CARBON en Ada

Carbon est un élément important de Mac OS X. Il propose une compatibilité avec l'interface de programmation (API) des versions antérieures de Mac OS. A partir d'un minimum de modification, Carbon permet de faire fonctionner sous Mac OS X quasiment toutes les applications. Carbon propose des API revues et corrigées : suppression des API peu utilisées ou périmées, modification des API peu ergonomiques ou mal conçues au départ, introduction de nouvelles API.
A travers les modules Carbon en Ada proposés avec gnat-osx, nous allons pouvoir utiliser la puissance de Mac OS X.

I. QuickDraw

QuickDraw est intégré dans la boite à outils du Macintosh comme une bibliothèque de fonctions que vous utiliserez pour effectuer la plupart des manipulations graphiques directes ou indirectes à travers d'autres bibliothèques comme les fenêtres, les menus. La version actuelle provient du Système 7, elle est aussi nommée "32 bits Color QuickDraw". Maintenant Carbon contient une grande partie de l'interface de programmation de QuickDraw. La plupart des déclarations sont incluses dans l'unité "ApplicationServices.qd.QuickDraw".

A. Les coordonnées du système

QuickDraw possède deux systèmes de coordonnées qui permettent de positionner des points sur l'écran : le système de coordonnées global et le système de coordonnées local. Chacun des systèmes mesure en pixel (l'affichage contient 72 pixels par pouces) depuis le point d'origine de coordonnées (0, 0) sur une dynamique allant de -32768 à 32767 pixels. Comme d'ordinaire sur un ordinateur, les coordonnées globales ont leur origine au niveau du coin supérieur gauche de l'écran avec un axe horizontal positif vers la droite et un axe vertical positif vers le bas.
Sur la figure ci-dessous le pixel est défini par sa coordonnée horizontale puis par sa coordonnée verticale:

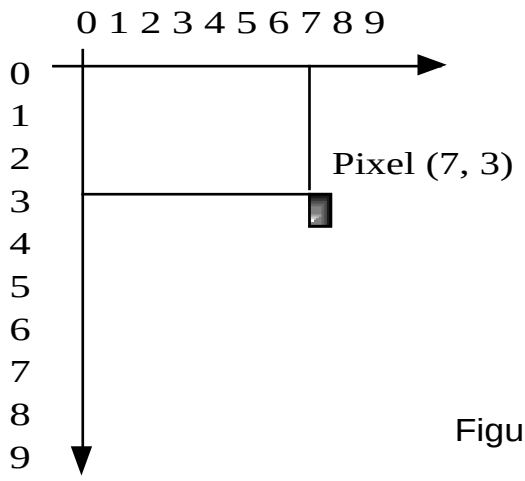

Figure 1

Le pixel se situe à droite et en dessous de l'intersection des lignes de coordonnées (7, 3).

QuickDraw déclare à travers l'unité "CoreServices.Carboncore.MacTypes" le type "Point" comme ceci :

```
type Point is
   record
      v : Short_Integer;
      h : Short_Integer;
   end record;
```

En fait un point possède plus de deux coordonnées dans un espace mathématique. Ici QuickDraw assimile un point à un pixel qui lui possède bien deux coordonnées sur l'écran.
Cependant, QuickDraw permet de définir des espaces graphiques indépendants qui possèdent chacun leur système de coordonnées. Ces espaces deviendront bien facilement des fenêtres ou autres objets graphiques. Cet espace graphique "GrafPort" à son origine au coin supérieur gauche, un axe horizontal vers la droite et un axe vertical vers le bas.
Sur la figure ci-dessous le pixel de cet espace graphique possède deux systèmes de coordonnées : un dans le système global (écran) et un dans le système local (espace graphique):

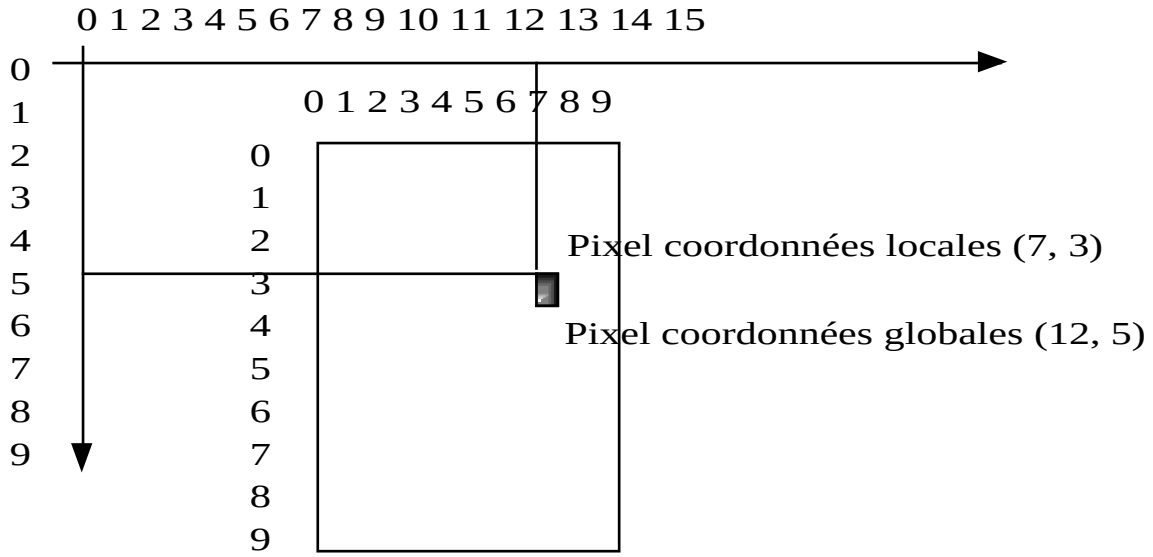

Figure 2

Pour convertir les coordonnées d'un système à l'autre QuickDraw fournit deux fonctions :

procedure LocalToGlobal (pt : access Point);
procedure GlobalToLocal (pt : access Point);

La conversion est opérationnelle si un espace graphique différent de l'écran est sélectionné.

