miniCopyright; EXTERNAL;
    {Write mini copyright at bottom of the screen}

PROCEDURE doInnovation; EXTERNAL;
    {Handle Innovation command}
PROCEDURE setScreenEstimate; EXTERNAL;
    {Screen layout for estimation purpose}
PROCEDURE Question; EXTERNAL;
    {Write question label}
PROCEDURE plotEstimator (estimatorPercent: LongInt ); EXTERNAL;
    {Draw gray subjective rectangle}
PROCEDURE writeEstimator ( estimatorPercent: LongInt ); EXTERNAL;
    {Write percent value of top of gray box}

PROCEDURE doImitation; EXTERNAL;
    {Handle Imitation command}

PROCEDURE doInfluence; EXTERNAL;
    {Handle Influence command}

PROCEDURE doPotential; EXTERNAL;
    {Handle Potential command}

PROCEDURE doReplace; EXTERNAL;
    {Handle Replace command}

PROCEDURE doExternalVC; EXTERNAL;
    {Handle ExternalVC command}

PROCEDURE doInternalVC; EXTERNAL;
    {Handle InternalVC command}

PROCEDURE doLearning; EXTERNAL;
    {Handle Learning command}
```

```
PROCEDURE doEnvironment; EXTERNAL;  
    {Handle Environment command}  
PROCEDURE setScreenEnvironment; EXTERNAL;  
    {Screen layout for environment parameters}
```

{-----}

{FILE Unit6.iPas: Interface Part for Unit6.code}

```
{ *****
#
#      Programme conçu et développé      #
#      par Jean-Marie Choffray,         #
#      avec l'aide de la FNEGE          #
#                                       #
# Copyright 1987      Tous droits réservés #
#                                       #
***** }
```

{-----}

PROCEDURE doAnalyzeChoice (theItem : INTEGER); EXTERNAL;
 {Handle choice from Analyze menu}

PROCEDURE doSummary; EXTERNAL;
 {Handle Summary command}

PROCEDURE displaySummary; EXTERNAL;
 {Display input data}

PROCEDURE doInnovationCur; EXTERNAL;
 {Handle InnovationCur command}

PROCEDURE setScreenCurve; EXTERNAL;
 {Set screen display for graphic diagnosis}

PROCEDURE setLabelsCurve; EXTERNAL;
 {Draw curve labels and mini copyright}

PROCEDURE plotInnovation; EXTERNAL;
 {Plot the Innovation curve}

FUNCTION stringToReal (myString : STR255): REAL; EXTERNAL;
 {STR255 -> REAL conversion}

FUNCTION exponential (x, y : REAL): REAL; EXTERNAL;
 {Compute x to the power of y}

PROCEDURE horizontal (y : INTEGER); EXTERNAL;
 {Draw a horizontal line at (230 - y)}

PROCEDURE dottedLine (y : INTEGER); EXTERNAL;
 {Draw a dotted line at (230 - y)}

PROCEDURE setActiveCircle (i: INTEGER; y, range, base : REAL); EXTERNAL;
 {Local coordinates of Cursor active circles 'i'}

PROCEDURE drawActiveCircle (i: INTEGER); EXTERNAL;
 {Draw Cursor active circle 'i'}

PROCEDURE doImitationCur; EXTERNAL;
 {Handle ImitationCur command}

PROCEDURE plotImitation; EXTERNAL;
 {Plot the Imitation curve}

PROCEDURE doPenetrationCur; EXTERNAL;
 {Handle PenetrationCur command}

PROCEDURE plotPenetration; EXTERNAL;
 {Plot the Penetration curve}

PROCEDURE doReplaceCur; EXTERNAL;
 {Handle ReplaceCur command}

PROCEDURE plotReplace; EXTERNAL;
 {Plot the Replace curve}

```
PROCEDURE doVolumeCur; EXTERNAL;  
  {Handle VolumeCur command}  
PROCEDURE plotVolume; EXTERNAL;  
  {Plot the Volume curve}
```

```
PROCEDURE doInternalVCCur; EXTERNAL;  
  {Handle InternalVCCur command}  
PROCEDURE plotInternalVC; EXTERNAL;  
  {Plot the InternalVC curve}
```

```
PROCEDURE doTotalCCur; EXTERNAL;  
  {Handle TotalCCur command}  
PROCEDURE plotTotalIC; EXTERNAL;  
  {Plot the TotalIC curve}
```

```
PROCEDURE doProfitCur; EXTERNAL;  
  {Handle ProfitCur command}  
PROCEDURE plotProfit; EXTERNAL;  
  {Plot the Profit curve}
```

{-----}

{FILE Unit7.iPas : Interface Part for Unit7.code}

```
{ *****
#
#      Programme conçu et développé      #
#      par Jean-Marie Choffray,          #
#      avec l'aide de la FNEGE           #
#                                         #
# Copyright 1987      Tous droits réservés #
#                                         #
***** }
```

{-----}

```
PROCEDURE doContent (whichWindow : WindowPtr); EXTERNAL;
    {Handle click in content region}
PROCEDURE menusEvaluation; EXTERNAL;
    {Activate evaluation menus items}
PROCEDURE adjustEstimator (mouseLocation : Point); EXTERNAL;
    {Adjust estimator's gray rectangle}
PROCEDURE menusSimulation; EXTERNAL;
    {Activate simulation menus items}
PROCEDURE fixSimulationMenu; EXTERNAL;
    {Update simulation menus items}
PROCEDURE displayCurEstimate; EXTERNAL;
    {Display curve's ordinate and abscissa}
PROCEDURE drawCurLabels (x : INTEGER; y : REAL); EXTERNAL;
    {Draw labels of curve's estimates}
PROCEDURE placeSpace (theValue : REAL; VAR theString : STR255); EXTERNAL;
    {Place ' 's when needed in a string}
PROCEDURE realToScientific (theValue : REAL; VAR theString : STR255); EXTERNAL;
    {REAL -> Scientific conversion}
PROCEDURE checkInput (VAR continue : BOOLEAN); EXTERNAL;
    {Check inputs and post alerts if need be}
PROCEDURE activateProduct (mousePoint : Point); EXTERNAL;
    {Deactivate/activate appropriate Edit record}
PROCEDURE deActivateProduct; EXTERNAL;
    {Deactivate appropriate Edit record}
PROCEDURE activateEnvironmt (mousePoint : Point); EXTERNAL;
    {Deactivate/activate appropriate Edit record}
PROCEDURE deActivateEnvironmt; EXTERNAL;
    {Deactivate appropriate Edit record}
PROCEDURE doSelect (thePoint : Point; theText : TEHandle); EXTERNAL;
    {Handle mouse-down in text rectangle}
PROCEDURE fixEditMenu; EXTERNAL;
    {Update edit menu items}

PROCEDURE doGoAway (whichWindow : WindowPtr); EXTERNAL;
    {Handle click in goaway region}

PROCEDURE doKeyStroke; EXTERNAL;
    {Handle keystroke events}
PROCEDURE doTyping; EXTERNAL;
    {Handle character typed from keyboard}

PROCEDURE doUpdate; EXTERNAL;
    {Handle update events}
```

```
PROCEDURE doActivate; EXTERNAL;  
  {Handle activate events}
```

```
{-----}
```

8.1. Unit1.Pas.

```

PROGRAM Unit1;                                {FILE Unit1.Pas: External Unit1 definition}

( *****
#
#      Programme conçu et développé          #
#      par Jean-Marie Choffray,              #
#      avec l'aide de la FNEGE               #
#
# Copyright 1987      Tous droits réservés  #
#
***** )

{$i HD:Pascal2:M1:MemTypes.ipas }             {TML interface to General Purpose TYPES}
{$i HD:Pascal2:M1:QuickDraw.ipas }            {TML interface to Macintosh Graphics Package}
{$i HD:Pascal2:M1:OSIntf.ipas }               {TML interface to the Operating System}
{$i HD:Pascal2:M1:ToolIntf.ipas }             {TML interface to Macintosh User Interface}
{$i HD:Pascal2:M1:PackIntf.ipas }             {TML interface to Macintosh Packages}
{$i HD:Pascal2:M1:MacPrint.ipas }             {TML interface to Macintosh Printing Mgr}
                                           {TML-RM-B-2, TML-UG-4-6}

{$i HD:Pascal2:Program:Globals.iPas }          {Declaration COMMON to Unit1... Unit7, and}
                                           {ProduitTest}

{-----}

PROCEDURE menusEdit; EXTERNAL;
  {Unit3: Activate Edit menu items}
PROCEDURE InitAppData; EXTERNAL;
  {Unit3: Initialize application data record}
PROCEDURE openFile (fileName : Str255; vNum : INTEGER); EXTERNAL;
  {Unit3: Open and read selected data file}

{-----}

PROCEDURE initialize; FORWARD;
  {One-time-only initialization}
PROCEDURE crash; FORWARD;
  {Return to Finder if major crash}
PROCEDURE setUpCursors; FORWARD;
  {Set up and lock cursors}
PROCEDURE setUpMenus; FORWARD;
  {Create program menus}
PROCEDURE menusFirst; FORWARD;
  {DeskTop menu bar}
PROCEDURE doStartup; FORWARD;
  {Process Finder startup information}
PROCEDURE readDeskScrap; FORWARD;
  {Desk Scrap -> TextEdit Scrap, adapt PasteItem}
PROCEDURE writeDeskScrap; FORWARD;
  {TextEdit Scrap -> Desk Scrap}

{-----}

{$S Code}                                {PROCEDURES and FUNCTIONS in Unit1.Code}

{-----}

PROCEDURE initialize;                      {One-time-only initialization}

```

VAR

theMask	:	INTEGER;	{New value for System Event Mask EM-16}
theMessage	:	INTEGER;	{Open 'O' or Print '1' from Finder SL-8}
nDocs	:	INTEGER;	{Nber of documents selected from Finder}

BEGIN {initialize}

MaxApplZone;	{Expand appl Heap zone to its limits MM-23}
MoreMasters;	{Extra pointer space at bottom of heap MM-30}
MoreMasters;	{This is 4 X 64 master pointers}
MoreMasters;	
MoreMasters;	

InitGraf(@thePort);	{Initialize QuickDraw global variables QD-34}
InitFonts;	{Read System Font if not in memory FM-9}
InitWindows;	{Init Window Mgr, draw desktop and empty}
	{menu bar WM-20}

InitMenus;	{Allocate space for Menu List, redraw}
	{empty menu bar MN-15}

TEInit;	{Init TextEdit, allocate handle to TextEdit}
	{Scrap TE-16}

InitDialogs (@crash);	{Init Dialog Mgr, point to restart procedure}
	{"crash" after a System error, DM-18}

FlushEvents (EveryEvent,0);	{Empty Event queue of any event left OE-5}
-----------------------------	--

setUpCursors;	{U1 : Set up and lock cursors}
InitCursor;	{Ready, set cursor to 'arrow' QD-39}

theMask := EveryEvent-KeyUpMask-MUPMask;	{Disable key-up and mouse-up events EM-16}
SetEventMask (theMask);	{Set Sys Evt Mask to all events OE-7, EM-16}

CountAppFiles (theMessage, nDocs);	{Decipher Finder information SL-8}
setUpMenus;	{U1 : Create program menus}
IF (nDocs = 0) THEN	{No document selected from Finder}
BEGIN	
menusFirst;	{U1 : DeskTop menu bar}
DrawMenuBar;	{Redraw MnBar according to menu list MN-19}
END; {IF}	

theText := NIL;	{Clear global pointers/handles}
theWindow := NIL;	

simulation := FALSE;	{Initialize simulation flag before 'doStartUp'}
errorFlag := FALSE;	{Initialize error flags before 'doStartUp'}

quitting := FALSE;	{Initialize quitting flags before 'doStartUp'}
finished := FALSE;	

scrapDirty := FALSE;	{TextEdit and Desk Scrap initially agree}
scrapCompare := InfoScrap^.ScrapCount + 1;	{Initial value of Scrap count SM-10, SM-14}

doStartUp;	{U1 : Process Finder startup information}
------------	---

END; {initialize}

```

PROCEDURE crash;                                {Return to Finder if major crash}

    BEGIN {crash}

        HUnlock(Handle(iBeam));                  {Unlock iBeam relocatable block MM-42}
        HUnlock(Handle(cross));                  {IMPORTANT Handle(cross) creates a handle}
        HUnlock(Handle(watch));                  {pointing to the same object as 'cross' MI-9}

        ExitToShell;                             {Release appl. Heap, return to Finder SL-9}

    END; {crash}

{-----}

PROCEDURE setUpCursors;                          {Set up and lock cursors}

    BEGIN {setUpCursors}

        iBeam := GetCursor (iBeamCursor);      {iBeamCursor from system res file TU-11}
        cross := GetCursor (CrossCursor);      {CrossCursor from system res file TU-11}
        watch := GetCursor (WatchCursor);      {WatchCursor from system res file TU-11}

        HLock(Handle(iBeam));                    {Lock iBeam relocatable block MM-42}
        HLock(Handle(cross));                    {IMPORTANT Handle(cross) creates a handle}
        HLock(Handle(watch));                    {pointing to the same object as 'cross' MI-9}

    END; {setUpCursors}

{-----}

PROCEDURE setUpMenus;                           {Create program menus}

    BEGIN {setUpMenus}

        appleMenu := GetMenu (appleID);        {Get Apple menu from resource file MN-16}
        AddResMenu (appleMenu, 'DRVR');        {Add names of desk accessories MN-18}
        InsertMenu (appleMenu, 0);              {Install at end of menu bar MN-19}

        fileMenu := GetMenu (fileID);          {Get File menu from resource file MN-16}
        InsertMenu (fileMenu, 0);               {Install at end of menu bar MN-19}

        editMenu := GetMenu (editID);          {Get Edit menu from resource file MN-16}
        InsertMenu (editMenu, 0);               {Install at end of menu bar MN-19}

        simulationMenu := GetMenu (simulationID); {Get Simulation menu from resource MN-16}
        InsertMenu (simulationMenu, 0);         {Install at end of menu bar MN-19}

        analyzeMenu := GetMenu (analyzeID);    {Get Analyze menu from resource file MN-16}
        InsertMenu (analyzeMenu, 0);            {Install at end of menu bar MN-19}

    END; {setUpMenus}

{-----}

PROCEDURE menusFirst;                           {DeskTop menu bar}

    BEGIN {menusFirst}

```



```

ELSE FOR thisDoc := 1 TO nDocs DO           {Otherwise loop through documents}
  BEGIN
    GetAppFiles (thisDoc, docInfo);         {Return information on 'thisDoc' SL-8}
    WITH docInfo DO

      IF fType = 'FGFI' THEN                {This is a File of type 'FGFI'}
        BEGIN
          simulation := FALSE;              {Set simulation flag to FALSE}
          theScreen := product;             {Set screen flag}
          textRecd := none;                 {Set Text edit flag}

          menusEdit;                        {U3: Activate Edit menu items}
                                           {No need for DrawMenuBar as it is followed}
                                           {by GetNewWindow and ShowWindow}

          theWindow := GetNewWindow ( windowID, @wdRecord, POINTER(-1));
          SetWindowTitle (theWindow, fName); {File name becomes window name WM-23}
          ShowWindow (theWindow);           {Show application window and place WM-24}
                                           {ACTIVATE and UPDATE Evts in queue}
          SetPort (theWindow);              {Set current GrafPort to theWindow QD-36}
          windowOn := TRUE;                 {Set application window flag}

          readDeskScrap;                    {U1: Desk Scrap -> TextEdit Scrap}
                                           {adapt PasteItem}
          TextFont (Geneva);                {Set current graphPort's Font QD-43, FM-6}
          TextSize (12);                    {Set grafPort type size to 12 QD-43}
          TextFace (1);                     {Select normal characters QD-43}
          TextMode (srcOr);                  {Set grafPort transfer mode QD-43, QD-78}

          initAppData;                      {U3: Initialize application data record}
          openFile (fName, vRefNum);        {U3: Open and read selected data file}
          ClrAppFiles (thisDoc);            {Tell Finder that 'thisDoc' is processed SL-8}

          GOTO 1001;                        {Exit doStartUp, process only one document!}
        END {THEN}

      ELSE                                  {The file type is NOT 'FGFI'}
        BEGIN
          ParamText (fName, "", "");        {File name into text of alert DL-30}
          ignore := StopAlert (wrongTypeID, NIL);
                                           {Post 'WrgTypeID' alert DM-28}

        END; {ELSE}
      END; {ELSE}

1001: END; {doStartUp}

-----}

PROCEDURE readDeskScrap;                   {Desk Scrap -> TextEdit Scrap}
                                           {adapt PasteItem}

VAR

  scrapLength : LongInt;                  {Length of Desk Scrap in bytes}
  ignore      : LongInt;                  {Dummy variable for Scrap offset}
  result      : OSErr;                    {Result code from Scrap transfer OU-9}

BEGIN {readDeskScrap}

```

```

IF scrapCompare <> InfoScrap^.ScrapCount THEN
    BEGIN
        scrapLength := GetScrap (NIL, 'TEXT', ignore);
        {Get length of Text Scrap data SM-12}
        {There is a Text item}
        IF scrapLength >= 0 THEN
            BEGIN
                result := TEFFromScrap;
                {Copy to TextEdit Scrap TE-23}
                IF result <> NoErr THEN
                    {Was there an error}
                    scrapLength := result;
                    {Set negative length OU-9, OU-31}
                END; {IF}

                IF scrapLength > 0 THEN
                    {There is a nonempty TEXT item}
                    BEGIN
                        IF (theScreen = product) OR (theScreen = environmt) THEN
                            EnableItem (editMenu, pasteItem); {Enable Paste command MN-24}
                        END {THEN}
                    ELSE
                        BEGIN
                            {There is an empty Text item}
                            TETSetScrapLen (0);
                            {Set TextEdit Scrap to 0 TE-24}
                            IF (theScreen = product) OR (theScreen = environmt) THEN
                                DisableItem (editMenu, pasteItem);
                            END; {ELSE}
                            {Disable Paste command MN-23}
                        END;
                    END;

                    scrapCompare := InfoScrap^.ScrapCount; {Save Scrap count for later use}

                END; {IF}
            END; {readDeskScrap}
        }-----}

PROCEDURE writeDeskScrap;
    {TextEdit Scrap -> Desk Scrap}

    VAR
        result : OSErr;
        {Result code from Scrap transfer OU-9}

    BEGIN {writeDeskScrap}

        IF scrapDirty THEN
            {Scrap changed since last read?}
            BEGIN
                scrapCompare := ZeroScrap;
                {Change scrapCount, save for later}
                {comparison SM-14}
                result := TETToScrap;
                {Copy TextEdit Scrap to Desk Scrap TE-23}
                scrapDirty := FALSE;
                {TextEdit and Desk Scraps agree}

            END; {IF}
        END; {writeDeskScrap}
    }-----}

{$S Main}
    {Empty PROCEDURE Unit1.Main}
    {To satisfy compiler syntax}
    BEGIN
    END.

```

8.2. Unit2.Pas.

```

PROGRAM Unit2;                                {FILE Unit2.Pas: External Unit2 definition}

{ ***** }
#
#      Programme conçu et développé          #
#      par Jean-Marie Choffray,              #
#      avec l'aide de la FNEGE               #
#
# Copyright 1987      Tous droits réservés  #
#
{ ***** }

{$i HD:Pascal2:MI:MemTypes.ipas }             {TML interface to General Purpose TYPES}
{$i HD:Pascal2:MI:QuickDraw.ipas }            {TML interface to Macintosh Graphics Package}
{$i HD:Pascal2:MI:OSIntf.ipas }              {TML interface to the Operating System}
{$i HD:Pascal2:MI:ToolIntf.ipas }            {TML interface to Macintosh User Interface}
{$i HD:Pascal2:MI:PackIntf.ipas }            {TML interface to Macintosh Packages}
{$i HD:Pascal2:MI:MacPrint.ipas }            {TML interface to Macintosh Printing Mgr}

{$i HD:Pascal2:Program:Globals.iPas }         {Declaration COMMON to Unit1 ... Unit7, and}
                                              {ProduitTest}

{-----}

PROCEDURE menusFirst; EXTERNAL;
  {Unit1: DeskTop menu bar}

PROCEDURE doFileChoice (theItem: INTEGER); EXTERNAL;
  {Unit3: Handle choice from File menu}
PROCEDURE doClose; EXTERNAL;
  {Unit3: Handle Close command}

PROCEDURE doEditChoice (theItem: INTEGER); EXTERNAL;
  {Unit4: Handle choice from Edit menu}

PROCEDURE doSimulationChoice (theItem: INTEGER); EXTERNAL;
  {Unit5: Handle choice from Simulation menu}
PROCEDURE doAnalyzeChoice (theItem: INTEGER); EXTERNAL;
  {Unit5: Handle choice from Analyze menu}

PROCEDURE doKeyStroke; EXTERNAL;
  {Unit7: Handle keystroke events}
PROCEDURE doUpdate; EXTERNAL;
  {Unit7: Handle update events}
PROCEDURE doActivate; EXTERNAL;
  {Unit7: Handle activate events}
PROCEDURE doContent (whichWindow: windowPtr); EXTERNAL;
  {Unit7: Handle click in content region}
PROCEDURE doGoAway (whichWindow: windowPtr); EXTERNAL;
  {Unit7: Handle click in go-away region}

{-----}

PROCEDURE mainLoop; FORWARD;
  {One pass of main program loop}
PROCEDURE fixCursor; FORWARD;
  {Adjust cursor for region of screen}
PROCEDURE doEvent; FORWARD;

```

```

{Get and process one event}
PROCEDURE doMouseDown; FORWARD;
{Handle mouse down events}
PROCEDURE doMenuClick; FORWARD;
{Handle click in menu bar}
PROCEDURE doMenuChoice (menuChoice : LongInt); FORWARD;
{Handle user's menu choice}
PROCEDURE doAppleChoice (theItem : INTEGER); FORWARD;
{Handle choice from Apple menu}
PROCEDURE doAbout; FORWARD;
{Handle About ProductTest... command}
PROCEDURE menusDesk; FORWARD;
{Activates desk accessory menu items}

```

```

-----
{$S Code}                                {PROCEDURES and FUNCTIONS in Unit2.Code}
-----

```

```

PROCEDURE mainLoop;                                {One pass of main program loop}

BEGIN {mainLoop}

  IF FrontWindow = NIL THEN                        {Is deskTop empty WM-26}
  BEGIN
    IF deskOn THEN                                {Coming from a desk accessory}
    BEGIN
      menusFirst;                                {U1 : DeskTop menu bar}
      DrawMenuBar;                                {Redraws MnBar according to menu list MN-19}
      deskOn := FALSE;                            {No desk accessory active}
    END {THEN}
  ELSE                                              {DeskTop still empty}
    IF windowOn THEN                              {Coming from an application window}
    BEGIN
      menusFirst;                                {U1 : DeskTop menu bar}
      DrawMenuBar;                                {Redraw MnBar according to menu list MN-19}
      windowOn := FALSE;                          {No window active}
    END {THEN}
  ELSE                                              {DeskTop is remaining empty}
    menusFirst;                                    {U1 : DeskTop menu bar}
  END {THEN}

  ELSE                                              {FrontWindow <> NIL WM-26}
  IF FrontWindow <> theWindow THEN
  BEGIN
    IF windowOn THEN                              {Coming from an application window}
    BEGIN
      menusDesk;                                  {U2 : Desk accessory menu items}
      DrawMenuBar;                                {Redraw MnBar according to menu list MN-19}
      windowOn := FALSE;                          {No window active}
    END {THEN}
  ELSE                                              {DeskTop contains accessories}
    menusDesk;                                    {U2 : Desk accessory menu items}
  END; {THEN}

  fixcursor;                                       {U2 : Adjust cursor for region of screen}
  SystemTask;                                     {Periodic system actions e.g. clock DS-10}

```

```

IF (theScreen = product) OR (theScreen = environmt) THEN
  BEGIN
    IF theText <> NIL THEN
      TEIdle (theText);
    END; {IF}
    doEvent;
  END; {mainLoop}

{-----}

PROCEDURE fixCursor;
  LABEL
    1002;
  VAR
    theData      : apiHandle;
    dataHandle   : Handle;
    mousePoint   : Point;
    i            : INTEGER;
  BEGIN {fixCursor}
    IF quitting THEN
      GOTO 1002;
    IF FrontWindow = NIL THEN
      BEGIN
        InitCursor;
        GOTO 1002;
      END; {IF}
    IF FrontWindow = theWindow THEN
      BEGIN
        dataHandle := Handle (GetWRefCon (theWindow));
        Hlock (dataHandle);
        theData := apiHandle (dataHandle);
        GetMouse (mousePoint);
        CASE theScreen OF
          product:
            WITH theData^^ DO
              IF PtInRect (mousePoint, divisionEditRecd^^.viewRect) THEN
                SetCursor (iBeam^^)
              ELSE IF PtInRect (mousePoint, brandEditRecd^^.viewRect) THEN
                SetCursor (iBeam^^)
              ELSE IF PtInRect (mousePoint, marketEditRecd^^.viewRect) THEN
                SetCursor (iBeam^^)
              ELSE IF PtInRect (mousePoint, evaluatorEditRecd^^.viewRect) THEN
                SetCursor (iBeam^^)
              ELSE

```

```

InitCursor;                                {Set cursor to 'arrow' QD-39}

innovation, imitation, influence, potential, replace, externalVC,
internalVC, learning:                      {Handle evaluation screens}
  IF PtInRect (mousePoint, activeRect) THEN
    SetCursor (cross^^)                    {Mouse in activeRect QD-47, QD-39}
  ELSE
    InitCursor;                            {Set cursor to 'arrow' QD-39}

environmnt:                                {Handle environmnt screen}
  WITH theData^^ DO
    IF PtInRect (mousePoint, adopterEditRecd^^.viewRect) THEN
      SetCursor (iBeam^^)                  {Mouse in Adopter Rectangle QD-47, QD-39}
    ELSE IF PtInRect (mousePoint, fCostEditRecd^^.viewRect) THEN
      SetCursor (iBeam^^)                  {Mouse in Fixed cost Rectangle QD-47, QD-39}
    ELSE IF PtInRect (mousePoint, productionEditRecd^^.viewRect) THEN
      SetCursor (iBeam^^)                  {Mouse in Production Rectangle QD-47, QD-39}
    ELSE IF PtInRect (mousePoint, priceEditRecd^^.viewRect) THEN
      SetCursor (iBeam^^)                  {Mouse in Price Rectangle QD-47, QD-39}
    ELSE
      InitCursor;                          {Set cursor to 'arrow' QD-39}

innovationPlot, imitationPlot, penetrationPlot, replacePlot,
volumePlot, internalVCPlot, totalCPlot, profitPlot:
  IF PtInRect (mousePoint, cursCircles[1].circle) AND
    ((theScreen <> imitationPlot) AND (theScreen <> replacePlot)) THEN
    SetCursor (cross^^)                    {Mouse in cursCircles[1] QD-47}
  ELSE IF (PtInRect (mousePoint, cursCircles[2].circle)) OR
    (PtInRect (mousePoint, cursCircles[3].circle)) OR
    (PtInRect (mousePoint, cursCircles[4].circle)) OR
    (PtInRect (mousePoint, cursCircles[5].circle)) THEN
    SetCursor (cross^^)                    {Mouse in cursCircles[1..5] QD-47}
  ELSE IF (PtInRect (mousePoint, cursCircles[6].circle)) OR
    (PtInRect (mousePoint, cursCircles[7].circle)) OR
    (PtInRect (mousePoint, cursCircles[8].circle)) OR
    (PtInRect (mousePoint, cursCircles[9].circle)) OR
    (PtInRect (mousePoint, cursCircles[10].circle)) THEN
    SetCursor (cross^^)                    {Mouse in cursCircles[6..10] QD-47}
  ELSE IF (PtInRect (mousePoint, cursCircles[11].circle)) OR
    (PtInRect (mousePoint, cursCircles[12].circle)) OR
    (PtInRect (mousePoint, cursCircles[13].circle)) OR
    (PtInRect (mousePoint, cursCircles[14].circle)) OR
    (PtInRect (mousePoint, cursCircles[15].circle)) THEN
    SetCursor (cross^^)                    {Mouse in cursCircles[10..15] QD-47}
  ELSE
    InitCursor;                            {Set cursor to 'arrow' QD-39}

summary:                                    {Handle summary screen}
  InitCursor;                              {Set cursor to 'arrow' QD-39}

END; {CASE}

HUnlock (dataHandle);                      {Unlock application data record MM-43}

END {IF}

ELSE                                        {A system window is active let desk mgr}
  {do nothing};                            {handle the situation}

```

```
1002: END;    {fixCursor}
```

```
{-----}
```

```
PROCEDURE doEvent;                                {Get and process one Event}

BEGIN {doEvent}

    errorFlag := FALSE;                            {Clear I/O error flag}

    IF GetNextEvent (EveryEvent, theEvent) THEN    {Get next event EM-21}

        CASE theEvent.what OF                      {Event code? EM-11}

            MouseDown:                             {Event code is MouseDown EM-12}
                IF NOT quitting THEN
                    doMouseDown;                    {U2: Handle mouse down events}

            KeyDown, AutoKey:                       {Event code is KeyDown or AutoKey EM-12}
                IF NOT quitting THEN
                    doKeyStroke;                    {U7: Handle keystroke events}

            UpdateEvt:                             {Event code is UpdateEvt EM-12}
                doUpdate;                           {U7: Handle update events}

            ActivateEvt:                           {Event code is ActivateEvt EM-12}
                doActivate;                         {U7: Handle activate events}

        END {CASE}

        ELSE IF quitting AND (theEvent.what = NilEvent) THEN
            BEGIN                                  {Closing up after a Quit command}
                IF FrontWindow <> NIL THEN          {There is a window on the screen}
                    doClose                         {U3: Handle Close command}
                ELSE
                    BEGIN
                        HUnlock(Handle(iBeam));      {Unlock iBeam relocatable block MM-42}
                        HUnlock(Handle(cross));      {IMPORTANT Handle(cross) creates a handle}
                        HUnlock(Handle(watch));      {pointing to the same object as 'cross' MI-09}
                        finished := TRUE;           {Signal end of program}
                    END; {ELSE}
                END; {IF}

        END; {doEvent}
```

```
{-----}
```

```
PROCEDURE doMouseDown;                            {Handle mouse down events}

VAR

    whichWindow : WindowPtr;                       {Window where mouse was pressed}
    thePart     : INTEGER;                         {Part of screen where mouse was pressed}

BEGIN {doMouseDown}

    thePart := FindWindow (theEvent.where, whichWindow);
```



```

        doAppleChoice(theItem);      {U2: Handle choice from Apple menu}
    END;

    fileID:                          {Menu ID for File menu}
    BEGIN
        Delay(30, nTicks);          {System waits for 1/2 second OU-22}
        doFileChoice(theItem);      {U3: Handle choice from File menu}
    END;

    editID:                          {Menu ID for Edit menu}
    BEGIN
        Delay(30, nTicks);          {System waits for 1/2 second OU-22}
        doEditChoice(theItem);      {U4: Handle choice from Edit menu}
    END;

    simulationID:                    {Menu ID for Simulation menu}
    BEGIN
        Delay(30, nTicks);          {System waits for 1/2 second OU-22}
        dosimulationChoice(theItem); {U5: Handle choice from Simulation menu}
    END;

    analyzeID:                       {Menu ID for Analyze menu}
    BEGIN
        Delay(30, nTicks);          {System waits for 1/2 second OU-22}
        doanalyzeChoice(theItem);   {U5: Handle choice from Analyze menu}
    END;

END; {CASE}

HiliteMenu(0);                      {Unhighlight the chosen menu title MM-22}

END; {IF}

END; {doMenuChoice}

{-----}

PROCEDURE doAppleChoice {(theItem : INTEGER)}; {Handle choice from Apple menu}

VAR

    accName   : STR255;              {Name of desk accessory}
    accNber   : INTEGER;             {Reference number of desk accessory}

BEGIN {doAppleChoice}

    CASE theItem OF                  {Item number of selected item}

        aboutItem:                   {Item number for about... command}
            doAbout;                  {U2: Handle About ProduiTest... command}

        OTHERWISE
            BEGIN

                deskOn := TRUE;        {A desk accessory is open}

                IF FrontWindow = NIL THEN {DeskTop is empty WM-26}
                    BEGIN

```

```

        menusDesk;                {U2: Activates desk accessory menu items}
        DrawMenuBar;              {Redraws MnBar according to menu list MN-19}
    END; {IF}

    GetItem (AppleMenu, theItem, accName);
                                   {Accessory name from desk menu item MN-23}
    accNber := OpenDeskAcc (accName); {Open desk accessory DS-07}

    END; {OTHERWISE}

END; {CASE}

END; {doAppleChoice}

{-----}

PROCEDURE doAbout;                {Handle About ProduiTest... command}

VAR

    ignore : INTEGER;             {Item number from About alert}

BEGIN {doAbout}

    ignore := Alert (AboutID, NIL); {Post About ProduiTest... stop alert DL-27}

END; {doAbout}

{-----}

PROCEDURE menusDesk;             {Activates desk accessory menu items}

BEGIN {menusDesk}

    EnableItem (appleMenu, 0);    {Enable Apple menu MN-24}

    EnableItem (fileMenu, 0);     {Initialization of File menu MN-24}
    DisableItem (fileMenu, newItem);
    DisableItem (fileMenu, openItem);

    SetItem (fileMenu, closeItem, 'Fermer'); {Remove fName from Close item MN-23}
    EnableItem (fileMenu, closeItem);
    DisableItem (fileMenu, saveItem);
    DisableItem (fileMenu, saveAsItem);
    DisableItem (fileMenu, printItem);
    DisableItem (fileMenu, quitItem);

    EnableItem (editMenu, 0);      {Enable Edit menu MN-24}
    EnableItem (editMenu, undoItem);
    EnableItem (editMenu, cutItem);
    EnableItem (editMenu, copyItem);
    DisableItem (editMenu, pasteItem);
    EnableItem (editMenu, clearItem);

    DisableItem (simulationMenu, 0); {Disable Simulation menu MN-24}
    DisableItem (analyzeMenu, 0);   {Disable Analyze menu MN-24}

END; {menusDesk}

```

{-----}

{SS Main}

{Empty PROCEDURE Unit2.Main}
{To satisfy compiler syntax}

BEGIN

END.

8.3. Unit3.Pas.

