QTilities ([QT Utilities]) & POSIXm2
-
un début de mode d'emploi

QTilities est un ensemble d'outils pour exploiter la fonctionnalité fournie par Apple QuickTime(tm). Il est composé en deux parties plus une librairie auxiliaire:

QTils : un mini 'toolkit' permettant d'exploiter (ouvrir, enregistrer et visualiser, …) des fichiers multimédia ainsi que d'accéder à certaines de leurs méta-données. Il donne également accès aux fonctions XML de QuickTime.

QTMovieSink : un autre 'toolkit' permettant de créer des vidéos (".mov") à partir de flux d'images.

POSIXm2 : un petit nombre de fonctions *POSIX* et de l'univers **C** plus général.

Une description de la procédure d'installation est en fin de document.

QTils

QTils prend la forme d'une dll (QTils.dll) avec son fichier en-tête (QTilities.h), associé à une interface pour Modula-2 (QTilsM2.mod et QTilsM2.def). Les fonctions exportées se divisent en quatre groupes: celles pour la visualisation/lecture des vidéos; celles pour gérer des actions QuickTime; celles pour manipuler d'objets (fichiers) vidéo et des fonctions de support. L'*API* QuickTime est encapsulée par la dll[1]: il n'y a pas besoin d'avoir le *SDK* QuickTime installé pour utiliser QTils.

Il est à noter qu'un "objet vidéo" (*Movie*) peut être obtenu à partir d'un fichier sur disque mais aussi à partir d'un serveur via une adresse http. On parlera donc d'*URL* au lieu de (nom de) fichier.

Il est également à noter que l'interface (API) Modula-2 est largement identique à l'API C/C++ (mêmes noms de fonctions et leurs arguments); les fonctions sont toutes contenues dans le tableau (*RECORD*) QTils. Un certain nombre de fonctions retournent des pointeurs vers des types *ARRAY[0..n] OF CHAR* (valeur de retour ou via un argument *VAR*); il s'agit en réalité de pointeurs vers des chaines de caractères de style C/C++ dont il ne faut pas présumer que la taille équivaut *n*. Le membre *theURL* des *QTMovieWindowH* est un autre exemple.

La plupart des fonctions adhèrent à la convention d'appel de QuickTime: elles retournent un code d'erreur, ou *noErr* (0) en cas de succès. Elles retournent donc leurs résultat(s) via des pointers (ou *VAR* en Modula-2).

Initialisation

QuickTime nécessite une initialisation pour être utilisable. Le module Modula-2 s'initialise automatiquement au démarrage du logiciel qui l'exploite, et cette initialisation inclut celle de QuickTime. Dans l'API C/C++, il faudra appeler *OpenQT()* avant chaque autre appel, et *CloseQT()* avant de quitter le logiciel.

```
#include "QTilities.h"

QTLSext ErrCode OpenQT();
QTLSext void CloseQT();
PROCEDURE OpenQT() : ErrCode;
```

[1] Cette notice décrit QTils dans le contexte MS Windows, mais QTils existe également sous Mac OS X, dans la forme un *framework* (C/C++/ObjC seulement).

(Voir les Notes pour la définition de *QTLSext*.)
En Modula-2, les résultats de l'initialisation sont aussi disponibles via deux fonctions; *QTOpened* retourne un *BOOLEAN* indiquant si l'initialisation à réussi ou non; *QTOpenError* retourne le code d'erreur (ou *noErr* en cas de succès). La fonction *QTClose()* est appelée automatiquement à la sortie du programme.

```
FROM QTilsM2 IMPORT *;

PROCEDURE QTilsAvailable() : BOOLEAN;
PROCEDURE QTOpened() : BOOLEAN;
PROCEDURE QTOpenError() : ErrCode;

CONST
  noErr = 0;
```

Fonctions de lecture/visualisation

Les fonctions permettant d'ouvrir un *Movie* pour visualisation/lecture utilisent un objet abstrait pour faire référence au *Movie* et à la fenêtre dans laquelle il est affiché. Cet objet est le *QTMovieWindowH*, un *handle* (pointer vers pointeur) sur QTMovieWindows. Les principaux types définis dans QTilities.h (QTilsM2.def):

C/C++	Modula-2
	`StringPtr = POINTER TO ARRAY OF CHAR;`
`typedef HWND NativeWindow;`	`NativeWindow = HWND;`
`typedef struct MovieType** Movie;`	`Movie = POINTER TO ADDRESS;`
`typedef struct QTMovieWindows {`	`QTMovieWindows= RECORD`
`  struct QTMovieWindows *self;`	`  self : POINTER TO QTMovieWindows;`
`  NativeWindow theView,`	`  theView : NativeWindow;`
`    *theViewPtr;`	`  theViewPtr : NativeWindowPtr;`
`  const char *theURL;`	`  theURL : StringPtr;`
`  int idx;`	`  idx : Int32;`
`  int isPlaying, isActive;`	`  isPlaying, isActive : Int32;`
`  long loadState;`	`  loadState : Int32;`
`  int wasStepped, wasScanned;`	`  wasStepped, wasScanned : Int32;`
`  int handlingEvent, shouldClose;`	`  handlingEvent, shouldClose : Int32;`
`  Movie theMovie;`	`  theMovie : Movie;`
`  QTMovieInfo *info;`	`  info : POINTER TO QTMovieInfo;`
`  void *user_data;`	`  user_data : ADDRESS;`
`  // ... private section ...`	`  (* section interne *)`
`} QTMovieWindows;`	`END;`
`typedef struct QTMovieWindows*`	`QTMovieWindowPtr = POINTER TO`
`      QTMovieWindowPtr;`	`  QTMovieWindows;`
`typedef QTMovieWindowPtr* QTMovieWindowH;`	`QTMovieWindowH= POINTER TO`
	`  QTMovieWindowPtr;`
	`CONST`
	`  NULL_QTMovieWindowH = CAST`
	`(QTMovieWindowH, NULL);`

C/C++	Modula-2

`typedef struct Rect {`
`  short top, left, bottom, right;`
`} Rect;`
	`QTRect    = RECORD`
	`  top, left, bottom, right : Int16;`
	`END;`
	`QTRectPtr  = POINTER TO QTRect;`

`typedef long TimeValue;`
`typedef struct QTMovieInfo {`
`  double frameRate, TCframeRate,`
`   duration,`
`   startTime;`
`  TimeValue timeScale;`
`  long startFrameNr;`
`  Rect naturalBounds;`
`  short controllerHeight;`
`} QTMovieInfo;// packed`
	`TimeValue  = Int32;`
	`QTMovieInfo  = RECORD`
	`  frameRate,  TCframeRate,`
	`   duration,`
	`   startTime : double;`
	`  timeScale  : TimeValue;`
	`  startFrameNr  : Int32;`
	`  naturalBounds   : QTRect;`
	`  controllerHeight : Int16;`
	`END;`

`typedef struct Cartesian {`
`  short horizontal, vertical;`
`} Cartesian;`
	`Cartesian  = RECORD`
	`  horizontal, vertical : Int16;`
	`END;`
	`CartesianPtr = POINTER TO Cartesian;`
	
	`CONST`
	`  NULL_Cartesian = CAST(CartesianPtr,`
	`NULL);`

`typedef struct TimeCodeTime {`
`  unsigned char hours;`
`  unsigned char minutes;`
`  unsigned char seconds;`
`  unsigned char frames;`
`} MovieFrameTime;`
	`MovieFrameTime= RECORD`
	`  hours,`
	`   minutes,`
	`   seconds,`
	`   frames : UInt8;`
	`END;`
	`MovieFrameTimePtr= POINTER TO`
	`MovieFrameTime;`

`typedef UInt16 EventKind;` `typedef unsigned short EventModifiers;`	`EventKind    = UInt16;` `EventModifiers= UInt16;`

`typedef struct EventRecord {`
`  EventKind what;`
`  UInt32 message;`
`  UInt32 when;`
`  Cartesian where;`
`  EventModifiers modifiers;`
`} EventRecord;`
	`EventRecord  = RECORD`
	`   what : EventKind;`
	`   message,`
	`    when : UInt32;`
	`   where : Cartesian;`
	`   modifiers : EventModifiers;`
	`END;`
	`EventRecordPtr= POINTER TO EventRecord;`

Ici, le membre *idx* est un compteur qui est augmenté à chaque ouverture d'un nouveau *QTMovieWindowH*. *theURL* contient le nom de fichier ou l'adresse http de la source de la vidéo. Les membres *isPlaying, isActive, loadState, wasStepped* et *wasScanned* sont des variables d'état dont les 2 dernières indiquent si l'utilisateur vient de faire avancer/reculer la vidéo d'un pas ou d'un saut. *user_data* permet d'associer des informations supplémentaires à chaque *QTMovieWindowH*. Finalement, *info* contient des informations spatio-temporelles, dont la fréquence, la taille "native" de la vidéo ainsi que la hauteur actuelle du contrôleur (0 si ce dernier est invisible). La fréquence est spécifiée également pour la piste TimeCode, si présente, car elle peut être légèrement différente (et est *a priori* une valeur entière).