Les valeurs peuvent très bien être négatives. Cela peut signifier d'une part que vous êtes en dehors de l'écran principal mais peut être sur un écran secondaire et d'autre part que vous êtes en dehors de l'espace affiché défini.

Routines de manipulation :
Initialisation :

```
procedure SetPt (
    pt : access Point;
    h  :       Short_Integer;
    v  :       Short_Integer  );
```

Addition :

```
procedure AddPt (
    src :       Point;
    dst : access Point  );
```

Soustraction :

```
procedure SubPt (
    src :       Point;
    dst : access Point  );
```

Test d'égalité:

```
function EqualPt (
    pt1 : Point;
    pt2 : Point  )
    return cBoolean;
```

B. Les rectangles

QuickDraw déclare à travers l'unité "CoreServices.Carboncore.MacTypes" le type "Rect" comme ceci :

```
type Rect is
  record
    top    : Short_Integer;
    left   : Short_Integer;
    bottom : Short_Integer;
    right  : Short_Integer;
  end record;
```

Les quatre valeurs sont respectivement les coordonnées du point en haut à gauche et du point en bas à droite de l'écran.

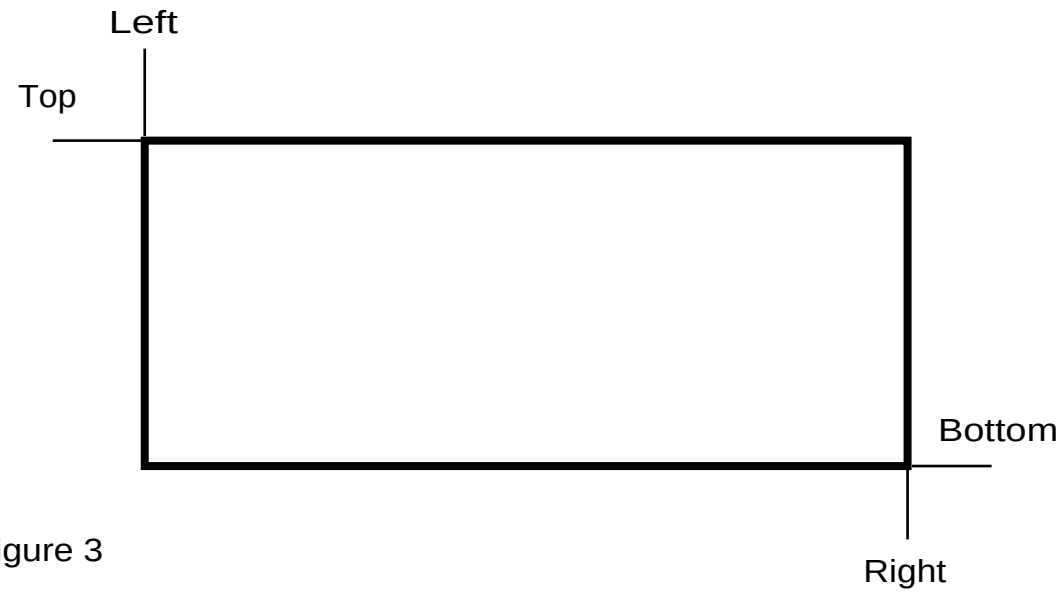

Figure 3

Routines de manipulation :
Initialisation :

```
procedure SetRect (
   r      : access Rect;
   left   :       Short_Integer;
   top    :       Short_Integer;
   right  :       Short_Integer;
   bottom :       Short_Integer  );
```

Décalage :

```
procedure OffsetRect (
   r  : access Rect;
   dH :       Short_Integer;
   dV :       Short_Integer  );
```

Changement de taille :

```
procedure insetRect (
    r  : access Rect;
    dH :        Short_Integer;
    dV :        Short_Integer  );
```

Intersection :

```
function SectRect (
    src1    : access Rect;
    src2    : access Rect;
    dstRect : access Rect  )
    return cBoolean;
```

Union :

```
procedure UnionRect (
    src1    : access Rect;
    src2    : access Rect;
    dstRect : access Rect  );
```

Test égalité :

```
function EqualRect (
    rect1 : access Rect;
    rect2 : access Rect  )
    return cBoolean;
```

Test rectangle vide :

```
function EmptyRect (
    r : access Rect )
    return cBoolean;
```

Test l'inclusion du point :

```
function PtInRect (
    pt :        Point;
    r  : access Rect   )
    return cBoolean;
```

Renvoi le plus petit rectangle défini par les deux points :

```
procedure Pt2Rect (
    pt1     :       Point;
    pt2     :       Point;
    dstRect : access Rect   );
```