```

PROGRAM Unit3;                                {FILE Unit3.Pas: External Unit3 definition}

( *****
#
#   Programme conçu et développé
#   par Jean-Marie Choffray,
#   avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
***** )

($I HD:Pascal2:M1:MemTypes.ipas )             {TML interface to General Purpose TYPES}
($I HD:Pascal2:M1:QuickDraw.ipas )            {TML interface to Macintosh Graphics Package}
($I HD:Pascal2:M1:OSIntf.ipas )               {TML interface to the Operating System}
($I HD:Pascal2:M1:ToolIntf.ipas )             {TML interface to Macintosh User Interface}
($I HD:Pascal2:M1:PackIntf.ipas )             {TML interface to Macintosh Packages}
($I HD:Pascal2:M1:MacPrint.ipas )             {TML interface to Macintosh Printing Mgr}

($I HD:Pascal2:Program:Globals.iPas )          {Declaration COMMON to Unit1...Unit7, and}
                                                {ProduitTest}

{-----}

PROCEDURE readDeskScrap; EXTERNAL;
  {Unit1: Desk Scrap -> TextEdit Scrap, adapt PasteItem}

PROCEDURE windowDirty (isDirty : BOOLEAN); EXTERNAL;
  {Unit4: Mark window as dirty or clean}

PROCEDURE doActivate; EXTERNAL;
  {Unit7: Handle activate events}

{-----}

PROCEDURE doFileChoice (theItem : INTEGER); FORWARD;
  {Handle choice from File menu}

PROCEDURE doNew; FORWARD;
  {Handle New command}
PROCEDURE menusEdit; FORWARD;
  {Activates Edit menu items}
PROCEDURE initAppData; FORWARD;
  {Initialize application data record}

PROCEDURE doOpen; FORWARD;
  {Handle Open... command}
PROCEDURE openFile (fileName : Str255; voiRefNum : INTEGER); FORWARD;
  {Open and read selected data file}
PROCEDURE ioCheck (resultCode : OSErr); FORWARD;
  {Check for I/O error}
PROCEDURE readFile (filePathNum : INTEGER); FORWARD;
  {Read application data file}
PROCEDURE decipherFile (theString : Str255); FORWARD;
  {Decipher application data file STRING}

PROCEDURE doClose; FORWARD;
  {Handle Close command}

```

```

PROCEDURE closeAppWindow; FORWARD;
    {Close application window}
PROCEDURE closeSysWindow; FORWARD;
    {Close system window}

```

```

PROCEDURE doSave; FORWARD;
    {Handle Save command}

```

```

PROCEDURE doSaveAs; FORWARD;
    {Handle SaveAs... command}
PROCEDURE writeFile (theFile : INTEGER; volNum : INTEGER); FORWARD;
    {Write application data to file}

```

```

PROCEDURE doPrint; FORWARD;
    {Handle Print command}

```

```

PROCEDURE doQuit; FORWARD;
    {Handle Quit command}

```

```

{-----}

```

```

{$S Code}                                {PROCEDURES and FUNCTIONS in Unit3.Code}

```

```

{-----}

```

```

PROCEDURE doFileChoice ((theItem : INTEGER));    {Handle choice from File menu}

```

```

    BEGIN {doFileChoice}

```

```

        CASE theItem OF

```

```

            newItem:      {Globals: item number for New command}
                doNew;      {U3: Handle New command}

```

```

            openItem:     {Globals: item number for Open command}
                doOpen;    {U3: Handle Open command}

```

```

            closeItem:    {Globals: item number for Close command}
                doClose;   {U3: Handle Close command}

```

```

            saveItem:     {Globals: item number for Save command}
                doSave;    {U3: Handle Save command}

```

```

            saveAsItem:   {Globals: item number for SaveAs... command}
                doSaveAs;  {U3: Handle SaveAs... command}

```

```

            printItem:    {Globals: item number for Print command}
                doPrint;   {U3: Handle Print command}

```

```

            quitItem:     {Globals: item number for Quit command}
                doQuit;    {U3: Handle Quit command}

```

```

        END; {CASE}

```

```

    END; {doFileChoice}

```

```

{-----}

```

```

PROCEDURE doNew;                                {Handle New command}

CONST

    windowID = 1001;                            {Resource ID for window template}

BEGIN {doNew}

    simulation := FALSE;                        {Set simulation flag to FALSE}
    theScreen := product;                      {Set screen flag}
    textRecd := none;                          {Set Text edit flag}

    menusEdit;                                {U3: Activate Edit menu items}
                                           {No need for DrawMenuBar as it is followed}
                                           {by GetNewWindow and ShowWindow}

    theWindow := GetNewWindow ( windowID,      {Make new window from template WM-22}
                               @wdRecord,      {Pointer to where to store wd record WM-21}
                               POINTER(-1));    {Place new window up front WM-22}
    ShowWindow (theWindow);                   {Show application window and place}
                                           {ACTIVATE and UPDATE Evts in queue}
    SetPort (theWindow);                     {Set current GrafPort to theWindow QD-36}
    windowOn := TRUE;                        {Set application window flag}

    readDeskScrap;                           {U1: Desk Scrap to TextEdit Scrap}
                                           {Adapt PasteItem}

    TextFont (Geneva);                       {Set current graphPort's Font QD-43, FM-06}
    TextSize (12);                           {Set grafPort type size to 12 QD-43}
    TextFace ([]);                           {Select normal characters QD-43}
    TextMode (srcOr);                         {Set grafPort transfer mode QD-43, QD-78}

    initAppData;                             {U3: Initialize application data record}

END; {doNew}

-----

PROCEDURE menusEdit;                            {Activates Edit menu items}

BEGIN {menusEdit}

    EnableItem (appleMenu, 0);                {Enable Apple menu MN-24}

    EnableItem (fileMenu, 0);                 {Initialization of File menu MN-24}
    DisableItem (fileMenu, newItem);
    DisableItem (fileMenu, openItem);
    EnableItem (fileMenu, closeItem);
    DisableItem (fileMenu, saveItem);
    EnableItem (fileMenu, saveAsItem);
    EnableItem (fileMenu, printItem);
    EnableItem (fileMenu, quitItem);

    EnableItem (editMenu, 0);                 {Enable Edit menu MN-24}
    DisableItem (editMenu, undoItem);
    DisableItem (editMenu, cutItem);
    DisableItem (editMenu, copyItem);
    DisableItem (editMenu, pasteItem);

```

```

DisableItem (editMenu, clearItem);

IF NOT simulation THEN                                {Data entry mode}
  BEGIN
    DisableItem (simulationMenu, 0);                  {Disable Simulation menu MN-23}
    DisableItem (analyzeMenu, 0);                     {Disable Analyze menu MN-23}
  END {THEN}
ELSE                                                    {Simulation mode}
  BEGIN
    EnableItem (simulationMenu, 0);                   {Enable Simulation menu MN-24}
    EnableItem (analyzeMenu, 0);                     {Enable Analyze menu MN-24}
  END; {ELSE}
END; {menusEdit}

{-----}

PROCEDURE initAppData;                                {Initialize application data record}

VAR

  theData      : apiHandle;                          {Handle to application data record}
  dataHandle   : Handle;                             {Untyped handle to create data record MI-9}

  destRect     : Rect;                               {Wrapping rectangle for edit records}
  viewRect     : Rect;                               {Clipping rectangle for edit records}

  textStr      : STRING[30];                         {Text string to be incorporated in record}
  textLength   : LongInt;                            {Length of text string}

BEGIN {initAppData}

  dataHandle := NewHandle (SIZEOF (apiData));          {Allocate application data record MM-32}
  SetWRefCon (theWindow, LongInt (dataHandle));       {Set window reference value WM-33}
  HLock (dataHandle);                                 {Lock application data record MM-42}
  theData := apiHandle (dataHandle);                  {Convert to an apiHandle}

  WITH theData^^ DO

    BEGIN

      SetRect (viewRect, 240, 71, 450, 94);           {Local Rect for Division view Rect LTRB QD-46}
      destRect := viewRect;
      InsetRect (destRect, textMargin, textMargin);
                                                    {Inset wrapping around rectangle QD-47}
      divisionEditRecd := TENew (destRect, viewRect);
                                                    {Create and initialize Edit record TE-16}
      textStr := 'La meilleure';                      {Initialization of Division Text}
      TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), divisionEditRecd);
                                                    {Incorporate copy of textStr TE- 17, TE-22}

      SetRect (viewRect, 240, 111, 450, 134);        {Local Rect for Brand view Rect LTRB QD-46}
      destRect := viewRect;
      InsetRect (destRect, textMargin, textMargin);
                                                    {Inset wrapping around rectangle QD-47}
      brandEditRecd := TENew (destRect, viewRect);
                                                    {Create and initialize Edit record TE-16}
      textStr := 'Le plus grand succès';              {Initialization of Brand Text}
    
```

```

TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), brandEditRecd);
{Incorporate copy of textStr TE- 17, TE-22}

SetRect (viewRect, 240, 151, 450, 174); {Local Rect for Market view Rect LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
{Inset wrapping around rectangle QD-47}
marketEditRecd := TENew (destRect, viewRect);
{Create and initialize Edit record TE-16}
textStr := 'Le monde entier';
{Initialization of Market Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), marketEditRecd);
{Incorporate copy of textStr TE- 17, TE-22}

SetRect (viewRect, 240, 191, 450, 214); {Loc Rect for Evaluator view Rect LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
{Inset wrapping around rectangle QD-47}
evaluatorEditRecd := TENew (destRect, viewRect);
{Create and initialize Edit record TE-16}
textStr := 'Vous !!!';
{Initialization of Evaluator Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), evaluatorEditRecd);
{Incorporate copy of textStr TE- 17, TE-22}

innovationPar := 10; {Initial values of parameters}
imitationPar := 20;
influencePar := 10;
potentialPar := 90;
replacePar := 6;
externalVCPAr := 10;
internalVCPAr := 20;
learningPar := 80;

SetRect (viewRect, 300, 71, 450, 94); {Local Rect for Adopter view Rect LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
{Inset wrapping around rectangle QD-47}
adopterEditRecd := TENew (destRect, viewRect);
{Create and initialize Edit record TE-16}
TESetJust (teJustRight, adopterEditRecd); {Set right justification of Text TE-21}
textStr := '100000';
{Initialization of Adopter Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), adopterEditRecd);
{Incorporate copy of textStr TE- 17, TE-22}

SetRect (viewRect, 300, 111, 450, 134); {Loc Rect for fCost LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
{Inset wrapping around rectangle QD-47}
fCostEditRecd := TENew (destRect, viewRect);
{Create and initialize Edit record TE-16}
TESetJust (teJustRight, fCostEditRecd); {Set right justification of Text TE-21}
textStr := '200';
{Initialization of fCost Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), fCostEditRecd);
{Incorporate copy of textStr TE- 17, TE-22}

SetRect (viewRect, 300, 151, 450, 174); {Local Production view Rect LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
{Inset wrapping around rectangle QD-47}

```

```

productionEditRecd := TENew (destRect, viewRect);
                                {Create and initialize Edit record TE-16}
TESetJust (teJustRight, productionEditRecd);
                                {Set right justification of Text TE-21}
textStr := '50000';
                                {Initialization of Production Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), productionEditRecd);
                                {Incorporate copy of textStr TE- 17, TE-22}

SetRect (viewRect, 300, 191, 450, 214); {Local Rect for Price view Rect LTRB QD-46}
destRect := viewRect;
InsetRect (destRect, textMargin, textMargin);
                                {Inset wrapping around rectangle QD-47}
priceEditRecd := TENew (destRect, viewRect);
                                {Create and initialize Edit record TE-16}
TESetJust (teJustRight, priceEditRecd); {Set right justification of Text TE-21}
textStr := '250';
                                {Initialization of Price Text}
TESetText (POINTER (ORD (@textStr) + 1), LENGTH (textStr), priceEditRecd);
                                {Incorporate copy of textStr TE- 17, TE-22}

dirty      := FALSE;           {Document is initially clean}
fileNumber := 0;               {Window has no associated file}
volNumber  := 0;               {Window has no associated volume}

END; {WITH}

HUnlock (dataHandle);          {Unlock application data record MM-43}

END; {InitAppData}

{-----}

PROCEDURE doOpen;               {Handle Open... command}

CONST

    windowID = 1001;            {Resource ID for window template}

VAR

    dlgOrigin    : Point;       {Top left corner of dialog box QD-7}
    theTypeList  : SFTypelist;  {List of file types to display PK-31}
    theReply     : SFReply;     {Data returned by Get dialog PK-25}

BEGIN {doOpen}

    SetPt (dlgOrigin, dlgLeft, dlgTop); {Set dialog box coordinates QD-65}
    theTypeList [0] := 'FGFI';          {Display only 'FGFI' files PK-31}

    SFGGetFile (dlgOrigin, '', NIL, 1, theTypeList, NIL, theReply);
                                {SFGGetFile dialog box PK-30}

    WITH theReply DO

        IF good THEN            {SFReply data TYPE PK-25}

            BEGIN
                simulation := FALSE;    {Set simulation flag to FALSE}
                theScreen := product;    {Set screen flag}
                textRecd := none;        {Set Text edit flag}
            END
        END
    END

```

```

    menusEdit;                                {U3: Activate Edit menu items}
                                              {No need for DrawMenuBar as it is followed}
                                              {by GetNewWindow and ShowWindow}

    theWindow := GetNewWindow ( windowID, @wdRecord, POINTER(-1));
    SetWTitle (theWindow, fName);              {File name becomes window WM-23}
    ShowWindow (theWindow);                    {Show application window and place}
                                              {ACTIVATE and UPDATE Evts in queue}
    SetPort (theWindow);                       {Set current GrafPort to theWindow QD-36}

    readDeskScrap;                             {U1: Desk Scrap -> TextEdit Scrap}
                                              {Adapt PasteItem}

    TextFont (Geneva);                         {Set current graphPort's Font QD-43, FM-06}
    TextSize (12);                             {Set grafPort type size to 12 QD-43}
    TextFace (II);                             {Select normal characters QD-43}
    TextMode (srcOr);                           {Set grafPort transfer mode QD-43, QD-78}

    initAppData;                               {U3: Initialize application data record}
    openFile (fName, vRefNum);                 {U3: Open and read selected data file}

    END; {IF}

END; {doOpen}

-----
PROCEDURE openFile ((fileName : Str255; voiRefNum : INTEGER));
                                              {Open and read selected data file}

    LABEL

    1002;                                       {Exit openFile LABEL}

    VAR

    theData      : apIHandle;                  {Handle to application data record}
    dataHandle    : Handle;                    {Untyped handle to create data record MI-9}

    filePathNum   : INTEGER;                   {Reference number of file access path}
    resultCode    : OSErr;                     {I/O error code OU-9}
    newCloseStr   : Str255;                    {New 'Close' item string}

    BEGIN {openFile}

        resultCode := FSOpen (fileName, voiRefNum, filePathNum);
                                              {Open the file, get file access path FL-18}
        ioCheck (resultCode);                 {U3: Check for I/O error}
        IF errorFlag THEN                     {There is an I/O error}
            GOTO 1002;                         {Skip rest of operation}

        dataHandle := Handle(GetWRefCon(theWindow));
        Hlock(dataHandle);                    {Get and lock Handle to data record WM-33}
        theData := apIHandle(dataHandle);      {Convert to apIHandle}

        WITH theData^^ DO
            BEGIN
                volNumber := voiRefNum;        {Save volume reference number}

```

```

        fileNumber := filePathNum;           {Save reference number of file access path}
    END; {WITH}

    HUnlock (dataHandle);                    {Unlock application data record MM-43}

    newCloseStr := CONCAT ('Fermer ', fileName);
    SetItem (fileMenu, closeItem, newCloseStr); {Set file name in Close item MN-23}

    readFile (filePathNum);                  {U3: Read application data file}

1002: END; {openFile}

-----}

PROCEDURE ioCheck {(resultCode : OSErr)};    {Check for I/O error OU-9}

    LABEL

        1003;                               {Exit ioCheck LABEL}

    VAR

        errorString : Str255;               {Error code in string form}
        ignore      : INTEGER;              {Item code returned by Stop alert}

    BEGIN {ioCheck}

        IF resultCode = NoErr THEN
            GOTO 1003;                       {Just return if no I/O error OU-09}

        NumToString (resultCode, errorString); {Convert error code to a string PK-20}
        ParamText (errorString, "", "", ""); {Substitute into text of alert DL-30}

        InitCursor;                          {Restore 'arrow' cursor QD-39, as the}
                                                {cursor is set to a 'watch' for read/write}
        ignore := StopAlert (IOErrID, NIL);   {Post alert DL-28}

        quitting := FALSE;                   {Cancel Quit command... if quitting with}
                                                {an unsaved, dirty data file}
        errorFlag := TRUE;                   {Force exit to main event loop}
                                                {doEvent sets errorFlag back to FALSE}

    1003: END; {ioCheck}

-----}

PROCEDURE readFile {(filePathNum : INTEGER)}; {Read application data file}
                                                {filePathNum = Ref Nber of file access path}

    LABEL

        1004;                               {Exit readFile LABEL}

    VAR

        theString : STR255;                 {Data string to be read}
        theLength  : LongInt;               {Length of the file in bytes}

        resultCode : OSErr;                 {I/O error code OU-9}

```

```

BEGIN {readFile}

    SetCursor (watch^^);                {Set watch cursor while reading QD-39}

    resultCode := GetEOF (filePathNum, theLength); {Return logical end-of-file FL-20}
    ioCheck (resultCode);                 {U3: Check for I/O error}
    IF errorFlag THEN                     {There is an I/O error}
        GOTO 1004;                       {Skip rest of operation}

    resultCode := SetFPos (filePathNum, FSFromStart, 0);
                                          {Set mark to first position FL-20}
    IOCheck (resultCode);                 {U3: Check for I/O error}
    IF errorFlag THEN                     {There is an I/O error}
        GOTO 1004;                       {Skip rest of operation}

    resultCode := FSRead (filePathNum, theLength, @theString);
                                          {Read file STRING into theString FL-19}
    IOCheck (resultCode);                 {U3: Check for I/O error}
    IF errorFlag THEN                     {There is an I/O error}
        GOTO 1004;                       {Skip rest of operation}

    resultCode := FSClose(filePathNum);    {Close application data file FM-22}
    IOCheck (resultCode);                 {U3: Check for IO error}
    IF errorFlag THEN                     {There is an I/O error}
        GOTO 1004;                       {Skip rest of operation}

    decipherFile (theString);             {Decipher theString content}

    windowDirty (FALSE);                 {U4: Mark window as clean}

    1004: END;    {readFile}

{-----}

PROCEDURE decipherFile ((theString : Str255));    {Decipher application data file STRING}

VAR

    theData      : apiHandle;             {Handle to application data record}
    dataHandle   : Handle;                {Untyped handle to create data record MI-9}

    workString   : STR255;                {Temporary string working space}
    index        : INTEGER;               {First occurrence of $ in theString}

BEGIN {decipherFile}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);                    {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle);     {Convert to apiHandle}

    WITH theData^^ DO

        BEGIN

            index := POS('$', theString);    {Search first occurrence of $ in theString}
            workString := COPY (theString, 1, index - 1);
                                          {Return Division substring}
            TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), divisionEditRecd);

```

```

DELETE (theString, 2, index - 1);      {Incorporate workString TE- 17, TE-22}
                                       {Delete Division substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return Brand substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), brandEditRecd);
                                       {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);      {Delete Brand substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return Market substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), marketEditRecd);
                                       {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);      {Delete Market substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return Evaluator substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), evaluatorEditRecd);
                                       {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);      {Delete Evaluator substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return innovationPar substring}
StringToNum (workString, innovationPar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);      {Delete innovationPar substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return imitationPar substring}
StringToNum (workString, imitationPar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);      {Delete imitationPar substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return influencePar substring}
StringToNum (workString, influencePar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);      {Delete influencePar substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                       {Return potentialPar substring}
StringToNum (workString, potentialPar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);      {Delete potentialPar substring}

index := POS('$', theString);          {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);

```

```

                                {Return replacePar substring}
StringToNum (workString, replacePar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);    {Delete replacePar substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return externalVCPPar substring}
StringToNum (workString, externalVCPPar);
                                {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);    {Delete externalVCPPar substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return internalVCPPar substring}
StringToNum (workString, internalVCPPar);
                                {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);    {Delete internalVCPPar substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return learningPar substring}
StringToNum (workString, learningPar); {Convert string to LongInt PK-21}
DELETE (theString, 2, index - 1);    {Delete learningPar substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return Adopter substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), adopterEditRecd);
                                {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);    {Delete Adopter substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return fCost substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), fCostEditRecd);
                                {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);    {Delete fCost substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return Production substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), productionEditRecd);
                                {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);    {Delete Production substring}

index := POS('$', theString);        {Search first occurrence of $ in theString}
workString := COPY (theString, 2, index - 2);
                                {Return Price substring}
TESetText (POINTER (ORD (@workString) + 1), LENGTH (workString), priceEditRecd);
                                {Incorporate workString TE- 17, TE-22}
DELETE (theString, 2, index - 1);    {Delete Price substring}

```

```

        END; {WITH}

        HUnlock (dataHandle);                {Unlock application data record MM-43}

    END; {decipherFile}

{-----}

PROCEDURE doClose;                          {Handle close command}

    BEGIN {doClose}

        IF FrontWindow = theWindow THEN    {Is the application window active? WM-26}
            closeAppWindow                  {U3: Close application window}
        ELSE
            closeSysWindow;                 {U3: Close system window}

    END; {doClose}

{-----}

PROCEDURE closeAppWindow;                   {Close application window}

    LABEL

        1005;                              {Exit closeAppWindow LABEL}

    CONST

        saveItem    = 1;                   {Item number for OK/oui button}
        cancelItem  = 2;                   {Item number for cancel/annuler button}
        discardItem = 3;                   {Item number for discard/non button}

    VAR

        theData      : apIHandle;          {Handle to application data record}
        dataHandle    : Handle;             {Untyped handle to create data record MI-9}

        theTitle      : Str255;             {Title of window}
        theItem        : INTEGER;           {Item number for save Alert}

        resultCode    : OSErr;              {I/O error code OU-9}
        thisWindow     : WindowPtr;         {Pointer to window beeing closed WM-11}
        newCloseStr    : Str255;           {New 'Close' item string}

    BEGIN {closeAppWindow}

        dataHandle := Handle(GetWRefCon(theWindow));
        Hlock(dataHandle);                  {Get and lock Handle to data record WM-33}
        theData := apIHandle(dataHandle);   {Convert to apIHandle}

        WITH theData^^ DO

            BEGIN

                IF (dirty) AND (fileNumber <> 0) THEN {There is a window with an associated file}
                    {whose content has been changed}

```

BEGIN

```
GetWTitle(theWindow,theTitle);    {Return window title WM-33}
ParamText(theTitle,"","");       {Window title into text of alert DL-30}
theItem := CautionAlert(saveID,NIL);
                                {Post 'saveID' alert DM-28}
```

CASE theItem OF

```

saveItem:                                     {Item number for OK/oui button}
BEGIN
    doSave;                                  {U3: Save application data on disk}
    IF errorFlag THEN                        {There is an I/O error}
        BEGIN
            HUnlock(dataHandle);             {Unlock application data record MM-43}
            GOTO 1005;                       {Exit closeAppWindow}
        END; {IF}
    END; {saveItem}

```

```
cancelItem:      {Item number for cancel/annuler button}
BEGIN
    quitting := FALSE;      {Cancel Quit command, if any}
    HUnlock(dataHandle);    {Unlock application data record MM-43}
    GOTO 1005;              {Exit closeAppWindow}
END; {cancelItem}
```

```
discardItem:      {item number for discard/non button}
                  {Do nothing};                          {Refuse saving a dirty app window}
```

END; {CASE}

END; {IF}

```

IF fileNumber <> 0 THEN                                {File associated with a 'clean' window?}
  BEGIN
    resultCode := FlushVol (NIL,volNumber);
                                                         {Write volume buffer content FM-12, FM-17}
    ioCheck (resultCode);                                {Check for IO error}
    IF errorFlag THEN                                    {There is an I/O error}
      BEGIN
        HUnlock(dataHandle);                            {Unlock application data record MM-43}
        GOTO 1005;                                       {Skip rest of operation}
      END; {IF}
    END; {IF}
  END; {IF}

```

TEDispose (divisionEditRecd);	{Dispose of division Edit Record TE-17}
TEDispose (brandEditRecd);	{Dispose of brand Edit Record TE-17}
TEDispose (marketEditRecd);	{Dispose of market Edit Record TE-17}
TEDispose (evaluatorEditRecd);	{Dispose of evaluator Edit Record TE-17}

TEDispose (adopterEditRecd);	{Dispose of adopter Edit Record TE-17}
TEDispose (fCostEditRecd);	{Dispose of fCost Edit Record TE-17}
TEDispose (productionEditRecd);	{Dispose of production Edit Record TE-17}
TEDispose (priceEditRecd);	{Dispose of price Edit Record TE-17}

END: {WITH}

```
HUnlock(dataHandle);           {Unlock application data record MM-43}
```

```

SetItem (fileMenu, closeItem, 'Fermer');      {Remove file name from Close item MN-23}

thisWindow := theWindow;                      {Save window pointer (doActivate will change)
                                              {theWindow}}
HideWindow (theWindow);                      {TheWindow -> invisible, FORCE DEACTIVATE}
                                              {and ACTIVATE events NOW! WM-23}
IF GetNextEvent (ActivateEvt, theEvent) THEN {Get and handle DEACTIVATE event}
  doActivate;
IF GetNextEvent (ActivateEvt, theEvent) THEN {Get and handle ACTIVATE event}
  doActivate;

CloseWindow(thisWindow);                     {Close window, release memory WM-22}

DisposHandle(dataHandle);                    {Release memory used by application data}
                                              {record MM-33, set in initAppData}

theText    := NIL;                          {Clear global Pointers}
theWindow  := NIL;

1005: END;   {closeAppWindow}

-----

PROCEDURE closeSysWindow;                    {Close System window}

VAR

  whichWindow  : WindowPeak;                {Pointer for access window fields WM-12}
  accNumber    : INTEGER;                   {Reference number of desk accessory}

BEGIN {closeSysWindow}

  whichWindow := WindowPeak (FrontWindow);  {Convert FrontWindow to WindowPeak WM-12}
  accNumber   := whichWindow.windowKind;    {Get ref number of desk accessory WM-12}
  CloseDeskAcc (accNumber);                 {Close desk accessory DS-08}

END;   {closeSysWindow}

-----

PROCEDURE doSave;                            {Handle Save command}

LABEL

  1020;                                       {Exit doSave label}

VAR

  fileName     : STR255;                    {File name of active window}
  filePathNum  : INTEGER;                   {Reference number of file access path}
  resultCode   : OSErr;                     {I/O error code OU-9}

  theData      : apiHandle;                 {Handle to application data record}
  dataHandle   : Handle;                    {Untyped handle to create data record MI-9}

BEGIN {doSave}

  dataHandle := Handle(GetWRefCon(theWindow));

```

```

Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
theData := ap!Handle(dataHandle); {Convert to ap!Handle}

WITH theData^^ DO
  IF fileNumber = 0 THEN      {No file is associated with the window}
    doSaveAs                  {U3: Handle SaveAs command}
  ELSE                        {A file is associated with the window}
    BEGIN
      GetWTitle(theWindow, fileName); {Get window title (file name) WM-23}
      resultCode := FSDPen(fileName, volNumber, filePathNum);
                                   {Open the file, get file access path FL-18}
      ioCheck(resultCode);        {U3: Check for I/O error}
      IF errorFlag THEN          {There is an I/O error}
        GOTO 1020;              {Skip rest of operation}
      fileNumber := filePathNum;  {Save reference number of file access path}
      writeFile(fileNumber, volNumber); {U3: Write application data to file}
    END; {ELSE}

  HUnlock(dataHandle);         {Unlock application data record MM-43}

1020: END; {doSave}
}-----}

PROCEDURE doSaveAs;           {Handle SaveAs command}

LABEL

  1006;                       {Exit doSaveAs label}

VAR

  theData    : ap!Handle;     {Handle to application data record}
  dataHandle : Handle;        {Untyped handle to create data record MI-9}

  dlgOrigin   : Point;        {Top-left corner of dialog box QD-7}

  theReply    : SFRReply;     {Data returned by put dialog PK-25}
  theInfo     : FInfo;        {File's Finder information FL-11}
  filePathNum : INTEGER;      {File Reference number}

  origName    : STR255;       {Title string for current file}
  newCloseStr : STR255;       {New 'Close' item string}

  ignore      : INTEGER;      {Item code returned by alert}
  resultCode  : OSErr;        {I/O error code OU-9}

BEGIN {doSaveAs}

  dataHandle := Handle(GetWRefCon(theWindow));
  Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
  theData := ap!Handle(dataHandle); {Convert to ap!Handle}
  WITH theData^^ DO
    IF fileNumber <> 0 THEN      {Is a file associated with the window?}
      getWTitle(theWindow, origName) {Get name of associated file WM-23}
    ELSE
      origName := 'sans titre';    {File name is 'sans titre'}

```



```

        GOTO 1007;                                {Skip rest of operation}

theLength := LENGTH (theString) + 1; {Add length byte MI-10, TML-RM-C-2}
resultCode := FSWrite (theFile, theLength, @theString);
                                                {Write theString to open file FL-19}
ioCheck (resultCode);                          {U3: Check for I/O error}
IF errorFlag THEN                              {There is an I/O error}
    GOTO 1007;                                {Skip rest of operation}

theLength := LENGTH (theString) + 1; {Add length byte}
resultCode := SetEOF (theFile, theLength); {Set logical end of file FL-21}
ioCheck (resultCode);                          {U3: Check for I/O error}
IF errorFlag THEN                              {There is an I/O error}
    GOTO 1007;                                {Skip rest of operation}

resultCode := FSClose(theFile);                 {Close application data file FM-22}
ioCheck (resultCode);                          {U3: Check for IO error}
IF errorFlag THEN                              {There is an I/O error}
    GOTO 1007;                                {Skip rest of operation}

resultCode := FlushVol (NIL, volNum);           {Write volume buffer content FM-12, FM-17}
ioCheck (resultCode);                          {U3: Check for IO error}

1007: END;    {writeString}

```

{-----}

BEGIN {writeFile}

```

theString := '';                                {Empty data string}

dataHandle := Handle(GetWRefCon(theWindow));
Hlock(dataHandle);                             {Get and lock Handle to data record WM-33}
theData := apiHandle(dataHandle);               {Convert to apiHandle}

WITH theData^^ DO

    BEGIN

        GetIText (divisionEditRecd^^.hText, workString);
                                                {Return text as a string DL-31}
        theString := CONCAT (theString, workString, '$');

        GetIText (brandEditRecd^^.hText, workString);
        theString := CONCAT (theString, workString, '$');

        GetIText (marketEditRecd^^.hText, workString);
        theString := CONCAT (theString, workString, '$');

        GetIText (evaluatorEditRecd^^.hText, workString);
        theString := CONCAT (theString, workString, '$');

        NumToString (innovationPar, workString);
                                                {Convert FK-20 and write innovationPar}
        theString := CONCAT (theString, workString, '$');

        NumToString (imitationPar, workString);
    
```

```

theString := CONCAT (theString, workString, '$');

NumToString (influencePar, workString);
theString := CONCAT (theString, workString, '$');

NumToString (potentialPar, workString);
theString := CONCAT (theString, workString, '$');

NumToString (replacePar, workString);
theString := CONCAT (theString, workString, '$');

NumToString (externalVCPAr, workString);
theString := CONCAT (theString, workString, '$');

NumToString (internalVCPAr, workString);
theString := CONCAT (theString, workString, '$');

NumToString (learningPar, workString);
theString := CONCAT (theString, workString, '$');


GetText (adopterEditRecd^.hText, workString);
                                {Return text as a string DL-31}
theString := CONCAT (theString, workString, '$');

GetText (fCostEditRecd^.hText, workString);
theString := CONCAT (theString, workString, '$');

GetText (productionEditRecd^.hText, workString);
theString := CONCAT (theString, workString, '$');

GetText (priceEditRecd^.hText, workString);
theString := CONCAT (theString, workString, '$');

END; {WITH}

HUnlock(dataHandle);           {Unlock application data record MM-43}

writeString (theString);       {U3: Write data string to file}
windowDirty (FALSE);          {U4: Mark window as clean}

END; {writeFile}

{-----}

PROCEDURE doPrint;              {Handle Print command}

BEGIN {doPrint}

    PrDrvOpen;                  {Open the Printer Driver PR-24}
    PrCtlCall (iPrEvtCtl, iPrEvtTop, 0, 0); {Print frontmost window PR-28}
    PrDrvClose;                  {Close the Printer Driver PR-24}

END; {doPrint}

{-----}

PROCEDURE doQuit;              {Handle Quit command}

```

```
BEGIN {doQuit}
```

```
    quitting := TRUE;
```

```
{Start closing down windows}
```

```
END; {doQuit}
```

```
{-----}
```

```
{$S Main}
```

```
{Empty PROCEDURE Unit3.Main}
```

```
{To satisfy compiler syntax}
```

```
BEGIN
```

```
END.
```

8.4. Unit4.Pas.

BEGIN {doEditChoice}

CASE theItem OF

```

undoItem:                                {Globals: item number for Undo command}
  IF NOT SystemEdit (undoCmd) THEN        {Not a desk accessory editing task DS-9}
    doUndo;                               {U4: Handle Undo command}

cutItem:                                 {Globals: item number for Cut command}
  BEGIN
    IF SystemEdit (cutCmd) THEN            {Desk accessory editing task DS-9}
      BEGIN
        EnableItem (EditMenu, CutItem);  {Enable relevant menu items MN-23, MN-24}
        EnableItem (EditMenu, CopyItem);
        EnableItem (EditMenu, ClearItem);
        DisableItem (EditMenu, PasteItem);
      END
    ELSE {THEN}
      doCut;                               {U4: Handle Cut command}
    END; {cutItem}

copyItem:                                {Globals: item number for Copy command}
  BEGIN
    IF SystemEdit (copyCmd) THEN           {Desk accessory editing task DS-9}
      BEGIN
        EnableItem (EditMenu, CutItem);  {Enable relevant menu items MN-23, MN-24}
        EnableItem (EditMenu, CopyItem);
        EnableItem (EditMenu, ClearItem);
        EnableItem (EditMenu, PasteItem);
      END {THEN}
    ELSE
      doCopy;                             {U4: Handle Copy command}
    END; {copyItem}

pasteItem:                               {Globals: item number for Paste command}
  BEGIN
    IF SystemEdit (pasteCmd) THEN          {Desk accessory editing task DS-9}
      BEGIN
        EnableItem (EditMenu, CutItem);  {Enable relevant menu items MN-23, MN-24}
        EnableItem (EditMenu, CopyItem);
        EnableItem (EditMenu, ClearItem);
        DisableItem (EditMenu, PasteItem);
      END {THEN}
    ELSE
      doPaste;                             {U4: Handle Paste command}
    END; {pasteItem}

clearItem:                               {Globals: item number for Clear command}
  BEGIN
    IF SystemEdit (clearCmd) THEN          {Desk accessory editing task DS-9}
      BEGIN
        EnableItem (EditMenu, CutItem);  {Enable relevant menu items MN-23, MN-24}
        EnableItem (EditMenu, CopyItem);
        EnableItem (EditMenu, ClearItem);
        DisableItem (EditMenu, PasteItem);
      END {THEN}
    ELSE

```

```

        doClear;                                {U4: Handle Clear command}
    END; {clearItem}

    END {case}

    END; {doEditChoice}

{-----}

PROCEDURE doUndo;                                {Handle Undo command}

    BEGIN {doUndo}

        {Do nothing}                            {Not implemented in this application}

    END; {doUndo}

{-----}

PROCEDURE doCut;                                {Handle Cut command}

    BEGIN {doCut}

        TECut (theText);                        {Cut the text selection TE-20}

        DisableItem (EditMenu, CutItem);        {Disable non relevant items MN-23, MN-24}
        DisableItem (EditMenu, CopyItem);
        DisableItem (EditMenu, ClearItem);
        EnableItem (EditMenu, PasteItem);

        scrapDirty := TRUE;                      {Mark scrap as dirty}
        windowDirty (TRUE);                     {U4: Mark window as dirty or clean}

    END; {doCut}

{-----}

PROCEDURE windowDirty {(isDirty : BOOLEAN)};    {Mark window as dirty or clean}

    VAR

        theData      : apIHandle;               {Handle to application data record}
        dataHandle    : Handle;                 {Untyped handle to create data record MI-9}

    BEGIN {windowDirty}

        dataHandle := Handle(GetWRefCon(theWindow));
        Hlock(dataHandle);                      {Get and lock Handle to data record WM-33}
        theData := apIHandle(dataHandle);       {Convert to apIHandle}

        WITH theData^^ DO

            BEGIN

                dirty := isDirty;                {Set 'dirty/clean' flag in data record}
                DisableItem (fileMenu, saveItem); {Disable Save command MN-23}

                IF (isDirty) AND (fileNumber <> 0) THEN {Want to set window as 'dirty'}

```

```

        EnableItem (fileMenu, saveItem);      {Enable Save command MN-24}

    END;  {WITH}

    HUnlock(dataHandle);                      {Unlock application data record MM-43}

    END;  {windowDirty}

{-----}

PROCEDURE doCopy;                            {Handle Copy command}

    BEGIN  {doCopy}

        TECopy (theText);                    {Copy the text selection TE-20}

        EnableItem (EditMenu, PasteItem);    {Enable Paste command MN-24}

        scrapDirty := TRUE;                  {Mark scrap as dirty}

    END;  {doCopy}

{-----}

PROCEDURE doPaste;                           {Handle Paste command}

    VAR

        sizeScrap   : INTEGER;               {Length in bytes of scrap}
        maxEditText : INTEGER;               {Maximum length of editing text}

    BEGIN  {doPaste}

        sizeScrap := TEGetScrapLen;          {Get scrap size TE-24}

        CASE textRecd OF                    {Set maximum length of concatenated string}
            division, brand, market, evaluator :
                maxEditText := 24;            {Maximum is 25 characters}
            adopter, fCost, production, price :
                maxEditText := 12;           {Maximum is 13 characters}
        END;  {CASE}

        IF (theText^.TELength + sizeScrap <= maxEditText) THEN
            {Limit editable text to authorized number}
            BEGIN

                TEPaste (theText);            {Paste the text selection TE-20}

                DisableItem (EditMenu, CutItem);    {Disable non relevant menu items MN-23}
                DisableItem (EditMenu, CopyItem);
                DisableItem (EditMenu, ClearItem);

                windowDirty(TRUE);            {U4: Mark window as dirty}

            END;  {IF}

        END;  {doPaste}

```

{-----}

```

PROCEDURE doClear;                                {Handle Clear command}

  BEGIN {doClear}

    TDelete (theText);                            {Delete the text selection TE-21}

    DisableItem (EditMenu, CutItem);               {Disable non relevant menu items MN-23}
    DisableItem (EditMenu, CopyItem);
    DisableItem (EditMenu, ClearItem);

    windowDirty(TRUE);                            {U4: Mark window as dirty}

  END; {doClear}

```

{-----}

```

{$S Main}                                         {Empty PROCEDURE Unit4.Main}
                                                {To satisfy compiler syntax}

BEGIN
END.

```

8.5. Unit5.Pas.