On obtient un *QTMovieWindowH* via *OpenQTMovieInWindow()*; cette fonction ouvre le *Movie* dont on spécifie l'*URL*; attache un contrôleur si demandé (à conseiller!) et affiche le tout dans une nouvelle fenêtre de la bonne taille. Le contrôleur est une interface graphique affichée sous la vidéo permettant à l'utilisateur de contrôler la lecture de la vidéo ainsi que de redimensionner la fenêtre. La fonction retourne le nouveau *QTMovieWindowH*, ou *NULL* en cas d'échec (appeler *LastQTError()* pour connaître le code d'erreur). NB : il est fortement conseillé d'ouvrir les *QTMovieWindows* sur le fil principal du programme.
La fenêtre peut être fermée via les méthodes standards du système d'exploitation; dans ce cas la fonction *CloseQTMovieWindow()* ferme les différentes ressources utilisées. La manière programmatique pour fermer un *QTMovieWindowH* est via *DisposeQTMovieWindow()* qui s'occupe également de fermer le *handle* (elle est donc à appeler même si la fenêtre a été fermée via une action directe de l'utilisateur). Attention toutefois de ne pas appeler *DisposeQTMovieWindow()* (ni *CloseQTMovieWindow()* d'ailleurs) tant qu'une fenêtre peut encore recevoir des événements.

```
void DisposeQTMovieWindow( QTMovieWindowH WI );
short CloseQTMovieWindow( QTMovieWindowH WI );
QTMovieWindowH OpenQTMovieInWindow( const char *theURL,
        int withController );

PROCEDURE DisposeQTMovieWindow( VAR wih : QTMovieWindowH );
PROCEDURE OpenQTMovieInWindow( theURL : ARRAY OF CHAR,
        withController : Int32 ) : QTMovieWindowH;
```

La version Modula-2 de *DisposeQTMovieWindow()* change la valeur de *wih* en *NULL_QTMovieWindowH*.

```
ErrCode ActivateQTMovieWindow( QTMovieWindowH wih );
```
active la vidéo et affiche la fenêtre devant toutes les autres fenêtres.

```
ErrCode QTMovieWindowSetGeometry( QTMovieWindowH wih, Cartesian *pos,
   Cartesian *size, double sizeScale, int setEnvelope );
PROCEDURE QTMovieWindowSetGeometry( wih : QTMovieWindowH, pos,
   size : CartesianPtr, sizeScale : double, setEnvelope : Int32) : ErrCode;
```
Modifie la position et/ou la taille de *wih*. *pos* et *size* peuvent être *NULL* (*NULL_Cartesian* en Modula-2) pour ne pas être pris en compte; l'échelle *sizeScale* s'applique soit sur la taille actuelle de la fenêtre, soit sur la taille *size*. *setEnvelope* indique si les dimensions prennent en compte la fenêtre entière (l'envelope) ou seulement le contenu (le *Movie* avec son contrôleur le cas échéant). Dans ce dernier cas, la fenêtre est redimensionnée et/ou déplacée de manière à ce que la taille et la position sur l'écran du contenu correspondent aux valeurs spécifiées.

```
ErrCode QTMovieWindowGetGeometry( QTMovieWindowH wih, Cartesian *pos,
      Cartesian  *size, int getEnvelope );
PROCEDURE QTMovieWindowGetGeometry( wih : QTMovieWindowH, pos : CartesianPtr,
      size : CartesianPtr, getEnvelope : Int32) : ErrCode;
```
Obtient position et/ou taille de *wih*. de la façon discutée pour *QTMovieWindowSetGeometry()*.

```
ErrCode QTMovieWindowPlay( QTMovieWindowH wih );
ErrCode QTMovieWindowStop( QTMovieWindowH wih );
PROCEDURE QTMovieWindowPlay( wih : QTMovieWindowH ) : ErrCode;
PROCEDURE QTMovieWindowStop( wih : QTMovieWindowH ) : ErrCode;
```
démarre/arrête la lecture de la vidéo dans la fenêtre indiquée.

```
ErrCode QTMovieWindowGetTime( QTMovieWindowH wih, double *t, int absolute );
ErrCode QTMovieWindowGetFrameTime( QTMovieWindowH wih, MovieFrameTime *ft,
      int absolute );
ErrCode QTMovieWindowSetTime( QTMovieWindowH wih, double t, int absolute );
ErrCode QTMovieWindowSetFrameTime( QTMovieWindowH wih, MovieFrameTime *ft,
      int absolute );
MovieFrameTime *secondsToFrameTime( double Time, double MovieFrameRate,
      MovieFrameTime *ft, int absolute );
PROCEDURE QTMovieWindowGetTime( wih : QTMovieWindowH,
      t : VAR double; absolute : Int32 ) : ErrCode;
PROCEDURE QTMovieWindowGetFrameTime( wih : QTMovieWindowH,
      ft : VAR MovieFrameTime; absolute : Int32 ) : ErrCode;
PROCEDURE QTMovieWindowSetTime( wih : QTMovieWindowH,
      t : double; absolute : Int32 ) : ErrCode;
PROCEDURE QTMovieWindowSetFrameTime( wih : QTMovieWindowH,
      ft : VAR MovieFrameTime; absolute : Int32 ) : ErrCode;
PROCEDURE secondsToFrameTime( t : double, freq : double,
      VAR ft : MovieFrameTime ) : MovieFrameTimePtr;
PROCEDURE FFTS( VAR ft : MovieFrameTime, freq : double ) : double;
```

QuickTime définit la notion du temps actuel comme le temps qui correspond à la trame affichée. Les fonctions ci-dessus permettent de connaître ou de modifier ce temps actuel, spécifié soit en secondes, soit en heures, minutes, secondes et nombre de trames. Cette dernière valeur varie entre 0 et *(*wih)->info->frameRate-1* (en Modula-2: *wih^^.info^.frameRate-1*) (NB: *frameRate* est une valeur réelle, le nombre de trames par seconde). Le temps *relatif* d'un *Movie* commence à *t=0s*, tandis que le temps *absolut* correspond à une heure de la journée si cette information est disponible (le temps initial, renseigné dans *info->startTime* (*info^.startTime*)). À noter quand même que le temps absolut retourné par les fonctions ci-dessus est basé sur la piste TimeCode, et peut donc être légèrement différent du temps calculé à partir du temps relatif et *info->startTime*, du aux contraintes d'arrondi.
secondsToFrameTime() permet de convertir un temps en secondes pour une fréquence (nombre de trames par seconde) donnée, *FFTS()* fait la conversion inverse. Pour obtenir le *FrameTime* absolut, passer la fréquence *info->TCframeRate* si positif!

Fonctions de gestion d'actions QuickTime

Un *QTMovieWindowH* contenant un *Movie* avec son contrôleur permet la lecture/exploration d'une vidéo. Un grand nombre des événements qui peuvent se produire donnent lieu à des *actions* que le contrôleur nous envoie, à priori **avant** que l'action en question soit réellement effectuée. Ce mécanisme permet par exemple d'être informé quand l'utilisateur change le temps actuel de la vidéo (en pas par pas ou par saut); quand la lecture d'une vidéo se termine; quand une vidéo distante est chargée complètement ou suffisamment pour commencer la lecture, etc. Pour chaque action, une fonction spécifique (à fournir par le programmeur) peut être appelée et reçoit alors le *QTMovieWindowH* ainsi qu'un pointeur vers l'information pertinente pour l'action en cours. Cette fonction, le *MCActionCallback*, peut (a priori) empêcher l'action d'avoir lieu en retournant une valeur non-zéro.

C/C++

```
typedef struct MCActions {
  short Step,
    Play,
    Activate,
    Deactivate,
    GoToTime,
    MouseDown,
    MovieClick,
    KeyUp,
    Suspend,
    Resume,
    LoadState,
    Finished,
    Idle,
    Start,
    Stop,
    Close;
} MCActions;



typedef int (*MCActionCallback)
  (QTMovieWindowH wi, void *params);


enum {
  kMovieLoadStateError = -1L,
  kMovieLoadStateLoading = 1000,
  kMovieLoadStateLoaded = 2000,
  kMovieLoadStatePlayable = 10000,
  kMovieLoadStatePlaythroughOK =
    20000,
  kMovieLoadStateComplete = 100000L
};
```

Modula-2

```
MCActions  = RECORD
    Step,
    Play,
    Activate,
    Deactivate,
    GoToTime,
    MouseDown,
    MovieClick,
    KeyUp,
    Suspend,
    Resume,
    LoadState,
    Finished,
    Idle,
    Start,
    Stop,
    Close   : Int16;
END;
MCActionsPtr  = POINTER TO MCActions;
MCActionParams= ADDRESS;

MCActionCallback = PROCEDURE(
  QTMovieWindowH,
  MCActionParams):Int32 [CDECL];

CONST
  kMovieLoadStateError = -1;
  kMovieLoadStateLoading = 1000;
  kMovieLoadStateLoaded = 2000;
  kMovieLoadStatePlayable = 10000;
  kMovieLoadStatePlaythroughOK =
    20000;
  kMovieLoadStateComplete = 100000;

VAR
  MCAction : MCActions;
```

L'action *Idle* est une "non-action" qui se produit régulièrement quand le moteur QuickTime est occupé avec autre chose que notre Movie; un *callback* pour cette action convient donc pour la traitement d'évènements non-graphiques, par exemple la communication avec un serveur.
NB: l'action *Start, Stop* et *Close* ne sont pas des actions QuickTime mais des actions QTils. Elle ne peuvent pas être empêchées!

```
void register_MCAction( QTMovieWindowH wih, short action, MCActionCallback
callback );
PROCEDURE register_MCAction( wih : QTMovieWindowH, action : Int16,
      callback : MCActionCallback );
```

enregistre le *callback* à appeler avant que l'événement *action* se produit dans *QTMovieWindowH*. À noter pour Modula-2: cette fonction doit être conforme à la convention d'appel *CDECL* (cf. la définition du type *MCActionCallback*).

```
MCActionCallback get_MCAction( QTMovieWindowH wih, short action );
PROCEDURE get_MCAction( wih : QTMovieWindowH, action : Int16 ) : MCActionCallback;
```

permet d'obtenir l'adresse de la fonction actuellement enregistrée pour l'action et le

QTMovieWindowH spécifiés.

```
void unregister_MCAction( QTMovieWindowH wih, short action );
PROCEDURE unregister_MCAction( wih : QTMovieWindowH, action : Int16 );
```

dissocie *callback* du *QTMovieWindowH*.