Renvoi l'angle (en degré, sens horaire) entre le centre du rectangle et le point :

```
procedure PtToAngle (
    r     : access Rect;
    pt    :        Point;
    angle : access Short_Integer );
```

C. Les régions

Une région est un autre type graphique abstrait d'une forme quelconque, sur laquelle nous pouvons effectuer les mêmes opérations que pour un rectangle. QuickDraw déclare à travers l'unité "ApplicationServices.qd.QuickDraw" le type "MacRegion" comme ceci :

```
type MacRegion is
   record
      rgnSize : UInt16; --  size in bytes; don't rely on it
      rgnBBox : Rect;
      --  enclosing rectangle; in Carbon use GetRegionBounds
   end record;
type RgnPtr is access MacRegion;
type RgnHandle is access RgnPtr;
```

En réalité seul le type RgnHandle nous permet de manipuler les régions. Une fois une région crée et ouverte tout tracé est inclut dans la région (il ne s'affiche plus dans l'espace graphique courant). La région est alors fermée pour terminer l'apprentissage.

Routines de manipulations :
Création :

```
function NewRgn return RgnHandle;
```

Ouverture :

```
procedure OpenRgn;
```

Fermeture :

```
procedure CloseRgn (
   dstRgn : RgnHandle );
```

Destruction :

```
procedure DisposeRgn (
   rgn : RgnHandle );
```

Copie :

```
procedure CopyRgn (
   srcRgn : RgnHandle;
   dstRgn : RgnHandle  );
```

Vide :

```
procedure SetEmptyRgn (
   rgn : RgnHandle );
```

Rectangle :

```
procedure SetRectRgn (
   rgn    : RgnHandle;
   left   : Short_Integer;
   top    : Short_Integer;
   right  : Short_Integer;
   bottom : Short_Integer  );
```

Ou :
procedure RectRgn (
rgn : RgnHandle;
r : access Rect);
Déplacement :
procedure OffsetRgn (
rgn : RgnHandle;
dH : Short_Integer;
dV : Short_Integer);
Changement de taille :
procedure InsetRgn (
rgn : RgnHandle;
dH : Short_Integer;
dV : Short_Integer);
Intersection :
procedure SectRgn (
srcRgnA : RgnHandle;
srcRgnB : RgnHandle;
dstRgn : RgnHandle);
Union :
procedure UnionRgn (
srcRgnA : RgnHandle;
srcRgnB : RgnHandle;
dstRgn : RgnHandle);
Différence :
procedure DiffRgn (
srcRgnA : RgnHandle;
srcRgnB : RgnHandle;
dstRgn : RgnHandle);
Différence entre l'union et l'intersection :
procedure XorRgn (
srcRgnA : RgnHandle;
srcRgnB : RgnHandle;
dstRgn : RgnHandle);
Test de l'inclusion d'un rectangle :
function RectInRgn (
r : access Rect;
rgn : RgnHandle)
return cBoolean;
Test d'égalité :
function EqualRgn (
rgnA : RgnHandle;
rgnB : RgnHandle)
return cBoolean;

Test vide :
```
function EmptyRgn (
   rgn : RgnHandle )
   return cBoolean;
```
Test de l'inclusion d'un point :
```
function PtInRgn (
   pt  : Point;
   rgn : RgnHandle )
   return cBoolean;
```

D. Les polygones

Une région est un autre type graphique abstrait d'une forme quelconque basé sur des segments de droite, sur laquelle nous pouvons effectuer les mêmes opérations que pour un rectangle.
QuickDraw déclare à travers l'unité "ApplicationServices.qd.QuickDraw" le type "MacPolygon" comme ceci :

```
type Polyarray is array (0 .. 0) of Point;
type MacPolygon is
   record
      polySize   : Short_Integer;
      polyBBox   : Rect;
      polyPoints : Polyarray;
   end record;
type PolyPtr is access MacPolygon;
type PolyHandle is access PolyPtr;
```

Remarque : le type Polyarray ne permet d'accéder qu'au premier élément du polygone. Ada n'aimant pas les structures non déterministes. Cela n'est pas gênant outre mesure car nous n'aurons pas besoin d'accéder directement à cette structure.

En réalité seul le type PolyHandle nous permet de manipuler les polygones. Une fois un polygone créé et ouvert, tout tracé (MoveTo, LineTo, Line) est inclut dans le polygone (il ne s'affiche plus dans l'espace graphique courant). Le polygone est alors fermé pour terminer l'apprentissage.

Routines de manipulations :
Création et ouverture :
```
function OpenPoly return PolyHandle;
```
Fermeture :
```
procedure ClosePoly;
```
Destruction :
```
procedure KillPoly (
   poly : PolyHandle );
```

Déplacement :
procedure OffsetPoly (
poly : PolyHandle;
dH : Short_Integer;
dV : Short_Integer);

E. La couleur

QuickDraw propose comme modèle de base pour la gestion de la couleur, le modèle dit RGB (Red, Green, Blue) ou RVB (Rouge, Vert, Bleu) en français.
QuickDraw déclare à travers l'unité "ApplicationServices.qd.QuickDraw" le type "RGBColor" comme ceci :

```
type RGBColor is
   record
      red   : UInt16; -- magnitude of red component
      green : UInt16; -- magnitude of green component
      blue  : UInt16; -- magnitude of blue component
   end record;
```

Cela permet de définir une couleur par synthèse additive de l'amplitude des couleurs fondamentales rouge, verte et bleue. Ce choix correspond en fait à la technologie employée dans un téléviseur ou un moniteur couleur pour ordinateur. Par contre nous pouvons définir beaucoup plus de couleurs que peut en créer aujourd'hui le dispositifs graphique des Macintosh : 2^{48} couleurs possibles contre au maximum 2^{24} soit 16 millions de fois plus de couleur. Pour cela QuickDraw se charge de trouver la meilleur combinaison correspondant à la couleur voulue avec le dispositif graphique de l'ordinateur utilisé.