```

PROGRAM Unit5;                                {FILE Unit1.Pas: External Unit5 definition}

( *****
#
#      Programme conçu et développé
#      par Jean-Marie Choffray,
#      avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
***** )

($i HD:Pascal2:M1:MemTypes.ipas  )      {TML interface to General Purpose TYPES}
($i HD:Pascal2:M1:QuickDraw.ipas )      {TML interface to Macintosh Graphics Package}
($i HD:Pascal2:M1:OSIntf.ipas    )      {TML interface to the Operating System}
($i HD:Pascal2:M1:ToolIntf.ipas  )      {TML interface to Macintosh User Interface}
($i HD:Pascal2:M1:PackIntf.ipas  )      {TML interface to Macintosh Packages}
($i HD:Pascal2:M1:MacPrint.ipas  )      {TML interface to Macintosh Printing Mgr}

($i HD:Pascal2:Program:Globals.iPas )      {Declaration COMMON to Unit1 ... Unit7, and}
                                           {ProduitTest}

(-----)

PROCEDURE deActivateProduct; EXTERNAL;
  {Unit7: Deactivate product Edit records}
PROCEDURE deActivateEnvironmt; EXTERNAL;
  {Unit7: Deactivate environmt Edit records}

(-----)

PROCEDURE desimulationChoice (theItem : INTEGER); FORWARD;
  {Handle choice from Simulation menu}
PROCEDURE miniClean; FORWARD;
  {Clean inside of application frame}
PROCEDURE doProduct; FORWARD;
  {Handle Product command}
PROCEDURE setScreenProduct; FORWARD;
  {Screen layout for product parameters}
PROCEDURE blackCircle ( h, V : INTEGER); FORWARD;
  {Draw a little black circle at h, v-6, h+6, v}
PROCEDURE miniCopyright; FORWARD;
  {Write mini copyright at bottom of the screen}

PROCEDURE doInnovation; FORWARD;
  {Handle Innovation command}
PROCEDURE setScreenEstimate; FORWARD;
  {Screen layout for estimation purpose}
PROCEDURE Question; FORWARD;
  {Write question label}
PROCEDURE plotEstimator (estimatorPercent: LongInt ); FORWARD;
  {Draw gray subjective rectangle}
PROCEDURE writeEstimator ( estimatorPercent: LongInt ); FORWARD;
  {Write percent value of top of gray box}

PROCEDURE doImitation; FORWARD;
  {Handle Imitation command}

```

```
PROCEDURE doInfluence; FORWARD;
    {Handle Influence command}
```

```
PROCEDURE doPotential; FORWARD;
    {Handle Potential command}
```

```
PROCEDURE doReplace; FORWARD;
    {Handle Replace command}
```

```
PROCEDURE doExternalVC; FORWARD;
    {Handle ExternalVC command}
```

```
PROCEDURE doInternalVC; FORWARD;
    {Handle InternalVC command}
```

```
PROCEDURE doLearning; FORWARD;
    {Handle Learning command}
```

```
PROCEDURE doEnvironmt; FORWARD;
    {Handle Environmt command}
PROCEDURE setScreenEnvironmt; FORWARD;
    {Screen layout for environmt parameters}
```

```
{-----}
```

```
{ $S Code }                                { PROCEDURES and FUNCTIONS in Unit5.Code }
```

```
{-----}
```

```
PROCEDURE doSimulationChoice {(theItem : INTEGER)};
    {Handle choice from simulation menu}
    LABEL
        1008;                                {Exit doSimulationChoice label}
    BEGIN {dosimulationChoice}
        CASE theItem OF
            productParItem:                  {Item number for Product Par command}
                IF theScreen = product THEN  {Do nothing if it is where we are}
                    GOTO 1008                {Skip rest of operation}
                ELSE
                    BEGIN
                        EnableItem (editMenu, 0);      {Enable Edit menu MN-24}
                        DisableItem (editMenu, undoItem); {Items will be activated automatically when}
                        DisableItem (editMenu, cutItem);  {the user selects a portion of text}
                        DisableItem (editMenu, copyItem);
                        DisableItem (editMenu, pasteItem);
                        DisableItem (editMenu, clearItem);
                        DrawMenuBar;                  {Redraw MnBar according to menu list MN-19}
                        IF theScreen = environmt THEN  {Coming from environmt screen}
                            deActivateEnvironmt;      {U7: Deactivate environmt Edit records}
                            miniClean;                 {U5: Clean inside of application frame}
                            doProduct;                 {U5: Handle ProductPar command}
                        END; {ELSE}
```

```

innovationParItem:
  IF theScreen = innovation THEN      {Do nothing if it is where we are}
    GOTO 1008                          {Skip rest of operation}
  ELSE
    CASE theScreen OF
      product, environmt:             {Coming from a Text editing screen}
        BEGIN
          DisableItem (editMenu, 0);  {Disable Edit menu MN-24}
          DrawMenuBar;                {Redraw MnBar according to menu list MN-19}
          IF theScreen = product THEN {Coming from product screen}
            deactivateProduct;        {U7: Deactivate product Edit records}
          IF theScreen = environmt THEN {Coming from environmt screen}
            deactivateEnvironmt;      {U7: Deactivate environmt Edit records}
          miniClean;                   {U5: Clean inside of application frame}
          doInnovation;                {U5: Handle InnovationPar command}
        END; {product, environmt}

      imitation, influence, potential, replace,
      externalVC, internalVC, learning:
        doInnovation;                 {U5: Handle InnovationPar command}

      OTHERWISE                       {Coming from an 'analyze' screen}
        BEGIN
          miniClean;                   {U5: Clean inside of application frame}
          doInnovation;                {U5: Handle InnovationPar command}
        END; {OTHERWISE}
    END; {CASE}

imitationParItem:
  IF theScreen = imitation THEN        {Do nothing if it is where we are}
    GOTO 1008                          {Skip rest of operation}
  ELSE
    CASE theScreen OF
      product, environmt:             {Coming from a Text editing screen}
        BEGIN
          DisableItem (editMenu, 0);  {Disable Edit menu MN-24}
          DrawMenuBar;                {Redraw MnBar according to menu list MN-19}
          IF theScreen = product THEN {Coming from product screen}
            deactivateProduct;        {U7: Deactivate product Edit records}
          IF theScreen = environmt THEN {Coming from environmt screen}
            deactivateEnvironmt;      {U7: Deactivate environmt Edit records}
          miniClean;                   {U5: Clean inside of application frame}
          doImitation;                 {U5: Handle ImitationPar command}
        END; {product, environmt}

      innovation, influence, potential, replace,
      externalVC, internalVC, learning:
        doImitation;                  {U5: Handle ImitationPar command}

      OTHERWISE                       {Coming from an 'analyze' screen}
        BEGIN
          miniClean;                   {U5: Clean inside of application frame}
          doImitation;                 {U5: Handle ImitationPar command}
        END; {OTHERWISE}
    END; {CASE}

influenceParItem:
  IF theScreen = influence THEN        {Do nothing if it is where we are}

```

```

GOTO 1008                                {Skip rest of operation}
ELSE
CASE theScreen OF
  product, environmt:                    {Coming from a Text editing screen}
  BEGIN
    DisableItem (editMenu, 0);           {Disable Edit menu MN-24}
    DrawMenuBar;                         {Redraw MnBar according to menu list MN-19}
    IF theScreen = product THEN           {Coming from product screen}
      deactivateProduct;                 {U7: Deactivate product Edit records}
    IF theScreen = environmt THEN         {Coming from environmt screen}
      deactivateEnvironmt;               {U7: Deactivate environmt Edit records}
    miniClean;                           {U5: Clean inside of application frame}
    doInfluence;                         {U5: Handle InfluencePar command}
  END; {product, environmt}

  innovation, imitation, potential, replace,
  externalVC, internalVC, learning:
    doInfluence;                         {U5: Handle InfluencePar command}

  OTHERWISE                              {Coming from an 'analyze' screen}
  BEGIN
    miniClean;                           {U5: Clean inside of application frame}
    doInfluence;                         {U5: Handle InfluencePar command}
  END; {OTHERWISE}

END; {CASE}

potentialParItem:
  IF theScreen = potential THEN           {Do nothing if it is where we are}
    GOTO 1008                             {Skip rest of operation}
  ELSE
  CASE theScreen OF
    product, environmt:                  {Coming from a Text editing screen}
    BEGIN
      DisableItem (editMenu, 0);         {Disable Edit menu MN-24}
      DrawMenuBar;                       {Redraw MnBar according to menu list MN-19}
      IF theScreen = product THEN         {Coming from product screen}
        deactivateProduct;               {U7: Deactivate product Edit records}
      IF theScreen = environmt THEN       {Coming from environmt screen}
        deactivateEnvironmt;             {U7: Deactivate environmt Edit records}
      miniClean;                         {U5: Clean inside of application frame}
      doPotential;                       {U5: Handle PotentialPar command}
    END; {product, environmt}

    innovation, imitation, influence, replace,
    externalVC, internalVC, learning:
      doPotential;                       {U5: Handle PotentialPar command}

    OTHERWISE                            {Coming from an 'analyze' screen}
    BEGIN
      miniClean;                         {U5: Clean inside of application frame}
      doPotential;                       {U5: Handle PotentialPar command}
    END; {OTHERWISE}

  END; {CASE}

replaceParItem:
  IF theScreen = replace THEN             {Do nothing if it is where we are}

```

```

GOTO 1008                                {Skip rest of operation}
ELSE
CASE theScreen OF
  product, environmt:                    {Coming from a Text editing screen}
    BEGIN
      DisableItem (editMenu, 0);         {Disable Edit menu MN-24}
      DrawMenuBar;                       {Redraw MnBar according to menu list MN-19}
      IF theScreen = product THEN        {Coming from product screen}
        deActivateProduct;              {U7: Deactivate product Edit records}
      IF theScreen = environmt THEN      {Coming from environmt screen}
        deActivateEnvironmt;            {U7: Deactivate environmt Edit records}
      miniClean;                         {U5: Clean inside of application frame}
      doReplace;                         {U5: Handle ReplacePar command}
    END; {product, environmt}

    innovation, imitation, influence, potential,
    externalVC, internalVC, learning:
      doReplace;                         {U5: Handle ReplacePar command}

  OTHERWISE                              {Coming from an 'analyze' screen}
    BEGIN
      miniClean;                         {U5: Clean inside of application frame}
      doReplace;                         {U5: Handle ReplacePar command}
    END; {OTHERWISE}

END; {CASE}

externalVCParItem:
  IF theScreen = externalVC THEN          {Do nothing if it is where we are}
    GOTO 1008                             {Skip rest of operation}
  ELSE
    CASE theScreen OF
      product, environmt:                {Coming from a Text editing screen}
        BEGIN
          DisableItem (editMenu, 0);     {Disable Edit menu MN-24}
          DrawMenuBar;                   {Redraw MnBar according to menu list MN-19}
          IF theScreen = product THEN    {Coming from product screen}
            deActivateProduct;           {U7: Deactivate product Edit records}
          IF theScreen = environmt THEN  {Coming from environmt screen}
            deActivateEnvironmt;         {U7: Deactivate environmt Edit records}
          miniClean;                     {U5: Clean inside of application frame}
          doExternalVC;                  {U5: Handle ExternalVCPar command}
        END; {product, environmt}

        innovation, imitation, influence, potential, replace,
        internalVC, learning:
          doExternalVC;                  {U5: Handle ExternalVCPar command}

      OTHERWISE                          {Coming from an 'analyze' screen}
        BEGIN
          miniClean;                     {U5: Clean inside of application frame}
          doExternalVC;                  {U5: Handle ExternalVCPar command}
        END; {OTHERWISE}

    END; {CASE}

internalVCParItem:
  IF theScreen = internalVC THEN          {Do nothing if it is where we are}

```

```

GOTO 1008                                {Skip rest of operation}
ELSE
CASE theScreen OF
product, environmt:                      {Coming from a Text editing screen}
BEGIN
DisableItem (editMenu, 0);               {Disable Edit menu MN-24}
DrawMenuBar;                             {Redraw MnBar according to menu list MN-19}
IF theScreen = product THEN              {Coming from product screen}
deActivateProduct;                       {U7: Deactivate product Edit records}
IF theScreen = environmt THEN            {Coming from environmt screen}
deActivateEnvironmt;                     {U7: Deactivate environmt Edit records}
miniClean;                               {U5: Clean inside of application frame}
doInternalVC;                            {U5: Handle InternalVCP command}
END;   {product, environmt}

innovation, imitation, influence, potential, replace,
externalVC, learning:
doInternalVC;                            {U5: Handle InternalVCP command}

OTHERWISE                                {Coming from an 'analyze' screen}
BEGIN
miniClean;                               {U5: Clean inside of application frame}
doInternalVC;                            {U5: Handle InternalVCP command}
END;   {OTHERWISE}

END;   {CASE}

learningParItem:
IF theScreen = learning THEN              {Do nothing if it is where we are}
GOTO 1008                                {Skip rest of operation}
ELSE
CASE theScreen OF
product, environmt:                      {Coming from a Text editing screen}
BEGIN
DisableItem (editMenu, 0);               {Disable Edit menu MN-24}
DrawMenuBar;                             {Redraw MnBar according to menu list MN-19}
IF theScreen = product THEN              {Coming from product screen}
deActivateProduct;                       {U7: Deactivate product Edit records}
IF theScreen = environmt THEN            {Coming from environmt screen}
deActivateEnvironmt;                     {U7: Deactivate environmt Edit records}
miniClean;                               {U5: Clean inside of application frame}
doLearning;                              {U5: Handle LearningPar command}
END;   {product, environmt}

innovation, imitation, influence, potential, replace,
externalVC, internalVC:
doLearning;                              {U5: Handle LearningPar command}

OTHERWISE                                {Coming from an 'analyze' screen}
BEGIN
miniClean;                               {U5: Clean inside of application frame}
doLearning;                              {U5: Handle LearningPar command}
END;   {OTHERWISE}

END;   {CASE}

environmtParItem:
IF theScreen = environmt THEN              {Do nothing if it is where we are}

```

```

        GOTO 1008                                {Skip rest of operation}
ELSE
    BEGIN
        EnableItem (editMenu, 0);                {Enable Edit menu MN-24}
        DisableItem (editMenu, undoItem);        {Items will be activated automatically when}
        DisableItem (editMenu, cutItem);          {the user selects a portion of text}
        DisableItem (editMenu, copyItem);
        DisableItem (editMenu, pasteItem);
        DisableItem (editMenu, clearItem);
        DrawMenuBar;                             {Redraw MnBar according to menu list MN-19}
        IF theScreen = product THEN               {Coming from product screen}
            deactivateProduct;                    {U7: Deactivate product Edit records}
            doEnvironmt;                          {U5: Handle EnvironmtPar command}
        END; {ELSE}

    END; {CASE}

1008: END;    {dosimulationChoice}

{-----}

PROCEDURE miniClean;                             {Clean inside of application frame}

VAR

    localScreenRect : Rect;                      {Local rect for application frame QD-9}

BEGIN {miniClean}

    SetRect ( localScreenRect, 25, 15, 465, 265); {Local coordinates of appl frame LTRB QD-46}
    FillRect ( localScreenRect, white);           {Erase localScreenRect QD-49, QD-34}

END; {miniClean}

{-----}

PROCEDURE doProduct;                             {Handle Product command}

BEGIN {doProduct}

    theScreen := product;                        {Set screen flag}
    textRecd := none;                           {Set initial text edit flag}

    setScreenProduct;                           {U5: Sreen layout for product parameters}

END; {doProduct}

{-----}

PROCEDURE setScreenProduct;                      {Screen layout for product parameters}

VAR

    theData      : apIHandle;                   {Handle to application data record}
    dataHandle    : Handle;                     {Untyped handle to create data record MI-9}
    titleRect     : Rect;                       {Local Rect for title QD-9}

    biggerRect    : Rect;                       {Work rect to frame Edit record boxes QD-9}

```

```

cleanRect    : Rect;                                {Work rect to clean up Edit record boxes QD-9}

BEGIN {setScreenProduct};

dataHandle := Handle (GetWRefCon (theWindow));
HLock (dataHandle);                                {Get and lock appl data record WM-33, MM-42}
theData := apIHandle (dataHandle);                  {Convert to an apIHandle}

PenSize (2,2);                                     {Set dimensions of graphic pen QD-41}
TextFace ([]);                                     {Select normal characters QD-43}

SetRect (screenRect, 20, 10, 470, 270);            {Set overall screen rectangle LTRB QD-46}
FrameRoundRect(screenRect, 10, 10);                {Draw overall screen rectangle QD-51}

SetRect (titleRect, 34, 20, 258, 50);              {Set product title rectangle LTRB QD-46}
FrameRoundRect(titleRect, 10, 10);                 {Draw product title rectangle QD-51}

PenSize (1,1);                                     {Set dimensions of graphic pen QD-41}

WITH theData^^ DO
  BEGIN

    biggerRect := divisionEditRecd^^.viewRect;
    cleanRect  := biggerRect;
    InSetRect (biggerRect, -2, -2);                  {Expand the rectangle QD-47}
    FrameRect(biggerRect);                            {Draw Division text edit frames QD-49}
    EraseRect(cleanRect);                            {Clean Division text edit frames QD-49}
    TextBox (divisionEditRecd^^.HText^, divisionEditRecd^^.TELength,
            divisionEditRecd^^.destRect, TEJustLeft);
                                                    {Draw non editable text in rectangle TE-23}

    biggerRect := brandEditRecd^^.viewRect;
    cleanRect  := biggerRect;
    InSetRect (biggerRect, -2, -2);                  {Expand the rectangle QD-47}
    FrameRect(biggerRect);                            {Draw Brand text edit frames QD-49}
    EraseRect(cleanRect);                            {Clean Brand text edit frames QD-49}
    TextBox (brandEditRecd^^.HText^, brandEditRecd^^.TELength,
            brandEditRecd^^.destRect, TEJustLeft);
                                                    {Draw non editable text in rectangle TE-23}

    biggerRect := marketEditRecd^^.viewRect;
    cleanRect  := biggerRect;
    InSetRect (biggerRect, -2, -2);                  {Expand the rectangle QD-47}
    FrameRect(biggerRect);                            {Draw Market text edit frames QD-49}
    EraseRect(cleanRect);                            {Clean Market text edit frames QD-49}
    TextBox (marketEditRecd^^.HText^, marketEditRecd^^.TELength,
            marketEditRecd^^.destRect, TEJustLeft);
                                                    {Draw non editable text in rectangle TE-23}

    biggerRect := evaluatorEditRecd^^.viewRect;
    cleanRect  := biggerRect;
    InSetRect (biggerRect, -2, -2);                  {Expand the rectangle QD-47}
    FrameRect(biggerRect);                            {Draw Evaluator text edit frames QD-49}
    EraseRect(cleanRect);                            {Clean Evaluator text edit frames QD-49}
    TextBox (evaluatorEditRecd^^.HText^, evaluatorEditRecd^^.TELength,
            evaluatorEditRecd^^.destRect, TEJustLeft);
                                                    {Draw non editable text in rectangle TE-23}

  END; {WITH}

```

```

TextFace ([Bold]);                                {Select bold characters QD-43}

MoveTo (40,39);                                   {Move pen (h,v) for product title QD-42}
DrawString ('Description du produit étudié');      {Product title QD-44}

blackCircle (40, 87);                             {U5: Draw a black circle at h,v-6, h+6,v}
MoveTo (55, 87);                                   {Move pen (h,v) for label#1 QD-42}
DrawString ('Division concernée...');             {Division label QD-44}

blackCircle (40, 127);                            {U5: Draw a black circle at h,v-6, h+6,v}
MoveTo (55, 127);                                 {Move pen (h,v) for label#2 QD-42}
DrawString ('Produit étudié...');                 {Product label QD-44}

blackCircle (40, 167);                            {U5: Draw a black circle at h,v-6, h+6,v}
MoveTo (55, 167);                                 {Move pen (h,v) for label#3 QD-42}
DrawString ('Marché visé...');                   {Market label QD-44}

blackCircle (40, 207);                            {U5: Draw a black circle at h,v-6, h+6,v}
MoveTo (55, 207);                                 {Move pen (h,v) for label#4 QD-42}
DrawString ('Evaluateur...');                    {Evaluator's label QD-44}

PenSize (2, 2);                                    {Set dimensions of graphic pen QD-41}

IF NOT simulation THEN
  BEGIN
    SetRect (okRect1, 370, 240, 450, 260); {Set rectangle for OK answer LTRB QD-46}
    FrameRoundRect(okRect1, 10, 10);         {Draw OK answer rectangle QD-51}
    MoveTo (400, 254);                       {Move pen (h,v) for OK title QD-42}
    DrawString ('OK');                       {OK label QD-44}
  END {THEN}
ELSE
  BEGIN
    {OK rect is set to Gray, i.e. inactive}
    SetRect (okRect1, 370, 240, 450, 260); {Set rectangle for OK answer LTRB QD-46}
    FrameRoundRect(okRect1, 10, 10);         {Draw OK answer rectangle QD-51}
    MoveTo (400, 254);                       {Move pen (h,v) for OK title QD-42}
    DrawString ('OK');                       {OK label QD-44}

    PenPat (Gray);                           {Set pen pattern QD-42, QD-34}
    PenMode (patBic);                        {Set pattern transfer mode QD-41, QD-29}

    SetRect (okRect1, 375, 243, 445, 257); {Set 'inside' of OK rectangle LTRB QD-46}
    PaintRect (okRect1);                     {Paint OK rect QD-47, QD-49}

    PenMode (patCopy);                       {Set pattern transfer mode QD-41, QD-29}
    PenPat (Black);                          {Set pen pattern QD-42, QD-34}
  END; {ELSE}

PenSize (1, 1);                                {Set dimensions of graphic pen QD-41}

TextFace ([I]);                                {Select normal characters QD-43}
miniCopyright;                                {U5: Write copyright at bottom of the screen}

HUnlock (dataHandle);                          {Unlock application data record MM-43}

END; {setScreenProduct}

```

{-----}

```

PROCEDURE blackCircle (( h, v : INTEGER));           {Draw a little black circle at h, v-6, h+6, v}

VAR

    blackRect : Rect;                               {Local rect for blackCircle}

BEGIN {blackCircle}

    PenSize (1, 1);                                 {Set dimensions of graphic pen QD-41}
    SetRect (blackRect, h, v-6, h+6, v);             {Set rectangle for blackCircle LTRB QD-46}
    FrameOval (blackRect);                           {Draw hollow outline of oval QD-49}
    FillOval (blackRect, Black);                     {Fill with black color QD-49}

END; {blackCircle}

{-----}

PROCEDURE miniCopyright;                             {Write mini copyright bottom of the screen}

BEGIN {miniCopyright}

    TextSize (9);                                    {Set grafPort type size to 9 QD-43}
    MoveTo (20, 282);                                {Move pen (h,v) for label QD-42}
    DrawString ('ProduitTest');                      {Write string QD-44}
    MoveTo (400, 282);                               {Move pen (h,v) for label QD-42}
    DrawString ('Copyright 1987');                   {Write string QD-44}
    TextSize (12);                                   {Set grafPort type size to 12 QD-43}

END; {miniCopyright}

{-----}

PROCEDURE doInnovation;                              {Handle Innovation command}

VAR

    theData      : apIHandle;                       {Handle to application data record}
    dataHandle   : Handle;                          {Untyped handle to create data record MI-9}

BEGIN {doInnovation}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                             {Get and lock a handle to data record WM-33}
    theData := apIHandle (dataHandle);               {Convert to apIHandle}

    theScreen := innovation;                         {Set screen flag}

    setScreenEstimate;                               {U5: Screen layout for estimation purpose}
    Question;                                         {U5: Write down innovation question}

    plotEstimator (theData^.innovationPar);          {U5: Draw initial gray subjective rectangle}
    writeEstimator (theData^.innovationPar);          {U5: Percent value of initial innovationPar}

    HUnlock (dataHandle);                            {Unlock application data record MM-43}

END; {doInnovation}

```

```

{-----}

PROCEDURE setScreenEstimate;           {Screen layout for estimation purpose}

BEGIN {setScreenEstimate}

    PenSize (2, 2);                    {Set dimensions of graphic pen QD-41}

    SetRect (screenRect, 20, 10, 470, 270); {Set overall screen rectangle LTRB QD-46}
    FrameRoundRect (screenRect, 10, 10);    {Draw overall screen rectangle QD-51}

    SetRect (questionRect, 40, 30, 390, 190); {Set rect for question label LTRB QD-46}
    FrameRoundRect (questionRect, 10, 10);    {Draw question rectangle QD-51}

    SetRect (subjectiveRect, 410, 30, 450, 250); {Set rect for sub estimation LTRB QD-46}
    FrameRect (subjectiveRect);              {Draw subjective rectangle QD-49}

    SetRect (estimatorRect, 40, 210, 390, 250); {Set rect for percent answer LTRB QD-46}
    FrameRoundRect (estimatorRect, 10, 10);    {Draw percent rectangle QD-51}

    TextFace ([Bold]);                  {Select bold characters QD-43}

    IF NOT simulation THEN
        BEGIN
            SetRect (okRect2, 350, 220, 380, 240); {Set rect for OK answer LTRB QD-46}
            FrameRoundRect (okRect2, 10, 10);    {Draw OK answer rectangle QD-51}
            MoveTo (356, 234);                  {Move pen for OK button (h, v) QD-42}
            DrawString ('OK');                  {Write string QD-44}
        END {THEN}
    ELSE
        BEGIN
            {OK rect is set to Gray, i.e. inactive}
            SetRect (okRect2, 350, 220, 380, 240); {Set rect for OK answer LTRB QD-46}
            FrameRoundRect (okRect2, 10, 10);    {Draw OK answer rectangle QD-51}
            MoveTo (356, 234);                  {Move pen for OK button (h, v) QD-42}
            DrawString ('OK');                  {Writes string QD-44}

            PenPat (Gray);                      {Set pen pattern QD-42, QD-34}
            PenMode (patBic);                   {Set pattern transfer mode QD-41, QD-29}

            SetRect (okRect2, 355, 223, 375, 237); {Set rect for OK answer LTRB QD-46}
            PaintRect (okRect2);                {Paint OK rect QD-47, QD-49}

            PenMode (patCopy);                  {Set pattern transfer mode QD-41, QD-29}
            PenPat (Black);                     {Set pen pattern QD-42, QD-34}
        END; {ELSE}

    PenSize (1, 1);                          {Set dimensions of graphic pen QD-41}

    MoveTo (50, 233);                         {Move pen location (h,v) QD-42}
    DrawString ('Estimation actuelle... ');    {Write string QD-44}
    TextFace ([]);                             {Select normal characters QD-43}

    miniCopyright;                            {U5: Write copyright at bottom of screen}

END; {setScreenEstimate}

{-----}

```

```

PROCEDURE Question;                                {Writes question label}

VAR

    cleanRect : Rect;                              {Cleaning rectangle QD-9}

BEGIN

    SetRect (cleanRect, 45, 35, 385, 185);        {Clean question rectangle LTRB QD-46}
    FillRoundRect (cleanRect, 3, 3, White);        {Fill question rect with 'white' QD-51}

    TextFace ([Bold]);                             {Select bold characters QD-43}
    MoveTo (50, 60);                               {Move pen location (h,v) QD-42}

    CASE theScreen OF
        innovation : DrawString ('Coefficient d''innovation. ');
        imitation  : DrawString ('Coefficient d''imitation. ');
        influence   : DrawString ('Coefficient d''influence. ');
        potential   : DrawString ('Marché potentiel. ');
        replace     : DrawString ('Coefficient de renouvellement. ');
        externalVC  : DrawString ('Coût des entrants. ');
        internalVC  : DrawString ('Coût de la valeur ajoutée. ');
        learning    : DrawString ('Taux d''apprentissage. ');
    END; {CASE}

    TextFace ([]);                                 {Select normal characters QD-43}
    MoveTo (50, 90);                               {Move pen location (h,v) QD-42}

    CASE theScreen OF
        innovation : DrawString ('Fraction des utilisateurs potentiels adoptant le ');
        imitation  : DrawString ('Fraction des utilisateurs potentiels adoptant le ');
        influence   : DrawString ('Renforcement de l''influence exercée par les ');
        potential   : DrawString ('Fraction du marché cible dont les besoins ');
        replace     : DrawString ('Fraction des utilisateurs ayant adopté le ');
        externalVC  : DrawString ('Part du coût variable, correspondant au ');
        internalVC  : DrawString ('Part du coût variable, correspondant à ');
        learning    : DrawString ('Fraction du coût unitaire correspondant ');
    END; {CASE}

    MoveTo (50, 110);                              {Move pen location (h,v) QD-42}
    CASE theScreen OF
        innovation : DrawString ('produit nouveau, indépendamment de toute ');
        imitation  : DrawString ('produit nouveau, à la suite de l''influence ');
        influence   : DrawString ('adopteurs antérieurs sur le comportement ');
        potential   : DrawString ('peuvent potentiellement être satisfaits ');
        replace     : DrawString ('produit nouveau, qui le rachèteront ');
        externalVC  : DrawString ('coût des entrants, exprimée en ');
        internalVC  : DrawString ('la valeur ajoutée du produit, exprimée ');
        learning    : DrawString ('à la valeur ajoutée du produit, lorsque ');
    END; {CASE}

    MoveTo (50, 130);                              {Move pen location (h,v) QD-42}
    CASE theScreen OF
        innovation : DrawString ('source d''influence extérieure. ');
        imitation  : DrawString ('exercée par les adopteurs antérieurs. ');
        influence   : DrawString ('des utilisateurs potentiels. ');
        potential   : DrawString ('par le produit nouveau. ');
        replace     : DrawString ('au cours de la prochaine période. ');
    END; {CASE}

```

```

    externalVC : DrawString ('pourcentage du prix de vente unitaire. ');
    internalVC : DrawString ('en pourcentage du prix de vente unitaire. ');
    learning    : DrawString ('la production cumulée double. ');
END; {CASE}

```

```

TextFace ([Bold]);           {Select bold characters QD-43}
MoveTo (50, 160);           {Move pen location (h,v) QD-42}
CASE theScreen OF
    innovation : DrawString ('Niveau initial de pénétration. ');
    imitation  : DrawString ('Vitesse de diffusion. ');
    influence  : DrawString ('Intensité de l'imitation. ');
    potential  : DrawString ('Pénétration cumulée maximale. ');
    replace    : DrawString ('Part de réachat. ');
    externalVC : DrawString ('Coût variable hors contrôle. ');
    internalVC : DrawString ('Coût variable sous contrôle. ');
    learning   : DrawString ('Pente d'expérience. ');
END; {CASE}

```

```

TextFace ([]);           {Select normal characters QD-43}

```

```

END; {Question}

```

```

{-----}

```

```

PROCEDURE plotEstimator ((estimatorPercent : LongInt));
                                {Draw gray subjective rectangle}

```

```

VAR

```

```

    estimatorLevel : LongInt;           {Top level of gray box}
    grayRect       : Rect;              {Gray subjective rectangle QD-9}
    cleanRect      : Rect;              {Cleaning rectangle QD-9}

```

```

BEGIN

```

```

    SetRect (cleanRect, 412, 32, 448, 248); {Clean subjective rectangle LTRB QD-46}
    FillRect (cleanRect, White);             {Fill rect with 'white' QD-49}

```

```

    estimatorLevel := ROUND (estimatorPercent / 100 * 220);
                                {Rescale top level of gray box}
    IF estimatorLevel <= 2 THEN {Minimum size of active rectangle}
        SetRect (activeRect, 410, 248, 450, 250) {Set active rectangle LTRB QD-46}
    ELSE
        BEGIN
            SetRect (activeRect, 410, 250 - estimatorLevel, 450, 250);
                                {Set active rectangle LTRB QD-46}
            IF estimatorLevel > 218 THEN {Avoid hitting top of subjective rectangle}
                SetRect (grayRect, 412, 32, 448, 248)
            ELSE
                {Draw standard gray box}
                SetRect (grayRect, 412, 250 - estimatorLevel, 448, 248);

            FillRect ( grayRect, LtGray); {Fill rect with 'ltGray' QD-49, QD-34}
        END; {ELSE}
    END;

```

```

END; {plotEstimator}

```

```

{-----}

```

```

PROCEDURE writeEstimator ((estimatorPercent: LongInt));
                                {Write percent value of top of gray box}

VAR

    alphaPercent   : STR255;           {String version of percentage}
    percentRect    : Rect;             {Rect for erasing changing percentage QD-9}
    lengthPercent  : INTEGER;          {Length of percent string}

BEGIN

    NumToString (estimatorPercent, alphaPercent);
                                {LongInt to STR255 conversion PK-20}
    lengthPercent := LENGTH ( alphaPercent);
    CASE lengthPercent OF
        1 : alphaPercent := CONCAT ( ' ', alphaPercent);
        2 : alphaPercent := CONCAT ( ' ', alphaPercent);
        3 : alphaPercent := CONCAT ( ' ', alphaPercent);
    END; {CASE}

    SetRect ( percentRect, 235, 215, 275, 245); {Clean percent rectangle LTRB QD-46}
    FillRect ( percentRect, White);             {Fill rect with 'white' QD-49; QD-34}

    TextFace ( [Bold] );                     {Select bold characters QD-43}
    MoveTo ( 235, 233 );                      {Move pen location (h,v) QD-42}
    DrawString (alphaPercent);                 {Write string QD-44}

    MoveTo ( 278, 233);                      {Move pen location (h,v) QD-42}
    DrawString ('pour cent');                 {Write string QD-44}
    TextFace ( [] );                          {Select normal characters QD-43}

END; {writeEstimator}

-----

PROCEDURE dolimitation;                {Handle Imitation command}

VAR

    theData      : apiHandle;           {Handle to application data record}
    dataHandle   : Handle;              {Untyped handle to create data record MI-9}

BEGIN {dolimitation}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                 {Get and lock a handle to data record WM-33}
    theData := apiHandle (dataHandle);   {Convert to apiHandle}

    theScreen := imitation;              {Set screen flag}

    setScreenEstimate;                  {U5: Screen layout for estimation purpose}
    Question;                           {U5: Write down imitation question}

    plotEstimator (theData^.imitationPar); {U5: Draw initial gray subjective rectangle}
    writeEstimator (theData^.imitationPar); {U5: Percent value of initial imitationPar}

    HUnlock (dataHandle);                {Unlock application data record MM-43}

END; {dolimitation}

```


VAR

theData : apIHandle; {Handle to application data record}
dataHandle : Handle; {Untyped handle to create data record MI-9}

BEGIN {doReplace}

dataHandle := Handle (GetWRefCon (theWindow));
Hlock (dataHandle); {Get and lock a handle to data record WM-33}
theData := apIHandle (dataHandle); {Convert to apIHandle}

theScreen := replace; {Set screen flag}

setScreenEstimate; {U5: Screen layout for estimation purpose}
Question; {U5: Write down replace question}

plotEstimator (theData^.replacePar); {U5: Draw initial gray subjective rectangle}
writeEstimator (theData^.replacePar); {U5: Percent value of initial replacePar}

HUnlock (dataHandle); {Unlock application data record MM-43}

END; {doReplace}

PROCEDURE doExternalVC; {Handle ExternalVC command}

VAR

theData : apIHandle; {Handle to application data record}
dataHandle : Handle; {Untyped handle to create data record MI-9}

BEGIN {doExternalVC}

dataHandle := Handle (GetWRefCon (theWindow));
Hlock (dataHandle); {Get and lock a handle to data record WM-33}
theData := apIHandle (dataHandle); {Convert to apIHandle}

theScreen := externalVC; {Set screen flag}

setScreenEstimate; {U5: Screen layout for estimation purpose}
Question; {U5: Write down externalVC question}

plotEstimator (theData^.externalVCPAr); {U5: Draw initial gray subjective rectangle}
writeEstimator (theData^.externalVCPAr); {U5: Percent value of initial variableVCPAr}

HUnlock (dataHandle); {Unlock application data record MM-43}

END; {doExternalVC}

PROCEDURE doInternalVC; {Handle InternalVC command}

VAR

theData : apIHandle; {Handle to application data record}

```

    dataHandle : Handle;           {Untyped handle to create data record MI-9}

BEGIN {doInternalVC}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);           {Get and lock a handle to data record WM-33}
    theData := apiHandle (dataHandle); {Convert to apiHandle}

    theScreen := internalVC;      {Set screen flag}

    setScreenEstimate;           {U5: Screen layout for estimation purpose}
    Question;                    {U5: Write down internalVC question}

    plotEstimator (theData^.internalVCPar); {U5: Draw initial gray subjective rectangle}
    writeEstimator (theData^.internalVCPar); {U5: Percent value of initial replacePar}

    HUnlock (dataHandle);        {Unlock application data record MM-43}

END; {doInternalVC}

{-----}

PROCEDURE doLearning;           {Handle Learning command}

VAR

    theData : apiHandle;         {Handle to application data record}
    dataHandle : Handle;         {Untyped handle to create data record MI-9}

BEGIN {doLearning}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);           {Get and lock a handle to data record WM-33}
    theData := apiHandle (dataHandle); {Convert to apiHandle}

    theScreen := learning;       {Set screen flag}

    setScreenEstimate;           {U5: Screen layout for estimation purpose}
    Question;                    {U5: Write down learning question}

    plotEstimator (theData^.learningPar); {U5: Draw initial gray subjective rectangle}
    writeEstimator (theData^.learningPar); {U5: Percent value of initial learningPar}