Dans la version Macintosh, il existent également des *NSMCActionCallback* avec les fonctions associées (*xxx_NSMCAction*), permettant de gérer les actions par des *sélecteurs* dédiés d'une instance de classe Objective C dont le *QTMovieWindowH* est un membre. L'initialisation pourrait prendre la forme

```
- (int) mPlay:(QTMovieWindowH)wih withParams:(void*)params
{ double t;
  if( (*wih)->wasScanned > 0 ){
    QTMovieWindowGetTime( wih, &t, 0 );
    [self didStepQTMovieView:(*wih)->theMovieView toRelativeTime:t];
  }
  return 0;
}

  register_NSMCAction(self, wih, MCAction()->Play,
     @selector(mPlay:withParams:),
     (NSMCActionCallback) [self methodForSelector:@selector(mPlay:withParams:)]);
```

ou, plus élégant:

```
#define REGISTER_NSMCACTION(wih,action,sel) register_NSMCAction(\
      self,(wih),(action),@selector(sel:withParams:),\
      (NSMCActionCallback) [self methodForSelector:@selector(sel:withParams:)])
//…
      REGISTER_NSMCACTION( wih, MCAction()->Play, mPlay );
```

Exemples de *callbacks* en Modula-2 qui interviennent quand l'utilisateur explore la vidéo; quand la lecture se termine; quand le chargement d'une vidéo distante progresse et quand un *QTMovieWindowH* est fermé.

```
FROM QTilsM2 IMPORT *;

PROCEDURE movieStep(wih : QTMovieWindowH; params : ADDRESS ) : Int32 [CDECL];
VAR
  t, t2 : double;
  steps : Int16;
  tstr : ARRAY[0..63] OF CHAR;
BEGIN
  steps := CAST(Int16, params);
  QTils.QTMovieWindowGetTime(wih, t, FALSE);
  IF steps = 0 THEN
    (* ceci est un cas spécifique à QTils! *)
    RealToStr( t, tstr );
    PostMessage( "Movie avancé en pas vers le nouveau temps:", tstr );
  ELSE
    (* steps est +1 ou -1 en fonction de la direction du pas *)
    t2 := t + CAST(double,steps) / wih^^.info^.frameRate;
    RealToStr( t2, tstr );
    PostMessage( "Movie va aller vers le nouveau temps:", tstr );
  END;
```

```
  RETURN 0;
END movieStep;

PROCEDURE moviePlay(wih : QTMovieWindowH; params : ADDRESS ) : Int32 [CDECL];
VAR
  t : double;
  tstr : ARRAY[0..63] OF CHAR;
BEGIN
  (* les actions 'Play' arrivent très régulièrement, mais le cas qui nous
   * intéresse est quand wasScanned>0, c.a.d. quand l'utilisateur vient de
   * faire avancer le temps actuel d'un saut. *)
  IF wih^^.wasScanned > 0 THEN
    QTils.QTMovieWindowGetTime(wih, t, FALSE);
    RealToStr( t, tstr );
    PostMessage( "Movie avancé en saut vers le nouveau temps:", tstr );
  END;
  RETURN 0;
END moviePlay;

PROCEDURE movieLoadState(wih : QTMovieWindowH; params : MCActionParams ) : Int32
[CDECL];
VAR
  code : ARRAY[0..32] OF CHAR;
  loadState : Int32;
BEGIN
  loadState := CAST(Int32,params);
  IntToStr( loadState, code );
  PostMessage( "Changement d'état de chargement", code );
  (* kMovieLoadStatePlaythroughOK: on peut lancer la lecture *)
  IF (loadState >= kMovieLoadStatePlaythroughOK)
     & (wih^^.loadState <= kMovieLoadStateComplete)
     & (wih^^.isPlaying=0) THEN
    QTils.QTMovieWindowPlay(wih);
  END;
  RETURN 0;
END movieLoadState;

PROCEDURE movieFinished(wih : QTMovieWindowH; params : ADDRESS ) : Int32 [CDECL];
VAR
  t : double;
  ft : MovieFrameTime;
BEGIN
  (* on envoie la vidéo à la moitié de sa durée *)
  QTils.QTMovieWindowSetTime(wih, wih^^.info^.duration/2.0, FALSE );
  err := QTils.QTMovieWindowGetTime(wih, t, FALSE);
  RETURN 0;
END movieFinished;

PROCEDURE movieClose(wih : QTMovieWindowH; params : ADDRESS ) : Int32 [CDECL];
VAR
  wiNum : ARRAY[0..16] OF CHAR;
BEGIN
  IntToStr( wih^^.idx, wiNum );
  (* ceci est une action dédiée de QTils *)
  PostMessage( "Fermeture du movie", wiNum );
  numQTWM := numQTWM - 1;
  RETURN 0;
```

```
END movieClose;

PROCEDURE movieKeyUp(wih : QTMovieWindowH; params : ADDRESS ) : Int32 [CDECL];
VAR
  evt : EventRecordPtr;
  size : Cartesian;
BEGIN
  IF( params = NIL )
    THEN
      RETURN 0;
  END;
  evt := CAST(EventRecordPtr, params);
  lastKeyUp := VAL(CHAR,evt^.message);
  CASE lastKeyUp OF
    'q', 'Q' :
       (*
         Il vaut mieux ne pas fermer des movies/fenêtres
         dans une MCActionCallback! On va quitter, pourtant:
       *)
       quitRequest := TRUE;
       (* ici on retourne TRUE pour annuler toute autre réaction ! *)
       RETURN 1;
    | '1' :
       (*
         taille actuelle du contenu de la fenêtre (= la vidéo, SANS
         le contrôlleur. Juste pour info…!
       *)
       QTils.QTMovieWindowGetGeometry( wih, NULL_Cartesian, ADR(size), 0 );
       (* on va mettre la vidéo à son échelle "1:1" : *)
       size.horizontal := wih^^.info^.naturalBounds.right;
       size.vertical := wih^^.info^.naturalBounds.bottom +
wih^^.info^.controllerHeight;
       QTils.QTMovieWindowSetGeometry( wih, NULL_Cartesian, ADR(size), 1.0, 0 );
       (* pour info, voyons si on a réussi: *)
       QTils.QTMovieWindowGetGeometry( wih, NULL_Cartesian, ADR(size), 0 );
    | 'c', 'C' :
       QTils.LogMsgEx( "QTMovieWindowToggleMCController() retourne %d",
         QTils.QTMovieWindowToggleMCController(wih) );
    ELSE
      (* noop *)
  END;
  RETURN 0;
END movieKeyUp;

PROCEDURE register_MCActions( qtwmH : QTMovieWindowH );
BEGIN
  IF QTils.QTMovieWindowH_Check(qtwmH) THEN
    QTils.register_MCAction( qtwmH, MCAction.Step, movieStep );
    QTils.register_MCAction( qtwmH, MCAction.Play, moviePlay );
    QTils.register_MCAction( qtwmH, MCAction.LoadState, movieLoadState );
    QTils.register_MCAction( qtwmH, MCAction.Finished, movieFinished );
    QTils.register_MCAction( qtwmH, MCAction.Close, movieClose );
    QTils.register_MCAction( qtwmH, MCAction.KeyUp, movieKeyUp );
    numQTWM := numQTWM + 1;
  END;
END register_MCActions;
```

Le code minimal pour afficher une vidéo avec la gestion définie ci-dessus serait, en Modula-2:

```
MODULE TstQTilsM2;

FROM QTilsM2 IMPORT *;

VAR
  numQTWM : UInt32;
  qtwmH : QTMovieWindowH;

(* fonctions MCActionCallback, register_MCActions() etc *)

BEGIN
  IF QTOpened() THEN
    numQTWM := 0;
    WITH QTils DO
      qtwmH := OpenQTMovieInWindow( "Sample.mov", 1 );
      (* on peut se passer de tester qtwmH ici *)
      register_MCActions( qtwmH );
      (* on teste soit si qtwmH <> NULL_QTMovieWindowH
       * soit si QTMovieWindowH_isOpen(qtwmH), soit s'il reste
       * des fenêtres ouvertes: *)
      WHILE (numQTWM > 0) DO
        n := PumpMessages(1)
      END;
      DisposeQTMovieWindow(qtwmH);
      LogMsg( "La Fin!" );
    END;
  END;
END TstQTilsM2.
```

Fonctions de manipulation d'objets vidéo

QuickTime permet non seulement l'affichage et la lecture de vidéos, mais également leur modification ... sans que cela nécessite un affichage. QTils exporte une toute petite partie de cette fonctionnalité, permettant essentiellement le transcodage (fonction bridée dans le lecteur QuickTime d'Apple à moins d'avoir la license "Pro"); l'enregistrement en tant que "séquence de référence" (*Movie* faisant référence à un autre fichier contenant la media source) et la lecture/modification des méta-données associées. Cette dernière fonctionnalité nécessite pour l'instant d'avoir les fichiers en-tête du SDK QuickTime en C/C++ mais est encapsulée (et simplifiée) dans l'interface Modula-2.

```
// open a movie from a file; pass NULL for dum1 & id, 0 or 1 for flags
ErrCode OpenMovieFromURL( Movie *newMovie, short flags, short *id,
      const char *URL, void *dataRef, OSType *type );
PROCEDURE OpenMovieFromURL( newMovie : VAR Movie, flags : Int16,
      theURL : ARRAY OF CHAR) : ErrCode;
```

ouvre l'objet QuickTime *theURL* spécifié et retourne un 'handle' dans newMovie. Passer *flags*=1 pour ouvrir le Movie en mode active. En C, passer NULL dans *id*, *dataRef* et *type* si ces informations ne vous intéressent pas.

```
unsigned short HasMovieChanged( Movie theMovie );
PROCEDURE HasMovieChanged( theMovie Movie ) : UInt16;
```