Définition de la couleur de dessin :
procedure RGBForeColor (
color : access RGBColor);
Définition de la couleur de fond :
procedure RGBBackColor (
color : access RGBColor);
Renvoie la couleur de dessin :
procedure GetForeColor (
color : access RGBColor);
Renvoie la couleur de fond :
procedure GetBackColor (
color : access RGBColor);

Plus facilement QuickDraw permet d'utiliser une couleur pré-définie :

```
blackColor   : constant := 33;
whiteColor   : constant := 30;
redColor     : constant := 205;
greenColor   : constant := 341;
blueColor    : constant := 409;
cyanColor    : constant := 273;
magentaColor : constant := 137;
yellowColor  : constant := 69;
```

Définition de la couleur de dessin :

```
procedure foreColor (
   color : Long_Integer );
```

Définition de la couleur de fond :

```
procedure backColor (
   color : Long_Integer );
```

F. Le crayon

Chaque espace graphique possède son propre crayon. QuickDraw utilise celui-ci pour dessiner toutes les formes utiles. Le crayon est caractérisé par sa position, sa taille, le mode d'écriture, le motif utilisé et l'indice de visibilité.

QuickDraw définit une structure de donnée associé au crayon :

```
type PenState is
   record
      pnLoc  : Point;
      pnSize : Point;
      pnMode : Short_Integer;
      pnPat  : pattern;
   end record;
```

Routines de manipulation :

Récupère l'état du crayon :

```
procedure GetPenState ( pnState : access PenState );
```

Positionne l'état du crayon :

```
procedure SetPenState ( pnState : access PenState );
```

Positionne l'état standard (taille : 1*1, mode d'écriture : copie, motif : noir) :

```
procedure PenNormal;
```

1. La position

Sa position est donnée en coordonnées locales de l'espace graphique.
Récupère la position :

```
procedure GetPen ( pt : access Point );
```

Fixe la position :

```
procedure MoveTo (
   h : Short_Integer;
   v : Short_Integer  );
```

Déplace le crayon :

```
procedure Move (
   dH : Short_Integer;
   dV : Short_Integer  );
```

2. La taille

La taille est donnée en nombre de pixels horizontaux et verticaux.
Fixe la taille :

```
procedure PenSize (
   width  : Short_Integer;
   height : Short_Integer  );
```

3. Le mode d'écriture

Le mode d'écriture indique l'opération effectuée au passage du crayon sur l'écran.
Les modes sont les suivants :

```
   patCopy          : constant := 8;
   patOr            : constant := 9;
   patXor           : constant := 10;
   patBic           : constant := 11;
   notPatCopy       : constant := 12;
   notPatOr         : constant := 13;
   notPatXor        : constant := 14;
   notPatBic        : constant := 15;
   blend            : constant := 32;
   addPin           : constant := 33;
   addOver          : constant := 34;
   subPin           : constant := 35;
   addMax           : constant := 37;
   adMax            : constant := 37;
   subOver          : constant := 38;
   adMin            : constant := 39;
   ditherCopy       : constant := 64;
   transparent      : constant := 36;
```

Nous devons considérer que chaque pixel du motif du crayon est un élément booléen sur lequel on applique une opération booléenne avec les pixels de l'écran.
patCopy : provoque le recouvrement du motif du crayon sur le fond,
patOr : provoque un "ou" logique entre le motif du crayon et du fond,
patXor : provoque un "ou" exclusif entre le motif du crayon et du fond,
patBic : provoque un "et" logique entre le motif du crayon et du fond,
notPatCopy : provoque le recouvrement de l'inverse du motif du crayon sur le fond,
notPatOr : provoque un "ou" logique entre l'inverse du motif du crayon et du fond,
notPatXor : provoque un "ou" exclusif entre l'inverse du motif du crayon et du fond,
notPatBic : provoque un "et" logique entre l'inverse du motif du crayon et du fond.

Nous devons considérer que chaque couleur des pixels du motif du crayon est une valeur sur laquelle on applique une opération avec les couleurs de l'écran.
blend : provoque une moyenne des couleurs,
addPin : provoque la somme des couleurs limitée à la valeur maximale,
addOver : provoque la somme des couleurs modulo la valeur maximale,
subPin : provoque la différence entre les couleurs limitée à la valeur minimale,
addMax, adMax : provoque la prise en compteur de la plus forte amplitude,
subOver : provoque la différence entre les couleurs modulo la valeur maximale,
adMin : provoque la prise en compte de la plus faible amplitude,
ditherCopy : provoque le mixage des couleurs,
transparent: ne prend pas en compte les pixels de même couleur que le fond.