    HUnlock (dataHandle);        {Unlock application data record MM-43}

END; {doLearning}

{-----}

PROCEDURE doEnvironmt;         {Handle Environmt command}

BEGIN {doEnvironmt}

    theScreen := environmt;      {Set screen flag}
    textRecd := none;            {Set initial text edit flag}
    miniClean;                   {U5: Clean inside of application frame}

    setScreenEnvironmt;          {U5: Screen layout for environmt parameters}

```

END; {doEnvironmt}

{-----}

PROCEDURE setScreenEnvironmt;

{Screen layout for environmt parameters}

VAR

theData	:	aplHandle;	{Handle to application data record}
dataHandle	:	Handle;	{Untyped handle to create data record MI-9}
titleRect	:	Rect;	{Local Rect for title QD-9}
biggerRect	:	Rect;	{Work Rect to frame edit record boxes QD-9}
cleanRect	:	Rect;	{Work Rect to clean up edit record boxes QD-9}

BEGIN {setScreenEnvironmt}

dataHandle := Handle (GetWRefCon (theWindow));	
HLock (dataHandle);	{Get and lock appl data record WM-33, MM-42}
theData := aplHandle (dataHandle);	{Convert to an aplHandle}

PenSize (2,2);	{Sets dimensions of graphic pen QD-41}
TextFace ([1]);	{Select normal characters QD-43}

SetRect (screenRect, 20, 10, 470, 270);	{Set overall screen rectangle LTRB QD-46}
FrameRoundRect(screenRect, 10, 10);	{Draw overall screen rectangle QD-51}

SetRect (titleRect, 34, 20, 278, 50);	{Set environmt title rectangle LTRB QD-46}
FrameRoundRect(titleRect, 10, 10);	{Draw environmt title screen rectangle QD-51}

PenSize (1,1);	{Sets dimensions of graphic pen QD-41}
----------------	--

WITH theData^^ DO
 BEGIN

biggerRect := adopterEditRecd^^.viewRect;	
cleanRect := biggerRect;	
InSetRect (biggerRect, -2, -2);	{Expand the rectangle QD-47}
FrameRect(biggerRect);	{Draw Adopter text edit frames QD-49}
EraseRect(cleanRect);	{Clean Adopter text edit frames QD-49}
TextBox (adopterEditRecd^^.HText^, adopterEditRecd^^.TELength,	
adopterEditRecd^^.destRect, TEJustRight);	{Draw non editable text in rect TE-22, TE-21}

biggerRect := fCostEditRecd^^.viewRect;	
cleanRect := biggerRect;	
InSetRect (biggerRect, -2, -2);	{Expand the rectangle QD-47}
FrameRect(biggerRect);	{Draw fCost text edit frames QD-49}
EraseRect(cleanRect);	{Clean fCost text edit frames QD-49}
TextBox (fCostEditRecd^^.HText^, fCostEditRecd^^.TELength,	
fCostEditRecd^^.destRect, TEJustRight);	{Draw non editable text in rect TE-23, TE-21}

biggerRect := productionEditRecd^^.viewRect;	
cleanRect := biggerRect;	
InSetRect (biggerRect, -2, -2);	{Expand the rectangle QD-47}
FrameRect(biggerRect);	{Draw Production text edit frames QD-49}

```

EraseRect(cleanRect);           {Clean Production text edit frames QD-49}
TextBox (productionEditRecd^.HText^, productionEditRecd^.TELength,
        productionEditRecd^.destRect, TEJustRight);
                                {Draw non editable text in rect TE-23, TE-21}

biggerRect := priceEditRecd^.viewRect;
cleanRect := biggerRect;
InSetRect (biggerRect, -2, -2); {Expand the rectangle QD-47}
FrameRect(biggerRect);          {Draw Price text edit frames QD-49}
EraseRect(cleanRect);           {Clean Price text edit frames QD-49}
TextBox (priceEditRecd^.HText^, priceEditRecd^.TELength,
        priceEditRecd^.destRect, TEJustRight);
                                {Draw non editable text in rect TE-23, TE-21}

END; {WITH}

PenSize (2, 2);                 {Set dimensions of graphic pen QD-41}
TextFace ([Bold]);              {Select Bold characters QD-43}

IF NOT simulation THEN
  BEGIN
    SetRect (okRect1, 370, 240, 450, 260); {Set rectangle for OK answer LTRB QD-46}
    FrameRoundRect(okRect1, 10, 10);        {Draw OK answer rectangle QD-51}
    MoveTo (400, 254);                      {Move pen for OK button (h, v) QD-42}
    DrawString ('OK');                      {OK label QD-44}
  END {THEN}
ELSE
  BEGIN
    SetRect (okRect1, 370, 240, 450, 260); {Set rectangle for OK answer LTRB QD-46}
    FrameRoundRect(okRect1, 10, 10);        {Draw OK answer rectangle QD-51}
    MoveTo (400, 254);                      {Move pen for OK button (h, v) QD-42}
    DrawString ('OK');                      {OK label QD-44}

    PenPat (Gray);                         {Set pen pattern QD-42, QD-34}
    PenMode (patBic);                      {Set pattern transfer mode QD-41, QD-29}

    SetRect (okRect1, 375, 243, 445, 257); {Set rectangle for OK answer LTRB QD-46}
    PaintRect (okRect1);                   {Paint OK rect QD-47, QD-49}

    PenMode (patCopy);                     {Set pattern transfer mode QD-41, QD-29}
    PenPat (Black);                        {Set pen pattern QD-42, QD-34}
  END; {ELSE}

PenSize (1, 1);                   {Set dimensions of graphic pen QD-41}

MoveTo (40, 39);                  {Move pen for environmt title (h, v) QD-42}
DrawString ('Description de l''environnement'); {Environmt title QD-44}

blackCircle (40, 87);             {Draw a little black circle at h, v-6, h+6, v}
MoveTo (55, 87);                 {Move pen for label#1 (h, v) QD-42}
DrawString ('Nbre d''adopteurs potentiels...'); {Adopter label QD-44}

blackCircle (40, 127);            {U5: Draw a black circle at h, v-6, h+6, v}
MoveTo (55, 127);                {Move pen for label#2 (h, v) QD-42}
DrawString ('Coût fixe total...'); {Fixed Cost label QD-44}

blackCircle (40, 167);            {U5: Draw a little circle at h, v-6, h+6, v}
MoveTo (55, 167);                {Move pen for label#3 (h, v) QD-42}
DrawString ('Palier d''expérience (unités)...'); {Production label QD-44}

```

blackCircle (40, 207);	{U5: Draw a little circle at h, v-6, h+6, v}
MoveTo (55, 207);	{Move pen for label#4 (h, v) QD-42}
DrawString ('Prix unitaire moyen...');	{Price label QD-44}
TextFace ([]);	{Select normal characters QD-43}
miniCopyright;	{U5: Write copyright at bottom of the screen}
HUnlock (dataHandle);	{Unlock application data record MM-43}
END; {setScreenEnvironmt}	
{-----}	
(\$S Main)	{Empty PROCEDURE Unit5.Main}
	{To satisfy compiler syntax}
BEGIN	
END.	

8.6. Unit6.Pas.

```

PROGRAM Unit6;                                     {FILE Unit6.Pas: External Unit6 definition}

( *****
#
#      Programme conçu et développé
#      par Jean-Marie Choffray,
#      avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
***** )

{$i HD:Pascal2:M1:MemTypes.ipas }                 {TML interface to General Purpose TYPES}
{$i HD:Pascal2:M1:QuickDraw.ipas }                 {TML interface to Macintosh Graphics Package}
{$i HD:Pascal2:M1:OSIntf.ipas }                     {TML interface to the Operating System}
{$i HD:Pascal2:M1:ToolIntf.ipas }                   {TML interface to Macintosh User Interface}
{$i HD:Pascal2:M1:PackIntf.ipas }                   {TML interface to Macintosh Packages}
{$i HD:Pascal2:M1:MacPrint.ipas }                   {TML interface to Macintosh Printing Mgr}

{$i HD:Pascal2:Program:Globals.iPas }               {Declaration COMMON to Unit1 ... Unit7, and
                                                    {ProduitTest}

(-----)

PROCEDURE miniClean; EXTERNAL;
    {Unit5: Clean inside of application frame}
PROCEDURE miniCopyright; EXTERNAL;
    {Unit5: Write mini copyright at bottom of the screen}

PROCEDURE deActivateProduct; EXTERNAL;
    {Unit7: Deactivate product Edit records}
PROCEDURE deActivateEnvironmt; EXTERNAL;
    {Unit7: Deactivate environmt Edit records}

(-----)

PROCEDURE doAnalyzeChoice (theItem : INTEGER); FORWARD;
    {Handle choice from Analyze menu}

PROCEDURE doSummary; FORWARD;
    {Handle Summary command}
PROCEDURE displaySummary; FORWARD;
    {Display input data}

PROCEDURE doInnovationCur; FORWARD;
    {Handle InnovationCur command}
PROCEDURE setScreenCurve; FORWARD;
    {Set screen display for graphic diagnosis}
PROCEDURE setLabelsCurve; FORWARD;
    {Draw curve labels and mini copyright}
PROCEDURE plotInnovation; FORWARD;
    {Plot the Innovation curve}
FUNCTION stringToReal (myString : STR255): REAL; FORWARD;
    {STR255 -> REAL conversion}
FUNCTION exponential (x, y : REAL): REAL; FORWARD;
    {Compute x to the power of y}
PROCEDURE horizontal (y : INTEGER); FORWARD;
    {Draw a horizontal line at (230 - y)}

```

```

PROCEDURE dottedLine (y : INTEGER); FORWARD;
    {Draw a dotted line at (230 - y)}
PROCEDURE setActiveCircle (i: INTEGER; y, range, base : REAL); FORWARD;
    {Local coordinates of Cursor active circles 'i'}
PROCEDURE drawActiveCircle (i: INTEGER); FORWARD;
    {Draw Cursor active circle 'i'}

PROCEDURE doImitationCur; FORWARD;
    {Handle ImitationCur command}
PROCEDURE plotImitation; FORWARD;
    {Plot the Imitation curve}

PROCEDURE doPenetrationCur; FORWARD;
    {Handle PenetrationCur command}
PROCEDURE plotPenetration; FORWARD;
    {Plot the Penetration curve}

PROCEDURE doReplaceCur; FORWARD;
    {Handle ReplaceCur command}
PROCEDURE plotReplace; FORWARD;
    {Plot the Replace curve}

PROCEDURE doVolumeCur; FORWARD;
    {Handle VolumeCur command}
PROCEDURE plotVolume; FORWARD;
    {Plot the Volume curve}

PROCEDURE doInternalVCCur; FORWARD;
    {Handle InternalVCCur command}
PROCEDURE plotInternalVC; FORWARD;
    {Plot the InternalVC curve}

PROCEDURE doTotalCCur; FORWARD;
    {Handle TotalCCur command}
PROCEDURE plotTotalC; FORWARD;
    {Plot the TotalC curve}

PROCEDURE doProfitCur; FORWARD;
    {Handle ProfitCur command}
PROCEDURE plotProfit; FORWARD;
    {Plot the Profit curve}

{-----}
{$S Code}                                {PROCEDURES and FUNCTIONS in Unit6.Code}
{-----}

PROCEDURE doAnalyzeChoice {(theItem : INTEGER)}; {Handle choice from Analyze menu}

LABEL

    1030;                                {Exit checkInput label}

VAR

    ignore      : INTEGER;                {Answer from alert}
    wkNum       : REAL;                   {Temporary REAL working space}

```

```

wkString      : STR255;                {Temporary STRING working space}

theData       : apiHandle;             {Handle to application data record}
dataHandle    : Handle;               {Untyped handle to create data record MI-9}

BEGIN {doAnalyzeChoice}

dataHandle := Handle(GetWRefCon(theWindow));
Hlock(dataHandle);                    {Get and lock Handle to data record WM-33}
theData := apiHandle(dataHandle);     {Convert to apiHandle}

WITH theData^^ DO

    BEGIN                                {Test major data inconsistencies}

        IF (innovationPar + imitationPar >= 100) THEN
            BEGIN                        {InnovationPar + imitationPar >= 100!!!}
                ignore := StopAlert(badModelID, NIL);
                                         {Post 'badModelID' stop alert DM-28}
                HUnlock(dataHandle);     {Unlock application data record MM-43}
                GOTO 1030;               {Skip rest of operation}
            END; {IF}

        IF (innovationPar < 1) AND (imitationPar < 1) THEN
            BEGIN                        {InnovationPar and imitationPar are zero!!!}
                ignore := StopAlert(nullPenetID, NIL);
                                         {Post 'nullPenetID' stop alert DM-28}
                HUnlock(dataHandle);     {Unlock application data record MM-43}
                GOTO 1030;               {Skip rest of operation}
            END; {IF}

        IF potentialPar < 1 THEN        {PotentialPar is zero!!!}
            BEGIN
                ignore := StopAlert(nullPotentialID, NIL);
                                         {Post 'nullPotentialID' stop alert DM-28}
                HUnlock(dataHandle);     {Unlock application data record MM-43}
                GOTO 1030;               {Skip rest of operation}
            END; {IF}

        GetIText(adopterEditRecd^^.hText, wkString);
                                         {Return adopterText as a STR255 DL-31}
        wkNum := stringToReal(wkString); {U6: STR255 -> REAL conversion}
        IF wkNum = 0 THEN
            BEGIN
                ignore := StopAlert(emptyTextID, NIL);
                                         {Post 'emptyTextID' alert DM-28}
                HUnlock(dataHandle);     {Unlock application data record MM-43}
                GOTO 1030;               {Skip rest of operation}
            END; {IF}

        GetIText(fCostEditRecd^^.hText, wkString);
                                         {Return fCostText as a STR255 DL-31}
        wkNum := stringToReal(wkString); {U6: STR255 -> REAL conversion}
        IF wkNum = 0 THEN
            BEGIN
                ignore := StopAlert(emptyTextID, NIL);
                                         {Post 'emptyTextID' alert DM-28}
                HUnlock(dataHandle);     {Unlock application data record MM-43}
            END;
        END;
    END;

```

```

        GOTO 1030;                                {Skip rest of operation}
    END;  {IF}

    GetIText (productionEditRecd^.hText, wkString);
                                                {Return productionText as a STR255 DL-31}
    wkNum := stringToReal (wkString);          {U6: STR255 -> REAL conversion}
    IF wkNum = 0 THEN
        BEGIN
            ignore := StopAlert (emptyTextID, NIL);
                                                {Post 'emptyTextID' alert DM-28}
            HUnlock (dataHandle);              {Unlock application data record MM-43}
            GOTO 1030;                          {Skip rest of operation}
        END;  {IF}

    GetIText (priceEditRecd^.hText, wkString);
                                                {Return priceText as a STR255 DL-31}
    wkNum := stringToReal (wkString);          {U6: STR255 -> REAL conversion}
    IF wkNum = 0 THEN
        BEGIN
            ignore := StopAlert (emptyTextID, NIL);
                                                {Post 'emptyTextID' alert DM-28}
            HUnlock (dataHandle);              {Unlock application data record MM-43}
            GOTO 1030;                          {Skip rest of operation}
        END;  {IF}

    END;  {WITH}

    HUnlock (dataHandle);                        {Unlock application data record MM-43}

    CASE theItem OF

        summaryItem:                            {Item number for summary data command}
            doSummary;                          {U6: Handle Summary command}

        innovationCurItem:                     {Item number for Innovation Cur command}
            doInnovationCur;                   {U6: Handle InnovationCur command}

        imitationCurItem:                     {Item number for Imitation Cur command}
            doImitationCur;                   {U6: Handle ImitationCur command}

        penetrationCurItem:                   {Item number for Penetration Cur command}
            doPenetrationCur;                 {U6: Handle PenetrationCur command}

        replaceCurItem:                      {Item number for Replace Cur command}
            doReplaceCur;                     {U6: Handle ReplaceCur command}

        volumeCurItem:                       {Item number for Volume Cur command}
            doVolumeCur;                      {U6: Handle VolumeCur command}

        internalVCCurItem:                    {Item number for InternalVC Cur command}
            doInternalVCCur;                   {U6: Handle InternalVCCur command}

        totalCCurItem:                       {Item number for TotalC Cur command}
            doTotalCCur;                       {U6: Handle TotalCCur command}

        profitCurItem:                      {Item number for Profit Cur command}
            doProfitCur;                      {U6: Handle ProfitCur command}

```

END; {CASE}

1030: END; {doAnalyzeChoice}

-----}

PROCEDURE doSummary; {Handle Summary command}

BEGIN {doSummary}

IF theScreen <> summary THEN
miniClean; {Clean inside of application frame}

IF theScreen = product THEN
BEGIN
deActivateProduct; {U7: Deactivate product Edit records}
DisableItem (editMenu, 0); {Disable Edit menu MN-24}
DrawMenuBar; {Redraw MnBar according to menu list MN-19}
END; {IF}

IF theScreen = environmt THEN
BEGIN
deActivateEnvironmt; {U7: Deactivate environmt Edit records}
DisableItem (editMenu, 0); {Disable Edit menu MN-24}
DrawMenuBar; {Redraw MnBar according to menu list MN-19}
END; {IF}

theScreen := summary; {Set screen flag}
displaySummary; {U6: Display input data}

END; {doSummary}

-----}

PROCEDURE displaySummary; {Display input data}

VAR

theData : apiHandle; {Handle to application data record}
dataHandle : Handle; {Untyped handle to create data record MI-9}
WorkString : STR255; {Temporary string working space}

-----}

PROCEDURE adaptString1 (VAR theString : STR255);
BEGIN {adaptString1} {Adapt length of parameter string}
{0 <= string value <= 100}

IF LENGTH (theString) = 1 THEN {Add spaces when needed for alignment}
theString := CONCAT (' ', theString);
IF LENGTH (theString) = 2 THEN
theString := CONCAT (' ', theString);
IF LENGTH (theString) = 3 THEN
theString := CONCAT (' ', theString);

END; {adaptString1}

-----}

```

PROCEDURE adaptString2 (VAR theString : STR255);
                                {Adapt length of environmt parameter strings}
                                {1digit <= string value <= 16 digits}
VAR
    nullString : STRING[40];      {Null string, for computation purpose}
    padString : STRING[40];      {Null string, used for padding}

BEGIN {adaptString2}

    nullString := '                '; {Initialize nullString to 40 'blanks'}
    padString := COPY (nullString, 1, (16 - LENGTH (theString)) # 2);

    theString := CONCAT (padString, theString); {Add spaces when needed for alignment}

END; {adaptString2}

-----

BEGIN {displaySummary}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle); {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

    WITH theData^^ DO

        BEGIN

            PenSize (2, 2); {Set dimensions of graphic pen QD-41}
            SetRect (screenRect, 20, 10, 470, 270); {Set overall screen rectangle LTRB QD-46}
            FrameRoundRect (screenRect, 10, 10); {Draw overall screen rectangle QD-51}

            PenSize (1, 1); {Set dimensions of graphic pen QD-41}

            miniCopyright; {U5: Write copyright at bottom of the screen}

            TextFace ([Bold]); {Select bold characters QD-43}
            MoveTo (40, 30); {Move pen (h,v) QD-42}
            DrawString ('Description du produit étudié');
            TextFace ([]); {Select normal characters QD-43}

            MoveTo (50, 45); {Move pen (h,v) QD-42}
            DrawString ('Division concernée...'); {Write string content QD-44}
            GetIText (divisionEditRecd^^.hText, workString);
            MoveTo (210, 45); {Return text as a string DL-31}
            DrawString (workString); {Write string content QD-44}

            MoveTo (50, 58); {Move pen (h,v) QD-42}
            DrawString ('Produit étudié...'); {Write string content QD-44}
            GetIText (brandEditRecd^^.hText, workString);
            MoveTo (210, 58); {Return text as a string DL-31}
            DrawString (workString); {Write string content QD-44}

            MoveTo (50, 71); {Move pen (h,v) QD-42}
            DrawString ('Marché visé...'); {Write string content QD-44}
            GetIText (marketEditRecd^^.hText, workString);
            MoveTo (210, 71); {Return text as a string DL-31}

```

DrawString (workString);	{Write string content QD-44}
MoveTo (50,84);	{Move pen (h,v) QD-42}
DrawString ('Evalueur...');	{Write string content QD-44}
GetIText (evaluatorEditRecd^.hText, workString);	
MoveTo (210,84);	{Return text as a string DL-31}
DrawString (workString);	{Write string content QD-44}
TextFace ([Bold]);	{Select bold characters QD-43}
MoveTo (40,104);	{Move pen (h,v) QD-42}
DrawString ('Paramètres utilisés');	{Write string content QD-44}
TextFace ([]);	{Select normal characters QD-43}
MoveTo (50,119);	{Move pen (h,v) QD-42}
DrawString ('Coeff. d'innovation...');	{Write string content QD-44}
NumToString (innovationPar,workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (210,119);	{Move pen (h,v) QD-42}
DrawString (workString);	{Writes string content QD-44}
MoveTo (250,119);	{Move pen (h,v) QD-42}
DrawString ('Coeff. d'imitation...');	{Write string content QD-44}
NumToString (imitationPar, workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (420,119);	{Move pen (h,v) QD-42}
DrawString (workString);	{Write string content QD-44}
MoveTo (50,132);	{Move pen (h,v) QD-42}
DrawString ('Coeff. d'influence...');	{Write string content QD-44}
NumToString (influencePar, workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (210,132);	{Move pen (h,v) QD-42}
DrawString (workString);	{Write string content QD-44}
MoveTo (250,132);	{Move pen (h,v) QD-42}
DrawString ('Marché potentiel...');	{Write string content QD-44}
NumToString (potentialPar, workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (420,132);	{Move pen (h,v) QD-42}
DrawString (workString);	{Write string content QD-44}
MoveTo (50,145);	{Move pen (h,v) QD-42}
DrawString ('Coeff. de renouvellement...');	{Write string content QD-44}
NumToString (replacePar, workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (210,145);	{Move pen (h,v) QD-42}
DrawString (workString);	{Write string content QD-44}
MoveTo (50,158);	{Move pen (h,v) QD-42}
DrawString ('Coût des entrants...');	{Write string content QD-44}
NumToString (externalVCPPar,workString);	{Convert to string type PK-20}
adaptString1 (workString);	{U6: Adapt length of parameter string}
MoveTo (210,158);	{Move pen (h,v) QD-42}
DrawString (workString);	{Write string content QD-44}
MoveTo (250,158);	{Move pen (h,v) QD-42}
DrawString ('Coût de la val. ajoutée...');	{Write string content QD-44}
NumToString (internalVCPPar,workString);	{Convert to string type PK-20}

```

adaptString1 (workString);           {U6: Adapt length of parameter string}
MoveTo (420,158);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

MoveTo (50,171);                      {Move pen (h,v) QD-42}
DrawString ('Taux d'apprentissage...'); {Write string content QD-44}
NumToString (learningPar, workString); {Convert to string type PK-20}
adaptString1 (workString);           {U6: Adapt length of parameter string}
MoveTo (210,171);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

TextFace ([Bold]);                    {Select bold characters QD-43}
MoveTo (40,191);                      {Move pen (h,v) QD-42}
DrawString ('Description de l'environnement');
TextFace ([]);                        {Select normal characters QD-43}

MoveTo (50,206);                      {Move pen (h,v) QD-42}
DrawString ('Adopteurs potentiels...'); {Write string content QD-44}
GetIText (adopterEditRecd^.hText, workString);
                                         {Return text as a string DL-31}
adaptString2 (workString);           {U6: Adapt length of environmt strings}
MoveTo (210,206);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

MoveTo (50,220);                      {Move pen (h,v) QD-42}
DrawString ('Coût fixe...');           {Write string content QD-44}
GetIText (fCostEditRecd^.hText, workString);
                                         {Return text as a string DL-31}
adaptString2 (workString);           {U6: Adapt length of environmt strings}
MoveTo (210,220);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

MoveTo (50,234);                      {Move pen (h,v) QD-42}
DrawString ('Palier d'expérience...'); {Write string content QD-44}
GetIText (productionEditRecd^.hText, workString);
                                         {Return text as a string DL-31}
adaptString2 (workString);           {U6: Adapt length of environmt strings}
MoveTo (210,234);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

MoveTo (50,248);                      {Move pen (h,v) QD-42}
DrawString ('Prix unitaire...');        {Write string content QD-44}
GetIText (priceEditRecd^.hText, workString);
                                         {Return text as a string DL-31}
adaptString2 (workString);           {U6: Adapt length of environmt strings}
MoveTo (210,248);                     {Move pen (h,v) QD-42}
DrawString (workString);              {Write string content QD-44}

END;  {WITH}

HUnlock (dataHandle);                 {Unlock application data record MM-43}

END;  {displaySummary}

(-----)

PROCEDURE doInnovationCur;            {Handle InnovationCur command}

```

LABEL

1009; {Exit doInnovationCur}

VAR

i : INTEGER; {Index variable for loop control}

BEGIN {doInnovationCur}

CASE theScreen OF

product:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateProduct; {U7: Deactivate product Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {product}

innovation, imitation, influence, potential, replace, externalVC,
 internalVC, learning, summary:

miniClean; {U5: Clean inside of application frame}

environmt:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateEnvironmt; {U7: Deactivate environmt Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {environmt}

imitationPlot, replacePlot:

FOR i:= 2 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

penetrationPlot:

FOR i:= 1 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

END; {CASE}

IF (theScreen = innovationPlot) AND (theEvent.what <> UpdateEvt) THEN

GOTO 1009; {This is where we are, skip rest of operation}

theScreen := innovationPlot;

{Set screen flag}

setScreenCurve;

{U6: Set screen display for graphic diagnosis}

setLabelsCurve;

{U6: Draw curve labels and mini copyright}

plotInnovation;

{U6: Plot Innovation curve}

1009: END; {doInnovationCur}

(-----)

PROCEDURE setScreenCurve;

{Set screen display for graphic diagnosis}

VAR

screenRect	:	Rect;	{Local version of screenRect QD-9}
cleanRect	:	Rect;	{Rectangle for cleaning purposes QD-9}
titleRect	:	Rect;	{Rectangle for graph title QD-9}

i	:	INTEGER;	{Index variable for loop control}
percentScreen	:	BOOLEAN;	{Percent screen flag}

BEGIN {setScreenCurve}

PenSize (2, 2);	{Set dimensions of graphic pen QD-41}
SetRect (screenRect, 20, 10, 470, 270);	{Set overall screen rectangle LTRB QD-46}
FrameRoundRect (screenRect, 10, 10);	{Draw overall screen rectangle QD-51}
PenSize (1, 1);	{Set dimensions of graphic pen QD-41}

SetRect (cleanRect, 72, 70, 450, 230);	{Set 'inside' graph cleaning rect LTRB QD-46}
FillRect (cleanRect, White);	{Fill it with 'white' QD-49, QD-34}

percentScreen := GetPixel (68, 95);	{Is it a 'percent' screen? QD-68. Get pixel}
	{below pen location QD-21}

IF percentScreen THEN	{Yes, it is}
-----------------------	--------------

CASE theScreen OF

innovationPlot, imitationPlot, penetrationPlot, replacePlot:	{Do nothing};
--	---------------

volumePlot, internalVCPlot, totalCPlot, profitPlot:	
---	--

BEGIN

SetRect(cleanRect, 30, 70, 71, 230);	{Set 'Y-labels' cleaning rect LTRB QD-46}
--------------------------------------	---

FillRect (cleanRect, White);	{Fill it with 'white' QD-49, QD-34}
------------------------------	-------------------------------------

END;

END {CASE}

ELSE	{No it is a 'minMax' screen}
------	------------------------------

CASE theScreen OF

innovationPlot, imitationPlot, penetrationPlot, replacePlot:	
--	--

BEGIN

SetRect(cleanRect, 30, 70, 71, 230);	{Set 'Y-labels' cleaning rect LTRB QD-46}
--------------------------------------	---

FillRect (cleanRect, White);	{Fill it with 'white' QD-49, QD-34}
------------------------------	-------------------------------------

END;

volumePlot, internalVCPlot, totalCPlot, profitPlot:	
---	--

{Do nothing};	
---------------	--

END; {CASE}

SetRect (titleRect, 37, 23, 201, 47);	{Set 'clean' rectangle for title LTRB QD-46}
FillRect (titleRect, White);	{Fill it with 'white' QD-49, QD-34}

PenSize (2, 2);	{Set dimensions of graphic pen QD-41}
SetRect (titleRect, 34, 20, 204, 50);	{Set title rectangle LTRB QD-46}
FrameRoundRect (titleRect, 10, 10);	{Draw title rectangle QD-51}

MoveTo (70, 230);	{Move pen (h,v) QD-42}
-------------------	------------------------

LineTo (390, 230);	{Draw x_axis QD-42}
--------------------	---------------------

PenSize (1, 1);	{Set dimensions of graphic pen QD-41}
-----------------	---------------------------------------

FOR i := 1 TO 15 DO

BEGIN

MoveTo((70 + i * 20), 230);	{Set x_axis intervals}
	{Move pen (h,v) QD-42}

LineTo ((70 + i * 20), 233);	{Draw down 3 pixels QD-42}
------------------------------	----------------------------

END; {FOR}

```

PenSize (2, 2);           {Set dimensions of graphic pen QD-41}
MoveTo (70, 230);         {Move pen (h,v) QD-42}
LineTo (70, 70);          {Draw y_axis QD-42}
PenSize (1, 1);           {Set dimensions of graphic pen QD-41}

CASE theScreen OF

    innovationPlot, imitationPlot, penetrationPlot, replacePlot:
        FOR i := 1 TO 10 DO
            BEGIN
                MoveTo(70, (230 - i * 15)); {Set y_axis intervals}
                LineTo (67, (230 - i * 15)); {Move pen (h,v) QD-42}
                END;                        {Draw back (left) 3 pixels QD-42}

    volumePlot, internalVCPlot, totalCPlot, profitPlot:
        BEGIN
            MoveTo(70, 80); {Set y_axis intervals}
            LineTo (67, 80); {Move pen (h,v) to MAX QD-42}
            MoveTo(70, 215); {Draw back (left) 3 pixels QD-42}
            LineTo (67, 215); {Move pen (h,v) to MIN QD-42}
            END;              {Draw back (left) 3 pixels QD-42}

        END; {CASE}

    END; {setScreenCurve}

{-----}

PROCEDURE setLabelsCurve;           {Draws curve labels and mini copyright}

VAR

    yLabel : STR255;               {Labels (%) for y_axis}
    i       : INTEGER;             {Index variable for loop control}

BEGIN {setLabelsCurve}

    TextFace ([Bold]);             {Select bold characters QD-43}

    CASE theScreen OF              {Write curve's title}

        innovationPlot:
            BEGIN
                MoveTo (49, 39);    {Move pen (h,v) QD-42}
                DrawString ('Courbe d''Innovation'); {Write string content QD-44}
            END; {innovationPlot}

        imitationPlot:
            BEGIN
                MoveTo (52, 39);    {Move pen (h,v) QD-42}
                DrawString ('Courbe d''Imitation'); {Write string content QD-44}
            END; {imitationPlot}

        penetrationPlot:
            BEGIN
                MoveTo (40, 39);    {Move pen (h,v) QD-42}
                DrawString ('Courbe de Pénétration'); {Write string content QD-44}
            END;
    END;

```

```

    END; {penetrationPlot}

replacePlot:
  BEGIN
    MoveTo (61, 39);           {Move pen (h,v) QD-42}
    DrawString ('Renouvellement'); {Write string content QD-44}
  END; {replacePlot}

volumePlot:
  BEGIN
    MoveTo (61, 39);           {Move pen (h,v) QD-42}
    DrawString ('Ventes Totales'); {Write string content QD-44}
  END; {volumePlot}

internalVCPlot:
  BEGIN
    MoveTo (60, 39);           {Move pen (h,v) QD-42}
    DrawString ('Coût Var Interne'); {Write string content QD-44}
  END; {internalVCPlot}

totalCPlot:
  BEGIN
    MoveTo (78, 39);           {Move pen (h,v) QD-42}
    DrawString ('Coût Total'); {Write string content QD-44}
  END; {totalCPlot}

profitPlot:
  BEGIN
    MoveTo (75, 39);           {Move pen (h,v) QD-42}
    DrawString ('Profit Brut'); {Write string content QD-44}
  END; {profitPlot}

END; {CASE}

TextFace([]);           {Select normal characters QD-43}
TextSize (9);           {Set grafPort type size to 9 QD-43}

MoveTo (168, 245); DrawString('5'); {Move pen (h,v) QD-42}
MoveTo (265, 245); DrawString('10'); {and write string content QD-44}
MoveTo (366, 245); DrawString('15');
MoveTo (390, 245); DrawString('Période'); {Write x_axis labels}

CASE theScreen OF

  innovationPlot, imitationPlot, penetrationPlot, replacePlot:
    FOR i := 1 TO 10 DO {Write y_axis labels}
      BEGIN
        MoveTo (34, (234 - i * 15)); {Move pen (h,v) QD-42}

        IF i = 10 THEN {100 % level}
          BEGIN
            NumToString (i * 10, yLabel); {Convert to string PK-20}
            yLabel := CONCAT (yLabel, '%');
          END {THEN}

        ELSE
          BEGIN
            NumToString (i * 10, yLabel); {Convert to string PK-20}

```

```

        yLabel := CONCAT (' ', yLabel, '%');
    END; {ELSE} {Add space as less than 100 % level}

    DrawString (yLabel); {Write string content QD-44}
END; {FOR}

volumePlot, internalVCPlot, totalCPlot, profitPlot:
    BEGIN {Write y-axis labels}
        TextFace([Bold]); {Select bold characters QD-43}
        MoveTo(40, 82); {Move pen (h,v) to MAX QD-42}
        DrawString ('Max'); {Write 'Max' QD-44}
        MoveTo(40, 217); {Move pen (h,v) to MIN QD-42}
        DrawString ('Min'); {Write 'Min' QD-44}
        TextFace([]); {Select normal characters QD-43}
    END;

END; {CASE}

miniCopyright; {US: Write copyright at bottom of screen}

END; {setLabelsCurve}

{-----}

PROCEDURE plotInnovation; {Plot the Innovation curve}

VAR

    P : REAL; {Cumulative adopters (%) in t}
    DP : REAL; {Adopters (%) in t}

    Y : REAL; {Cumulative innovators (%) in t}
    DY : REAL; {Innovators (%) in t}

    i, j : INTEGER; {Index variables for loop control}

    adopterString : STR255; {Temp STRING working space}
    adopterReal : REAL; {Temp REAL working space}

    curveDone : BOOLEAN; {Curve 'done' flag}

    theData : apiHandle; {Handle to application data record}
    dataHandle : Handle; {Untyped handle to create data record MI-9}

BEGIN {plotInnovation}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle); {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

    PenSize(1,1); {Set dimensions of graphic pen QD-41}
    curveDone := FALSE; {Initialize and set curve flag}

    i := 1; {Start at period one}

    WITH theData^^ DO

        BEGIN

```

```

GetIText (adopterEditRecd^.hText, adopterString);
                                {Return Text as a STRING DL-31}
adopterReal := stringToReal (adopterString);
                                {U6: STR255 -> REAL conversion}
P := innovationPar *
    potentialPar / 100;          {Cumulative adopters (%) in t = 1}
Y := P;                          {Cumulative innovators (%) in t = 1}

horizontal (TRUNC (potentialPar * 1.5)); {U6: Draw Maximum asymptote}
horizontal (TRUNC (Y * 1.5));          {U6: Draw minimum asymptote}

REPEAT

    IF Y >= potentialPar THEN      {We hit the max asymptote}
    BEGIN
        MoveTo (90 + (i-1) * 20, 230 - TRUNC (potentialPar * 1.5));
        setActiveCircle (i, potentialPar, 100, adopterReal);
        curveDone := TRUE;        {U6: Local coordinates of active circles 'i'}
    END {THEN}

    ELSE                          {The curve progresses normally}
    BEGIN
        MoveTo (90 + (i-1) * 20, 230 - TRUNC (Y * 1.5));
        setActiveCircle (i, Y, 100, adopterReal);
    END; {ELSE}                  {U6: Local coordinates of active circles 'i'}

    DP := ( innovationPar +
            imitationPar *
            exponential ((P/potentialPar), ((100 - influencePar)/100)) ) *
            (potentialPar - P) / 100; {Adopters (%) in t = i + 1}

    DY := innovationPar * (potentialPar - P) / 100;
                                {Innovators (%) in t = i + 1}

    P := P + DP;                  {Cumulative adopters (%) in t = i + 1}
    Y := Y + DY;                  {Cumulative innovators (%) in t = i + 1}

    IF Y >= potentialPar THEN      {After last increment the curve hits potential}
    BEGIN
        LineTo (90 + i * 20, 230 - TRUNC (potentialPar * 1.5));
        drawActiveCircle (i);      {U6: Draw Cursor active circle 'i'}
        curveDone := TRUE;
        FOR j := i + 1 TO 15 DO
        BEGIN                      {Move successively to all cycles}
            MoveTo (90 + j * 20, 230 - TRUNC (potentialPar * 1.5));
            setActiveCircle(j, potentialPar, 100, adopterReal);
            drawActiveCircle (j);    {U6: Draw Cursor active circle 'j'}
        END; {FOR}
    END {THEN}

    ELSE                          {The curve does not hit potential}
    BEGIN
        LineTo (90 + i * 20, 230 - TRUNC (Y * 1.5));
        drawActiveCircle (i);      {U6: Draw Cursor active circle 'i'}
        IF i = 14 THEN             {Last cycle, draw last bubble}
        BEGIN

```