retourne une valeur non-nulle si *theMovie* a été modifié.

```
ErrCode SaveMovie( Movie theMovie );
PROCEDURE SaveMovie( theMovie Movie ) : ErrCode;
```
enregistre *theMovie* dans son fichier sur disque (sans vérifier s'il y a eu des modifications!).

```
ErrCode SaveMovieAsRefMov( const char *dstURL, Movie theMovie );
PROCEDURE SaveMovieAsRefMov( theURL : ARRAY OF CHAR, theMovie : Movie) : ErrCode;
```
enregistre l'objet theMovie en tant que séquence de référence.

```
ErrCode SaveMovieAs( char **URL, Movie theMovie, int noDialog );
PROCEDURE SaveMovieAs( theURL : ARRAY OF CHAR, theMovie Movie,
    noDialog : Int32) : ErrCode;
```
enregistre *theMovie* dans un fichier qui peut être spécifié et/ou sélectionné via la fenêtre dialogue qui est affichée si *noDialog* est zéro.

```
ErrCode CloseMovie( Movie *theMovie );
PROCEDURE CloseMovie(VAR theMovie : Movie) : ErrCode;
```
ferme *theMovie* – qui sera NULL au retour.

```
PROCEDURE GetMovieTrackCount(theMovie : Movie) : Int32;
```
obtient le nombre de pistes dans *theMovie*., tous types confondus (vidéo, audio, TimeCode, etc). C'est la fonction QuickTime du même nom qui est appelée directement.

```
double GetMovieTimeResolution(Movie theMovie);
PROCEDURE GetMovieTimeResolution(theMovie : Movie) : Real64 [CDECL];
```
le laps de temps minimal qui peut être représenté sur l'échelle de temps de *theMovie*.

```
ErrCode FindTextInMovie( Movie theMovie, char *text, int displayResult,
       Track *inoutTrack, double *foundTime, long *foundOffset, char **foundText )
PROCEDURE FindTextInMovie( theMovie Movie; clef: ARRAY OF CHAR;
       displayResult: Int32; VAR foundTrackNr: Int32; VAR foundTime: Real64;
       VAR foundOffset: Int32; VAR foundText: ARRAY OF CHAR ) : ErrCode [CDECL];
```
Trouve "clef" dans *theMovie*, et retourne la numéro de la piste trouvée, le temps où le texte a été trouvé ainsi que où dans le texte (la chaine) complet la clef se trouve (*offset* par rapport au début de la chaine). Si *displayResult* <> 0, *theMovie* est envoyé à *foundTime* et le texte trouvé est mis en surbrillance (si sa piste est visible). Retourne la chaine de texte complète dans *foundText*.
Cherche dans le texte contenu dans des pistes (y inclus les noms de chapitre), *pas* dans les méta-données!

```
long (*GetMovieChapterCount)( Movie theMovie );
PROCEDURE GetMovieChapterCount( theMovie: Movie ) : Int32 [CDECL];
```
Retourne le nombre de chapitres dont *theMovie* dispose.

```
ErrCode (*GetMovieIndChapter)( Movie theMovie, long ind, double *time, char **text
);
PROCEDURE GetMovieIndChapter( theMovie: Movie; idx: Int32,
      VAR time: Real64; VAR text: ARRAY OF CHAR ) : ErrCode [CDECL];
```
Retourne le temps (en secondes) et le titre du chapitre *idx* de *theMovie*.

```
ErrCode MovieAddChapter( Movie theMovie, Track refTrack, const char *name,
      double time, double duration );
PROCEDURE MovieAddChapter( theMovie: Movie; refTrackNr: Int32;
      title: ARRAY OF CHAR; time: Real64; duration: Real64 ) : ErrCode [CDECL];
```

Ajoute un titre de chapitre au temps et de la durée spécifié. refTrackNr désigne la piste à laquelle ce chapitre est associé et peut être 0 pour laisser QTils faire un choix raisonnable. Un durée de 0 équivaut une durée d'une trame (cf. GetMovieTimeResolution).

D'autres fonctions existent et sont (pour l'instant, uniquement) documentées ailleurs.

Les méta-données qui peuvent être associées à un *Movie* (et enregistrées dans le fichier correspondant; titre, auteur, date, etc) prennent la forme d'une clé d'annotation, une valeur (chaine de caractères) et optionnellement une indication de la langue utilisée (une autre chaine de caractères). QuickTime permet de faire de telles annotations au niveau du *Movie*, mais également au niveau des pistes. Par souci de simplification, QTils exporte seulement la notion du numéro de piste et non pas les pistes en tant qu'objets (à multiple types).

C/C++

```
typedef enum AnnotationKeys {
  akAuthor='auth', akComment='cmmt',
  akCopyRight='cprt',
  akDisplayName='name',
  akInfo='info', akKeywords='keyw',
  akDescr='desc', akFormat='orif',
  akSource='oris', akSoftware='soft',
  akWriter='wrtr', akYear='year',
  akCreationDate='@day'
} AnnotationKeys;
```

Modula-2

```
AnnotationKey   = UInt32;
AnnotationKeys= RECORD
  akAuthor, akComment,
  akCopyRight,
  akDisplayName,
  akInfo, akKeywords,
  akDescr, akFormat,
  akSource, akSoftware,
  akWriter, akYear,
  akCreationDate : AnnotationKey;
END;
MDLangStr = ARRAY[0..16] OF CHAR;
VAR
  MetaData : AnnotationKeys;
```

NB: en C/C++, '*auth*' n'est pas une chaine de caractères mais une variable de type entier dont les 4 octets se représentent par les caractères US-ASCII 'a', 'u', 't' et 'h', c.a.d. '*auth*' *!=* "*auth*".

Les fonctions *AddMetaDataStringToTrack()* et *AddMetaDataStringToMovie()* ajoutent la valeur spécifiée à la valeur déjà existante, le cas échéant. La spécification de la langue est optionnelle (passer *lang=NULL* pour utiliser la valeur par défaut); le format de la chaine est de la forme *fr_FR* pour le français (France) ou *en_GB* pour l'anglais "européen".
Les fonctions GetMetaDataStringFromTrack() et GetMetaDataStringFromMovie() retournent une copie de la valeur, si la clé spécifiée existe. Passer *lang=NULL* si la langue est sans intérêt.
NB: les copie(s) de valeur retournée(s) doivent être libérées après usage (C/C++). En Modula-2, les fonctions *GetMetaDataStringFromXXXX* copient les valeurs dans des *VAR ARRAY OF CHAR* qui ne nécessitent aucune dés-allocation.
Pour connaître la bonne longueur, utiliser *GetMetaDataStringLengthFromTrack()* ou *GetMetaDataStringLengthFromMovie()*.

```
ErrCode AddMetaDataStringToTrack( Movie theMovie, Track theTrack,
        AnnotationKeys key, const char *value,
        const char *lang );
ErrCode AddMetaDataStringToMovie( Movie theMovie, AnnotationKeys key,
        const char *value, const char *lang );
ErrCode GetMetaDataStringFromTrack( Movie theMovie, Track theTrack,
```

```
        AnnotationKeys key, char **value, char **lang );
ErrCode GetMetaDataStringFromMovie( Movie theMovie, AnnotationKeys key,
        char **value, char **lang );

PROCEDURE AddMetaDataStringToTrack( theMovie : Movie, trackNr : Int32,
        key : AnnotationKey, value : ARRAY OF CHAR,
        lang : MDLangStr ) : ErrCode;
PROCEDURE AddMetaDataStringToMovie( theMovie : Movie, key : AnnotationKey,
        value : ARRAY OF CHAR, lang : MDLangStr): ErrCode;
PROCEDURE GetMetaDataStringFromTrack( theMovie : Movie, trackNr : Int32,
        key : AnnotationKey, VAR value : ARRAY OF CHAR,
        VAR lang : ARRAY OF CHAR ) : ErrCode;
PROCEDURE GetMetaDataStringFromMovie( theMovie Movie, key : AnnotationKey,
        VAR value : ARRAY OF CHAR,
        VAR lang : ARRAY OF CHAR ) : ErrCode;
PROCEDURE GetMetaDataStringLengthFromTrack( theMovie : Movie, trackNr : Int32,
        key : AnnotationKey, VAR len : UInt32 ) : ErrCode;
PROCEDURE GetMetaDataStringLengthFromMovie( theMovie Movie, key : AnnotationKey,
        VAR len : UInt32 ) : ErrCode;
```

Fonctions de support - XML

Parmi l'éventail de formats que QuickTime sait lire, on trouve certains basés sur le XML. L'API dispose donc d'un certain nombre de routines pour importer et interpréter des fichiers XML, et QTils fournit un accès d'un plus haut niveau à cette fonctionnalité. Le principe est relativement simple. On construit une description du contenu qu'on cherche à lire, une sorte de clef codée qui décrit les éléments et leurs éventuels attributs. Cette clef compilée est ensuite passée à une routine, avec un nom de fichier à importer. Si le fichier en question correspond bien au format attendu, cette routine retourne une représentation binaire, un *XMLDoc* qui reprend l'arborescence présente dans le **contenu** XML. Étant donné que le XML est un format très générique et puissant, le *XMLDoc* retourné peut être relativement complexe et ne permettra pas, en général, d'obtenir directement les valeurs/données d'intérêt. QTils propose quelques fonctions pour manipuler ces contenus, éléments et attributs, dans le but de faciliter l'extraction des données. On note que bien que le XML est un format hiérarchique, il n'existe pas de moyen de marquer un élément comme l'élément racine dans la clef codée. En revanche, cet élément est traité séparément dans la représentation binaire du fichier XML (le *XMLDoc*), ce qui fait que certaines des routines décrites ci-dessous existent pour un élément 'lambda', et pour l'élément racine.