Fixe le mode d'écriture :
procedure PenMode (
mode : Short_Integer);

4. Le motif utilisé

Fixe le motif noir et blanc du crayon :
procedure PenPat (
pat : access pattern);
Fixe le motif couleur du crayon :
procedure PenPixPat (
pp : PixPatHandle);

5. L'indice de visibilité

L'indice de visibilité s'incrémente pour être visible ou se décrémente pour être invisible. Pratiquement seule les valeurs positive ou nulle permettent au crayon de laisser une trace. Au départ l'indice a la valeur 0.

Décrémente la visibilité :
procedure HidePen;
Incrémente la visibilité :
procedure ShowPen;

G. Les figures

QuickDraw fournit un ensemble de routines pour dessiner toutes sortes de figures :

- points “Pixel”,
- lignes “Line”,
- rectangles “Rect”,
- ovales “Oval”,
- rectangles aux angles arrondis “RoundRect”,
- arcs “Arc”,
- polygones “Poly”,
- régions “Rgn”.

Les figures sont tracées sur l’espace graphique courant avec les caractéristiques courantes du crayon.

Tracé d’un point :
Appel de Line (0, 0);
Tracé d’un point en couleur:

```
procedure SetCPixel (
   h    :       Short_Integer;
   v    :       Short_Integer;
   cPix : access RGBColor      );
```

Test d’un point noir :

```
function GetPixel (
   h : Short_Integer;
   v : Short_Integer  )
   return cBoolean;
```

Renvoie la couleur d’un point :

```
procedure GetCPixel (
   h    :       Short_Integer;
   v    :       Short_Integer;
   cPix : access RGBColor      );
```

Tracé d’une ligne à partir du point courant jusqu’à la position définie :

```
procedure LineTo (
   h : Short_Integer;
   v : Short_Integer  );
```

Tracé d'une ligne par déplacement du point courant :

```
procedure Line (
   dH : Short_Integer;
   dV : Short_Integer  );
```

Mis à part les points et les lignes, nous pouvons effectuer sur chacune des figures un certains nombres d'opérations :

- dessin "Frame",
- remplissage avec le motif du crayon "Paint",
- remplissage avec le motif du fond "Erase",
- inversion des pixels "Invert",
- remplissage avec le motif noir et blanc fourni "Fill",
- remplissage avec le motif couleur fourni "FillC".

Ces opérations s'inscrivent toutes dans un rectangle support.

Les opérations combinées :

```
procedure FrameRect (
   r : access Rect );
procedure PaintRect (
   r : access Rect );
procedure EraseRect (
   r : access Rect );
procedure InvertRect (
   r : access Rect );
procedure FillRect (
   r   : access Rect;
   pat : access pattern );
procedure FillCRect (
   r  : access Rect;
   pp :       PixPatHandle );

procedure FrameOval (
   r : access Rect );
procedure PaintOval (
   r : access Rect );
procedure EraseOval (
   r : access Rect );
procedure InvertOval (
   r : access Rect );
procedure FillOval (
   r   : access Rect;
   pat : access pattern );
procedure FillCOval (
   r  : access Rect;
   pp :       PixPatHandle );
```

```
procedure FrameRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer  );
procedure PaintRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer  );
procedure EraseRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer  );
procedure InvertRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer  );
procedure FillRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer;
    pat        : access pattern        );
procedure FillCRoundRect (
    r          : access Rect;
    ovalWidth  :        Short_Integer;
    ovalHeight :        Short_Integer;
    pp         :        PixPatHandle   );

procedure FrameArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer  );
procedure PaintArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer  );
procedure EraseArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer  );
procedure InvertArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer  );
```

```
procedure FillArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer;
    pat        : access pattern        );
procedure FillCArc (
    r          : access Rect;
    startAngle :        Short_Integer;
    arcAngle   :        Short_Integer;
    pp         :        PixPatHandle   );

procedure FramePoly (
    poly : PolyHandle );
procedure PaintPoly (
    poly : PolyHandle );
procedure ErasePoly (
    poly : PolyHandle );
procedure InvertPoly (
    poly : PolyHandle );
procedure FillPoly (
    poly :        PolyHandle;
    pat  : access pattern     );
procedure FillCPoly (
    poly : PolyHandle;
    pp   : PixPatHandle );

procedure FrameRgn (
    rgn : RgnHandle );
procedure PaintRgn (
    rgn : RgnHandle );
procedure EraseRgn (
    rgn : RgnHandle );
procedure InvertRgn (
    rgn : RgnHandle );
procedure FillRgn (
    rgn :        RgnHandle;
    pat : access pattern    );
procedure FillCRgn (
    rgn : RgnHandle;
    pp  : PixPatHandle );
```

H. Les motifs

QuickDraw fournit des structures de motifs pour le dessin et le remplissage des des formes.