```

        setActiveCircle (15, Y, 100, adopterReal);
                                {U6: Local coordinates of active circle '15'}
        drawActiveCircle (15);   {U6: Draw Cursor active circle '15'}
        curveDone := TRUE;
    END; {IF}
END; {ELSE}

    i := i+1;                    {Go to next cycle}

UNTIL curveDone; {ELSE}

END; {WITH}

HUnlock (dataHandle);           {Unlock application data record MM-43}

END; {plotInnovation}

{-----}

FUNCTION stringToReal ((myString : STR255): REAL);
                                {STR255 -> REAL conversion}
VAR
    i      : INTEGER;           {Index variable for loop control}
    result : REAL;              {Function result}

BEGIN {stringToReal}

    result := 0;                {Initialize function value}

    FOR i := 1 TO LENGTH (myString) DO
        result := result * 10 + ORD (myString[i]) - ORD('0');

    stringToReal := result;      {Set function value}

END; {stringToReal}

{-----}

FUNCTION exponential ((x, y : REAL): REAL);    {Computes x to the power of y}

BEGIN {exponential}

    exponential := EXP ( LN (x) * y);

END; {exponential}

{-----}

PROCEDURE horizontal ((y : INTEGER));          {Draws a horizontal line at (230 - y)}

BEGIN {horizontal}

    PenSize (1,1);              {Set dimensions of graphic pen QD-41}
    MoveTo (70, 230 - y);       {Move pen (h,v) QD-42}
    LineTo (370, 230 - y);      {Draw line 300 pixels on the right QD-42}

END; {horizontal}

```

{-----}

```

PROCEDURE dottedLine ((y : INTEGER));           {Draws a dotted line at (230 - y)}

VAR

  h : INTEGER;                                {index variable for loop control}

BEGIN {dottedLine}

  PenSize (1,1);                             {Set dimensions of graphic pen QD-41}
  h := 75;                                     {Initial value of index variable}

  REPEAT
    MoveTo (h, 230 - y);                      {Move pen (h,v) QD-42}
    LineTo (h + 5, 230 - y);                  {Draw line 5 pixels on the right QD-42}
    h := h + 10;                              {Increment value of 'h' by 10}
  UNTIL h = 375;

END; {dottedLine}

```

{-----}

```

PROCEDURE setActiveCircle ((i: INTEGER; y, range, base : REAL));
                                                    {Local coordinates of Cursor active circle 'i'}
                                                    {i is the array index variable (abscissa)}
                                                    {diameter of circles is equal to 6 pixels}

BEGIN {setActiveCircle}

  CASE theScreen OF

    innovationPlot, imitationPlot, penetrationPlot, replacePlot:
      BEGIN {Percent graphic plot}
        cursCircles[i].circle.Top    := 230 - 3 - TRUNC (y * 150 / range);
                                                    {150 pixels available}
        cursCircles[i].circle.Bottom := 230 + 3 - TRUNC (y * 150 / range);
      END;

    volumePlot, internalVCPlot, totalCPlot, profitPlot:
      BEGIN {MinMax graphic plot}
        cursCircles[i].circle.Top    := 215 - 3 - TRUNC (y * 135 / range);
                                                    {135 pixels available}
        cursCircles[i].circle.Bottom := 215 + 3 - TRUNC (y * 135 / range);
      END;

  END; {CASE}

  cursCircles[i].circle.Left  := 70 - 3 + (i) * 20;
  cursCircles[i].circle.Right := 70 + 3 + (i) * 20;
  cursCircles[i].predictor    := i;

```

CASE theScreen OF

```

  innovationPlot, imitationPlot, penetrationPlot, replacePlot:
    BEGIN
      cursCircles[i].percentResponse := y;
      cursCircles[i].nominalResponse := y * base / 100;
                                                    {Set nominal response to actual nber of }
    END;

```

```

        END;                                {innovators, imitators, adopters, repeaters}

volumePlot, internalVCPlot, totalCPlot, profitPlot:
    BEGIN
        cursCircles[i].percentResponse := y;
        cursCircles[i].nominalResponse := base;
                                         {Set nominal response to Minimum. A trick}
        END;                                {to pass information to U#7: DrawcurLabels}

    END; {CASE}

END; {setActiveCircle}

{-----}

PROCEDURE drawActiveCircle {(i: INTEGER)};    {Draw Cursor active circle 'i'}

VAR

    Circle : Rect;                            {Circle to be drawn QD-9}

BEGIN {drawActiveCircle}

    SetRect (circle, cursCircles[i].circle.Left,
              cursCircles[i].circle.Top,
              cursCircles[i].circle.Right,
              cursCircles[i].circle.Bottom);
                                         {Set rect surrounding active circles QD-46}
    FillOval (circle, White);              {Fill circle with 'white' QD-50, QD-34}
    FrameOval (circle);                    {Draw circle outline QD-50}

END; {drawActiveCircle}

{-----}

PROCEDURE dolimitationCur;                    {Handle ImitationCur command}

LABEL

    1010;                                     {Exit dolimitationCur}

VAR

    i : INTEGER;                             {Index variable for loop control}

BEGIN {dolimitationCur}

    CASE theScreen OF

        product:
            BEGIN
                DisableItem (editMenu, 0);    {Disable Edit menu MN-24}
                DrawMenuBar;                  {Redraw MnBar according to menu list MN-19}
                deactivateProduct;            {U7: Deactivate product Edit records}
                miniClean;                    {U5: Clean inside of application frame}
            END; {product}

        innovation, imitation, influence, potential, replace, externalVC,

```

```

internalVC, learning, summary :
    miniClean;                                     {U5: Clean inside of application frame}

environmt:
    BEGIN
        DisableItem (editMenu, 0);                 {Disable Edit menu MN-24}
        DrawMenuBar;                                {Redraw MnBar according to menu list MN-19}
        deactivateEnvironmt;                         {U7: Deactivate environmt Edit records}
        miniClean;                                   {U5: Clean inside of application frame}
    END;    {environmt}

replacePlot:
    FOR i:= 2 TO 15 DO                               {Check for an active circle on x-axis}
        IF (cursCircles[i].circle.bottom>= 229) THEN
            FillRect (cursCircles[i].circle, 'white'); {Fill it with 'white' QD-49, QD-34}

innovationPlot, penetrationPlot:
    FOR i:= 1 TO 15 DO                               {Check for an active circle on x-axis}
        IF (cursCircles[i].circle.bottom>= 229) THEN
            FillRect (cursCircles[i].circle, 'white'); {Fill it with 'white' QD-49, QD-34}

END;    {CASE}

IF (theScreen = imitationPlot) AND (theEvent.what <> UpdateEvt) THEN
    GOTO 1010;                                         {This is where we are, skip rest of operation}

theScreen := imitationPlot;                           {Set screen flag}
setScreenCurve;                                       {U6: Set screen display for graphic diagnosis}
setLabelsCurve;                                       {U6: Draw curve labels and mini copyright}
plotImitation;                                        {U6: Plot the Imitation curve}

1010: END;    {doImitationCur}

{-----}

PROCEDURE plotImitation;                               {Plots the Imitation curve}

VAR

    P            : REAL;                               {Cumulative adopters (%) in t}
    DP           : REAL;                               {Adopters (%) in t}

    Y            : REAL;                               {Cumulative imitators (%) in t}
    DY           : REAL;                               {Imitators (%) in t}

    i, j         : INTEGER;                           {Index variables for loop control}

    adopterString : STR255;                            {Temp STRING working space}
    adopterReal   : REAL;                              {Temp REAL working space}

    curveDone     : BOOLEAN;                          {Curve 'done' flag}

    theData       : apIHandle;                        {Handle to application data record}
    dataHandle    : Handle;                          {Untyped handle to create data record M1-9}

BEGIN {plotImitation}

    dataHandle := Handle(GetWRefCon(theWindow));

```

```

Hlock(dataHandle);                                {Get and lock Handle to data record WM-33}
theData := apIHandle(dataHandle);                  {Convert to apIHandle}

PenSize(1,1);                                     {Set dimensions of graphic pen QD-41}
curveDone:= FALSE;                                {Initialize and set curve flag}

i:= 2;                                              {Start in t = 2}

WITH theData^^ DO

  BEGIN

    GetIText (adopterEditRecd^^.hText, adopterString);
                                                    {Return Text as a STRING DL-31}
    adopterReal := stringToReal (adopterString);
                                                    {U6: STR255 -> REAL conversion}

    IF innovationPar = 0 THEN                      {Initialize diffusion process}
      P := 0.0001                                  {Cumulative adopters (%) in t = 1}
    ELSE
      P := innovationPar *
          potentialPar / 100;                      {Cumulative adopters (%) in t = 1}

    DP:= ( innovationPar +
            imitationPar *
            exponential ((P/potentialPar), ((100 - influencePar)/100))) *
          (potentialPar - P) / 100;                {Adopters (%) in t = 2 }

    DY:=  imitationPar *
            exponential ((P/potentialPar), ((100 - influencePar)/100))) *
          (potentialPar - P) / 100;                {Imitators (%) in t = 2}

    P := P + DP;                                  {Cumulative adopters (%) in t = 2}
    Y := DY;                                       {Cumulative imitators (%) in t = 2}

    horizontal (TRUNC (potentialPar * 1.5)); {U6: Draw maximum asymptote}
    IF innovationPar = 0 THEN
      horizontal (0)                               {U6: Draw minimum (0) asymptote}
    ELSE
      horizontal (TRUNC (Y * 1.5));                {U6: Draw minimum asymptote}

  REPEAT

    IF Y >= potentialPar THEN                      {We hit the max asymptote}
      BEGIN
        MoveTo (90 + (i-1) * 20, 230 - TRUNC (potentialPar * 1.5));
        setActiveCircle (i, potentialPar, 100, adopterReal);
        curveDone := TRUE;                        {U6: Local coordinates of active circles 'i'}
      END {THEN}

    ELSE                                           {The curve progresses normally}
      BEGIN
        MoveTo (90 + (i-1) * 20, 230 - TRUNC (Y * 1.5));
        setActiveCircle (i, Y, 100, adopterReal);
      END; {ELSE}                                {U6: Local coordinates of active circles 'i'}

    DP:= ( innovationPar +
            imitationPar *

```



```
BEGIN {doPenetrationCur}
```

```
  CASE theScreen OF
```

```
    product:
```

```
      BEGIN
```

```
        DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
        DrawMenuBar;                    {Redraw MnBar according to menu list MN-19}
        deActivateProduct;              {U7: Deactivate product Edit records}
        miniClean;                      {U5: Clean inside of application frame}
      END; {product}
```

```
    innovation, imitation, influence, potential, replace, externalVC,
```

```
    internalVC, learning, summary:
```

```
      miniClean;                        {U5: Clean inside of application frame}
```

```
    environmt:
```

```
      BEGIN
```

```
        DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
        DrawMenuBar;                    {Redraw MnBar according to menu list MN-19}
        deActivateEnvironmt;            {U7: Deactivate environmt Edit records}
        miniClean;                      {U5: Clean inside of application frame}
      END; {environmt}
```

```
    imitationPlot, replacePlot:
```

```
      FOR i:= 2 TO 15 DO                  {Check for an active circle on x-axis}
        IF (cursCircles[i].circle.bottom>= 229) THEN
          FillRect (cursCircles[i].circle, 'white'); {Fill it with 'white' QD-49, QD-34}
```

```
    innovationPlot:
```

```
      FOR i:= 1 TO 15 DO                  {Check for an active circle on x-axis}
        IF (cursCircles[i].circle.bottom>= 229) THEN
          FillRect (cursCircles[i].circle, 'white'); {Fill it with 'white' QD-49, QD-34}
```

```
  END; {CASE}
```

```
  IF (theScreen = penetrationPlot) AND (theEvent.what <> UpdateEvt) THEN
```

```
    GOTO 1011;                          {This is where we are, skip rest of operation}
```

```
  theScreen := penetrationPlot;
```

```
    {Set screen flag}
```

```
  setScreenCurve;
```

```
    {U6: Set screen display for graphic diagnosis}
```

```
  setLabelsCurve;
```

```
    {U6: Draw curve labels and mini copyright}
```

```
  plotPenetration;
```

```
    {U6: Plot the Penetration curve}
```

```
1011: END; {doPenetrationCur}
```

```
{-----}
```

```
PROCEDURE plotPenetration;
```

```
    {Plots the Penetration curve}
```

```
  VAR
```

```
    P      : REAL;
```

```
    {Cumulative adopters (%) in t}
```

```
    DP     : REAL;
```

```
    {Adopters (%) in t}
```

```
    Y      : REAL;
```

```
    {Cumulative adopters (%) in t}
```

```
    DY     : REAL;
```

```
    {Adopters (%) in t}
```

```

i, j          : INTEGER;           {Index variables for loop control}

adopterString : STR255;           {Temp STRING working space}
adopterReal   : REAL;            {Temp REAL working space}

curveDone     : BOOLEAN;         {Curve 'done' flag}

theData       : apIHandle;       {Handle to application data record}
dataHandle    : Handle;         {Untyped handle to create data record MI-9}

BEGIN {plotPenetration}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
    theData := apIHandle(dataHandle); {Convert to apIHandle}

    PenSize(1,1);               {Set dimensions of graphic pen QD-41}
    curveDone := FALSE;         {Initialize and set curve flag}

    i := 1;                     {Start at period one}

    WITH theData^^ DO

        BEGIN

            GetIText(adopterEditRecd^^.hText, adopterString);
                                {Return Text as a STRING DL-31}
            adopterReal := stringToReal(adopterString);
                                {U6: STR255 -> REAL conversion}

            IF innovationPar = 0 THEN {Initialize diffusion process}
                P := 0.0001 {Cumulative adopters (%) in t = 1}
            ELSE
                P := innovationPar *
                    potentialPar / 100; {Cumulative adopters (%) in t = 1}

            Y := P; {Cumulative adopters (%) in t = 1}

            horizontal(TRUNC(potentialPar * 1.5)); {U6: Draw maximum asymptote}
            IF innovationPar = 0 THEN
                horizontal(0) {U6: Draw minimum (0) asymptote}
            ELSE
                horizontal(TRUNC(Y * 1.5)); {U6: Draw minimum (0) asymptote}

            REPEAT

                IF Y >= potentialPar THEN {We hit the max asymptote}
                    BEGIN
                        MoveTo(90 + (i-1) * 20, 230 - TRUNC(potentialPar * 1.5));
                        setActiveCircle(i, potentialPar, 100, adopterReal);
                        curveDone := TRUE; {U6: Local coordinates of active circles 'i'}
                    END {THEN}

                ELSE {The curve progresses normally}
                    BEGIN
                        MoveTo(90 + (i-1) * 20, 230 - TRUNC(Y * 1.5));
                        setActiveCircle(i, Y, 100, adopterReal);

```



```

i : INTEGER;                                {Index variable for loop control}

BEGIN {doReplaceCur}

CASE theScreen OF

    product:
        BEGIN
            DisableItem (editMenu, 0);        {Disable Edit menu MN-24}
            DrawMenuBar;                       {Redraw MnBar according to menu list MN-19}
            deActivateProduct;                 {U7: Deactivate product Edit records}
            miniClean;                         {U5: Clean inside of application frame}
        END; {product}

    innovation, imitation, influence, potential, replace, externalVC,
    internalVC, learning, summary:
        miniClean;                            {U5: Clean inside of application frame}

    environmt:
        BEGIN
            DisableItem (editMenu, 0);        {Disable Edit menu MN-24}
            DrawMenuBar;                       {Redraw MnBar according to menu list MN-19}
            deActivateEnvironmt;               {U7: Deactivate environmt Edit records}
            miniClean;                         {U5: Clean inside of application frame}
        END; {environmt}

    imitationPlot:
        FOR i:= 2 TO 15 DO                    {Check for an active circle on x-axis}
            IF (cursCircles[i].circle.bottom >= 229) THEN
                FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

    innovationPlot, penetrationPlot:
        FOR i:= 1 TO 15 DO                    {Check for an active circle on x-axis}
            IF (cursCircles[i].circle.bottom >= 229) THEN
                FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

END; {CASE}

IF (theScreen = replacePlot) AND (theEvent.what <> UpdateEvt) THEN
    GOTO 1012;                                {This is where we are, skip rest of operation}

theScreen := replacePlot;                     {Set screen flag}
setScreenCurve;                              {U6: Set screen display for graphic diagnosis}
setLabelsCurve;                              {U6: Draw curve labels and mini copyright}
plotReplace;                                 {U6: Plot the Replace curve}

1012: END; {doReplaceCur}

```

{-----}

```

PROCEDURE plotReplace;                        {Plot the Replace curve}

```

```

VAR

```

```

P          : REAL;                          {Cumulative adopters (%) in t}
DP         : REAL;                          {Adopters (%) in t}

DY         : REAL;                          {Repeaters (%) in t}

```

```

i, j          : INTEGER;           {Index variables for loop control}

adopterString : STR255;           {Temp STRING working space}
adopterReal   : REAL;            {Temp REAL working space}

curveDone     : BOOLEAN;          {Curve 'done' flag}

theData       : apiHandle;        {Handle to application data record}
dataHandle    : Handle;          {Untyped handle to create data record MI-9}

BEGIN {plotReplace}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);             {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

    PenSize(1,1);                 {Set dimensions of graphic pen QD-41}
    curveDone := FALSE;           {Initialize and set curve flag}

    i := 2;                       {Start at period two}

    WITH theData DO

        BEGIN

            GetIText(adopterEditRecd^.hText, adopterString);
                                     {Return Text as a STRING DL-31}
            adopterReal := stringToReal(adopterString);
                                     {U6: STR255 -> REAL conversion}

            IF innovationPar = 0 THEN {Initialize diffusion process}
                P := 0.0001          {Cumulative adopters (%) in t = 1}
            ELSE
                P := innovationPar *
                    potentialPar / 100; {Cumulative adopters (%) in t = 1}

            DP := ( innovationPar +
                    imitationPar *
                    exponential((P/potentialPar), ((100 - influencePar)/100)) ) *
                    (potentialPar - P) / 100; {Adopters (%) in t = 2}

            DY := P * replacePar / 100; {Repeaters (%) in t = 2}

            P := P + DP;               {Cumulative adopters (%) in t = 2}

            horizontal(TRUNC(potentialPar * 1.5)); {U6: Draw maximum asymptote}
            horizontal(TRUNC(DY * 1.5)); {U6: Draw minimum asymptote}

            REPEAT

                IF DY >= potentialPar THEN {We hit the max asymptote}
                    BEGIN
                        MoveTo(90 + (i-1) * 20, 230 - TRUNC(potentialPar * 1.5));
                        setActiveCircle(i, potentialPar, 100, adopterReal);
                        curveDone := TRUE; {U6: Local coordinates of active circles 'i'}
                    END {THEN}
                END
            UNTIL curveDone;
        END
    END

```

```

ELSE
    BEGIN
        MoveTo (90 + (i-1) * 20, 230 - TRUNC (DY * 1.5));
        setActiveCircle (i, DY, 100, adopterReal);
    END; {ELSE}
    {U6: Local coordinates of active circles 'i'}

DP := ( innovationPar +
        imitationPar *
        exponential ((P/potentialPar), ((100 - influencePar)/100))) *
        (potentialPar - P) / 100; {Adopters (%) in t = i + 1}

DY := P * replacePar / 100; {Repeaters (%) in t = i + 1}

P := P + DP; {Cumulative adopters (%) in t = i + 1}

IF DY >= potentialPar THEN {After last increment the curve hits potential}
    BEGIN
        LineTo (90 + i * 20, 230 - TRUNC (potentialPar * 1.5));
        drawActiveCircle (i); {U6: Draw Cursor active circle 'i'}
        curveDone := TRUE;
        FOR j := i + 1 TO 15 DO
            BEGIN
                {Move successively to all cycles}
                MoveTo (90 + j * 20, 230 - TRUNC (potentialPar * 1.5));
                setActiveCircle (j, potentialPar, 100, adopterReal);
                drawActiveCircle (j); {U6: Draw Cursor active circle 'j'}
            END; {FOR}
        END {THEN}

    ELSE {The curve does not hit potential}
        BEGIN
            LineTo (90 + i * 20, 230 - TRUNC (DY * 1.5));
            drawActiveCircle (i); {U6: Draw Cursor active circle 'i'}
            IF i = 14 THEN {Last cycle, draw last bubble}
                BEGIN
                    setActiveCircle (15, DY, 100, adopterReal);
                    {U6: Local coordinates of active circle '15'}
                    drawActiveCircle (15); {U6: Draw Cursor active circle '15'}
                    curveDone := TRUE;
                END; {IF}
            END; {ELSE}

            i := i + 1; {Go to next cycle}

        UNTIL curveDone; {ELSE}

    END; {WITH}

    HUnlock (dataHandle); {Unlock application data record MM-43}

END; {plotReplace}

{-----}

PROCEDURE doVolumeCur; {Handle VolumeCur command}

LABEL

1013; {Exit doVolumeCur}

```

VAR

i : INTEGER; {Index variable for loop control}

BEGIN {doVolumeCur}

CASE theScreen OF

product:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateProduct; {U7: Deactivate product Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {product}

innovation, imitation, influence, potential, replace, externalVC,
 internalVC, learning, summary:

miniClean; {U5: Clean inside of application frame}

environmt:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateEnvironmt; {U7: Deactivate environmt Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {environmt}

imitationPlot, replacePlot:

FOR i:= 2 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

innovationPlot, penetrationPlot:

FOR i:= 1 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

END; {CASE}

IF (theScreen = volumePlot) AND (theEvent.what <> UpdateEvt) THEN

GOTO 1013; {This is where we are, skip rest of operation}

theScreen := VolumePlot; {Set screen flag}
 setScreenCurve; {U6: Set screen display for graphic diagnosis}
 setLabelsCurve; {U6: Draw curve labels and mini copyright}
 plotVolume; {U6: Plot the Volume curve}

1013: END; {doVolumeCur}

----->

PROCEDURE plotVolume; {Plots the Volume curve}

VAR

P : REAL; {Cumulative adopters (%) in t}

```

DP          : REAL;           {Adopters (%) in t}

i           : INTEGER;        {Index variables for loop control}

adopterString : STR255;       {Temp STRING working space}
adopterReal   : REAL;         {Temp REAL working space}

theData      : apiHandle;      {Handle to application data record}
dataHandle   : Handle;        {Untyped handle to create data record MI-9}

buyers       : ARRAY [1..15] OF REAL; {Buyers (adoption + repeat) in t}
buyersMax    : REAL;          {Maximum of buyers (adoption + repeat)}
buyersMin    : REAL;          {Minimum of buyers (adoption + repeat)}
buyersRange  : REAL;          {Range of buyers ( A + R ) over 15 periods}

```

{-----}

```

PROCEDURE peakBuyers;          {Max, min, range of buyers}

```

```

VAR

```

```

    j : INTEGER;               {Index variables for loop control}

```

```

BEGIN {peakBuyers}

```

```

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

```

```

    WITH theData DO

```

```

        BEGIN

```

```

            IF innovationPar = 0 THEN      {Initialize diffusion process}
                P := 0.0001                {Cumulative adopters (%) in t = 1}
            ELSE
                P := innovationPar *
                    potentialPar / 100;     {Cumulative adopters (%) in t = 1}

```

```

            buyers[1] := P;                {Buyers (adoption + repeat) in t = 1}
            buyersMax := P;                {Maximum of buyers (adoption + repeat)}
            buyersMin := P;                {Minimum of buyers (adoption + repeat)}

```

```

            FOR j := 2 TO 15 DO

```

```

                BEGIN

```

```

                    DP := ( innovationPar +
                        imitationPar *
                        exponential ((P/potentialPar), ((100 - influencePar)/100))) *
                        (potentialPar - P) / 100; {Adopters (%) in t = j}

```

```

                    IF P + DP >= potentialPar THEN {Cumulative adopters (%) = potential}
                        DP := 0;

```

```

                    buyers[j] := DP + (P * replacePar / 100);
                        {Buyers (adoption + repeat) in t = j}

```

```

                    IF buyersMax < buyers[j] THEN

```

```

        buyersMax := buyers[j]; {Maximum of buyers = buyers[j]}
    IF buyersMin > buyers[j] THEN
        buyersMin := buyers[j]; {Minimum of buyers = buyers[j]}

    P := P + DP; {Cumulative adopters (%) in t = j}

END; {FOR}

buyersRange := buyersMax - buyersMin;
{Range of buyers ( A + R ) over 15 periods}

END; {WITH}

HUnlock (dataHandle); {Unlock application data record MM-43}

END; {peakBuyers}

-----)

BEGIN {plotVolume}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle); {Get and lock Handle to data record 'WM-33'}
    theData := apIHandle(dataHandle); {Convert to apIHandle}

    PenSize(1,1); {Set dimensions of graphic pen QD-41}

    WITH theData DO

        BEGIN

            GetIText (adopterEditRecd^.hText, adopterString);
            {Return Text as a STRING DL-31}
            adopterReal := stringToReal (adopterString);
            {U6: STR255 -> REAL conversion}

            peakBuyers; {Max, min, range of buyers}

            dottedLine (150); {U6: Draw a dotted line at (80)}
            dottedLine (15); {U6: Draw a dotted line at (215)}

            MoveTo (90, 215 - TRUNC ((buyers[1] - buyersMin) * 135 / buyersRange));
            {Move to buyers (A + R) in t = 1}

            FOR i := 1 TO 15 DO

                BEGIN

                    setActiveCircle ( i, (buyers[i] - buyersMin), buyersRange, buyersMin);
                    {U6: Local coordinates of active circles 'i'}

                    IF i <= 14 THEN
                        BEGIN
                            LineTo (90 + i * 20, 215 - TRUNC ((buyers[i + 1] - buyersMin) * 135
                                / buyersRange));
                                {Line to buyers (A + R) in t = i + 1}
                            drawActiveCircle (i); {U6: Draw Cursor active circle 'i'}
                        END {THEN}
                    ELSE
                        drawActiveCircle (15); {U6: Draw Cursor active circle '15'}
                END
            END
        END
    END

```

```

        END; {FOR}

    END; {WITH}

    HUnlock (dataHandle);           {Unlock application data record MM-43}

    END; {plotVolume}

{-----}

PROCEDURE doInternalVCCur;          {Handle InternalVCCur command}

    LABEL

    1014;                          {Exit doInternalVCCur}

    VAR

    i : INTEGER;                   {Index variable for loop control}

    BEGIN {doInternalVCCur}

        CASE theScreen OF

            product:
                BEGIN
                    DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
                    DrawMenuBar;                    {Redraw MnBar according to menu list MN-19}
                    deActivateProduct;              {U7: Deactivate product Edit records}
                    miniClean;                      {U5: Clean inside of application frame}
                END; {product}

            innovation, imitation, influence, potential, replace, externalVC,
            internalVC, learning, summary:
                miniClean;                          {U5: Clean inside of application frame}

            environmt:
                BEGIN
                    DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
                    DrawMenuBar;                    {Redraw MnBar according to menu list MN-19}
                    deActivateEnvironmt;            {U7: Deactivate environmt Edit records}
                    miniClean;                      {U5: Clean inside of application frame}
                END; {environmt}

            imitationPlot, replacePlot:
                FOR i:= 2 TO 15 DO                   {Check for an active circle on x-axis}
                    IF (cursCircles[i].circle.bottom>= 229) THEN
                        FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}
                    END IF;
                END FOR;

            innovationPlot, penetrationPlot:
                FOR i:= 1 TO 15 DO                   {Check for an active circle on x-axis}
                    IF (cursCircles[i].circle.bottom>= 229) THEN
                        FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}
                    END IF;
                END FOR;

        END; {CASE}

        IF (theScreen = internalVCPlot) AND (theEvent.what <> UpdateEvt) THEN
            GOTO 1014;                               {This is where we are, skip rest of operation}
        END IF;
    END;

```



```

ELSE
  P := innovationPar *
    potentialPar / 100;           {Cumulative adopters (%) in t = 1}

  cumBuyers[1] := P;             {Cumulative buyers (A + B) in t = 1}

  FOR j := 2 TO 15 DO

    BEGIN

      DP := ( innovationPar +
        imitationPar *
        exponential ((P/potentialPar), ((100 - influencePar)/100)) ) *
        (potentialPar - P) / 100; {Adopters (%) in t = j}

      IF P + DP >= potentialPar THEN {Cumulative adopters (%) = potential}
        DP := 0;

      cumBuyers[j] := cumBuyers[j-1] + DP + (P * replacePar / 100);
        {Cumulative buyers (A + R) in t = j}

      P := P + DP;                {Cumulative adopters (%) in t = j}

    END; {FOR}

  END; {WITH}

  HUnlock (dataHandle);           {Unlock application data record M1-43}

  END; {assessBuyers}

{-----}

BEGIN {plotInternalVC}

  dataHandle := Handle(GetWRefCon(theWindow));
  Hlock(dataHandle);              {Get and lock Handle to data record WM-33}
  theData := apiHandle(dataHandle); {Convert to apiHandle}

  PenSize(1,1);                  {Set dimensions of graphic pen QD-41}
  assessBuyers;                  {U6: Assess cumulative buyers (A + R) in t}

  WITH theData DO

    BEGIN

      GetIText (adopterEditRecd^.hText, adopterString);
        {Return Text as a STRING DL-31}
      adopterReal := stringToReal (adopterString);
        {U6: STR255 -> REAL conversion}
      GetIText (productionEditRecd^.hText, productionString);
      productionReal := stringToReal (productionString);

      eta := - (ln (learningPar/100) / ln (2)); {Assess internalVC elasticity}

      IF (cumBuyers[1] * adopterReal / 100) > productionReal THEN
        BEGIN {Assess STARTING point of internalVC curve}
          reductionFactor := exponential
            ((cumBuyers[1] * adopterReal / 100) / productionReal,

```

```

        - eta);
    internalVCNow := internalVCPar * reductionFactor;
    internalVCMax := internalVCNow;
END {THEN}
ELSE
BEGIN
    internalVCNow := internalVCPar;
    internalVCMax := internalVCNow;
END; {ELSE}

IF (cumBuyers[15] * adopterReal / 100) > productionReal THEN
BEGIN
    {Assess ENDING point of internalVC curve}
    reductionFactor := exponential
        ((cumBuyers[15] * adopterReal / 100) / productionReal,
        - eta);
    internalVCMin := internalVCPar * reductionFactor;
END {THEN}
ELSE
    internalVCMin := internalVCPar;

internalVCRange := internalVCMax - internalVCMin;

dotedLine (150); {U6: Draw a doted line at 80}
dotedLine (15); {U6: Draw a doted line at 215}

MoveTo (90, 215 - TRUNC ((internalVCNow - internalVCMin) * 135
    / internalVCRange));
    {InternalVC estimate in t = 1}
FOR i := 1 TO 15 DO
    BEGIN
        setActiveCircle ( i, (internalVCNow - internalVCMin), internalVCRange,
            internalVCMin);
        IF i <= 14 THEN {U6: Local coordinates of active circles 'i'}
            BEGIN
                IF (cumBuyers[i+1] * adopterReal / 100) > productionReal THEN
                BEGIN
                    reductionFactor := exponential
                        ((cumBuyers[i+1] * adopterReal/100) / productionReal,
                        - eta);
                    internalVCNow := internalVCPar * reductionFactor;
                END {THEN}
                ELSE
                    internalVCNow := internalVCPar;

                LineTo (90 + i * 20, 215 - TRUNC ((internalVCNow - internalVCMin) * 135
                    / internalVCRange));
                    drawActiveCircle (i); {U6: Draw Cursor active circle 'i'}
                END {THEN}
            ELSE
                drawActiveCircle (15); {U6: Draw Cursor active circle '15'}

        END; {FOR}

    END; {WITH}

HUnlock (dataHandle); {Unlock application data record MM1-43}

```

END; {plotInternalVC}

{-----}

PROCEDURE doTotalCCur; {Handle TotalCCur command}

LABEL

1015; {Exit doTotalCCur}

VAR

i : INTEGER; {Index variable for loop control}

BEGIN {doTotalCCur}

CASE theScreen OF

product:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateProduct; {U7: Deactivate product Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {product}

innovation, imitation, influence, potential, replace, externalVC,
 internalVC, learning, summary:

miniClean; {U5: Clean inside of application frame}

environmt:

BEGIN

DisableItem (editMenu, 0); {Disable Edit menu MN-24}
 DrawMenuBar; {Redraw MnBar according to menu list MN-19}
 deactivateEnvironmt; {U7: Deactivate environmt Edit records}
 miniClean; {U5: Clean inside of application frame}
 END; {environmt}

imitationPlot, replacePlot:

FOR i:= 2 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

innovationPlot, penetrationPlot:

FOR i:= 1 TO 15 DO {Check for an active circle on x-axis}
 IF (cursCircles[i].circle.bottom>= 229) THEN
 FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

END; {CASE}

IF (theScreen = totalCPlot) AND (theEvent.what <> UpdateEvt) THEN

GOTO 1015; {This is where we are, skip rest of operation}

theScreen := totalCPlot;

{Set screen flag}

setScreenCurve;

{U6: Set screen display for graphic diagnosis}

setLabelsCurve;

{U6: Draw curve labels and mini copyright}

plotTotalC;

{U6: Plot the TotalC curve}

1015: END; {doTotalCCur}

{-----}

PROCEDURE plotTotalC; {Plots the TotalC curve}

VAR

P	: REAL;	{Cumulative adopters (%) in t}
DP	: REAL;	{Adopters (%) in t}
i	: INTEGER;	{Index variables for loop control}
adopterString	: STR255;	{Temp STRING working space}
fCostString	: STR255;	{Temp STRING working space}
productionString	: STR255;	{Temp STRING working space}
priceString	: STR255;	{Temp STRING working space}
adopterReal	: REAL;	{Temp REAL working space}
fCostReal	: REAL;	{Temp REAL working space}
productionReal	: REAL;	{Temp REAL working space}
priceReal	: REAL;	{Temp REAL working space}
theData	: apiHandle;	{Handle to application data record}
dataHandle	: Handle;	{Untyped handle to create data record MI-9}
cumBuyers	: ARRAY [1..15] OF REAL;	{Cumulative buyers (adoption + repeat) in t}
totalCMax	: REAL;	{Maximum value of totalC}
totalCMin	: REAL;	{Minimum value of totalC}
totalCRange	: REAL;	{Range of totalC estimates}
totalCNow	: REAL;	{Current value of totalC}
reductionFactor	: REAL;	{Experience slope}
eta	: REAL;	{InternalVC elasticity}

{-----}

PROCEDURE assessBuyers; {Assess cumulative buyers (A + R) in t}

VAR

j : INTEGER; {Index variable for loop control}

BEGIN {assessBuyers}

dataHandle := Handle(GetWRefCon(theWindow));
 Hlock(dataHandle); {Get and lock Handle to data record WM-33}
 theData := apiHandle(dataHandle); {Convert to apiHandle}

WITH theData^^ DO

BEGIN

IF innovationPar = 0 THEN	{Initialize diffusion process}
P := 0.0001	{Cumulative adopters (%) in t = 1}
ELSE	

```

    P := innovationPar *
        potentialPar / 100;           {Cumulative adopters (%) in t = 1}

    cumBuyers[1] := P;                {Cumulative buyers (A + B) in t = 1}

    FOR j := 2 TO 15 DO

        BEGIN

            DP := ( innovationPar +
                    imitationPar *
                    exponential ((P/potentialPar), ((100 - influencePar)/100))) *
                    (potentialPar - P) / 100; {Adopters (%) in t = j}

            IF P + DP >= potentialPar THEN {Cumulative adopters (%) = potential}
                DP := 0;

            cumBuyers[j] := cumBuyers[j-1] + DP + (P * replacePar / 100);
                                {Cumulative buyers (A + R) in t = j}
            P := P + DP;         {Cumulative adopters (%) in t = j}

        END; {FOR}

    END; {WITH}

    HUnlock (dataHandle);         {Unlock application data record MM-43}

    END; {assessBuyers}

{-----}

BEGIN {plotTotalC}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

    PenSize(1,1);               {Set dimensions of graphic pen QD-41}
    assessBuyers;               {U6: Assess cumulative buyers (A + R) in t}

    WITH theData DO

        BEGIN

            GetIText (adopterEditRecd^.hText, adopterString);
                                {Return Text as a STRING DL-31}
            adopterReal := stringToReal (adopterString);
                                {U6: STR255 -> REAL conversion}

            GetIText (fCostEditRecd^.hText, fCostString);
            fCostReal := stringToReal (fCostString);

            GetIText (productionEditRecd^.hText, productionString);
            productionReal := stringToReal (productionString);

            GetIText (priceEditRecd^.hText, priceString);
            priceReal := stringToReal (priceString);

```

```

eta := - (ln (learningPar/100) / ln (2)); {Assess internalVC elasticity}

IF (cumBuyers[1] * adopterReal / 100) > productionReal THEN
  BEGIN
    {Assess STARTING point of totalC curve}
    reductionFactor := exponential
      ((cumBuyers[1] * adopterReal / 100) / productionReal,
      - eta);
    totalCNow := internalVCPar * reductionFactor
      + externalVCPar
      + (fCostReal / cumBuyers[1]) * (100 / priceReal);
    totalCMax := totalCNow;
  END {THEN}
ELSE
  BEGIN
    totalCNow := internalVCPar
      + externalVCPar
      + (fCostReal / cumBuyers[1]) * (100 / priceReal);
    totalCMax := totalCNow;
  END; {ELSE}