En grandes lignes, l'exploitation d'un *XMLDoc* se fait ainsi:

1. On vérifie que l'élément racine est bien celui attendu.
2. On extrait le contenu des éléments valides de l'élément racine (une matrice), un par un
    3. On parcourt la matrice de contenu obtenue en 2. ou 5.: on extrait chaque élément valide (*theElement*) en succession
        4. on identifie l'élément (*theElement.identifier*) et on lit les attributs et/ou les éléments dont on connait l'existence.
        5. Si l'élément peut contenir d'autres éléments, on extrait son contenu et le passe à l'étape 3.

Cette discussion n'entrera pas dans le détail des différents types associés, et sera centré sur l'interface Modula-2 (les procédures marquées [CDECL] sont présentes dans QTils.dll et donc également accessibles en C).

```
CONST
```

```
   elementFlagAlwaysSelfContained = 1;    (* Element doesn't have contents or clos-
ing tag even if it doesn't end with />, as in the HTML <img> tag*)
   elementFlagPreserveWhiteSpace  = 2;    (* Preserve whitespace in content, de-
fault is to remove it *)
   xmlParseFlagAllowUppercase     = 1;    (* Elements and attributes do not have to
be lowercase (strict XML), but can be upper or mixed case as in HTML*)
   xmlParseFlagAllowUnquotedAttributeValues = 2;  (* Attributes values do not have
to be enclosed in quotes (strict XML), but can be left unquoted if they contain no
spaces*)
   xmlParseFlagEventParseOnly     = 4;    (* Do event parsing only*)
   xmlParseFlagPreserveWhiteSpace = 8;    (* Preserve whitespace throughout the
document*)
PROCEDURE ParseXMLFile( theURL: ARRAY OF CHAR; xmlParser: ComponentInstance;
             flags: Int32; VAR xmlDoc: XMLDoc ) : ErrCode [CDECL];
```

ouvre le fichier *theURL* et tente de décoder son contenu avec la clef codée *xmlParser*. En cas de succès, le résultat est stocké dans *xmlDoc* et le code de retour sera *noErr*. Sinon, un code d'erreur est retourné, dont une description peut être obtenu avec *Check4XMLError*. À noter qu'on peut réutiliser pour lire plusieurs fichiers XML. L'argument *flags* est la somme (BOR) de la sélection pertinente des constantes ci-dessus.

```
PROCEDURE Check4XMLError( xmlParser: ComponentInstance; err: ErrCode;
             theURL: ARRAY OF CHAR,; VAR descr: Str255 ) : ErrCode [CDECL];
```

retourne une description/explication du code d'erreur *err* dans la chaine *descr*. À note qu'un message explicatif sera aussi disponible dans *QTils.lastSSLogMsg* immédiatement après l'appel à *ParseXMLFile*.

```
CONST
   nameSpaceIDNone = 0;
XMLnameSpaceID : UInt32;
```

un code pour identifier le *namespace* dans lequel sont attendus les éléments et attributs suivants.

```
PROCEDURE XMLParserAddElement( VAR xmlParser: ComponentInstance;
             elementName: ARRAY OF CHAR; elementID: UInt32;
             elementFlags: Int32 ) : ErrCode [CDECL];
```

définit un élément du fichier XML et l'ajoute à la clef codée. L'élément est défini par son nom (trouvé dans le fichier XML) ainsi que par un identifiant unique par lequel il sera identifiable dans le code.

```
CONST
  attributeValueKindCharString    = 0;
  attributeValueKindDouble        = attributeValueKindCharString;
  attributeValueKindInteger       = 1;    (* Number*)
  attributeValueKindPercent       = 2;    (* Number or percent*)
  attributeValueKindBoolean       = 4;    (* "true" or "false"*)
  attributeValueKindOnOff         = 8;    (* "on" or "off"*)
  attributeValueKindColour        = 16;   (* Either "#rrggbb" or a color name*)
  attributeValueKindEnum          = 32;   (* one of a number of strings; the enum
strings are passed as a zero-separated, Real64-zero-terminated C string in the at-
tributeKindValueInfo param*)
  attributeValueKindCaseSensEnum  = 64;   (* one of a number of strings; the enum
strings are passed as for attributeValueKindEnum, but the values are case-
sensitive*)
  MAX_ATTRIBUTE_VALUE_KIND     = attributeValueKindCaseSensEnum;
  attributeNotFound                 = -1;
PROCEDURE XMLParserAddElementAttribute( VAR xmlParser: ComponentInstance;
             elementID: UInt32; attrName: ARRAY OF CHAR;
             attrID: UInt32; attrType: UInt32 ) : ErrCode [CDECL];
```

définit un attribut pour l'élément *elementID* (qui doit déjà être défini) et l'ajoute à la clef codée. L'attribut est défini par son nom (trouvé dans le fichier XML) ainsi que par un identifiant. Ce code doit être unique dans le contexte de l'élément, c-a-d que le même code peut être utilisé pour plusieurs éléments. *attrType* définit le type de variable dont il s'agit.

```
PROCEDURE DisposeXMLParser( VAR xmlParser: ComponentInstance;
            VAR xmlDoc: XMLDoc; parserToo: Int32 ) : ErrCode [CDECL];
```

détruit le document *xmlDOC* et/ou la clef codée *xmlParser* si *parserToo* est *TRUE*. Il est acceptable de passer un *xmlDoc = nil* si le but est de détruire uniquement la clef codée.

```
PROCEDURE GetAttributeIndex( attrs: XMLAttributePtr; ID: UInt32;
            VAR idx; Int32 ) : ErrCode [CDECL];
```

obtient l'indice dans la matrice *attrs* où se trouve l'attribut avec l'identifiant *ID*.

```
PROCEDURE GetStringAttribute( VAR element: XMLElement; ID: UInt32;
            VAR str: ARRAY OF CHAR ) : ErrCode [CDECL];
PROCEDURE GetIntegerAttribute( VAR element: XMLElement; ID: UInt32;
            VAR num:  Int32 ) : ErrCode [CDECL];
PROCEDURE GetShortAttribute( VAR element: XMLElement; ID: UInt32;
            VAR num: Int16 ) : ErrCode [CDECL];
PROCEDURE GetDoubleAttribute( VAR element*) XMLElement; ID: UInt32;
            VAR num: Real64 ) : ErrCode [CDECL];
PROCEDURE GetBooleanAttribute( VAR element: XMLElement; ID: UInt32;
            VAR bool: UInt8 ) : ErrCode [CDECL];
```

routines qui obtiennent la valeur de l'attribut *ID* de l'élément *element* (qui n'est pas modifié). La routine *GetDoubleAttribute* sait lire les flottants en notation internationale et française et gère le NaN et les ±Inf.

```
PROCEDURE XMLRootElementID( xmlDoc: XMLDoc ) : UInt32;
```

retourne l'identifiant de l'élément racine, pour vérifier rapidement si le document *xmlDoc* est proprement enraciné.

```
CONST
   xmlIdentifierInvalid           = 0;
   xmlIdentifierUnrecognized      = 0FFFFFFFFh;
   xmlContentTypeInvalid          = 0;
   xmlContentTypeElement          = 1;
   xmlContentTypeCharData         = 2;
PROCEDURE XMLContentKind( theContent: XMLContentArrayPtr;
            element: CARDINAL ) : UInt32;
```

retourne un code descriptif de l'élément numéro *element* de la matrice de contenu *theContent*. Servira surtout à vérifier si
XMLContentKind(theContent, indice) <> xmlContentTypeInvalid

```
PROCEDURE XMLRootElementContentKind( xmlDoc: XMLDoc; element: CARDINAL ) : UInt32;
```

Comme *XMLContentKind* mais pour l'élément racine du document *xmlDoc*.

```
PROCEDURE XMLContentOfElementOfRootElement( xmlDoc: XMLDoc; element: CARDINAL;
            VAR theContent: XMLContentArrayPtr ) : BOOLEAN;
```

retourne la matrice de contenu *theContent* de l'élément *element* de l'élément racine du document *xmlDoc*. Sans doute une des deux fonctions centrales pour les documents simples où l'élément racine n'apparaît qu'à la racine et non pas de façon récursive. *theContent* contiendra alors les différents autres éléments dans leur orde d'apparition dans le fichier.

```
PROCEDURE XMLContentOfElement( VAR parentElement: XMLElement;
            VAR theElements: XMLContentArrayPtr ) : BOOLEAN;
```

retourne la matrice de contenu *theContent* de l'élément *parentElement*. Cette routine est utilisée pour les éléments qui contiennent d'autres éléments au lieu ou en plus d'avoir seulement des attributs.

```
PROCEDURE XMLElementOfContent( theContent: XMLContentArrayPtr;
            element: CARDINAL; VAR theElement: XMLElement ) : BOOLEAN;
```

extrait l'élément numéro *element* de la matrice de contenu *theContent*. L'autre fonction centrale: le *theElement* obtenu peut être comparé aux identifiants connus (*theElement.identifier*), et puis les attributs connus extraits via une des routines *GetXXXXAttribute*. Si l'élément contient d'autres éléments, il faut en extraire la matrice de contenu via *XMLContentOfElement(theElement, theElementsContents)* et parcourir les éléments de cette matrice.