Deux types de motifs sont à disposition, d'une part les motifs noir et blanc "Pattern" et les motifs couleurs "PixPat" définis à travers l'unité "ApplicationServices.qd.QuickDraw" comme ceci :

```
type Patternarray is array (0 .. 7) of UInt8;
type pattern is
  record
    pat : Patternarray;
  end record;
type PatternPtr is access pattern;
type PatPtr is access pattern;
type PatHandle is access PatPtr;

type pixPat is
  record
    pATType   : Short_Integer; -- type of pattern
    patMap    : PixMapHandle;  -- the pattern's pixMap
    patData   : Handle;        -- pixmap's data
    patXData  : Handle;        -- expanded Pattern data
    patXValid : Short_Integer;
    -- flags whether expanded Pattern valid
    patXMap : Handle;
    -- Handle to expanded Pattern data
    pat1Data : pattern; -- old-Style pattern/RGB color
  end record;
type PixPatPtr is access pixPat;
type PixPatHandle is access PixPatPtr;
```

Un motif du crayon est défini par une grille de 8 par 8 pixels.
Un certain nombre de motifs sont pré-définis dans la structure interne de QuickDraw.

Récupère les motifs noir et blanc pré-définis :

```
function GetQDGlobalsDarkGray (
  dkGray : access pattern )
  return PatternPtr;
function GetQDGlobalsLightGray (
  ltGray : access pattern )
  return PatternPtr;
```

function GetQDGlobalsGray (
gray : access pattern)
return PatternPtr;
function GetQDGlobalsBlack (
black : access pattern)
return PatternPtr;
function GetQDGlobalsWhite (
white : access pattern)
return PatternPtr;

Des motifs personnalisés peuvent être utilisés à partir de ressources.
En réalité ces structures ne sont pas à utiliser directement mais au travers de ressources (voir le chapitre sur les ressources- à venir) :

Récupère le motif noir et blanc désigné en ressource :
function GetPattern (
patternID : Short_Integer)
return PatHandle;
Récupère le motif couleur désigné en ressource :
function GetPixPat (
patID : Short_Integer)
return PixPatHandle;

I. Le texte

Le texte dans QuickDraw pourrait faire partie des formes de dessin car il ne s'agit pas de texte éditable, juste un affichage à l'écran. Le texte éditable est vu au paragraphe édition de texte (à venir). Par contre l'affichage par QuickDraw bénéficie de tout les paramétrages de circonstance : la police des caractères, le style, le mode d'écriture, la taille.

Les routines utilitaires sont définies par l'unité ApplicationServices.QD.QuickdrawText.

1. La police

Établit la police utilisée pour l'affichage :
procedure TextFont (
font : Short_Integer);
Dans Carbon *TextFont* s'utilise avec *FMGetFontFamilyFromName* défini dans l'unité *ApplicationServices.QD.Fonts*. Cette fonction retourne un identificateur de police sur la base d'une chaîne de caractère représentant le nom d'une police, par exemple :
TextFont(Short_Integer(FMGetFontFamilyFromName(To_Pascal("Chicago"))));

2. Le style

QuickDraw déclare à travers l'unité "CoreServices.Carboncore.MacTypes" le type "StyleParameter" comme ceci :

```
subtype style is Short_Integer;
subtype StyleParameter is style;
```

C'est à l'utilisateur de définir les constantes correspondantes aux différents styles utilisables :

```
normal          := 0 -- normal,
bold            := 1 -- gras,
italic          := 2 -- italique,
underline       := 4 -- souligné,
outline         := 8 -- en relief,
shadow          := 16#10# -- ombré,
condense        := 16#20# -- condensé,
extend          := 16#40# -- élargi.
```

Les styles sont tous combinables les uns aux autres.
Établit le style utilisé pour l'affichage

```
procedure textFace (
    face : StyleParameter );
```

3. Le mode d'écriture

Le mode d'écriture indique l'opération effectuée par l'affichage sur l'écran.
Les modes sont les suivants :

```
--  transfer modes
srcCopy          : constant := 0;
srcOr            : constant := 1;
srcXor           : constant := 2;
srcBic           : constant := 3;
notSrcCopy       : constant := 4;
notSrcOr         : constant := 5;
notSrcXor        : constant := 6;
notSrcBic        : constant := 7;
--  Special Text Transfer Mode
grayishTextOr    : constant := 49;
hilitetransfermode : constant := 50;
hilite           : constant := 50;
blend            : constant := 32;
addPin           : constant := 33;
addOver          : constant := 34;
subPin           : constant := 35;
addMax           : constant := 37;
```

```
adMax          : constant := 37;
subOver        : constant := 38;
adMin          : constant := 39;
ditherCopy     : constant := 64;
transparent    : constant := 36;
```

Nous devons considérer que chaque pixel des caractères est un élément booléen sur lequel on applique une opération booléenne avec les pixels de l'écran.
srcCopy : provoque le recouvrement du caractère sur le fond,
srcOr : provoque un "ou" logique entre le caractère et du fond,
srcXor : provoque un "ou" exclusif entre le caractère et du fond,
srcBic : provoque un "et" logique entre le caractère et du fond,
notSrcCopy : provoque le recouvrement de l'inverse du caractère sur le fond,
notSrcOr : provoque un "ou" logique entre l'inverse du caractère et du fond,
notSrcXor : provoque un "ou" exclusif entre l'inverse du caractère et du fond,
notSrcBic : provoque un "et" logique entre l'inverse du caractère et du fond.
grayishTextOr : provoque l'affichage du text en grisé, à n'utiliser que sur écran,
hilitetransfermode ou
hilite : provoque le remplacement du la couleur de fond des caractères par la couleur de surlignage.