IF (cumBuyers[15] * adopterReal / 100) > productionReal THEN
  BEGIN
    {Assess ENDING point of totalC curve}
    reductionFactor := exponential
      ((cumBuyers[15] * adopterReal / 100) / productionReal,
      - eta);
    totalCMin := internalVCPar * reductionFactor
      + externalVCPar
      + (fCostReal / cumBuyers[15]) * (100 / priceReal);
  END {THEN}
ELSE
  totalCMin := internalVCPar
    + externalVCPar
    + (fCostReal / cumBuyers[15]) * (100 / priceReal);

totalCRange := totalCMax - totalCMin;

dotedLine (150); {U6: Draw a doted line at 80}
dotedLine (15); {U6: Draw a doted line at 215}

MoveTo (90, 215 - TRUNC ((totalCNow - totalCMin) * 135
  / totalCRange));
{TotalC estimate in period t = 1}

FOR i := 1 TO 15 DO

  BEGIN

    setActiveCircle ( i, (totalCNow - totalCMin), totalCRange,
      totalCMin);

    IF i <= 14 THEN {U6: Local coordinates of active circles 'i'}
      BEGIN
        IF (cumBuyers[i+1] * adopterReal / 100) > productionReal THEN
          BEGIN
            reductionFactor := exponential
              ((cumBuyers[i+1] * adopterReal / 100) / productionReal,
              - eta);
            totalCNow := internalVCPar * reductionFactor
              + externalVCPar
              + (fCostReal / cumBuyers[i+1]) * (100 / priceReal);
          END
        END
      END
    END
  END

```

```

        END    {THEN}
    ELSE
        totalCNow := internalVCPAr
                    + externalVCPAr
                    + (fCostReal / cumBuyers[i+1]) * (100 / priceReal);

        LineTo (90 + i * 20, 215 - TRUNC ((totalCNow - totalCMin) * 135
                                            / totalCRange));
        drawActiveCircle (i);      {U6: Draw Cursor active circle 'i'}
    END    {THEN}
ELSE
    drawActiveCircle (15);      {U6: Draw Cursor active circle '15'}

END;    {FOR}

END;    {WITH}

HUnlock (dataHandle);          {Unlock application data record MM-43}

END;    {plotTotalC}

{-----}

PROCEDURE doProfitCur;          {Handle ProfitCur command}

LABEL

    1016;                        {Exit doProfitCur}

VAR

    i : INTEGER;                 {Index variable for loop control}

BEGIN {doProfitCur}

    CASE theScreen OF

        product:
            BEGIN
                DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
                DrawMenuBar;                     {Redraw MnBar according to menu list MN-19}
                deActivateProduct;               {U7: Deactivate product Edit records}
                miniClean;                       {U5: Clean inside of application frame}
            END; {product}

        innovation, imitation, influence, potential, replace, externalVC,
        internalVC, learning, summary:
            miniClean;                          {U5: Clean inside of application frame}

        environmt:
            BEGIN
                DisableItem (editMenu, 0);      {Disable Edit menu MN-24}
                DrawMenuBar;                     {Redraw MnBar according to menu list MN-19}
                deActivateEnvironmt;             {U7: Deactivate environmt Edit records}
                miniClean;                       {U5: Clean inside of application frame}
            END; {environmt}

        imitationPlot, replacePlot:

```

```

FOR i:= 2 TO 15 DO                                {Check for an active circle on x-axis}
  IF (cursCircles[i].circle.bottom>= 229) THEN
    FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

innovationPlot, penetrationPlot:
  FOR i:= 1 TO 15 DO                                {Check for an active circle on x-axis}
    IF (cursCircles[i].circle.bottom>= 229) THEN
      FillRect (cursCircles[i].circle, White); {Fill it with 'white' QD-49, QD-34}

END; {CASE}

IF (theScreen = profitPlot) AND (theEvent.what <> UpdateEvt) THEN
  GOTO 1016;                                         {This is where we are, skip rest of operation}

theScreen := profitPlot;                            {Set screen flag}
setScreenCurve;                                    {U6: Set screen display for graphic diagnosis}
setLabelsCurve;                                    {U6: Draw curve labels and mini copyright}
plotProfit;                                         {U6: Plot the Profit curve}

1016: END; {doProfitCur}

{-----}

PROCEDURE plotProfit;                               {Plots the Profit curve}

VAR

  P          : REAL;                               {Cumulative adopters (%) in t}
  DP         : REAL;                               {Adopters (%) in t}

  i          : INTEGER;                            {Index variable for loop control}

  adopterString : STR255;                          {Temp STRING working space}
  fCostString   : STR255;                          {Temp STRING working space}
  productionString : STR255;                        {Temp STRING working space}
  priceString   : STR255;                          {Temp STRING working space}

  adopterReal   : REAL;                             {Temp REAL working space}
  fCostReal     : REAL;                             {Temp REAL working space}
  productionReal : REAL;                             {Temp REAL working space}
  priceReal     : REAL;                             {Temp REAL working space}

  theData      : apIHandle;                         {Handle to application data record}
  dataHandle   : Handle;                           {Untyped handle to create data record MI-9}

  buyers       : ARRAY [1..15] OF REAL;            {Buyers (adoption + repeat) in t}
  cumBuyers    : ARRAY [1..15] OF REAL;            {Cumulative buyers (adoption + repeat) in t}
  profit       : ARRAY [1..15] OF REAL;            {Profit level in t}

  reductionFactor : REAL;                          {Experience slope}
  totalCNow      : REAL;                           {Current value of totalC}

  eta          : REAL;                             {InternalVC elasticity}

  profitMin     : REAL;                             {Minimum value of profit}
  profitMax     : REAL;                             {Maximum value of profit}
  profitRange   : REAL;                             {Range of profit estimates over 15 periods}

```

```
{-----}
```

```

PROCEDURE profitEstimates;           {Estimates profit level per period}

VAR

    j          : INTEGER;           {Index variable for loop control}

BEGIN {profitEstimates}

    dataHandle := Handle(GetWRefCon(theWindow));
    Hlock(dataHandle);               {Get and lock Handle to data record WM-33}
    theData := apiHandle(dataHandle); {Convert to apiHandle}

    WITH theData DO

        BEGIN

            GetIText (adopterEditRecd^.hText, adopterString);
                                     {Return Text as a STRING DL-31}
            adopterReal := stringToReal (adopterString);
                                     {U6: STR255 -> REAL conversion}

            GetIText (fCostEditRecd^.hText, fCostString);
            fCostReal := stringToReal (fCostString);

            GetIText (productionEditRecd^.hText, productionString);
            productionReal := stringToReal (productionString);

            GetIText (priceEditRecd^.hText, priceString);
            priceReal := stringToReal (priceString);

            eta := - (ln (learningPar / 100) / ln (2));
                                     {Assess internalVC elasticity}

            IF innovationPar = 0 THEN   {Initialize diffusion process}
                P := 0.0001            {Cumulative adopters (%) in t = 1}
            ELSE
                P := innovationPar *
                    potentialPar / 100; {Cumulative adopters (%) in t = 1}

            buyers[1] := P;             {Buyers (A + R) in t = 1}
            cumBuyers[1] := P;          {Cumulative buyers (A + R) in t = 1}

            IF (cumBuyers[1] * adopterReal / 100) > productionReal THEN
                BEGIN
                    reductionFactor := exponential
                                     ((cumBuyers[1] * adopterReal / 100) / productionReal,
                                      - eta);
                    totalCNow := internalVCPPar * reductionFactor
                               + externalVCPPar
                               + (fCostReal / cumBuyers[1]) * (100 / priceReal);
                END {THEN}
            ELSE
                totalCNow := internalVCPPar
                               + externalVCPPar
                               + (fCostReal / cumBuyers[1]) * (100 / priceReal);
        END
    END

```

```

profit[1] := (buyers[1] * adopterReal / 100)
            * (priceReal * (100 - totalCNow) / 100);
profitMin := profit[1];           {Initialize minimum profit}
profitMax := profit[1];           {Initialize maximum profit}

FOR j := 2 TO 15 DO

    BEGIN

        DP := ( innovationPar +
                imitationPar *
                exponential ((P/potentialPar), ((100 - influencePar)/100))) *
                (potentialPar - P) / 100; {Adopters (%) in t = j}

        IF P + DP >= PotentialPar THEN {Cumulative adopters (%) = potential}
            DP := 0;

        buyers[j] := DP + (P * replacePar / 100);
                                {Buyers (A + P) in t = j}
        cumBuyers[j] := cumBuyers[j-1] + DP + (P * replacePar / 100);
                                {Cumulative buyers (A + P) in t = j}

        IF (cumBuyers[j] * adopterReal / 100) > productionReal THEN
            BEGIN
                reductionFactor := exponential
                    ((cumBuyers[j] * adopterReal / 100) / productionReal,
                     - eta);
                totalCNow := internalVCPAr * reductionFactor
                    + externalVCPAr
                    + (fCostReal / cumBuyers[j]) * (100 / priceReal);
            END {THEN}

        ELSE
            totalCNow := internalVCPAr
                + externalVCPAr
                + (fCostReal / cumBuyers[j]) * (100 / priceReal);

        profit[j] := ( buyers[j] * adopterReal / 100)
                    * (priceReal * (100 - totalCNow) / 100);

        IF profitMax < profit[j] THEN {Update 'extreme' profit estimates}
            profitMax := profit[j];
        IF profitMin > profit[j] THEN
            profitMin := profit[j];

        P := P + DP; {Computes cumulative adopters}

    END; {FOR}

END; {WITH}

profitRange := profitMax - profitMin;

HUnlock (dataHandle); {Unlock application data record MM-43}

END; {profitEstimates}

{-----}

```

```

BEGIN {plotProfit}

dataHandle := Handle(GetWRefCon(theWindow));
Hlock(dataHandle);           {Get and lock Handle to data record WM-33}
theData := apIHandle(dataHandle); {Convert to apIHandle}

PenSize(1,1);                {Set dimensions of graphic pen QD-41}

profitEstimates;             {U6: Estimates profit level per period}

WITH theData^^ DO

  BEGIN

    IF profitMax <= 0 THEN
      BEGIN
        horizontal(150);      {U6: Draw 'zero' profit line at top of diagram}
        horizontal(149);      {U6: Double the line!!!}
      END {THEN}
    ELSE IF profitMin <= 0 THEN
      BEGIN
        MoveTo(70, 215 - TRUNC(-profitMin * 135 / profitRange));
        LineTo(370, 215 - TRUNC(-profitMin * 135 / profitRange));
        {Draw 'zero' profit line if there are losses}
        MoveTo(390, 219 - TRUNC(-profitMin * 135 / profitRange));
        { Move pen (h,v) QD-42}
        TextSize(9);          {Sets grafPort type size to 9 QD-43}
        DrawString('Point-mort'); { 'Point-mort' label QD-44}
        TextSize(12);         {Sets grafPort type size to 9 QD-43}
      END; {ELSE}

      dottedLine(150);        {Draw a dotted line at 80}
      dottedLine(15);         {Draw a dotted line at 215}

      MoveTo(90, 215 - TRUNC((profit[1] - profitMin) * 135 / profitRange));
      {Move to profit estimate in period [1]}

      FOR i := 1 TO 15 DO

        BEGIN

          setActiveCircle(i, (profit[i] - profitMin), profitRange, profitMin);
          {U6: Local coordinates of active circles 'i'}

          IF i <= 14 THEN
            BEGIN
              LineTo(90 + i * 20,
                    215 - TRUNC((profit[i + 1] - profitMin) * 135 / profitRange));
              drawActiveCircle(i); {U6: Draw Cursor active circle 'i'}
            END {THEN}

          ELSE
            drawActiveCircle(15); {U6: Draw Cursor active circle '15'}

          END; {FOR}

        END; {WITH}

      HUnlock(dataHandle);      {Unlock application data record MM-43}
    
```

```
END; {plotProfit}
```

```
{-----}
```

```
{$$ Main}
```

```
{Empty PROCEDURE Unit6.Main}
```

```
{To satisfy compiler syntax}
```

```
BEGIN
```

```
END.
```

8.7. Unit7.Pas.

```

PROGRAM Unit7;                                     {FILE Unit7.Pas: External Unit7 definition}

( *****
#
#   Programme conçu et développé
#   par Jean-Marie Choffray,
#   avec l'aide de la FNEGE
#
# Copyright 1987      Tous droits réservés
#
***** )

{$i HD:Pascal2:M1:MemTypes.ipas }                 {TML interface to General Purpose TYPES}
{$i HD:Pascal2:M1:QuickDraw.ipas }                 {TML interface to Macintosh Graphics Package}
{$i HD:Pascal2:M1:OSIntf.ipas }                     {TML interface to the Operating System}
{$i HD:Pascal2:M1:ToolIntf.ipas }                   {TML interface to Macintosh User Interface}
{$i HD:Pascal2:M1:PackIntf.ipas }                   {TML interface to Macintosh Packages}
{$i HD:Pascal2:M1:MacPrint.ipas }                   {TML interface to Macintosh Printing Mgr}

{$i HD:Pascal2:Program:Globals.iPas }               {Declaration COMMON to Unit1 ... Unit7, and}
                                                    {ProduitTest}

{-----}

PROCEDURE menusFirst; EXTERNAL;
    {Unit1: DeskTop menu bar}
PROCEDURE readDeskScrap; EXTERNAL;
    {Unit1: Desk Scrap -> TextEdit Scrap, adapt PasteItem}
PROCEDURE writeDeskScrap; EXTERNAL;
    {Unit1: TextEdit Scrap to Desk Scrap}

PROCEDURE menusDesk; EXTERNAL;
    {Unit2: Activate desk accessory menu items}
PROCEDURE doMenuChoice (menuChoice : LongInt); EXTERNAL;
    {Unit2: Handle user's menu choice}

PROCEDURE doClose; EXTERNAL;
    {Unit3: Handle Close command}
PROCEDURE menusEdit; EXTERNAL;
    {Unit3: Activate only Edit menu items}

PROCEDURE windowDirty (isDirty : BOOLEAN); EXTERNAL;
    {Unit4: Mark window as dirty or clean}

PROCEDURE miniClean; EXTERNAL;
    {Unit5: Clean inside of application frame}
PROCEDURE setScreenProduct; EXTERNAL;
    {Unit5: Screen layout for product parameters}
PROCEDURE blackCircle (h, v : INTEGER); EXTERNAL;
    {Unit5: Draw a little black circle at h, v-6, h+6, v}
PROCEDURE setScreenEnvironmt; EXTERNAL;
    {Unit5: Screen layout for environmt parameters}
PROCEDURE writeEstimator (estimatorPercent : LongInt); EXTERNAL;
    {Unit5: Write percent value of top of grey box}
PROCEDURE doInnovation; EXTERNAL;
    {Unit5: Handle Innovation command}
PROCEDURE doImitation; EXTERNAL;
    {Unit5: Handle Imitation command}

```

```

PROCEDURE doInfluence; EXTERNAL;
    {Unit5: Handle Influence command}
PROCEDURE doPotential; EXTERNAL;
    {Unit5: Handle Potential command}
PROCEDURE doReplace; EXTERNAL;
    {Unit5: Handle Replace command}
PROCEDURE doExternalVC; EXTERNAL;
    {Unit5: Handle ExternalVC command}
PROCEDURE doInternalVC; EXTERNAL;
    {Unit5: Handle InternalVC command}
PROCEDURE doLearning; EXTERNAL;
    {Unit5: Handle Learning command}
PROCEDURE doEnvironmt; EXTERNAL;
    {Unit5: Handle Environmt command}

PROCEDURE displaySummary; EXTERNAL;
    {Unit6: Display input data}
PROCEDURE doInnovationCur; EXTERNAL;
    {Unit6: Handle InnovationCur command}
FUNCTION stringToReal (myString : STR255): REAL; EXTERNAL;
    {Unit6: STR255 -> REAL conversion}
PROCEDURE doImitationCur; EXTERNAL;
    {Unit6: Handle ImitationCur command}
PROCEDURE doPenetrationCur; EXTERNAL;
    {Unit6: Handle PenetrationCur command}
PROCEDURE doReplaceCur; EXTERNAL;
    {Unit6: Handle ReplaceCur command}
PROCEDURE doVolumeCur; EXTERNAL;
    {Unit6: Handle VolumeCur command}
PROCEDURE doInternalVCCur; EXTERNAL;
    {Unit6: Handle InternalVCCur command}
PROCEDURE doTotalCCur; EXTERNAL;
    {Unit6: Handle TotalCCur command}
PROCEDURE doProfitCur; EXTERNAL;
    {Unit6: Handle ProfitCur command}
FUNCTION exponential (x, y : REAL): REAL; EXTERNAL;
    {Unit6: Compute x to the power of y}

{-----}

PROCEDURE doContent (whichWindow : WindowPtr); FORWARD;
    {Handle click in content region}
PROCEDURE menusEvaluation; FORWARD;
    {Activate evaluation menus items}
PROCEDURE adjustEstimator (mouseLocation : Point); FORWARD;
    {Adjust estimator's gray rectangle}
PROCEDURE menusSimulation; FORWARD;
    {Activate simulation menus items}
PROCEDURE fixSimulationMenu; FORWARD;
    {Update simulation menus items}
PROCEDURE displayCurEstimate; FORWARD;
    {Display curve's ordinate and abscissa}
PROCEDURE drawCurLabels (x : INTEGER; y : REAL); FORWARD;
    {Draw labels of curve's estimates}
PROCEDURE placeSpace (theValue : REAL; VAR theString: STR255); FORWARD;
    {Place ' 's when needed in a string}
PROCEDURE realToScientific (theValue : REAL; VAR theString: STR255); FORWARD;
    {REAL -> Scientific conversion}

```

```

PROCEDURE checkinput (VAR continue : BOOLEAN); FORWARD;
    {Check inputs and post alerts if need be}
PROCEDURE activateProduct (mousePoint : Point); FORWARD;
    {Deactivate/activate appropriate Edit record}
PROCEDURE deactivateProduct; FORWARD;
    {Deactivate appropriate Edit record}
PROCEDURE activateEnvironmt (mousePoint : Point); FORWARD;
    {Deactivate/activate appropriate Edit record}
PROCEDURE deactivateEnvironmt; FORWARD;
    {Deactivate appropriate Edit record}
PROCEDURE doSelect (thePoint : Point; theText : THandle); FORWARD;
    {Handle mouse-down in text rectangle}
PROCEDURE fixEditMenu; FORWARD;
    {Update edit menu items}

PROCEDURE doGoAway (whichWindow : WindowPtr); FORWARD;
    {Handle click in goaway region}

PROCEDURE doKeyStroke; FORWARD;
    {Handle keystroke events}
PROCEDURE doTyping; FORWARD;
    {Handle character typed from keyboard}

PROCEDURE doUpdate; FORWARD;
    {Handle update events}

PROCEDURE doActivate; FORWARD;
    {Handle activate events}

{-----}

{$S Code}                                {PROCEDURES and FUNCTIONS in Unit7.Code}

{-----}

PROCEDURE doContent((whichWindow : WindowPtr)); {Handle click in content region}

VAR

    theData      : apIHandle;              {Handle to application data record}
    dataHandle   : Handle;                 {Untyped handle to create data record M1-9}

    thePoint     : Point;                  {Location of mouse click in local coord QD-7}
    continue     : BOOLEAN;                {Check input flag}

    cleanRect    : Rect;                   {Temporary Rect for cleaning purposes QD-9}

BEGIN {doContent}

    dataHandle := Handle (GetWRefCon (theWindow));
    HLock (dataHandle);                    {Get and lock appl data record WM-33, MM-42}
    theData := apIHandle (dataHandle);     {Convert to apIHandle}

    IF whichWindow <> FrontWindow THEN     {Is 'clicked' window the front most WM-26}
        SelectWindow (whichWindow)        {Activate the window WM-23}
    ELSE
        {Click in application window}
        BEGIN

```

```

thePoint := theEvent.Where;           {Get point in global coordinates EM-11}
GlobalToLocal (thePoint);             {Convert to local coordinates QD-66}

CASE theScreen OF

  product:
    BEGIN
      activateProduct (thePoint);      {U7: Deactivate/activate Edit records}
      IF textRecd <> none THEN
        doSelect (thePoint, theText);  {U7: Handle mouse-down in text rectangle}
      IF PtlInRect(thePoint, okRect1) AND NOT simulation THEN
        BEGIN
          checkInput (continue);        {U7: Check input and post alert if need be}
          IF continue THEN
            BEGIN
              deactivateProduct;         {U7: Deactivate appropriate edit record}
              menusEvaluation;           {U7: Adapt menus items for evaluation phase}
              drawMenuBar;               {Redraw MnBar according to menu list MN-19}
              miniClean;                 {U7: Clean inside of application frame}
              doInnovation;              {U5: Mouse hit OK, calibrate innovation}
            END; {IF}
          END; {IF}
        END; {product}

    innovation, imitation, influence, potential, replace,
    externalVC, internalVC, learning:
      BEGIN
        IF PtlInRect(thePoint, activeRect) THEN
          BEGIN
            adjustEstimator (thePoint);  {U7: Adjust estimator's grey rectangle}
            windowDirty (TRUE);          {U4: Window -> dirty, change Save item}
          END; {IF}

        IF PtlInRect(thePoint, okRect2) AND NOT simulation THEN
          {Mouse hit OK, calibrate parameters}

          CASE theScreen OF
            innovation :
              BEGIN
                checkInput (continue);    {U7: Check input and post alert if need be}
                IF continue THEN
                  doImitation;            {U5: Handle imitation command}
                END; {innovation}
            imitation :
              BEGIN
                checkInput (continue);    {U7: Check input and post alert if need be}
                IF continue THEN
                  doInfluence;           {U5: Handle influence command}
                END; {imitation}
            influence :
              BEGIN
                checkInput (continue);    {U7: Check input and post alert if need be}
                IF continue THEN
                  doPotential;           {U5: Handle potential command}
                END; {influence}
            potential :
              BEGIN
                checkInput (continue);    {U7: Check input and post alert if need be}
                IF continue THEN

```

```

        doReplace;          {U5: Handle replace command}
    END; {potential}
replace :
    BEGIN
        checkInput (continue); {U7: Check input and post alert if need be}
        IF continue THEN
            doExternalVC;      {U5: Handle externalVC command}
        END; {replace}
    externalVC :
        BEGIN
            checkInput (continue); {U7: Check input and post alert if need be}
            IF continue THEN
                doInternalVC;   {U5: Handle internalVC command}
            END; {externalVC}
    internalVC :
        BEGIN
            checkInput (continue); {U7: Check input and post alert if need be}
            IF continue THEN
                doLearning;      {U5: Handle learning command}
            END; {internalVC}
    learning :
        BEGIN
            checkInput (continue); {U7: Check input and post alert if need be}
            IF continue THEN
                BEGIN
                    menusEdit;    {U3: Adapt menus items for environmt screen}
                    IF (theData^.dirty) AND (theData^.fileNumber <> 0) THEN
                        EnableItem (fileMenu, saveItem);
                    drawMenuBar;   {Redraw MnBar according to menu list MN-19}
                    doEnvironmt;   {U5: Handle environmt command}
                END; {IF}
            END; {learning}

    END; {CASE}

END; {evaluation screens}

environmt:
    BEGIN
        activateEnvironmt (thePoint); {U7: Deactivate/activate Edit records}
        IF textRead <> none THEN
            doSelect (thePoint, theText); {U7: Handle mouse-down in text rectangle}
        IF PtInRect(thePoint, okRect1) AND NOT simulation THEN
            BEGIN
                checkInput (continue); {U7: Check input and post alert if need be}
                IF continue THEN
                    BEGIN
                        deActivateEnvironmt;
                        {U7: Deactivate appropriate Edit record}
                        simulation := TRUE; {Set simulation flag}
                        menusEdit;          {U3: Adapt Edit menus for simulation mode}
                        IF (theData^.dirty) AND (theData^.fileNumber <> 0) THEN
                            EnableItem (fileMenu, saveItem);
                        fixSimulationMenu; {U7: Update simulation menu items}
                        DrawMenuBar;      {Redraw MnBar according to menu list MN-19}

                        PenPat (Gray);     {Set pen pattern QD-42, QD-34}
                        PenMode (patBio);  {Set pattern transfer mode QD-41, QD-29}
                    END;
                END;
            END;
        END;
    END;

```

```

        SetRect (okRect1, 375, 243, 445, 257);
        PaintRect (okRect1); {Set and paint OK rect in Gray QD-47, QD-49}

        PenMode (patCopy); {Set pattern transfer mode QD-41, QD-29}
        PenPat (Black);     {Set pen pattern QD-42, QD-34}
    END; {IF}
END; {IF}
END; {environmt}

innovationPlot, imitationPlot, penetrationPlot, replacePlot,
volumePlot, internalVCPlot, totalCPlot, profitPlot:
    displayCurEstimate; {U7: Display curve's ordinate and abscissa};

END; {CASE}

END; {ELSE}

HUnlock (dataHandle); {Unlock application data record MM-43}

END; {doContent}

{-----}

PROCEDURE menusEvaluation; {Activate evaluation menus items}

VAR

    theData : apiHandle; {Handle to application data record}
    dataHandle : Handle; {Untyped handle to create data record MM-9}

BEGIN {menusEvaluation}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle); {Get and lock appl data record WM-33, MM-42}
    theData := apiHandle (dataHandle); {Convert to apiHandle}

    EnableItem (appleMenu, 0); {Enable Apple menu MM-24}

    EnableItem (fileMenu, 0); {Enable File menu MM-24}
    disableItem (fileMenu, newItem);
    disableItem (fileMenu, openItem);
    EnableItem (fileMenu, closeItem);

    DisableItem (fileMenu, saveItem);
    IF (theData^.dirty) AND (theData^.fileNumber <> 0) THEN
        EnableItem (fileMenu, saveItem); {Enable Save command MM-24}

    EnableItem (fileMenu, saveAsItem);
    EnableItem (fileMenu, printItem);
    EnableItem (fileMenu, quitItem);

    DisableItem (editMenu, 0); {Disable Edit menu MM-23}
    DisableItem (simulationMenu, 0); {Disable simulation menu MM-23}
    DisableItem (analyzeMenu, 0); {Disable Analyze menu MM-23}

    HUnlock (dataHandle); {Unlock application data record MM-43}

```

END; {menusEvaluation}

{-----}

PROCEDURE adjustEstimator ({mouseLocation : Point});

{Adjust estimator's gray rectangle}

VAR

grayRect : Rect; {Estimator's gray rectangle QD-9}
whiteRect : Rect; {Complement to estimator's gray rect QD-9}
subjectiveRect : Rect; {Subjective estimation rectangle QD-9}

loc1, loc2 : INTEGER; {Test change in mouse location}
done : BOOLEAN; {End of adjustment, mouse up}

theData : apiHandle; {Handle to application data record}
dataHandle : Handle; {Handle for locking application data MI-9}

{-----}

PROCEDURE drawGrayRect; {Draw gray rectangle}

BEGIN

CASE mouseLocation.v OF

31, 32: {Just below top line of subjective rectangle}

BEGIN

SetRect (grayRect, 412, 34, 448, 248);
{Local coordinates for grey Rect LTRB QD-46}
FillRect (grayRect, Ltgray); {Make it light grey QD-49}
SetRect (whiteRect, 412, 33, 448, 33);
{Local coordinates for white Rect LTRB QD-46}
FillRect (whiteRect, White); {Make it white QD-49}
writeEstimator (ROUND ((250 - mouseLocation.v) / 2.2));
{U5: Write percent value of top of grey box}
SetRect (activeRect, 410, mouseLocation.v, 450, 250);
END; {Local coordinates for active Rect LTRB QD-46}

248, 249 : {Just above bottom line of subjective rectangle}

BEGIN

SetRect (grayRect, 412, 247, 448, 247);
{Local coordinates for grey Rect LTRB QD-46}
FillRect (grayRect, Ltgray); {Make it light grey QD-49}
SetRect (whiteRect, 412, 32, 448, 246);
{Local coordinates for white Rect LTRB QD-46}
FillRect (whiteRect, White); {Make it white QD-49}
writeEstimator (ROUND ((250 - mouseLocation.v) / 2.2));
{U5: Write percent value of top of grey box}
SetRect (activeRect, 410, 248, 450, 250)
END; {Min coordinates for active Rect LTRB QD-46}

OTHERWISE {General case}

BEGIN

IF (mouseLocation.v <= 30) THEN {Top line of subjective rectangle}
BEGIN
SetRect (grayRect, 412, 32, 448, 248);
{Local coordinates for grey Rect LTRB QD-46}

```

        FillRect (grayRect, Ltgray);    {Make it light grey QD-49}
        writeEstimator (100);           {U5: Write percent value of top of grey box}
        SetRect (activeRect, 410, 30, 450, 250);
    END;                                {Local coordinates for active Rect LTRB QD-46}

    IF (mouseLocation.v >= 33) AND (mouseLocation.v <= 247) THEN
    BEGIN
        SetRect (grayRect, 412, mouseLocation.v, 448, 248);
                                     {Local coordinates for grey Rect LTRB QD-46}
        FillRect (grayRect, Ltgray);    {Make it light grey QD-49}
        SetRect (whiteRect, 412, 32, 448, mouseLocation.v - 1);
                                     {Local coordinates for white Rect LTRB QD-46}
        FillRect (whiteRect, White);    {Make it white QD-49}
        writeEstimator ( ROUND ( (250 - mouseLocation.v) / 2.2 ));
                                     {U5: Write percent value of top of grey box}
        SetRect (activeRect, 410, mouseLocation.v, 450, 250);
    END;                                {Local coordinates for active Rect LTRB QD-46}

    IF (mouseLocation.v >= 250) THEN    {Bottom line of subjective rectangle}
    BEGIN
        SetRect (whiteRect, 412, 32, 448, 248);
                                     {Local coordinates for white Rect LTRB QD-46}
        FillRect (whiteRect, white);    {Make it white QD-49}
        writeEstimator (0);             {U5: Write percent value of top of grey box}
        SetRect ( activeRect, 410, 248, 450, 250);
    END;                                {Local coordinates for active Rect LTRB QD-46}

    END;    {OTHERWISE}

    END; {CASE}

    END; {drawGrayRect}

    {-----}

    BEGIN {adjustEstimator}

        drawgrayRect;                {U7: Initial drawing of gray rectangle}
        SetRect (subjectiveRect, 410, 30, 470, 270); {Local copy of subjective rect LTRB QD-46}

        loc1 := mouseLocation.v;      {Initial location of mouse}

        REPEAT

            done := NOT StillDown;     {Test whether button still down EM-23}
            GetMouse (mouseLocation);  {Location in... local coordinates EM-23}
            loc2 := mouseLocation.v;   {New location of mouse}

            IF (loc1 <> loc2) THEN      {Change in mouse location?}
            BEGIN
                IF (PtInRect(mouseLocation, subjectiveRect)) THEN
                    drawgrayRect;      {Is it still in subjective rectangle QD-47}
                END;                   {U7: Draw corresponding grey rectangle}
                loc1 := loc2;           {Adapt initial location of mouse}
            END;

        UNTIL done;

        dataHandle := Handle (GetWRefCon (theWindow));

```

```

Hlock (dataHandle);                                {Get and lock a handle to data record WM-33}
theData := apiHandle (dataHandle);                  {Convert to apiHandle}

IF mouseLocation.v >= 250 THEN                       {Limit to bottom line of subjective rect}
  mouseLocation.v := 250;
IF mouseLocation.v <= 30 THEN                       {Limit to top line of subjective rect}
  mouseLocation.v := 30;

CASE theScreen OF                                   {Calibrated parameters}

  innovation   : theData^.innovationPar := ROUND ( (250 - mouseLocation.v) / 2.2);
  imitation    : theData^.imitationPar  := ROUND ( (250 - mouseLocation.v) / 2.2);
  influence    : theData^.influencePar   := ROUND ( (250 - mouseLocation.v) / 2.2);
  potential    : theData^.potentialPar   := ROUND ( (250 - mouseLocation.v) / 2.2);
  replace      : theData^.replacePar     := ROUND ( (250 - mouseLocation.v) / 2.2);
  externalVC   : theData^.externalVCPar  := ROUND ( (250 - mouseLocation.v) / 2.2);
  internalVC   : theData^.internalVCPar   := ROUND ( (250 - mouseLocation.v) / 2.2);
  learning     : theData^.learningPar    := ROUND ( (250 - mouseLocation.v) / 2.2);

END; {CASE}

IF simulation THEN
  fixSimulationMenu;                                {U7: Update simulation menus items}

HUnlock (dataHandle);                              {Unlock application data record}

END; {adjustEstimator}

-----}

PROCEDURE menusSimulation;                          {Activate simulation menus items}

VAR

  theData      : apiHandle;                         {Handle to application data record}
  dataHandle    : Handle;                           {Untyped handle to lock data record MI-9}

BEGIN {menusSimulation}

  dataHandle := Handle (GetWRefCon (theWindow));
  Hlock (dataHandle);                                {Get and lock appl data record WM-33, MM-42}
  theData := apiHandle (dataHandle);                  {Convert to apiHandle}

  EnableItem  (appleMenu, 0);                        {Enable Apple menu MI-24}

  EnableItem  (fileMenu, 0);                         {Enable File menu MN-24}
  disableItem (fileMenu, newItem);
  disableItem (fileMenu, openItem);
  EnableItem  (fileMenu, closeItem);

  DisableItem (fileMenu, saveItem);
  IF (theData^.dirty) AND (theData^.fileNumber <> 0) THEN
    EnableItem (fileMenu, saveItem);                 {Enable Save command MN-24}

  EnableItem  (fileMenu, saveAsItem);
  EnableItem  (fileMenu, printItem);
  EnableItem  (fileMenu, quitItem);

```

```

DisableItem (editMenu, 0);           {Disable Edit menu MN-23}
EnableItem (simulationMenu, 0);      {Enable simulation menu MN-24}
EnableItem (analyzeMenu, 0);         {Enable Analyze menu MN-24}

HUnlock (dataHandle);                {Unlock application data record MM-43}

```

```
END; {menusSimulation}
```

```
{-----}
```

```
PROCEDURE fixSimulationMenu;          {Update simulation menus items}
```

```
VAR
```

```

theData      : apIHandle;             {Handle to application data record}
dataHandle   : Handle;               {Untyped handle to lock data record MI-9}

```

```
BEGIN {fixSimulationMenu}
```

```

dataHandle := Handle (GetWRefCon (theWindow));
HLock (dataHandle);                  {Get and lock appl data record WM-33, MM-42}
theData := apIHandle (dataHandle);   {Convert to an apIHandle}

```

```
WITH theData^^ DO
```

```
BEGIN
```

```

IF innovationPar <> 0 THEN            {Enable/disable innovationCurItem cmd}
  EnableItem (analyzeMenu, innovationCurItem)
ELSE
  {Enable menu MN-24}
  DisableItem (analyzeMenu, innovationCurItem);

```

```

IF imitationPar <> 0 THEN              {Enable/disable imitationCurItem cmd}
  EnableItem (analyzeMenu, imitationCurItem)
ELSE
  {Enable menu MN-24}
  DisableItem (analyzeMenu, imitationCurItem);

```

```

IF replacePar <> 0 THEN                {Enable/disable replaceCurItem cmd}
  EnableItem (analyzeMenu, replaceCurItem)
ELSE
  {Enable menu MN-24}
  DisableItem (analyzeMenu, replaceCurItem);

```

```

IF internalVCPAr <> 0 THEN             {Enable/disable internalVCCurItem cmd}
  EnableItem (analyzeMenu, internalVCCurItem)
ELSE
  {Enable menu MN-24}
  DisableItem (analyzeMenu, internalVCCurItem);

```

```
END; {WITH}
```

```
HUnlock (dataHandle);                {Unlock application data record MM-43}
```

```
END; {fixSimulationMenu}
```

```
{-----}
```

```
PROCEDURE displayCurEstimate;        {Display curve's ordinate and abscissa}
```

```
VAR
```

```

displayRect : Rect;           {Rectangle for curve parameters QD-9}
thePoint : Point;             {Mouse location in local coordinates QD-7}

done : BOOLEAN;               {End of 'show', mouse up}
i, j : INTEGER;               {Index variables for loop control}

BEGIN {displayCurEstimate}

  thePoint := theEvent.Where;   {Get point in global coordinates EM-11}
  GlobalToLocal (thePoint);     {Convert to local coordinates QD-66}

  IF (theScreen = imitationPlot) OR (theScreen = replacePlot) THEN
    i := 2;                     {Curve starts at period two}
  ELSE
    i := 1;                     {Curve starts at period one}

  FOR j := i TO 15 DO
    IF PtInRect (thePoint, cursCircles[j].circle) THEN
      {Is mouse in active circle's rectangle QD-47}
      BEGIN
        REPEAT
          done := NOT StillDown; {Test whether button still down EM-23}
          drawCurLabels (cursCircles[j].predictor, cursCircles[j].percentResponse)
          UNTIL done;            {U7: Draw labels of curve's estimates}

          SetRect (displayRect, 236, 18, 460, 73);
          {Coord for enlarged display Rect LTRB QD-46}
          FillRect (displayRect, White); {Clean-up display rectangle QD-49}
        END; {IF}

      END; {displayCurEstimate}

    }-----}

PROCEDURE drawCurLabels (( x: INTEGER; y : REAL));
{Draw labels of curve's estimates}
{Cycle = x and penetration = y estimates}