Un exemple: le code qui était utilisé par QTVODm2 pour lire les fichiers XML '*design*'. Ici, on commence par définir un tableau constant qui décrit la squelette des fichiers XML qu'on veut pouvoir lire :

```
CONST

  recordAttributeValueTypeCharString    = 0;
  recordAttributeValueTypeInteger       = 1;          (* Number*)
  recordAttributeValueTypePercent       = 2;          (* Number or percent*)
  recordAttributeValueTypeBoolean       = 4;          (* "true" or "false"*)
  recordAttributeValueTypeOnOff         = 8;          (* "on" or "off"*)
  recordAttributeValueTypeDouble        = (1+MAX_ATTRIBUTE_VALUE_KIND);
  MAX_RECORD_ATTRIBUTE_VALUE_TYPE       = recordAttributeValueTypeDouble;

TYPE
   XML_ITEM_TYPES = ( xml_element, xml_attribute );
   XML_RECORD =
      RECORD
         CASE itemType : XML_ITEM_TYPES OF
            xml_element :
               elementTag : ARRAY[0..63] OF CHAR;
               elementID : UInt32;
            | xml_attribute :
               attributeTag : ARRAY[0..63] OF CHAR;
               attributeID, attributeType : UInt32;
         END;

   END;
   VODDESIGN_PARSER = ARRAY[0..21] OF XML_RECORD;

CONST
  element_vodDesign = 1;
  element_frequency = 2;
  element_frequence = 3;
  element_scale = 4;
  element_utc = 5;
  element_echelle = 6;
  element_channels = 7;
  element_canaux = 8;
  element_parsing = 9;
  element_lecture = 10;
```

```
  element_transcoding = 11;
  element_transcodage = 12;
  attr_freq = 1;
  attr_scale = 2;
  attr_forward = 4;
  attr_pilot = 6;
  attr_left = 8;
  attr_right = 10;
  attr_zone = 11;
  attr_dst = 12;
  attr_flLR = 13;
  attr_usevmgi = 14;
  attr_log = 15;
  attr_codec = 16;
  attr_bitrate = 17;
   (*
      Représentation lisible du 'design' de nos fichiers XML:
      les éléments et leurs attributs connus:
   *)
   xml_design_parser = VODDESIGN_PARSER{
        (* l'élément racine: *)
        {xml_element, "vod.design", element_vodDesign},
        {xml_element, "frequency", element_frequency},
       {xml_attribute, "fps", attr_freq, recordAttributeValueTypeDouble},
        {xml_element, "frequence", element_frequence},
           {xml_attribute, "tps", attr_freq, recordAttributeValueTypeDouble},
        {xml_element, "utc", element_utc},
           {xml_attribute, "zone", attr_zone, recordAttributeValueTypeDouble},
           {xml_attribute, "dst", attr_dst, recordAttributeValueTypeBoolean},
        {xml_element, "scale", element_scale},
           {xml_attribute, "factor", attr_scale, recordAttributeValueTypeDouble},
        {xml_element, "echelle", element_echelle},
           {xml_attribute, "facteur", attr_scale, recordAttributeValueTypeDouble},
        {xml_element, "channels", element_channels},
           {xml_attribute, "forward", attr_forward, recordAttributeValueTypeInte-
ger},
           {xml_attribute, "pilot", attr_pilot, recordAttributeValueTypeInteger},
           {xml_attribute, "left", attr_left, recordAttributeValueTypeInteger},
           {xml_attribute, "right", attr_right, recordAttributeValueTypeInteger},
        {xml_element, "canaux", element_canaux},
           {xml_attribute, "avant", attr_forward, recordAttributeValueTypeInteger},
           {xml_attribute, "pilote", attr_pilot, recordAttributeValueTypeInteger},
           {xml_attribute, "gauche", attr_left, recordAttributeValueTypeInteger},
           {xml_attribute, "droite", attr_right, recordAttributeValueTypeInteger}
   };
```

La routine qui va créer la clef codée à partir de ce tableau :

```
PROCEDURE CreateXMLParser( VAR xmlParser : ComponentInstance; VAR errors : Int32 )
: ErrCode;
VAR
   xmlErr : ErrCode;
   i, N : CARDINAL;
   lastElement : UInt32;
BEGIN

   (*
```

```
      Le nombre d'éléments dans le tableau d'initialisation:
   *)
   N := SIZE(xml_design_parser) / SIZE(xml_design_parser[0]);

   errors := 0;
  lastElement := 0;

   (*
      on parcourt le tableau xml_design_parser, ajoutant les informations
      à la clef codée:
   *)
   FOR i := 0 TO N-1 DO
      CASE xml_design_parser[i].itemType OF
         xml_element :
               xmlErr := XMLParserAddElement( xmlParser,
                  xml_design_parser[i].elementTag, xml_design_parser[i].elementID,
                  0, errors );
               IF xmlErr = noErr
                  THEN
                     lastElement := xml_design_parser[i].elementID;
                     QTils.LogMsgEx( "Nouvel élément #%d '%s'",
                        xml_design_parser[i].elementID,
                        xml_design_parser[i].elementTag );
               END;
       | xml_attribute :
               xmlErr := XMLParserAddElementAttribute( xmlParser, lastElement,
                  xml_design_parser[i].attributeTag,
                  xml_design_parser[i].attributeID,
                  xml_design_parser[i].attributeType, errors );
               IF xmlErr = noErr
                  THEN
                     QTils.LogMsgEx(
                        "Nouvel attribute #%d '%s' type %d pour élément #%d",
                        xml_design_parser[i].attributeID,
                        xml_design_parser[i].attributeTag,
                        xml_design_parser[i].attributeType, lastElement );
               END;
      END;
   END;
   RETURN xmlErr;
END CreateXMLParser;
```

Une fois qu'on a obtenu une clef codée, on peut l'utiliser pour lire des fichiers XML (autant de fois qu'on souhaiterait). Ici, on définit une routine qui prend un nom de fichier et une référence vers un tableau où stocker les valeurs de paramètres lues du fichier. La clef codée est obtenu d'une variable globale; si elle est `NIL`, la routine appelle d'abord *CreateXMLParser* pour créer la clef.

```
PROCEDURE ReadDefaultVODDescription( fName : URLString;
            VAR descr : VODDescription ) : ErrCode;
VAR
   xmlErr : ErrCode;
   errDescr : URLString;
   errors : Int32;
   elm : CARDINAL;
BEGIN
```

```
   IF xmlParser = NIL
      THEN
         xmlErr := CreateXMLParser( xmlParser, errors );
      ELSE
         xmlErr := noErr;
         errors := 0;
   END;

   IF errors = 0
      THEN
         (*
            vidons lastSSLogMsg - ParseXMLFile y laissera un message en cas d'erreur
         *)
         QTils.lastSSLogMsg^[0] := CHR(0);
         (*
            On lit <fName> et on tente d'interpreter son contenu selon les
            spécifications stockées dans xmlParser.
         *)
         xmlErr := QTils.ParseXMLFile( fName, xmlParser,
                   xmlParseFlagAllowUppercase
                   BOR xmlParseFlagAllowUnquotedAttributeValues
                   BOR elementFlagPreserveWhiteSpace,
                   xmldoc
         );
   END;
   IF xmlErr <> noErr THEN
      Assign( QTils.lastSSLogMsg^, errDescr );
      IF ( xmlErr = -2000 )
         THEN
            Append( " (fichier inexistant)", errDescr );
         ELSIF ( QTils.lastSSLogMsg^[0] <> CHR(0) )
            THEN
               PostMessage( "QTVODm2", errDescr );
      END;
   END;

   IF xmlErr = noErr
      THEN
         IF QTils.XMLRootElementID(xmldoc) = element_vodDesign
            THEN
               QTils.LogMsgEx( "Lectures des paramètres depuis '%s'", fName );
               ReadXMLDoc( fName, xmldoc, descr );
            ELSE
               QTils.LogMsgEx(
                  "'%s' est du XML valide mais n'a pas l'élément racine '%s'\n",
                  fName, xml_design_parser[0].elementTag );
         END;
   END;
   QTils.DisposeXMLParser( xmlParser, xmldoc, 0 );
   RETURN xmlErr;
END ReadDefaultVODDescription;
```

La lecture d'un *XMLDoc* se fait à partir de l'élément racine: on extrait le contenu de chaque élément valide en succession, et on le lit:

```
PROCEDURE ReadXMLDoc( fName : URLString; xmldoc : XMLDoc;
             VAR descr : VODDescription );
```

```
VAR
   theContent : XMLContentArrayPtr;
   elm : CARDINAL;
BEGIN
   elm := 0;
   (*
      on commence à lire à l'indice passée dans <elm>, à partir de 0,
      tant qu'on trouve du contenu valide dans la racine:
   *)
   WHILE QTils.XMLRootElementContentKind(xmldoc, elm) <> xmlContentTypeInvalid DO
      (* on obtient un pointeur vers le contenu *)
      IF QTils.XMLContentOfElementOfRootElement( xmldoc, elm, theContent )
         THEN
            (*
               on lit le contenu. ReadXMLContent augmentera elm;
               la plupart du temps ce sera elle qui fait toute la lecture
            *)
            ReadXMLContent( fName, theContent, descr, elm );
      END;
   END;
END ReadXMLDoc;
```