Nous devons considérer que chaque couleur des caractères est une valeur sur laquelle on applique une opération avec les couleurs de l'écran.
blend : provoque une moyenne des couleurs,
addPin : provoque la somme des couleurs limitée à la valeur maximale,
addOver : provoque la somme des couleurs modulo la valeur maximale,
subPin : provoque la différence entre les couleurs limitée à la valeur minimale,
addMax, adMax : provoque la prise en compteur de la plus forte amplitude,
subOver : provoque la différence entre les couleurs modulo la valeur maximale,
adMin : provoque la prise en compte de la plus faible amplitude,
ditherCopy : provoque le mixage des couleurs,
transparent: ne prend pas en compte les pixels de même couleur que le fond.

Fixe le mode d'écriture :

```
procedure TextMode (
    mode : Short_Integer );
```

4. La taille

La taille est indiquée en nombre de points (0 à 32767), le point valant 1/72^{ième} de pouce. Pour un écran dont la résolution est de 72 pixels par pouce, le point vaut un pixel. La valeur initiale 0 spécifie la taille de la police du système (12 points). Si la taille spécifiée n'est pas définie par la police, chaque caractère est calculé à partir de la taille de police existante la plus proche.

Établit la police utilisée pour l'affichage :

procedure TextSize (
Size : Short_Integer);

5. L'affichage

L'affichage débute à la position courante du crayon :
La position du crayon est déplacée de la largeur du caractère.
Le texte est affiché avec la couleur du crayon.

QuickDraw utilise les types suivants à travers l'unité "CoreServices.Carboncore.MacTypes" :

Caractère :

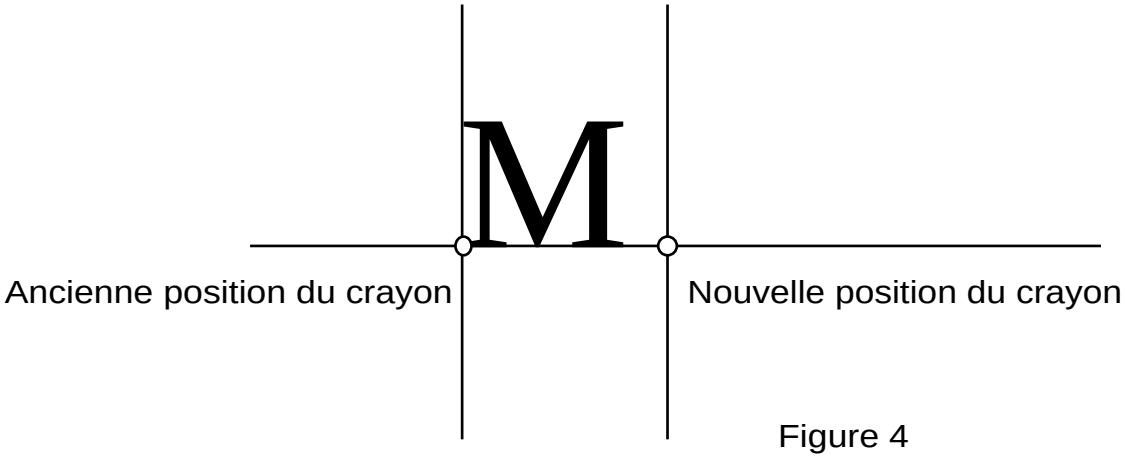

Figure 4

subtype CharParameter is Character;

Chaîne de caractères :

type pString is array (Natural range <>) of aliased Character;
subtype Str255 is pString (0 .. 255);

Conversion d'une chaîne de caractères Ada en chaîne de caractères Pascal - taille de la chaîne dans le premier caractère, limitée à 255 caractères (fonction déclarée dans l'unité "PStrings") :

function To_Pascal (item : in String) return Str255;

Tampon de caractères :

type Void_Ptr is access all Long_Integer;
function Convert is new

Conversion de pointeur :

Unchecked_Conversion(Integer, MacTypes.Void_Ptr);

Pointeur nul :

Nulvoidptr : MacTypes.Void_Ptr:= Convert(0);

Affiche un caractère :

procedure DrawChar (
ch : CharParameter);

Affiche une chaîne de caractères :

```
procedure DrawString (
    s : Str255 );
```

Affiche un tampon de caractères :

```
procedure DrawText (
    textBuf   : Void_Ptr;
    firstByte : Short_Integer;
    ByteCount : Short_Integer  );
```

6. La largeur

Les largeurs sont retournées en nombre de pixels. C'est à dire la valeur du déplacement du crayon si le caractère était dessiné.

Retourne la largeur d'un caractère :

```
function CharWidth (
    ch : CharParameter )
    return Short_Integer;
```

Retourne la largeur d'une chaîne de caractères :

```
function StringWidth (
    s : Str255 )
    return Short_Integer;
```

Retourne la largeur d'un tampon de caractères :

```
function TextWidth (
    textBuf   : Void_Ptr;
    firstByte : Short_Integer;
    ByteCount : Short_Integer  )
    return Short_Integer;
```

7. Le surlignage

Établit la couleur utilisée pour le surlignage :

```
procedure hiliteColor (
    color : access RGBColor );
```

J. L'espace graphique

L'espace graphique ou port graphique (GrafPort) est la structure support de tout dessin. Elle enregistre les différents réglages du crayon, des motifs, de la couleur, du texte. Elle contient aussi le plan mémoire support des tracés.