VAR

  wkString1 : STR255; {Temporary working space of type STRING}
  wkString2 : STR255; {Temporary working space of type STRING}
  wkString3 : STR255; {Temporary working space of type STRING}
  wkString4 : STR255; {Temporary working space of type STRING}

  displayRect : Rect; {Rectangle for curve parameters QD-9}

  theData : apIHandle; {Handle to application data record}
  dataHandle : Handle; {Untyped handle to create data record MI-9}

  adopterReal : REAL; {Temp REAL working space}
  fCostReal : REAL; {Temp REAL working space}
  priceReal : REAL; {Temp REAL working space}

  number : REAL; {Nber of innovators, imitators, adopters}
  turnover : REAL; {Corresponding turnover}

  nominalCost : REAL; {Nominal unit cost}

```

```

costReduction      : REAL;                {Cost reduction factor}

profit             : REAL;                {Nominal profit level [per period]}
twoBillions        : REAL;                {Constant equal to two billions}

{-----}

PROCEDURE adaptString2 (VAR theString : STR255);
                                {Adapt length of string if <= 13 char}
VAR
    nullString : STRING[40];              {Null string, for computation purpose}
    padString  : STRING[40];              {Null string, used for padding}

BEGIN {adaptString2}

    nullString := ' '; {Initialize nullString to 40 'blanks'}
    padString := COPY (nullString, 1, (13 - LENGTH (theString)));

    theString := CONCAT (padString, theString); {Add spaces when needed for alignment}

END; {adaptString2}

{-----}

BEGIN {drawCurLabels}

    twoBillions := 2 * exponential (10, 9); {Compute constant}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle); {Get and lock appl data record WM-33, MM-42}
    theData := apiHandle (dataHandle); {Convert to apiHandle}

    WITH theData^^ DO
        BEGIN
            GetText(adopaterEditRecd^.hText,wkString1);
                                {Return text as a string DL-31}
            adopaterReal := stringToReal (wkString1); {U6: STR255 -> REAL conversion}

            GetText(fCostEditRecd^.hText,wkString1);
            fCostReal := stringToReal (wkString1);

            GetText(priceEditRecd^.hText,wkString1);
            priceReal := stringToReal (wkString1);
        END; {WITH}

    PenSize (2, 2); {Set dimensions of graphic pen QD-41}
    SetRect (displayRect, 240, 20, 456, 71); {Set display rectangle LTRB QD-46}
    FrameRoundRect (displayRect, 10,10); {Draw display rectangle QD-51}
    PenSize (1, 1); {Set dimensions of graphic pen QD-41}

    CASE theScreen OF

        innovationPlot, imitationPlot, penetrationPlot, replacePlot:
            BEGIN
                TextSize (9); {Sets grafPort type size to 9 QD-43}

                blackCircle (250, 33); {U5: Draw a little black circle}
            END
    END

```

```

MoveTo (265, 33);           {Move pen (h,v) QD-42}
DrawString ('Période...');  {'Période' label QD-44}

blackCircle (250, 48);      {U5: Draw a little black circle}
MoveTo (265, 48);           {Move pen (h,v) QD-42}
CASE theScreen OF
  innovationPlot:
    DrawString ('Innovateurs...'); {'Innovateurs' label QD-44}
  imitationPlot:
    DrawString ('Imitateurs...'); {'Imitateurs' label QD-44}
  penetrationPlot:
    DrawString ('Adopteurs...'); {'Adopteurs' label QD-44}
  replacePlot:
    DrawString ('Réacheteurs...'); {'Réacheteurs' label QD-44}
END; {CASE}

blackCircle (250, 63);      {U5: Draw a little black circle}
MoveTo (265, 63);           {Move pen (h,v) QD-42}
DrawString ('Chiffre d'affaires...'); {'Chiffre d'affaires' label QD-44}

number := y * adopterReal / 100; {Computes number of innovators, imitators,...}
turnover := number * priceReal; {Computes corresponding turnover}

NumToString (x, wkString1); {'Période' integer conversion PK-20}
adaptString2 (wkString1); {U7: Adapt length of string if <= 13 char}

NumToString (TRUNC(y), wkString2); {'Innovation,...' integer conversion PK-20}
wkString2 := CONCAT ('{',wkString2, '§');
adaptString2 (wkString2); {U7: Adapt length of string if <= 13 char}

IF number <= twoBillions THEN
  BEGIN
    placeSpace (number, wkString3); {U7: Place ' 's when needed in a string}
    adaptString2 (wkString3); {U7: Adapt length of string if <= 13 char}
  END {THEN}
ELSE
  BEGIN
    realToScientific(number, wkString3);
    adaptString2 (wkString3); {U7: REAL -> Scientific conversion}
    adaptString2 (wkString3); {U7: Adapt length of string if <= 13 char}
  END; {ELSE}

IF turnover <= twoBillions THEN
  BEGIN
    placeSpace (turnover, wkString4);
    adaptString2 (wkString4); {U7: Place ' 's when needed in a string}
    adaptString2 (wkString4); {U7: Adapt length of string if <= 13 char}
  END {THEN}
ELSE
  BEGIN
    realToScientific(turnover, wkString4);
    adaptString2 (wkString4); {U7: REAL -> Scientific conversion}
    adaptString2 (wkString4); {U7: Adapt length of string if <= 13 char}
  END; {ELSE}

TextFont (Monaco); {Set current GrafPort's Font QD-43, FM-26}

```

```

MoveTo (360, 33);           {Move pen (h,v) QD-42}
DrawString (wkString1);     {'Période' estimate label QD-44}
MoveTo (360, 48);           {Move pen (h,v) QD-42}
DrawString (wkString3);     {'Innovateurs,...' estimate label QD-44}
MoveTo (360, 63);           {Move pen (h,v) QD-42}
DrawString (wkString4);     {'Chiffre d'affaires' estimate label QD-44}

TextFont (Geneva);          {Set current GrafPort's Font QD-43, FM-26}
TextSize (12);              {Sets grafPort type size to 12 QD-43}
END;  {innovationPlot}

```

volumePlot:

```

BEGIN
  TextSize (9);              {Sets grafPort type size to 9 QD-43}

  blackCircle (250, 33);     {U5: Draw a little black circle}
  MoveTo (265, 33);          {Move pen (h,v) QD-42}
  DrawString ('Période...');  {'Période' label QD-44}

  blackCircle (250, 48);     {U5: Draw a little black circle}
  MoveTo (265, 48);          {Move pen (h,v) QD-42}
  DrawString ('Acheteurs...'); {'Acheteurs' label QD-44}

  blackCircle (250, 63);     {U5: Draw a little black circle}
  MoveTo (265, 63);          {Move pen (h,v) QD-42}
  DrawString ('Chiffre d'affaires...'); {'Chiffre d'affaires' label QD-44}

  number := (y + cursCircles[1].nominalResponse) * adopterReal / 100;
                                     {Computes number of buyers}
  turnover := number * priceReal;    {Computes corresponding turnover}

  NumToString (x, wkString1);        {'Période' integer conversion PK-20}
  adaptString2 (wkString1);          {U7: Adapt length of string if <= 13 char}

  IF number <= twoBillions THEN
    BEGIN
      placeSpace (number, wkString2); {U7: Place ' 's when needed in a string}
      adaptString2 (wkString2);        {U7: Adapt length of string if <= 13 char}
    END {THEN}
  ELSE
    BEGIN
      realToScientific(number, wkString2);
                                     {U7: REAL -> Scientific conversion}
      adaptString2 (wkString2);        {U7: Adapt length of string if <= 13 char}
    END; {ELSE}

  IF turnover <= twoBillions THEN
    BEGIN
      placeSpace (turnover, wkString3);
                                     {U7: Place ' 's when needed in a string}
      adaptString2 (wkString3);        {U7: Adapt length of string if <= 13 char}
    END {THEN}
  ELSE
    BEGIN
      realToScientific(turnover, wkString3);
                                     {U7: REAL -> Scientific conversion}
      adaptString2 (wkString3);        {U7: Adapt length of string if <= 13 char}
    END; {ELSE}

```

```

    TextFont (Monaco);                {Set current GrafPort's Font QD-43, FM-26}

    MoveTo (360, 33);                 {Move pen (h,v) QD-42}
    DrawString (wkString1);           {'Période' estimate label QD-44}
    MoveTo (360, 48);                 {Move pen (h,v) QD-42}
    DrawString (wkString2);           {'Acheteurs,...' estimate label QD-44}
    MoveTo (360, 63);                 {Move pen (h,v) QD-42}
    DrawString (wkString3);           {'Chiffre d'affaires' estimate label QD-44}

    TextFont (Geneva);                {Set current GrafPort's Font QD-43, FM-26}
    TextSize (12);                    {Sets grafPort type size to 12 QD-43}
END;  {volumePlot}

```

internalVCPlot:

```

BEGIN
    TextSize (9);                     {Sets grafPort type size to 9 QD-43}

    blackCircle (250, 33);            {U5: Draw a little black circle}
    MoveTo (265, 33);                 {Move pen (h,v) QD-42}
    DrawString ('Période...');        {'Période' label QD-44}

    blackCircle (250, 48);            {U5: Draw a little black circle}
    MoveTo (265, 48);                 {Move pen (h,v) QD-42}
    DrawString ('Coût Var Interne...'); {'Coût Var interne' label QD-44}

    blackCircle (250, 63);            {U5: Draw a little black circle}
    MoveTo (265, 63);                 {Move pen (h,v) QD-42}
    DrawString ('Réduction [%]...');  {'Réduction [%]' label QD-44}

    nominalCost := (y + cursCircles[1].nominalResponse) * priceReal / 100;
                                           {Computes nominal unit cost}
    costReduction := (y + cursCircles[1].nominalResponse) * 100
                    / theData^.internalVCPAr;
                                           {Computes costReduction factor}

    NumToString (x, wkString1);        {'Période' integer conversion PK-20}
    adaptString2 (wkString1);          {U7: Adapt length of string if <= 13 char}

    IF nominalCost <= twoBillions THEN
        BEGIN
            placeSpace (nominalCost, wkString2);
                                           {U7: Place ' 's when needed in a string}
            adaptString2 (wkString2);    {U7: Adapt length of string if <= 13 char}
        END {THEN}
    ELSE
        BEGIN
            realToScientific (nominalCost, wkString2);
                                           {U7: REAL -> Scientific conversion}
            adaptString2 (wkString2);    {U7: Adapt length of string if <= 13 char}
        END; {ELSE}

    IF costReduction <= twoBillions THEN
        BEGIN
            placeSpace (costReduction, wkString3);
                                           {U7: Place ' 's when needed in a string}
            adaptString2 (wkString3);    {U7: Adapt length of string if <= 13 char}
        END {THEN}
    END

```

```

ELSE
  BEGIN
    realToScientific(costReduction, wkString3);
    {U7: REAL -> Scientific conversion}
    adaptString2 (wkString3);
    {U7: Adapt length of string if <= 13 char}
  END; {ELSE}

  TextFont (Monaco);
  {Set current GrafPort's Font QD-43, FM-26}

  MoveTo (360, 33);
  {Move pen (h,v) QD-42}
  DrawString (wkString1);
  {'Période' estimate label QD-44}
  MoveTo (360, 48);
  {Move pen (h,v) QD-42}
  DrawString (wkString2);
  {'Coût Var interne' estimate label QD-44}
  MoveTo (360, 63);
  {Move pen (h,v) QD-42}
  DrawString (wkString3);
  {'Réduction [%]' estimate label QD-44}

  TextFont (Geneva);
  {Set current GrafPort's Font QD-43, FM-26}
  TextSize (12);
  {Sets grafPort type size to 12 QD-43}
END; {internalVCPlot}

totalCPlot:
  BEGIN
    TextSize (9);
    {Sets grafPort type size to 9 QD-43}

    blackCircle (250, 33);
    {U5: Draw a little black circle}
    MoveTo (265, 33);
    {Move pen (h,v) QD-42}
    DrawString ('Période...');
    {'Période' label QD-44}

    blackCircle (250, 48);
    {U5: Draw a little black circle}
    MoveTo (265, 48);
    {Move pen (h,v) QD-42}
    DrawString ('Coût total...');
    {'Coût total' label QD-44}

    blackCircle (250, 63);
    {U5: Draw a little black circle}
    MoveTo (265, 63);
    {Move pen (h,v) QD-42}
    DrawString ('Réduction [%]...');
    {'Réduction [%]' label QD-44}

    nominalCost := (y + cursCircles[1].nominalResponse) * priceReal / 100;
    {Computes nominal unit total cost}
    costReduction := (y + cursCircles[1].nominalResponse) * 100
      / (cursCircles[1].percentResponse
        + cursCircles[1].nominalResponse);
    {Computes total cost reduction factor}

    NumToString (x, wkString1);
    {'Période' integer conversion PK-20}
    adaptString2 (wkString1);
    {U7: Adapt length of string if <= 13 char}

    IF nominalCost <= twoBillions THEN
      BEGIN
        placeSpace (nominalCost, wkString2);
        {U7: Place ' 's when needed in a string}
        adaptString2 (wkString2);
        {U7: Adapt length of string if <= 13 char}
      END {THEN}
    ELSE
      BEGIN
        realToScientific (nominalCost, wkString2);
        {U7: REAL -> Scientific conversion}
        adaptString2 (wkString2);
        {U7: Adapt length of string if <= 13 char}
      END; {ELSE}

```

```

IF costReduction <= twoBillions THEN
  BEGIN
    placeSpace (costReduction, wkString3);
    adaptString2 (wkString3);
  END {THEN}
ELSE
  BEGIN
    realToScientific (costReduction, wkString3);
    adaptString2 (wkString3);
  END {ELSE}

  TextFont (Monaco);
  MoveTo (360, 33);
  DrawString (wkString1);
  MoveTo (360, 48);
  DrawString (wkString2);
  MoveTo (360, 63);
  DrawString (wkString3);

  TextFont (Geneva);
  TextSize (12);
END; {totalCPlot}

profitPlot:
  BEGIN
    TextSize (9);

    blackCircle (250, 38);
    MoveTo (280, 38);
    DrawString ('Période...');

    IF (y + cursCircles[1].nominalResponse) >= 0 THEN
      BEGIN
        blackCircle (250, 58);
        MoveTo (280, 58);
        DrawString ('Profit...');
      END {THEN}
    ELSE
      BEGIN
        blackCircle (250, 58);
        MoveTo (280, 58);
        DrawString ('Perte...');
      END {THEN}

      profit := y + cursCircles[1].nominalResponse;

      NumToString (x, wkString1);
      adaptString2 (wkString1);

      IF (profit >= - twoBillions) AND (profit <= twoBillions) THEN
        BEGIN
          placeSpace (profit, wkString2);
          adaptString2 (wkString2);
        END
      END IF
    END IF
  END

```

```

        END      {THEN}
    ELSE
        BEGIN
            realToScientific (profit, wkString2);
                                {U7: REAL -> Scientific conversion}
            adaptString2 (wkString2);
                                {U7: Adapt length of string if <= 13 char}
            END;      {ELSE}

            TextFont (Monaco);
                                {Set current GrafPort's Font QD-43, FM-26}

            MoveTo (360, 38);
                                {Move pen (h,v) QD-42}
            DrawString (wkString1);
                                {'Période' estimate label QD-44}
            MoveTo (360, 58);
                                {Move pen (h,v) QD-42}
            DrawString (wkString2);
                                {'Profit/Perte' estimate label QD-44}

            TextFont (Geneva);
                                {Set current GrafPort's Font QD-43, FM-26}
            TextSize (12);
                                {Sets grafPort type size to 12 QD-43}
            END;      {profitPlot}

        END;      {CASE}

        HUnlock (dataHandle);
                                {Unlock application data record MM-43}

    END;      {drawCurLabels}

}-----}

PROCEDURE placeSpace ((theValue : REAL; VAR theString : STR255));
                                {Place ' 's when needed in a string}
    LABEL
                                {Min = -2.0 E+9, max = 2.0 E+9}

        1001;
                                {Exit placeSpace Label}

    VAR

        myValue   : LongInt;
                                {longInt version of 'theValue'}
        myLength   : INTEGER;
                                {Length of string}

    BEGIN      {placeSpace}

        myValue   := TRUNC (theValue);
                                {REAL -> LongInt conversion}
        NumToString (myValue, theString);
                                {LongInt -> STR255 conversion PK-20}

        myLength := LENGTH (theString);
                                {Get length of current string}

        IF myValue >= 0 THEN
                                {Value is POSITIVE}
            BEGIN

                IF (myLength > 3) AND (myLength < 7) THEN
                    BEGIN
                                {Value > 999 and < 999 999}
                        INSERT (' ', theString, LENGTH (theString) - 2);
                        GOTO 1001;
                                {Enter 'thousands' space, exit placeSpace}
                    END;

                IF (myLength > 6) AND (myLength < 10) THEN
                    BEGIN
                                {Value > 999 999 and < 999 999 999}
                        INSERT (' ', theString, LENGTH (theString) - 2);
                                {Enter 'thousands' space, and exit placeSpace}
                    END;
                END;
            END;
        END;
    END;

```

```

        INSERT (' ', theString, LENGTH (theString) - 6);
        GOTO 1001;                                {Enter 'millions' space}
    END; {THEN}

    IF (myLength = 10) THEN                        {Value > 999 999 999 and <= 2 000 000 000}
    BEGIN
        INSERT (' ', theString, 2);              {Enter 'billions' space}
        INSERT (' ', theString, 6);              {Enter 'millions' space}
        INSERT (' ', theString, 10);             {Enter 'thousands' space}
    END; {THEN}

END {THEN}

ELSE                                              {Value is NEGATIVE}
BEGIN

    IF (myLength > 4) AND (myLength < 8) THEN
    BEGIN                                         {Value > -999 999 and < -999}
        INSERT (' ', theString, LENGTH (theString) - 2);
        GOTO 1001;                                {Enter 'thousands' space, exit placeSpace}
    END;

    IF (myLength > 7) AND (myLength < 11) THEN
    BEGIN                                         {Value > -999 999 999 and < -999 999}
        INSERT (' ', theString, LENGTH (theString) - 2);
                                                {Enter 'thousands' space, and exit placeSpace}
        INSERT (' ', theString, LENGTH (theString) - 6);
        GOTO 1001;                                {Enter 'millions' space}
    END; {THEN}

    IF (myLength = 11) THEN                      {Value > -2.0 E+9 and <= -999 999 999}
    BEGIN
        INSERT (' ', theString, 3);              {Enter 'billions' space}
        INSERT (' ', theString, 7);              {Enter 'millions' space}
        INSERT (' ', theString, 11);             {Enter 'thousands' space}
    END; {THEN}

END; {ELSE}

1001: END; {placeSpace}

{-----}

PROCEDURE realToScientific ((theValue : REAL; VAR theString: STR255));
                                {REAL -> Scientific conversion}
    LABEL                        {2 000 000 000 < val < 1 000 000 000 000}

    1002;                        {Exit realToScientific Label}

VAR

    myValue          : LongInt;          {LongInt version of fractionnal portion}

    oneThousandBillions : REAL;          {Constant equal to one thousand billions}
    oneHundredBillions  : REAL;          {Constant equal to one hundred billions}
    tenBillions          : REAL;          {Constant equal to ten billions}

    oneHundredMillions  : REAL;          {Constant equal to one hundred millions}

```

```

tenMillions      : REAL;           {Constant equal to ten millions}
oneMillion       : REAL;           {Constant equal to one million}

BEGIN    {realToScientific}

    oneThousandBillions := exponential (10, 12); {Compute constant}
    oneHundredBillions  := exponential (10, 11); {Compute constant}
    tenBillions         := exponential (10, 10); {Compute constant}

    oneHundredMillions  := exponential (10, 8);  {Compute constant}
    tenMillions         := exponential (10, 7);  {Compute constant}
    oneMillion          := exponential (10, 6);  {Compute constant}

    IF theValue >= 0 THEN                {Value is POSITIVE}

        BEGIN

            IF (theValue < oneThousandBillions) THEN {Val < 1 000 000 000 000}

                BEGIN

                    IF (theValue < tenBillions) THEN {Val < 10 000 000 000}

                        BEGIN

                            myValue := TRUNC (theValue / oneMillion);
                            NumToString (myValue, theString);
                                {Type conversion PK-20}
                            INSERT ('.', theString, 2); {Enter comma in fractionnal portion}
                            theString := CONCAT (theString, ' E+09');
                            GOTO 1002; {Skip rest of operation}
                        END; {IF}

                    IF (theValue < oneHundredBillions) THEN

                        BEGIN {Val < 100 000 000 000}

                            myValue := TRUNC (theValue / tenMillions);
                            NumToString (myValue, theString);
                                {Type conversion PK-20}
                            INSERT ('.', theString, 2); {Enter comma in fractionnal portion}
                            theString := CONCAT (theString, ' E+10');
                            GOTO 1002; {Skip rest of operation}
                        END; {IF}

                        {Val < 1 000 000 000 000}
                        myValue := TRUNC (theValue / oneHundredMillions);
                        NumToString (myValue, theString); {Type conversion PK-20}
                        INSERT ('.', theString, 2); {Enter comma in fractionnal portion}
                        theString := CONCAT (theString, ' E+11');

                    END {THEN}

                ELSE

                    theString := 'Très élevé !!!';

                END {THEN}

            ELSE

                {Value is NEGATIVE}

                BEGIN

```

```

wkNum      : REAL;                                {Temporary REAL space}

{-----}

FUNCTION testLoss: BOOLEAN;                        {Test if externalVCPAr + internalVCPAr > 100}

BEGIN {testLoss}

    testLoss := TRUE;                              {Set checkInput flag}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                             {Get and lock appl data record WM-33, MM-42}
    theData := apiHandle (dataHandle);              {Convert to apiHandle}

    WITH theData^^ DO

        IF (externalVCPAr + internalVCPAr > 100) THEN
            BEGIN                                  {Penetration pricing strategy!!!}
                theItem := CautionAlert (lossID, NIL);
                CASE theItem OF                    {Post 'lossID' caution alert DM-28}
                    no  : testLoss := TRUE;        {Set testLoss value}
                    yes : testLoss := FALSE;       {Set testLoss value}
                END; {CASE}
            END; {WITH}

            HUnlock (dataHandle);                  {Unlock application data record MM-43}

        END; {testLoss}

{-----}

BEGIN {checkInput}

    continue := FALSE;                             {Set checkInput flag}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                             {Get and lock appl data record WM-33, MM-42}
    theData := apiHandle (dataHandle);              {Convert to apiHandle}

    WITH theData^^ DO

        CASE theScreen OF

            product:
                IF (divisionEditRecd^^.TELength <= 1) THEN
                    {Division text length <= 1 character}
                    theItem := StopAlert (emptyTextID, NIL)
                    {Post 'emptyTextID' stop alert DM-28}

                ELSE IF (brandEditRecd^^.TELength <= 1) THEN
                    {Brand text length <= 1 character}
                    theItem := StopAlert (emptyTextID, NIL)
                    {Post 'emptyTextID' stop alert DM-28}

                ELSE IF (marketEditRecd^^.TELength <= 1) THEN
                    {Market text length <= 1 character}
                    theItem := StopAlert (emptyTextID, NIL)
                    {Post 'emptyTextID' stop alert DM-28}

```

```

IF (theValue > -oneThousandBillions) THEN {Val > (-) 1 000 000 000 000}

BEGIN

    IF (theValue > - tenBillions) THEN    {Val > (-) 10 000 000 000}
    BEGIN
        myValue := TRUNC (theValue / oneMillion);
        NumToString (myValue, theString);
                                                {Type conversion PK-20}
        INSERT ('.', theString, 3);    {Enter comma in fractionnal portion}
        theString := CONCAT (theString, ' E+09');
        GOTO 1002;                      {Skip rest of operation}
    END; {IF}

    IF (theValue > - oneHundredBillions) THEN
    BEGIN                                {Val > (-) 100 000 000 000}
        myValue := TRUNC (theValue / tenMillions);
        NumToString (myValue, theString);
                                                {Type conversion PK-20}
        INSERT ('.', theString, 3);    {Enter comma in fractionnal portion}
        theString := CONCAT (theString, ' E+10');
        GOTO 1002;                      {Skip rest of operation}
    END; {IF}

    {Val > (-) 1 000 000 000 000}
    myValue := TRUNC (theValue / oneHundredMillions);
    NumToString (myValue, theString); {Type conversion PK-20}
    INSERT ('.', theString, 3);    {Enter comma in fractionnal portion}
    theString := CONCAT (theString, ' E+11');

    END    {THEN}

ELSE
    theString := 'Très faible !!!';

END; {ELSE}

1002: END;    {realToScientific}

```

----->

PROCEDURE checkInput ((VAR continue : BOOLEAN)); {Check input and post alert if need be}

LABEL

1011; {Exit checkInput label}

CONST

no = 1; {First item in alert box}
yes = 2; {Second item in alert box}

VAR

theItem : INTEGER; {Item number for save Alert}
theData : apIHandle; {Handle to application data record}
dataHandle : Handle; {Untyped handle to lock data record MI-9}
wkString : STR255; {Temporary STR255 space}

```

ELSE IF (evaluatorEditRecd^.TELength <= 1) THEN
    {Evaluator text length <= 1 character}
    theItem := StopAlert (emptyTextID, NIL)
    {Post 'emptyTextID' stop alert DM-28}

ELSE continue := TRUE;
    {All text items > 1 character}

innovation:
BEGIN

    continue := TRUE;
    {Set checkInput flag}

    IF innovationPar <= 5 THEN
        {InnovationPar is less than 5 %}
        BEGIN
            theItem := CautionAlert (estimatorLowID, NIL);
            CASE theItem OF
                no : continue := TRUE;
                yes : continue := FALSE;
            END; {CASE}
            {Post 'estimatorLowID' alert DM-28}
            {Set checkInput flag}
            {Set checkInput flag}
        END; {IF}

    IF innovationPar >= 15 THEN
        {InnovationPar is more than 15 %}
        BEGIN
            theItem := CautionAlert (estimatorHighID, NIL);
            CASE theItem OF
                no : continue := TRUE;
                yes : continue := FALSE;
            END; {CASE}
            {Post 'estimatorHighID' alert DM-28}
            {Set checkInput flag}
            {Set checkInput flag}
        END; {IF}

    END; {innovation}

imitation:
BEGIN

    continue := TRUE;
    {Set checkInput flag}

    IF (innovationPar + imitationPar >= 100) THEN
        BEGIN
            theItem := StopAlert (badModelID, NIL);
            {Post 'badModelID' stop alert DM-28}
            continue := FALSE;
            GOTO 1011;
        END;
        {Set checkInput flag}
        {Skip rest of operation}
        {IF}

    IF imitationPar <= 15 THEN
        {ImitationPar is less than 15 %}
        BEGIN
            IF (innovationPar < 1) AND (imitationPar < 1) THEN
                BEGIN
                    theItem := StopAlert (nullPenetID, NIL);
                    {Post 'nullPenetID' stop alert DM-28}
                    continue := FALSE;
                    GOTO 1011;
                END;
                {Set checkInput flag}
                {Skip rest of operation}
                {IF}

            theItem := CautionAlert (estimatorLowID, NIL);
            CASE theItem OF
                {Post 'estimatorLowID' alert DM-28}

```

```

        no : continue := TRUE;    {Set checkInput flag}
        yes : continue := FALSE;  {Set checkInput flag}
    END; {CASE}
END; {IF}

IF imitationPar >= 30 THEN        {ImitationPar is more than 30 %}
    BEGIN
        theItem := CautionAlert (estimatorHighID, NIL);
        CASE theItem OF          {Post 'estimatorHighID' alert DM-28}
            no : continue := TRUE;    {Set checkInput flag}
            yes : continue := FALSE;  {Set checkInput flag}
        END; {CASE}
    END; {IF}

END; {imitation}

influence:
    BEGIN

        continue := TRUE;        {Set checkInput flag}

        IF influencePar >= 50 THEN    {InfluencePar is more than 50 %}
            BEGIN
                theItem := CautionAlert (estimatorHighID, NIL);
                CASE theItem OF      {Post 'estimatorHighID' alert DM-28}
                    no : continue := TRUE;    {Set checkInput flag}
                    yes : continue := FALSE;  {Set checkInput flag}
                END; {CASE}
            END; {IF}

        END; {influence}

potential:
    BEGIN

        continue := TRUE;        {Set checkInput flag}

        IF potentialPar <= 50 THEN    {PotentialPar is less than 50 %}
            BEGIN
                IF potentialPar < 1 THEN    {PotentialPar is zero!!!}
                    BEGIN
                        theItem := StopAlert (nullPotentialID, NIL);
                        {Post 'nullPotentialID' stop alert DM-28}
                        continue := FALSE;    {Set checkInput flag}
                        GOTO 1011;          {Skip rest of operation}
                    END; {IF}

                    theItem := CautionAlert (estimatorLowID, NIL);
                    CASE theItem OF      {Post 'estimatorLowID' alert DM-28}
                        no : continue := TRUE;    {Set checkInput flag}
                        yes : continue := FALSE;  {Set checkInput flag}
                    END; {CASE}
                END; {IF}

            END;

            IF potentialPar >= 95 THEN    {PotentialPar is more than 95 %}
                BEGIN
                    theItem := CautionAlert (estimatorHighID, NIL);
                    CASE theItem OF      {Post 'estimatorHighID' alert DM-28}

```

```

        no : continue := TRUE;    {Set checkInput flag}
        yes : continue := FALSE; {Set checkInput flag}
    END; {CASE}
END; {IF}

END; {potential}

replace:
BEGIN

    continue := TRUE;            {Set checkInput flag}

    IF replacePar <= 5 THEN      {ReplacePar is less than 5 %}
    BEGIN
        theItem := CautionAlert (estimatorLowID, NIL);
        CASE theItem OF
            no : continue := TRUE;    {Set checkInput flag}
            yes : continue := FALSE; {Set checkInput flag}
        END; {CASE}
    END; {IF}

    IF replacePar >= 15 THEN     {ReplacePar is more than 15 %}
    BEGIN
        theItem := CautionAlert (estimatorHighID, NIL);
        CASE theItem OF
            no : continue := TRUE;    {Set checkInput flag}
            yes : continue := FALSE; {Set checkInput flag}
        END; {CASE}
    END; {IF}

END; {replace}

externalVC:
BEGIN

    continue := TRUE;            {Set checkInput flag}

    IF externalVCPAr <= 5 THEN  {ExternalVCPAr is less than 5 %}
    BEGIN
        theItem := CautionAlert (estimatorLowID, NIL);
        CASE theItem OF
            no : continue := TRUE;    {Set checkInput flag}
            yes : continue := FALSE; {Set checkInput flag}
        END; {CASE}
    END; {IF}

    IF externalVCPAr >= 50 THEN {ExternalVCPAr is more than 50 %}
    BEGIN
        theItem := CautionAlert (estimatorHighID, NIL);
        CASE theItem OF
            no : continue := TRUE;    {Set checkInput flag}
            yes : continue := FALSE; {Set checkInput flag}
        END; {CASE}
    END; {IF}

END; {externalVC}

internalVC:

```

BEGIN

```

continue := TRUE;           {Set checkInput flag}

IF internalVCPAr <= 5 THEN   {InternalVCPAr is less than 5 %}
  BEGIN
    IF (externalVCPAr < 1) AND (internalVCPAr < 1) THEN
      BEGIN
        {ExternalVCPAr and internalVCPAr are zero!!!}
        theItem := StopAlert (nullCostID, NIL);
        {Post 'nullPenetID' stop alert DM-28}
        continue := FALSE;   {Set checkInput flag}
        GOTO 1011;           {Skip rest of operation}
      END; {IF}

    IF NOT testLoss THEN     {Test if externalVCPAr + internalVCPAr > 100}
      BEGIN
        continue := FALSE;   {Set checkInput flag}
        GOTO 1011;           {Skip rest of operation}
      END; {THEN}

    theItem := CautionAlert (estimatorLowID, NIL);
    CASE theItem OF
      no : continue := TRUE;   {Set checkInput flag}
      yes : continue := FALSE; {Set checkInput flag}
    END; {CASE}

  END; {IF}

  IF internalVCPAr >= 60 THEN {InternalVCPAr is more than 60 %}
    BEGIN
      IF NOT testLoss THEN   {Test if externalVCPAr + internalVCPAr > 100}
        BEGIN
          continue := FALSE; {Set checkInput flag}
          GOTO 1011;         {Skip rest of operation}
        END; {THEN}

        theItem := CautionAlert (estimatorHighID, NIL);
        CASE theItem OF
          no : continue := TRUE; {Set checkInput flag}
          yes : continue := FALSE; {Set checkInput flag}
        END; {CASE}
      END; {IF}

    IF (internalVCPAr > 5) AND (internalVCPAr < 60) THEN
      BEGIN
        {5 % < internalVCPAr < 60 %}
        IF NOT testLoss THEN {Test if externalVCPAr + internalVCPAr > 100}
          BEGIN
            continue := FALSE; {Set checkInput flag}
            GOTO 1011;         {Skip rest of operation}
          END; {THEN}
        END; {IF}
      END; {internalVC}
    END;
  END;

```

learning:

BEGIN

```

continue := TRUE;           {Set checkInput flag}

```

```

IF learningPar <= 60 THEN      {LearningPar is less than 60 %}
  BEGIN
    theItem := CautionAlert (estimatorLowID, NIL);
    CASE theItem OF           {Post 'estimatorLowID' alert DM-28}
      no  : continue := TRUE;  {Set checkInput flag}
      yes : continue := FALSE; {Set checkInput flag}
    END; {CASE}
  END; {IF}

END; {learning}

environmt:
  BEGIN
    continue := TRUE;          {Set checkInput flag}

    GetIText (adopterEditRecd^.hText, wkString);
                                {Return adopterText as a STR255 DL-31}
    wkNum := stringToReal (wkString); {U6: STR255 -> REAL conversion}

    IF wkNum = 0 THEN
      BEGIN
        theItem := StopAlert (emptyTextID, NIL);
                                {Post 'emptyTextID' alert DM-28}
        continue := FALSE;      {Set checkInput flag}
        GOTO 1011;              {Skip rest of routine}
      END; {IF}

    GetIText (fCostEditRecd^.hText, wkString);
                                {Return fCostText as a STR255 DL-31}
    wkNum := stringToReal (wkString); {U6: STR255 -> REAL conversion}

    IF wkNum = 0 THEN
      BEGIN
        theItem := StopAlert (emptyTextID, NIL);
                                {Post 'emptyTextID' alert DM-28}
        continue := FALSE;      {Set checkInput flag}
        GOTO 1011;              {Skip rest of routine}
      END; {IF}

    GetIText (productionEditRecd^.hText, wkString);
                                {Return productionText as a STR255 DL-31}
    wkNum := stringToReal (wkString); {U6: STR255 -> REAL conversion}

    IF wkNum = 0 THEN
      BEGIN
        theItem := StopAlert (emptyTextID, NIL);
                                {Post 'emptyTextID' alert DM-28}
        continue := FALSE;      {Set checkInput flag}
        GOTO 1011;              {Skip rest of routine}
      END; {IF}

    GetIText (priceEditRecd^.hText, wkString);
                                {Return priceText as a STR255 DL-31}
    wkNum := stringToReal (wkString); {U6: STR255 -> REAL conversion}

    IF wkNum = 0 THEN
      BEGIN
        theItem := StopAlert (emptyTextID, NIL);

```

```

        continue := FALSE;           {Post 'emptyTextID' alert DM-28}
        GOTO 1011;                   {Set checkInput flag}
    END; {IF}                         {Skip rest of routine}

    END; {environmt}

    END; {CASE}

    HUnlock (dataHandle);             {Unlock application data record MM-43}

1011: END; {checkInput}

{-----}

PROCEDURE activateProduct ((mousePoint: Point)); {Deactivate/activate appropriate Edit record}

VAR

    theData      : aplHandle;         {Handle to application data record}
    dataHandle    : Handle;           {Untyped handle to lock data record MI-9}

BEGIN {activateProduct}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                {Get and lock appl data record WM-33, MM-42}
    theData := aplHandle (dataHandle); {Convert to aplHandle}

    deActivateProduct;                 {U7: Deactivate appropriate Edit record}

    WITH theData DO

        IF PtinRect (mousePoint, divisionEditRecd^.viewRect) THEN
            BEGIN                     {Is mouse in division Edit Rectangle QD-47}
                textRecd := division; {Set text record flag}
                TEActivate (divisionEditRecd); {Activate selected Edit record TE-19}
                theText := divisionEditRecd; {TheText set to currently active Edit record}
            END

        ELSE IF PtinRect (mousePoint, brandEditRecd^.viewRect) THEN
            BEGIN                     {Is mouse in brand Edit Rectangle QD-47}
                textRecd := brand;     {Set text record flag}
                TEActivate (brandEditRecd); {Activate selected Edit record TE-19}
                theText := brandEditRecd; {TheText set to currently active Edit record}
            END

        ELSE IF PtinRect (mousePoint, marketEditRecd^.viewRect) THEN
            BEGIN                     {Is mouse in market Edit Rectangle QD-47}
                textRecd := market;    {Set text record flag}
                TEActivate (marketEditRecd); {Activate selected Edit record TE-19}
                theText := marketEditRecd; {TheText set to currently active Edit record}
            END