La lecture du contenu d'un élément 'lambda' se fait de façon comparable à la lecture du contenu de l'élément racine : on extrait et traite les éléments valides en succession. Ici, le traitement des éléments individuels se fait via une sous-routine :

```
PROCEDURE ReadXMLContent( fName : URLString; theContent : XMLContentArrayPtr;
        VAR descr : VODDescription; VAR elm : CARDINAL );
VAR
  xmlErr : ErrCode;
  bool : UInt8;
  element_content : XMLContentArrayPtr;
  theElement : XMLElement;

  (*
    la procédure qui lit chaque élément individuel;
    les attributs, ou les (sous) éléments.
  *)
  PROCEDURE ReadXMLElementAttributes( theElement : XMLElement;
        VAR descr : VODDescription );
  VAR
    idx : CARDINAL;
  BEGIN
    CASE theElement.identifier OF

      element_frequency,
      element_frequence :
         xmlErr := QTils.GetDoubleAttribute( theElement, attr_freq,
                    descr.frequency );
           IF (xmlErr <> attributeNotFound )
            THEN
              QTils.LogMsgEx( "attr #%d freq=%g (%d)",
                    VAL(INTEGER,attr_freq), descr.frequency, xmlErr );
           END;

      | element_utc :
         xmlErr := QTils.GetDoubleAttribute( theElement, attr_zone,
```

```
                descr.timeZone );
      IF (xmlErr <> attributeNotFound )
        THEN
          QTils.LogMsgEx( "attr #%d timeZone=%g (%d)",
                VAL(INTEGER,attr_zone), descr.timeZone, xmlErr );
      END;
    xmlErr := QTils.GetBooleanAttribute( theElement, attr_dst, bool );
      IF (xmlErr <> attributeNotFound )
        THEN
          descr.DST := (bool <> 0);
          QTils.LogMsgEx( "attr #%d DST=%hd (%d)",
                VAL(INTEGER,attr_dst), VAL(Int16,bool), xmlErr );
      END;

  | element_scale,
  element_echelle :
    xmlErr := QTils.GetDoubleAttribute( theElement, attr_scale,
                descr.scale );
      IF (xmlErr <> attributeNotFound )
        THEN
          QTils.LogMsgEx( "attr #%d scale=%g (%d)",
                VAL(INTEGER,attr_scale), descr.scale, xmlErr );
      END;

  | element_channels,
  element_canaux :
    xmlErr := QTils.GetIntegerAttribute( theElement, attr_forward,
                descr.channels.forward );
      IF (xmlErr <> attributeNotFound )
        THEN
          ClipInt( descr.channels.forward, 1, 4 );
          QTils.LogMsgEx( "attr #%d chForward=%d (%d)",
                VAL(INTEGER,attr_forward), descr.channels.forward,
                xmlErr );
      END;
    xmlErr := QTils.GetIntegerAttribute( theElement, attr_pilot,
                descr.channels.pilot );
      IF (xmlErr <> attributeNotFound )
        THEN
          ClipInt( descr.channels.pilot, 1, 4 );
          QTils.LogMsgEx( "attr #%d chPilot=%d (%d)",
                VAL(INTEGER,attr_pilot), descr.channels.pilot,
                xmlErr );
      END;
    xmlErr := QTils.GetIntegerAttribute( theElement, attr_left,
                descr.channels.left );
      IF (xmlErr <> attributeNotFound )
        THEN
          ClipInt( descr.channels.left, 1, 4 );
          QTils.LogMsgEx( "attr #%d chLeft=%d (%d)",
                VAL(INTEGER,attr_left), descr.channels.left,
                xmlErr );
      END;
    xmlErr := QTils.GetIntegerAttribute( theElement, attr_right,
                descr.channels.right );
      IF (xmlErr <> attributeNotFound )
        THEN
```

```
            ClipInt( descr.channels.right, 1, 4 );
            QTils.LogMsgEx( "attr #%d chRight=%d (%d)",
                  VAL(INTEGER,attr_right), descr.channels.right,
                  xmlErr );
          END;

    | element_vodDesign :
       (*
        ceci est le 'root element', la racine, qui n'a que des éléments,
        pas d'attributs. Si le fichier contient l'élément racine ailleurs
        qu'à la racine, le membre 'contents' de theELement sera un pointer
        vers un array de XMLContent, comme xmldoc^.rootElement.contents .
        On peut donc appeler ReadXMLContent de façon récursive.
        Bien sur il n'y a aucune utilité à pouvoir écrire des fichiers avec
        une liste de paramètres de configuration de façon récursive...
       *)
       IF QTils.XMLContentOfElement( theElement, element_content )
        THEN
          idx := 0;
          ReadXMLContent( fName, element_content, descr, idx );
       END;

    | xmlIdentifierUnrecognized :
       IF (theElement.name <> NIL) AND (theElement.attributes^.name <> NIL)
        THEN
          QTils.LogMsgEx( "élément inconnu <%s %s /> rencontré dans %s",
            theElement.name, theElement.attributes^.name, fName );
        ELSE
          QTils.LogMsgEx( "élément inconnu rencontré dans %s", fName );
       END;
       PostMessage( "QTVODm2", QTils.lastSSLogMsg^ );

    ELSE
      xmlErr := paramErr;
   END;
  END ReadXMLElementAttributes;

(* ReadXMLContent body code *)
BEGIN
  (* tant qu'on trouve du contenu valide... *)
  WHILE ( QTils.XMLContentKind( theContent, elm ) <> xmlContentTypeInvalid ) DO
    (*
     et pour chaque élément trouvé (on ne gère pas le cas
     xmlContentTypeCharData)
    *)
    IF QTils.XMLElementOfContent( theContent, elm, theElement )
     THEN
       (* on lit les attributs qu'on connait de chaque élément *)
       QTils.LogMsgEx(
          "Scanne attributs et/ou éléments de l'élément #%d (entrée %d)",
          theElement.identifier, elm );
       ReadXMLElementAttributes( theElement, descr );
    END;
    INC(elm);
  END;
END ReadXMLContent;
```

Depuis, les définitions de *XML_RECORD*, *CreateXMLParser* et *ReadXMLContent* ont évolué et ont été incluses dans QTilsM2 :

```
TYPE
  XMLAttributeParseCallback = PROCEDURE( (*theElement*) XMLElement, (*elementNr*)
UInt32, (*designTable*) ARRAY OF XML_RECORD,
                (*designEntry*) UInt32, (*fName*) ARRAY OF CHAR);
  XML_RECORD =
    RECORD
      CASE itemType : XML_ITEM_TYPES OF
        xml_element :
          elementTag : ARRAY[0..63] OF CHAR;
          elementID : UInt32;
        | xml_attribute :
          attributeTag : ARRAY[0..63] OF CHAR;
          attributeID, attributeType : UInt32;
          CASE useHandler : BOOLEAN OF
            FALSE :
              parsed : ADDRESS;
            | TRUE :
              handler : XMLAttributeParseCallback;
          END;
      END;
  END;
```

```
PROCEDURE CreateXMLParser( VAR xmlParser : ComponentInstance; xml_design_parser :
       ARRAY OF XML_RECORD;  VAR errors :  Int32 ) : ErrCode;
```

Construit une clef codée *xmlParser* à partir du tableau *xml_design_parser* (une sorte de DTD simplifié). Le tableau d'entrée décrit la structure hiérarchique attendu dans les fichiers XML à lire, et donne une adresse où la valeur lue de chaque attribut est stockée. Ce sera typiquement l'adresse (d'un membre) d'une variable temporaire/intermédiaire.

```
PROCEDURE ReadXMLContent( fName : ARRAY OF CHAR; theContent : XMLContentArrayPtr;
       xml_design_parser : ARRAY OF XML_RECORD; VAR elm : CARDINAL );
```

Lit le contenu donné, obtenu du fichier *fName* en cherchant une correspondance entre *theContent* et le tableau *xml_design_parser*. En fonction de *useHandler,* la ou les valeur(s) sont lues directement et stockées à l'adresse renseignée dans ce tableau, ou bien à lire et stocker par le *handler*. Cette option est pour les cas de figure plus complexes où l'interprétation d'un attribut donné dépend du contexte, *mais aussi* pour la lecture de chaines de caractère (*recordAttributeValueTypeCharString*). En effet, le tableau ne peut référencer que l'adresse de la chaine destination et pas sa dimension.