Deux types de ports graphiques sont à disposition, d'une part le port graphique noir et blanc "GrafPort" - celui ci n'est plus valide avec Carbon - et le port graphique couleur "CGrafPort" défini à travers l'unité "ApplicationServices.qd.QuickDraw" comme ceci :

```
type CGrafPort is
   record
      device     : Short_Integer; --  not available in Carbon
      portPixMap : PixMapHandle;
      --  in Carbon use GetPortPixMap
      portVersion : Short_Integer; --  in Carbon use IsPortColor
      GrafVars    : Handle;        --  not available in Carbon
      chExtra     : Short_Integer;
      --  in Carbon use GetPortChExtra
      pnLocHFrac : Short_Integer;
      --  in Carbon use Get/SetPortFracHPenLocation
      portRect : Rect;
      --  in Carbon use Get/SetPortBounds
      visRgn : RgnHandle;
      --  in Carbon use Get/SetPortVisibleRegion
      clipRgn : RgnHandle;
      --  in Carbon use Get/SetPortClipRegion
      bkPixPat : PixPatHandle;
      --  in Carbon use GetPortBackPixPat or BackPixPat
      rgbFgColor : RGBColor;
      --  in Carbon use GetPortForeColor or RGBForeColor
      rgbBkColor : RGBColor;
      --  in Carbon use GetPortBackColor or RGBBackColor
      pnLoc : Point;
      --  in Carbon use GetPortPenLocation or MoveTo
      pnSize : Point;
      --  in Carbon use Get/SetPortPenSize
      pnMode : Short_Integer;
      --  in Carbon use Get/SetPortPenMode
      pnPixPat : PixPatHandle;
      --  in Carbon use Get/SetPortPenPixPat
      fillPixPat : PixPatHandle;
      --  in Carbon use GetPortFillPixPat
      pnVis : Short_Integer;
      --  in Carbon use GetPortPenVisibility or Show/HidePen
      txFont : Short_Integer;
      --  in Carbon use GetPortTextFont or TextFont
      txFace : StyleField;
      --  in Carbon use GetPortTextFace or TextFace
      -- StyleField occupies 16-bits, but only first 8-bits are used
      txMode : Short_Integer;
      --  in Carbon use GetPortTextMode or TextMode
      txSize : Short_Integer;
      --  in Carbon use GetPortTextSize or TextSize
      spExtra : Fixed;
      --  in Carbon use GetPortSpExtra or SpaceExtra
```

```
      fgColor    : Long_Integer;  --  not available in Carbon
      bkColor    : Long_Integer;  --  not available in Carbon
      colrBit    : Short_Integer; --  not available in Carbon
      patStretch : Short_Integer; --  not available in Carbon
      picSave    : Handle;
      --  in Carbon use IsPortPictureBeingDefined
      rgnSave : Handle;
      --  in Carbon use IsPortRegionBeingDefined
      polySave : Handle;
      --  in Carbon use IsPortPolyBeingDefined
      grafProcs : CQDProcsPtr;
      --  in Carbon use Get/SetPortGrafProcs
   end record;
   pragma pack(CGrafPort);
type CGrafPortPtr is access CGrafPort;
type CGrafPtr is access CGrafPort;
```

Le type du port graphique ne sera jamais manipulé directement mais à travers les routines qui sont indiquées dans le texte du type “CGrafPort”.

Il est tout à fait possible de réaliser des tracés sur des ports graphiques différents. Cela est utile pour réaliser des animations. Cela est même nécessaire pour réaliser un affichage avec de multiples fenêtres. Le dessin s’effectue toujours sur le port graphique courant.

Création d’un port graphique :

```
function CreateNewPort return CGrafPtr;
```

Destruction d’un port graphique :

```
procedure DisposePort (
   port : CGrafPtr );
procedure GetPort (
   port : access GrafPtr );
```

Définit le port graphique courant :

```
procedure SetPort (
   port : CGrafPtr );
```

et

```
procedure SetPort (
   port : GrafPtr );
```

La vocation finale d’un port graphique est d’être rattaché à un dispositif de visualisation qui peut être soit une fenêtre de l’écran soit l’imprimante.
La fonction de liaison avec la fenêtre courante est définie à travers l’unité “Carbon.HIToolbox.MacWindows” comme ceci :

Retourne le port graphique de la fenêtre courante :

```
function GetWindowPort (
   window : WindowRef )
   return CGrafPtr;
```

Remarque : la fonction “GetPort” retourne le type “GrafPtr” et non “CGrafPtr”, il s’agit certainement d’un oubli. Cependant, ces deux fonctions ont pour unique vocation de sauvegarder l’espace graphique courant avant de définir celui de l’application et de le restaurer après, comme sur l’exemple ci dessous :

```
procedure DrawWindow (
      Window : WindowRef ) is
  TempRect :    aliased Rect;
  CurPort :    aliased GrafPtr;
begin
   GetPort(CurPort'access);
  SetPort(GetWindowPort(Window));
  BeginUpdate(Window);
  EraseRect(GetWindowPortBounds(Window, TempRect'access));
  DrawControls(Window);
  DrawGrowIcon(Window);
  DrawContents; -- routine d’affichage du contenu
  EndUpdate(Window);
   SetPort(CurPort);
end;
```

Suite au prochain numéro...

Pascal Pignard, juin 2002.