        ELSE IF PtinRect (mousePoint, evaluatorEditRecd^.viewRect) THEN
            BEGIN                     {Is mouse in evaluator Edit Rectangle QD-47}
                textRecd := evaluator;  {Set text record flag}
                TEActivate (evaluatorEditRecd); {Activate selected Edit record TE-19}
                theText := evaluatorEditRecd; {TheText set to currently active Edit record}
            END
    END

```

```

END;

HUnlock (dataHandle);           {Unlock application data record MM-43}

END;  {activateProduct}

{-----}

PROCEDURE deActivateProduct;    {Deactivate appropriate Edit record}

VAR

  theData      : apIHandle;     {Handle to application data record}
  dataHandle    : Handle;       {Untyped handle to lock data record MI-9}

BEGIN {deActivateProduct}

  dataHandle := Handle (GetWRefCon (theWindow));
  Hlock (dataHandle);           {Get and lock appl data record WM-33, MM-42}
  theData := apIHandle (dataHandle); {Convert to apIHandle}

  WITH theData DO

    CASE textRecd OF           {Deactivate any previous Edit record TE-19}

      division:
        BEGIN
          TEDeactivate (divisionEditRecd); {Deactivate selected Edit record TE-19}
          textRecd := none;                {Set text record flag to 'nothing'}
          theText := NIL;                  {TheText set to NIL}
        END; {division}

      brand:
        BEGIN
          TEDeactivate (brandEditRecd);    {Deactivate selected Edit record TE-19}
          textRecd := none;                {Set text record flag to 'nothing'}
          theText := NIL;                  {TheText set to NIL}
        END; {brand}

      market:
        BEGIN
          TEDeactivate (marketEditRecd);   {Deactivate selected Edit record TE-19}
          textRecd := none;                {Set text record flag to 'nothing'}
          theText := NIL;                  {TheText set to NIL}
        END; {market}

      evaluator:
        BEGIN
          TEDeactivate (evaluatorEditRecd); {Deactivate selected Edit record TE-19}
          textRecd := none;                {Set text record flag to 'nothing'}
          theText := NIL;                  {TheText set to NIL}
        END; {evaluator}

    END; {CASE}

  HUnlock (dataHandle);           {Unlock application data record MM-43}

END; {deActivateProduct}

```



```

theData      : apIHandle;           {Handle to application data record}
dataHandle   : Handle;              {Untyped handle to lock data record MI-9}

BEGIN {deActivateEnvironmt}

  dataHandle := Handle (GetWRefCon (theWindow));
  Hlock (dataHandle);                {Get and lock appl data record WM-33, MM-42}
  theData := apIHandle (dataHandle); {Convert to apIHandle}

  WITH theData DO

    CASE textRecd OF                 {Deactivate any previous Edit record TE-19}

      adopter :
        BEGIN
          TEdesactivate (adopterEditRecd); {Deactivate selected Edit record TE-19}
          textRecd := none;                 {Set text record flag to 'nothing'}
          theText := NIL;                   {TheText set to NIL}
        END; {adopter}

      fCost:
        BEGIN
          TEdesactivate (fCostEditRecd);    {Deactivate selected Edit record TE-19}
          textRecd := none;                 {Set text record flag to 'nothing'}
          theText := NIL;                   {TheText set to NIL}
        END; {fCost}

      production:
        BEGIN
          TEdesactivate (productionEditRecd); {Deactivate selected Edit record TE-19}
          textRecd := none;                 {Set text record flag to 'nothing'}
          theText := NIL;                   {TheText set to NIL}
        END; {production}

      price:
        BEGIN
          TEdesactivate (priceEditRecd);    {Deactivate selected Edit record TE-19}
          textRecd := none;                 {Set text record flag to 'nothing'}
          theText := NIL;                   {TheText set to NIL}
        END; {price}

    END; {CASE}

  HUnlock (dataHandle);               {Unlock application data record MM-43}

END; {deActivateEnvironmt}

{-----}

PROCEDURE doSelect ((thePoint : Point; thetext : TEHandle));
                                     {Handle mouse-down in text rectangle}

VAR

  extend : BOOLEAN;                  {Extend text selection (Shift-Click) TE-18}

BEGIN {doSelect}

```

```

WITH theEvent DO
    extend := (BitAnd(modifiers, ShiftKey) <> 0);
                                {Is Shift key down? EM-15}
    TEClick(thePoint, extend, theText);
                                {Highlight selection range TE-18}
    fixEditMenu;
                                {U7: Enable/Disable editing commands}

END; {doSelect}

{-----}

PROCEDURE fixEditMenu;
                                {Enable/Disable editing commands}

BEGIN {fixEditMenu}

    DisableItem(editMenu, undoItem);
                                {Undo command sits idle}

    Hlock(Handle(theText));
                                {Lock selected edit record MM-42}

    WITH theText DO

        IF SelStart = SelEnd THEN
                                {Selection range is empty TE-09}
            BEGIN
                DisableItem(editMenu, cutItem);
                                {Disable menu items that operate on a}
                DisableItem(editMenu, copyItem);
                                {nonempty selection MN-24}
                DisableItem(editMenu, clearItem);
            END {THEN}

        ELSE
            BEGIN
                EnableItem(editMenu, cutItem);
                                {Enable menu items that operate on a}
                EnableItem(editMenu, copyItem);
                                {nonempty selection MN-24}
                EnableItem(editMenu, clearItem);
            END; {ELSE}

        HUnlock(Handle(theText));
                                {Unlock selected edit record MM-43}

        IF TEGetScrapLen = 0 THEN
                                {TextEdit Scrap is empty TE-24}
            DisableItem(editMenu, pasteItem);
                                {Disable PasteItem MN-23}
        ELSE
                                {Scrap is nonempty}
            EnableItem(editMenu, pasteItem);
                                {Enable PasteItem MN-24}

        END; {fixEditMenu}

{-----}

PROCEDURE doGoAway(whichWindow:windowPtr); {Handle click in goaway region}

BEGIN {doGoAway}

    IF whichWindow <> FrontWindow THEN
                                {Is it an inactive window? WM-26}
        SelectWindow(whichWindow);
                                {If so, activate it WM-23}

    ELSE IF TrackGoAway(theWindow, theEvent.where) THEN
                                {Click in goAway region of appl wd WM-27}
        doClose;
                                {U3: Handle close command}

    END; {doGoAway}

```

-----}

```

PROCEDURE doKeyStroke;                                {Handle keystroke events}

VAR

  ch          : CHAR;                                {Character that was typed}
  chInt       : INTEGER;                             {INTEGER value of character typed}

  menuChoice  : LongInt;                             {Menu ID and item number for keyboard}
                                                    {equivalents}

BEGIN {doKeyStroke}

  chInt := theEvent.message MOD 256;                  {Extract character code EM-16, EM-28}
  ch    := CHR (chInt);                              {Convert to a character}

  IF BitAnd (theEvent.modifiers, CmdKey) <> 0 THEN
    BEGIN
      IF theEvent.what <> AutoKey THEN                {Command key down EM-15}
                                                {Ignore repeats EM-12}
      BEGIN
        menuChoice := MenuKey (ch);                  {Get menu equivalent MN-22}
        DoMenuChoice (menuChoice);                  {Handle user's menu choice}
      END {IF}
    END {THEN}
  ELSE
    doTyping;                                         {U7: Handle character typed from keyboard}

END; {doKeyStroke}

```

-----}

```

PROCEDURE doTyping;                                  {Handle character typed from keyboard}

TYPE

  charSet     = SET OF CHAR;                         {Set of valid keys for environmt screen}

VAR

  ch          : CHAR;                                {Character that was typed}
  chInt       : INTEGER;                             {INTEGER value of character typed}

  inputLength : INTEGER;                             {Maximum length of input string}
  validKeys   : charSet;                             {Valid keys for environmt screen}

BEGIN {doTyping}

  chInt := theEvent.message MOD 256;                  {Extract character code EM-16, EM-28}
  ch    := CHR (chInt);                              {Convert to a character}

  CASE theScreen OF

    product:
      IF textRecd <> none THEN                        {There is an 'active' Text record}
      BEGIN
        inputLength := 25;                            {Limit editable text to 25 characters}
        IF (theText^.TELength <= inputLength - 1) OR (chInt = 8) THEN
          {Accept backspace BSp = $08 EM-08}

```

```

IF (chInt <> 3) AND (chInt <> 9) AND (chInt <> 13) THEN
    {Filter for unaccepted keyboard entries EM-08}
    {enter, 'tab', 'return'}

    BEGIN
        TEKey (ch,theText);      {Process character TE-19}
        DisableItem (editMenu, cutItem);
                                {Disable menu items that operate on a}
                                {nonempty selection MN-24}
        DisableItem (editMenu, copyItem);
        DisableItem (editMenu, clearItem);
        DisableItem (editMenu, pasteItem);

        windowDirty (TRUE);      {U4: Mark window as dirty}
    END; {IF}
END; {product}

environmt:
IF textRecd <> none THEN        {There is an 'active' Text record}
    BEGIN
        validKeys := ['0','1','2','3','4','5','6','7','8','9'];
                                {Acceptable set comprises integers 0..9}
        inputLength := 13;      {Limit editable text to 13 digits}

        IF (theText^.TELength <= inputLength - 1) OR (chInt = 8) THEN
            {Accept backspace BSp = $06 EM-08}
            IF (chInt = 8) OR (ch IN validKeys) THEN
                BEGIN
                    TEKey (ch,theText);      {Process character TE-19}
                    DisableItem (editMenu, cutItem);
                                            {Disable menu items that operate on a}
                                            {nonempty selection MN-24}
                    DisableItem (editMenu, copyItem);
                    DisableItem (editMenu, clearItem);
                    DisableItem (editMenu, pasteItem);

                    windowDirty (TRUE);      {U4: Mark window as dirty}
                END; {IF}
            END; {environmt}

        END; {CASE}

    END; {doTyping}

{-----}

PROCEDURE doUpdate;              {Handle update events}

VAR

    savePort      : GrafPtr;      {Pointer to previous port QD-18}

    theData        : apIHandle;    {Handle to application data record}
    dataHandle     : Handle;       {Untyped handle to lock application data MI-9}

    BEGIN {doUpdate}

        GetPort (savePort);        {Save current grafPort QD-36}
        SetPort (theWindow);        {Make theWindow the current port QD-36}

```

```

BeginUpdate(theWindow);                {Restrict visible region to update reg WM-32}

dataHandle := Handle (GetWRefCon (theWindow));
Hlock (dataHandle);                    {Get and lock appl data record WM-33, MM-42}
theData := apiHandle (dataHandle);     {Convert to apiHandle}

WITH theData DO

  CASE theScreen OF

    product:
      BEGIN
        CASE textRecd OF
          division    : TEUpdate(divisionEditRecd^.viewRect, divisionEditRecd);
          brand       : TEUpdate(brandEditRecd^.viewRect, brandEditRecd);
          market      : TEUpdate(marketEditRecd^.viewRect, marketEditRecd);
          evaluator    : TEUpdate(evaluatorEditRecd^.viewRect, evaluatorEditRecd);
        END; {CASE}                      {Redraw the text after an update evt TE-22}
        setScreenProduct;                {U5: Screen layout for product parameters}
      END;

    environmt :
      BEGIN
        CASE textRecd OF
          adopter      : TEUpdate(adopterEditRecd^.viewRect, adopterEditRecd);
          fCost         : TEUpdate(fCostEditRecd^.viewRect, fCostEditRecd);
          production    : TEUpdate(productionEditRecd^.viewRect, productionEditRecd);
          price         : TEUpdate(priceEditRecd^.viewRect, priceEditRecd);
        END; {CASE}                      {Redraw the text after an update evt TE-22}
        setScreenEnvironmt;              {U5: Screen layout for environmt parameters}
      END;

  END; {CASE}

HUnlock (dataHandle);                  {Unlock application data record}

CASE theScreen OF

  innovation      : doInnovation;        {Update innovation screen}
  imitation       : doImitation;         {Update imitation screen}
  influence       : doInfluence;         {Update influence screen}
  potential       : doPotential;         {Update potential screen}
  replace         : doReplace;           {Update replace screen}
  externalVC      : doExternalVC;        {Update externalVC screen}
  internalVC      : doInternalVC;        {Update internalVC screen}
  learning        : doLearning;          {Update learning screen}

  summary         : displaySummary;      {Update summary screen}
  innovationPlot  : doInnovationCur;    {Update innovation curve screen}
  imitationPlot   : doImitationCur;     {Update imitation curve screen}
  penetrationPlot : doPenetrationCur;   {Update penetration curve screen}
  replacePlot     : doReplaceCur;       {Update replace curve screen}
  volumePlot      : doVolumeCur;        {Update volume curve screen}
  internalVCPlot  : doInternalVCCur;     {Update internalVC curve screen}
  totalCPlot      : doTotalCCur;         {Update totalC curve screen}
  profitPlot      : doProfitCur;        {Update profit curve screen}

END; {CASE}

```

```

EndUpdate (theWindow);           {Restore original visible region WM-32}
SetPort(savePort);               {Restore original port QD-36}

END; {doUpdate}

{-----}

PROCEDURE doactivate;             {Handle activate events}

CONST

    changeFlag    = $0002;        {Mask to get 'change bit' from Evt.modifiers}
                                   {EM-15}

VAR

    whichWindow   : WindowPtr;     {Pointer to active window WM-11}
    fName         : STR255;        {Window (file) name}
    newCloseItem  : STR255;        {New label for Close item}

    theData       : apIHandle;     {Handle to application data record}
    dataHandle    : Handle;        {Untyped handle to lock application data MI-9}

BEGIN {doActivate}

    whichWindow := WindowPtr (theEvent.message); {Convert LongInt to Pointer}
    SetPort (whichWindow);             {Make whichWindow the current port QD-36}

    dataHandle := Handle (GetWRefCon (theWindow));
    Hlock (dataHandle);                {Get and lock appl data record WM-33, MM-42}
    theData := apIHandle (dataHandle); {Convert to apIHandle}

    WITH theData^^ DO

        IF BitAnd (theEvent.modifiers, ActiveFlag) <> 0 THEN
            {Appl window -> ACTIVE TU-08, EM-15}

            BEGIN

                CASE theScreen OF

                    product:
                        CASE textRecd OF

                            division:
                                {IMPORTANT set text Edit flag for 'update'}
                                BEGIN
                                    TEActivate (divisionEditRecd);
                                    {Activate division Edit Record TE-19}
                                    theText := divisionEditRecd;
                                END; {division}

                            brand:
                                BEGIN
                                    TEActivate (brandEditRecd);
                                    {Activate brand Edit Record TE-19}
                                    theText := brandEditRecd;
                                END; {brand}

                            market:

```

```

BEGIN
    TEActivate (marketEditRecd);
                                {Activate market Edit Record TE-19}
    theText := marketEditRecd;
END; {market}

evaluator:
BEGIN
    TEActivate (evaluatorEditRecd);
                                {Activate evaluator Edit Record TE-19}
    theText := evaluatorEditRecd;
END; {evaluator}

END; {CASE}

environmt:
CASE textRecd OF

    adopter:
                                {IMPORTANT set text edit flag for 'update'}
        BEGIN
            TEActivate (adopterEditRecd);
                                {Activate adopter Edit Record TE-19}
            theText := adopterEditRecd;
        END; {adopter}

    fCost:
        BEGIN
            TEActivate (fCostEditRecd);
                                {Activate fixed cost Edit Record TE-19}
            theText := fCostEditRecd;
        END; {fCost}

    production:
        BEGIN
            TEActivate (productionEditRecd);
                                {Activate production Edit Record TE-19}
            theText := productionEditRecd;
        END; {production}

    price:
        BEGIN
            TEActivate (priceEditRecd);
                                {Activate price Edit Record TE-19}
            theText := priceEditRecd;
        END; {price}

END; {CASE}

END; {CASE}

IF BitAnd (theEvent.modifiers, changeFlag) <> 0 THEN
                                {Coming from a SYS window TU-08, EM-15}
CASE theScreen OF

    product, environmt:
        BEGIN
            menusEdit;          {U3: Activate only Edit menu items}
            readDeskScrap;      {U1: Desk Scrap -> ToolBox Scrap}

```

```

DisableItem (fileMenu, saveItem);
                                {Disable Save command MN-23}
IF (theData^.dirty) AND (fileNumber <> 0) THEN
    EnableItem (fileMenu, saveItem);

IF (whichWindow = theWindow) AND (fileNumber <> 0) THEN
    BEGIN
        GetWindowTitle (theWindow, fName);
                                {Get window (file) name WM-23}
        newCloseItem := CONCAT ('Fermer ', fName);
        SetItem (fileMenu, closeItem, newCloseItem);
                                {Add file name in close command MN-23}
    END; {IF}

IF simulation THEN
    DrawMenuBar;                {Redraw MnBar according to menu list MN-19}

END; {product, environmt}

innovation, imitation, influence, potential, replace,
externalVC, internalVC, learning:
BEGIN
    IF NOT simulation THEN      {In calibration mode}
        BEGIN
            menusEvaluation;    {U7: Activate evaluation menu items}
            IF (whichWindow = theWindow) AND (fileNumber <> 0) THEN
                BEGIN
                    GetWindowTitle (theWindow, fName);
                                {Get window (file) name WM-23}
                    newCloseItem := CONCAT ('Fermer ', fName);
                    SetItem (fileMenu, closeItem, newCloseItem);
                                {Add file name in close command MN-23}
                END; {IF}
            DrawMenuBar;        {Redraw MnBar according to menu list MN-19}
        END {THEN}
    ELSE
        BEGIN
            menusSimulation;     {U7: Activate simulation menu items}
            fixSimulationMenu;   {U7: Update simulation menu items}
            IF (whichWindow = theWindow) AND (fileNumber <> 0) THEN
                BEGIN
                    GetWindowTitle (theWindow, fName);
                                {Get window (file) name WM-23}
                    newCloseItem := CONCAT ('Fermer ', fName);
                    SetItem (fileMenu, closeItem, newCloseItem);
                                {Add file name in close command MN-23}
                END; {IF}
            DrawMenuBar;        {Redraw MnBar according to menu list MN-19}
        END; {ELSE}
    END; {Calibration screens}

summary, innovationPlot, imitationPlot, penetrationPlot, replacePlot,
volumePlot, internalVCPlot, totalICPlot, profitPlot:
BEGIN
    menusSimulation;           {U7: Activate simulation menu items}
    fixSimulationMenu;         {U7: Update simulation menu items}
    IF (whichWindow = theWindow) AND (fileNumber <> 0) THEN

```

```

        BEGIN
            GetWTitle (theWindow, fName);
                                {Get window (file) name WM-23}
            newCloseItem := CONCAT ('Ferner ', fName);
            SetItem (fileMenu, closeItem, newCloseItem);
                                {Add file name in close command MN-23}

            END; {IF}
            DrawMenuBar;          {Redraw MnBar according to menu list MN-19}
        END; {Results screens}

    END {CASE}

ELSE                                {No desk accessory was active, coming from}
    CASE theScreen OF                {DeskTop}
        product:
            BEGIN
                menusEdit;           {U3: Activate only Edit menu items}
                IF (whichWindow = theWindow) AND (fileNumber <> 0) THEN
                    BEGIN
                        GetWTitle (theWindow, fName);
                                    {Get window (file) name WM-23}
                        newCloseItem := CONCAT ('Ferner ', fName);
                        SetItem (fileMenu, closeItem, newCloseItem);
                                    {Add file name in close command MN-23}

                        END; {IF}
                        DrawMenuBar;   {Redraw MnBar according to menu list MN-19}
                    END; {product}
                END; {CASE}
            END {THEN}

ELSE                                {Appl window -> INACTIVE TU-08, EM-15}
    BEGIN

        theText      := NIL;          {Clear global pointer}

        CASE theScreen OF

            product:
                CASE textRecd OF        {Deactivate any previous Edit record TE-19}
                    division      : TEDeactivate (divisionEditRecd);
                    brand          : TEDeactivate (brandEditRecd);
                    market        : TEDeactivate (marketEditRecd);
                    evaluator      : TEDeactivate (evaluatorEditRecd);
                END; {CASE}

            environmt:
                CASE textRecd OF        {Deactivate any previous Edit record TE-19}
                    adopter       : TEDeactivate (adopterEditRecd);
                    fCost         : TEDeactivate (fCostEditRecd);
                    production    : TEDeactivate (productionEditRecd);
                    price         : TEDeactivate (priceEditRecd);
                END; {CASE}

        END; {CASE}

    IF BitAnd (theEvent.modifiers, changeFlag) <> 0 THEN
        CASE theScreen OF
            {Exiting to a SYSTEM window TU-08, EM-15}

```

```

product, environmt:
  BEGIN
    menusDesk;           {U2: Activate desk accessory menu items}
    writeDeskScrap;       {U1: ToolBox Scrap -> Desk Scrap}
    IF simulation THEN
      DrawMenuBar;        {Redraw MnBar according to menu list MN-19}
    END;  {product, environmt}

  OTHERWISE
    BEGIN
      menusDesk;           {U2: Activate desk accessory menu items}
      DrawMenuBar;         {Redraw MnBar according to menu list MN-19}
    END;  {OTHERWISE}
  END;  {CASE}

END;  {WITH}

HUnlock (dataHandle);    {Unlock application data record MM-43}

END;  {doActivate}

{-----}

{$S Main}                {Empty PROCEDURE Unit7.Main}
                          {To satisfy compiler syntax}

BEGIN
END.

```

9. ProduiTest.Res

```
* FILE: ProduitTest.R
```

```
* *****
* #
* #      Programme conçu et développé      #
* #      par Jean-Marie Choffray,          #
* #      avec l'aide de la FNEGE           #
* #
* # Copyright 1987      Tous droits réservés #
* #
* *****
```

```
HD:Pascal2:Resources:ProduitTest/Rsrc.Rel
```

```
*                                     ;;Name of Res file to be created
```

```
*-----Version data of the application
```

```
Type FGPR = STR
```

```
,0 (32)
```

```
;;Res ID is 0 ST-05, and purgeable RM-13
```

```
Jean-Marie Choffray Version 1.0 March 1, 1987
```

```
;;Version data of ProduitTest ST-05
```

```
*-----File reference resources ST-05
```

```
Type FREF
```

```
,128
```

```
;;Resource ID for the application
```

```
APPL 0
```

```
;;File type and local ID of icon list ST-05
```

```
,129
```

```
;;Resource ID for created files
```

```
FGFI 1
```

```
;;File type and local ID of icon list ST-05
```

```
*-----Resource type for a Bundle ST-06, ST-08
```

```
Type BNDL
```

```
,128
```

```
;;Resource ID
```

```
FGPR 0
```

```
;;Creator signature and ID of Vers data ST-06
```

```
ICN#
```

```
;;Res type for icon list
```

```
0 128 1 129
```

```
;;Maps local to actual ID for Appl icons
```

```
FREF
```

```
;;Res type for File references
```

```
0 128 1 129
```

```
;;Maps local to actual ID for File icons
```

```
*-----Resource type for user defined icons
```

```
Type ICN# = GNRL
```

```
;;Creation of ICN# resource type
```

```
,128 (32)
```

```
;;Res ID (Attribute byte = purgeable) RM-13
```

```
.H
```

```
;;Application icons
```

```
07FFFFFFC
```

```
08000006
```

```
1000000B
```

```
2FFFFFFE5
```

```
40000011
```

```
80100009
```

```
80920009
```

```
80440009
```

```
90100009
```

```
B9BB0009
```

```
90100009
```

```
904403C9
```

```
90921CC9
```

```
90106649
```

```
9010B309
```

```
90019989
```

```
9001CCC9
```



```

#-----Resource type for a window
Type WIND
    ,1001
    Sans titre                ;;Window title
    45 10 332 502            ;;Top left bottom right
    Invisible GoAway          ;;Evident!!!
    4                        ;;Window def ID (noGrowDocProc 'WM-09)
    0                        ;;Reference value
#-----Resource type for a menu
Type MENU
#-----Desk accessory menu
    ,2001                    ;;AppleID
    \14                      ;;HexaD 14, AppleMark
    A propos de ProduitTest...
    (-                      ;;Counts as an item in list

#-----File menu
    ,2002                    ;;FileID
    Fichier                  ;;Menu title
    Nouveau /N               ;;Menu items, /keyboard equivalent
    Ouvrir
    (-                      ;;Counts as an item in list
    Fermer
    Enregistrer
    Enregistrer Sous...
    (-                      ;;Counts as an item in list
    Imprimer l'écran
    (-                      ;;Counts as an item in list
    Quitter /Q

#-----Edit menu
    ,2003                    ;;EditID
    Editer                   ;;Menu title
    Annuler /Z               ;;Menu items, /keyboard equivalent
    (-                      ;;Counts as an item in list
    Couper /X
    Copier /C
    Coller /V
    Effacer

#-----Simulation menu
    ,2004                    ;;SimulationID
    Paramètres               ;;Menu title
    Définition Projet        ;;Menu items
    (-                      ;;Counts as an item in list
    Taux d'Innovation
    Taux d'Initiation
    Taux d'Influence
    Part Accessible
    (-                      ;;Counts as an item in list
    Taux de Renouvellement
    (-                      ;;Counts as an item in list
    Coût Var Externe
    Coût Var Interne
    Taux d'Apprentissage
    (-                      ;;Counts as an item in list
    Environnement

```

```

*-----Results menu
,2005                                ;;ResultsID
Analyse                             ;;Menu title
Synthèse des Données                ;;Menu items
(-                                  ;;Counts as an item in list
Innovation
Imitation
Pénétration
(-                                  ;;Counts as an item in list
Ventes de Renouvellement
Ventes Totales
(-                                  ;;Counts as an item in list
Coût Var Interne
Coût Total
(-                                  ;;Counts as an item in list
Profit Brut

*-----Resource type for alerts
Type ALRT
*-----Stop alert "AboutID"
,3001                                ;;AboutID alert
100 100 300 400                    ;;Top left bottom right
3001                                ;;Resource ID of item list
4444                                ;;Outline OK(4), box, no sound DL-36

*-----Stop alert "CantPrintID"
,3002                                ;;CantPrintID alert
100 100 220 400                    ;;Top left bottom right
3002                                ;;Resource ID of item list
5555                                ;;Outline OK(4), box, sound1 DL-36

*-----Stop alert "WrongTypeID"
,3003                                ;;WrongTypeID alert
100 100 220 400                    ;;Top left bottom right
3003                                ;;Resource ID of item list
5555                                ;;Outline OK(4), box, sound1 DL-36

*-----Caution alert "SaveID"
,3004                                ;;SaveID alert
100 100 220 400                    ;;Top left bottom right
3004                                ;;Resource ID of item list
0555                                ;;Outline Cancel, OK(3), box, sound1 DL-36

*-----Stop alert "IOErrID"
,3005                                ;;IOErrID alert
100 100 220 400                    ;;Top left bottom right
3005                                ;;Resource ID of item list
5555                                ;;Outline OK(4), box, sound1 DL-36

*-----Stop alert "EmptyTextID"
,3006                                ;;EmptyTextID alert
100 100 220 400                    ;;Top left bottom right
3006                                ;;Resource ID of item list
5555                                ;;Outline OK(4), box, sound1 DL-36

*-----Caution alert "EstimatorHighID"

```

,3007	;;EstimatorHighID alert
100 100 220 400	;;Top left bottom right
3007	;;Resource ID of item list
DDDD	;;Outline Cancel(4), box, sound1 DL-36
*-----Caution alert "EstimatorLowID"	
,3008	;;EstimatorLowID alert
100 100 220 400	;;Top left bottom right
3008	;;Resource ID of item list
DDDD	;;Outline Cancel(4), box, sound1 DL-36
*-----Caution alert "LossID"	
,3009	;;LossID alert
100 100 220 400	;;Top left bottom right
3009	;;Resource ID of item list
5555	;;Outline OK(4), box, sound1 DL-36
*-----Stop alert "NullPenetID"	
,3010	;;NullPenetID alert
100 100 220 400	;;Top left bottom right
3010	;;Resource ID of item list
5555	;;Outline OK(4), box, sound1 DL-36
*-----Stop alert "NullCostID"	
,3011	;;NullCostID alert
100 100 220 400	;;Top left bottom right
3011	;;Resource ID of item list
5555	;;Outline OK(4), box, sound1 DL-36
*-----Stop alert "NullPotentialID"	
,3012	;;NullPotentialID alert
100 100 220 400	;;Top left bottom right
3012	;;Resource ID of item list
5555	;;Outline OK(4), box, sound1 DL-36
*-----Stop alert "BadModelID"	
,3013	;;BadModelID alert
100 100 220 400	;;Top left bottom right
3013	;;Resource ID of item list
5555	;;Outline OK(4), box, sound1 DL-36
*-----Resource type for Item Lists	
Type DITL	
*-----Item list for the AboutID Stop alert	
,3001	;;Resource ID
12	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
155 210 180 230	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel Button. Second in alert !
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
IconItem Disabled	;;Icon dialog item
20 20 52 52	;;Top left bottom right
130	;;Resource ID of Icon

StaticText Disabled 20 60 35 280 Système de Diagnostic	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 40 60 55 280 Version 1.0.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 65 20 80 280 Développé par Jean-Marie Choffray,	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 80 20 95 280 Professeur de Management à l'ESSEC,	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 95 20 110 280 en collaboration avec la FNEGE.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 110 20 125 280 Copyright 1987. Tous droits réservés.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 140 20 155 200 B.P.105.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 155 20 170 200 Cergy 95021, FRANCE.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 170 20 185 200 Tel: (1) 30 38 38 00.	;;Static text dialog item ;;Top left bottom right ;;Message
*-----Item list for the CantPrintID Stop alert	
,3002	;;Resource ID
5	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 68 20 88 280 Ce document	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 88 20 108 280 ne peut être imprimé!	;;Static text dialog item ;;Top left bottom right ;;Message

*-----Item list for the WrongTypeID Stop alert

,3003	;;Resource ID
6	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
48 20 68 280	;;Top left bottom right
Le fichier	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 280	;;Top left bottom right
"0	;;"0" is to be replaced at execution time DL-30
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
ne peut être ouvert!	;;Message

*-----Item list for the SaveID Caution alert

,3004	;;Resource ID
7	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
Oui	;;Message
Button Enabled	;;Cancel button. Always second in alert DL-13
45 210 65 285	;;Top left bottom right
Annuler	;;Message
Button Enabled	;;Button dialog item
75 210 95 285	;;Top left bottom right
Non	;;Message
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
48 20 68 200	;;Top left bottom right
Le fichier	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 200	;;Top left bottom right
"0 a été modifié.	;;"0" is to be replaced at execution time DL-30
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
Voulez-vous le remplacer?	;;Message

*-----Item list for the IDErrID Stop alert

,3005	;;Resource ID
6	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
48 20 68 200	;;Top left bottom right
I/O erreur	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 280	;;Top left bottom right
Code : "0.	;;"0" is to be replaced at execution time DL-30
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
L'opération en cours est annulée.	;;Message

*-----Item list for the EmptyTextID Stop alert

,3006	;;Resource ID
5	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 290	;;Top left bottom right
Les paramètres 'produit' et 'environmt'	;;Message
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
doivent obligatoirement être définis!	;;Message

*-----Item list for the EstimatorHighID Caution alert

,3007	;;Resource ID
6	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
Non	;;Message

Button Enabled 45 210 65 285 Oui	;;Cancel button. Always second in alert DL-13 ;;Top left bottom right ;;Message
StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 48 20 68 200 Cet estimateur	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 68 20 88 200 paraît à priori très élevé.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 88 20 108 280 Souhaitez-vous en revoir l'évaluation?	;;Static text dialog item ;;Top left bottom right ;;Message
*-----Item list for the EstimatorLowID Caution alert	
,3008 6 Button Enabled 15 210 35 285 Non	;;Resource ID ;;Number of items in list ;;OK button. Always first in alert DL-13 ;;Top left bottom right ;;Message
Button Enabled 45 210 65 285 Oui	;;Cancel button. Always second in alert DL-13 ;;Top left bottom right ;;Message
StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 48 20 68 200 Cet estimateur	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 68 20 88 200 paraît à priori très faible.	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 88 20 108 280 Souhaitez-vous en revoir l'évaluation?	;;Static text dialog item ;;Top left bottom right ;;Message
*-----Item list for the LossID Caution alert	
,3009 6 Button Enabled 15 210 35 285 Non	;;Resource ID ;;Number of items in list ;;OK button. Always first in alert DL-13 ;;Top left bottom right ;;Message
Button Enabled 45 210 65 285 Oui	;;Cancel button. Always second in alert DL-13 ;;Top left bottom right ;;Message

StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 48 20 68 200 Le coût variable total est	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 68 20 88 280 supérieur au prix de vente!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 88 20 108 280 Changement du coût variable interne?	;;Static text dialog item ;;Top left bottom right ;;Message
*-----Item list for the NullPenetID Stop alert	
,3010	;;Resource ID
6	;;Number of items in list
Button Enabled 15 210 35 285	;;OK button. Always first in alert DL-13 ;;Top left bottom right
OK	;;Message
Button Disabled 150 400 180 450 Annuler	;;Cancel button. Always second in alert DL-13 ;;Outside of range to be 'invisible' ;;Message (not visible!!!)
StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 48 20 68 280 Vos estimateurs des taux	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 68 20 88 280 d'innovation et d'imitation	;;Static text dialog item ;;Top left bottom right ;;Message
StaticText Disabled 88 20 108 280 ne peuvent être tous deux nuls!	;;Static text dialog item ;;Top left bottom right ;;Message
*-----Item list for the NullCostID Stop alert	
,3011	;;Resource ID
6	;;Number of items in list
Button Enabled 15 210 35 285	;;OK button. Always first in alert DL-13 ;;Top left bottom right
OK	;;Message
Button Disabled 150 400 180 450 Annuler	;;Cancel button. Always second in alert DL-13 ;;Outside of range to be 'invisible' ;;Message (not visible!!!)
StaticText Disabled 15 70 35 200 ATTENTION!!!	;;Static text dialog item ;;Top left bottom right ;;Message

StaticText Disabled	;;Static text dialog item
48 20 68 280	;;Top left bottom right
Vos estimateurs du coût variable	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 280	;;Top left bottom right
(entrants et valeur ajoutée)	;;Message
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
ne peuvent être tous deux nuls!	;;Message
#-----Item list for the NullPotentialID Stop alert	
,3012	;;Resource ID
5	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 280	;;Top left bottom right
La pénétration potentielle maximale	;;Message
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
de votre produit ne peut être nulle!	;;Message
#-----Item list for the BadModelID Stop alert	
,3013	;;Resource ID
5	;;Number of items in list
Button Enabled	;;OK button. Always first in alert DL-13
15 210 35 285	;;Top left bottom right
OK	;;Message
Button Disabled	;;Cancel button. Always second in alert DL-13
150 400 180 450	;;Outside of range to be 'invisible'
Annuler	;;Message (not visible!!!)
StaticText Disabled	;;Static text dialog item
15 70 35 200	;;Top left bottom right
ATTENTION!!!	;;Message
StaticText Disabled	;;Static text dialog item
68 20 88 280	;;Top left bottom right
Paramètres non cohérents !	;;Message
StaticText Disabled	;;Static text dialog item
88 20 108 280	;;Top left bottom right
(innovation + imitation) > 1.0	;;Message

*-----Resource type for user defined icons

Type ICON = GNRL

,130 (32)

;;Res ID (Attribute byte = purgeable)

.H

07FFFFFFC

08000006

1000000B

2FFFFFFE5

40000011

80100009

80920009

80440009

90100009

B9BB0009

90100009

904403C9

90921CC9

90106649

9010B309

90019989

9001CCC9

90026649

9FFFFFFC9

9AAA0009

95540009

9AAC7F09

95583E09

9AA01C09

95C00809

9A000049

9FFFFFFE9

8000004A

80000004

80000008

40000010

3FFFFFFE0

*-----ATTENTION, the last line needs a "return"

10. ProduiTest.Link

!PAS\$Xfer

/Globals -4

HD:Pascal2:Program:ProdiTest

HD:Pascal2:RSL:PAS\$Library

HD:Pascal2:M1:OSTraps

HD:Pascal2:M1:ToolTraps

HD:Pascal2:M1:PackTraps

HD:Pascal2:M1:PrLink

<

HD:Pascal2:Program:ProdiTest/Seg1

<

HD:Pascal2:Unit1:Unit1/Code

<

HD:Pascal2:Program:ProdiTest/Seg2

<

HD:Pascal2:Unit2:Unit2/Code

<

HD:Pascal2:Program:ProdiTest/Seg3

<

HD:Pascal2:Unit3:Unit3/Code

<

HD:Pascal2:Program:ProdiTest/Seg4

<

HD:Pascal2:Unit4:Unit4/Code

<

HD:Pascal2:Program:ProdiTest/Seg5

<

HD:Pascal2:Unit5:Unit5/Code

<

HD:Pascal2:Program:ProdiTest/Seg6

<

HD:Pascal2:Unit6:Unit6/Code

<

HD:Pascal2:Program:ProdiTest/Seg7

<

HD:Pascal2:Unit7:Unit7/Code

/Resources

HD:Pascal2:Resources:ProdiTest/Rsrc

/Bundle

/Type 'APPL' 'FGPR'

\$