Et la lecture des fichiers XML est donc devenu :

```
TYPE
  VODDESIGN_PARSER = ARRAY[0..29] OF XML_RECORD;

VAR
  xmlVD : VODDescription;

CONST
  _d = FALSE;
  _fn = TRUE;
  xml_design_parser = VODDESIGN_PARSER{
     {xml_element, "vod.design", element_vodDesign},
     {xml_element, "frequency", element_frequency},
```

```
      {xml_attribute, "fps", attr_freq, recordAttributeValueTypeDouble,
        _d, ADR(xmlVD.frequency) },
     {xml_element, "frequence", element_frequence},
      {xml_attribute, "tps", attr_freq, recordAttributeValueTypeDouble,
        _d, ADR(xmlVD.frequency)},
     {xml_element, "utc", element_utc},
      {xml_attribute, "zone", attr_zone, recordAttributeValueTypeDouble,
        _d, ADR(xmlVD.timeZone)},
      {xml_attribute, "dst", attr_dst, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.DST)},
      {xml_attribute, "flipleftright", attr_flLR, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.flipLeftRight)},
      {xml_attribute, "flipgauchedroite", attr_flLR, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.flipLeftRight)},
     {xml_element, "scale", element_scale},
      {xml_attribute, "factor", attr_scale, recordAttributeValueTypeDouble,
        _d, ADR(xmlVD.scale)},
     {xml_element, "echelle", element_echelle},
      {xml_attribute, "facteur", attr_scale, recordAttributeValueTypeDouble,
        _d, ADR(xmlVD.scale)},
     {xml_element, "channels", element_channels},
      {xml_attribute, "forward", attr_forward, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.forward)},
      {xml_attribute, "pilot", attr_pilot, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.pilot)},
      {xml_attribute, "left", attr_left, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.left)},
      {xml_attribute, "right", attr_right, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.right)},
     {xml_element, "canaux", element_canaux},
      {xml_attribute, "avant", attr_forward, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.forward)},
      {xml_attribute, "pilote", attr_pilot, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.pilot)},
      {xml_attribute, "gauche", attr_left, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.left)},
      {xml_attribute, "droite", attr_right, recordAttributeValueTypeInteger,
        _d, ADR(xmlVD.channels.right)},
     {xml_element, "parsing", element_parsing},
      {xml_attribute, "usevmgi", attr_usevmgi, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.useVMGI)},
      {xml_attribute, "log", attr_log, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.log)},
     {xml_element, "lecture", element_lecture},
      {xml_attribute, "avecvmgi", attr_usevmgi, attributeValueKindBoolean,
        _d, ADR(xmlVD.useVMGI)},
      {xml_attribute, "journal", attr_log, recordAttributeValueTypeBoolean,
        _d, ADR(xmlVD.log)},
     {xml_element, "transcoding.mp4", element_transcoding},
      {xml_attribute, "codec", attr_codec, recordAttributeValueTypeCharString, _fn,
GetCodecString },
      {xml_attribute, "bitrate", attr_bitrate, recordAttributeValueTypeCharString, _fn,
GetBitRateString},
     {xml_element, "transcodage.mp4", element_transcodage},
      {xml_attribute, "codec", attr_codec, recordAttributeValueTypeCharString, _fn,
GetCodecString},
      {xml_attribute, "taux", attr_bitrate, recordAttributeValueTypeCharString, _fn,
```

```
GetBitRateString}
  };

PROCEDURE GetCodecString( theElement : XMLElement; elementEntry : UInt32;
  designTable : ARRAY OF XML_RECORD;
  designEntry : UInt32; fName : ARRAY OF CHAR ) : ErrCode;
BEGIN
  IF theElement.attributes^.identifier = attr_codec
    THEN
      Assign( theElement.attributes^.valueStr^, xmlVD.codec );
      RETURN noErr;
    ELSE
      RETURN paramErr;
  END;
END GetCodecString;

PROCEDURE GetBitRateString( theElement : XMLElement; elementEntry : UInt32;
  designTable : ARRAY OF XML_RECORD;
  designEntry : UInt32; fName : ARRAY OF CHAR ) : ErrCode;
BEGIN
  IF theElement.attributes^.identifier = attr_bitrate
    THEN
      Assign( theElement.attributes^.valueStr^, xmlVD.bitRate );
      RETURN noErr;
    ELSE
      RETURN paramErr;
  END;
END GetBitRateString;

PROCEDURE ReadXMLDoc( fName : URLString; xmldoc : XMLDoc; VAR descr :
VODDescription );
VAR
  theContent : XMLContentArrayPtr;
  elm : CARDINAL;
BEGIN
  elm := 0;
  (*
    on commence à lire à l'indice passée dans <elm>, à partir de 0,
    tant qu'on trouve du contenu valide dans la racine:
  *)
  WHILE ( QTils.XMLRootElementContentKind(xmldoc, elm) <> xmlContentTypeInvalid ) DO
    (* on obtient un pointeur vers le contenu *)
    IF QTils.XMLContentOfElementOfRootElement( xmldoc, elm, theContent )
      THEN
        (*
          on lit le contenu. ReadXMLContent augmentera elm;
          la plupart du temps ce sera elle qui fait toute la lecture
        *)
        QTils.ReadXMLContent(fName,theContent,xml_design_parser,elm);
        (*
          xml_design_parser contient les adresses des membres de la variable
          globale xmlVD; c'est elle qu'on copie après dans la destination:
        *)
        descr := xmlVD;
    END;
  END;
END ReadXMLDoc;
```

En C, le support pour l'interprétation automatique est légèrement différent : QTils fournit les définitions suivantes :

```
enum xmlRecordAttributeTypes {
  recordAttributeValueTypeCharString  = 0,
  recordAttributeValueTypeInteger  = 1L << 0, //!<    Number
  recordAttributeValueTypePercent  = 1L << 1, //!<    Number or percent
  recordAttributeValueTypeBoolean  = 1L << 2, //!<    "true" or "false"
  recordAttributeValueTypeOnOff    = 1L << 3, //!<    "on" or "off"
  recordAttributeValueTypeDouble   = (1+MAX_ATTRIBUTE_VALUE_KIND), //!<    floating
point number
  MAX_RECORD_ATTRIBUTE_VALUE_TYPE  = recordAttributeValueTypeDouble
};

/*!
  Callback to handle the occurrence of a specific element/attribute in XML file fName
  @param theElement the XML element being considered
  @param elementEntry  the element's number (largely informational)
  @param designTable the design table describing the XML file
  @param designEntry the index into the design table that matches the current
element/attribute pair
  @param fName  the XML file name
 */
typedef ErrCode (*XMLAttributeParseCallback)(XMLElement *theElement,
              size_t elementEntry, struct XML_Record *designTable,
              size_t designEntry, const char *fName);

typedef struct XML_Record {
  xml_Item_Types itemType;
  union content {
    struct xml_element_content {
      char Tag[64];
      UInt32 ID, unused;
    } element;
    struct xml_attribute_content {
      char Tag[64];
      UInt32 ID, Type;
    } attribute;
  } xml;
  union localread {
    void *parsed;
    XMLAttributeParseCallback handler;
  } reading;
} XML_Record;

QTLSext ErrCode CreateXMLParser( ComponentInstance *xmlParser,
            XML_Record *xml_design_parser, unsigned int N, int *errors );
QTLSext ErrCode DisposeXMLParser( ComponentInstance *xmlParser, XMLDoc *xmldoc,
            int parserToo );
QTLSext void ReadXMLContent( const char *fName, XMLContent *theContent,
            XML_Record *design, size_t designLength, size_t *elm );
```

Lit le contenu donné, obtenu du fichier *fName* en cherchant une correspondance entre *theContent* et le tableau *xml_design_parser*. La ou les valeur(s) sont lues directement et stockées à l'adresse renseignée dans ce tableau.

```
QTLSext void ReadXMLContentWithHandlers( const char *fName, XMLContent *theContent,
            XML_Record *design, size_t designLength, size_t *elm );
```

Lit le contenu donné, obtenu du fichier *fName* en cherchant une correspondance entre *theContent* et le tableau *xml_design_parser*. La ou les valeur(s) sont à lire et stocker par le *handler*. (Pour les cas de figure plus complexes où l'interprétation d'un attribut donné dépend du contexte).

L'exemple Modula-2 ci-dessus devient donc, en (Objective) C :

```
enum xmlItems { element_vodDesign = 1,
  element_frequency = 2,
  element_frequence = 3,
  element_scale = 4,
  element_utc = 5,
  element_echelle = 6,
  element_channels = 7,
  element_canaux = 8,
  element_parsing = 9,
  element_lecture = 10,
  element_transcoding = 11;
  element_transcodage = 12;
  attr_freq = 1,
  attr_scale = 2,
  attr_forward = 4,
  attr_pilot = 6,
  attr_left = 8,
  attr_right = 10,
  attr_zone = 11,
  attr_dst = 12,
  attr_flLR = 13,
  attr_usevmgi = 14,
  attr_log = 15 };

VODDescription xmlVD;

XML_Record xml_design_parser[30] = {
    {xml_element, "vod.design", element_vodDesign},
    {xml_element, "frequency", element_frequency},
      {xml_attribute, "fps", attr_freq, recordAttributeValueTypeDouble,
         &xmlVD.frequency},
    {xml_element, "frequence", element_frequence},
      {xml_attribute, "tps", attr_freq, recordAttributeValueTypeDouble,
         &xmlVD.frequency},
    {xml_element, "utc", element_utc},
      {xml_attribute, "zone", attr_zone, recordAttributeValueTypeDouble,
         &xmlVD.timeZone},
      {xml_attribute, "dst", attr_dst, recordAttributeValueTypeBoolean,
         &xmlVD.DST},
      {xml_attribute, "flipleftright", attr_flLR, recordAttributeValueTypeBoolean,
         &xmlVD.flipLeftRight},
      {xml_attribute, "flipgauchedroite", attr_flLR, recordAttributeValueTypeBoolean,
         &xmlVD.flipLeftRight},
    {xml_element, "scale", element_scale},
      {xml_attribute, "factor", attr_scale, recordAttributeValueTypeDouble,
         &xmlVD.scale},
    {xml_element, "echelle", element_echelle},
      {xml_attribute, "facteur", attr_scale, recordAttributeValueTypeDouble,
         &xmlVD.scale},
    {xml_element, "channels", element_channels},
```

```
    {xml_attribute, "forward", attr_forward, recordAttributeValueTypeInteger,
       &xmlVD.channels.forward},
    {xml_attribute, "pilot", attr_pilot, recordAttributeValueTypeInteger,
       &xmlVD.channels.pilot},
    {xml_attribute, "left", attr_left, recordAttributeValueTypeInteger,
       &xmlVD.channels.left},
    {xml_attribute, "right", attr_right, recordAttributeValueTypeInteger,
       &xmlVD.channels.right},
  {xml_element, "canaux", element_canaux},
    {xml_attribute, "avant", attr_forward, recordAttributeValueTypeInteger,
       &xmlVD.channels.forward},
    {xml_attribute, "pilote", attr_pilot, recordAttributeValueTypeInteger,
       &xmlVD.channels.pilot},
    {xml_attribute, "gauche", attr_left, recordAttributeValueTypeInteger,
       &xmlVD.channels.left},
    {xml_attribute, "droite", attr_right, recordAttributeValueTypeInteger,
       &xmlVD.channels.right},
  {xml_element, "parsing", element_parsing},
    {xml_attribute, "usevmgi", attr_usevmgi, recordAttributeValueTypeBoolean,
       &xmlVD.useVMGI},
    {xml_attribute, "log", attr_log, recordAttributeValueTypeBoolean,
       &xmlVD.log},
  {xml_element, "lecture", element_lecture},
    {xml_attribute, "avecvmgi", attr_usevmgi, recordAttributeValueTypeBoolean,
       &xmlVD.useVMGI},
    {xml_attribute, "journal", attr_log, recordAttributeValueTypeBoolean,
       &xmlVD.log}
  {xml_element, "transcoding.mp4", element_transcoding},
    // xmlVD.codec sera allouée!
    {xml_attribute, "codec", attr_codec, recordAttributeValueTypeCharString,
       &xmlVD.codec},
    // xmlVD.bitRate sera allouée!
    {xml_attribute, "bitrate", attr_bitrate, recordAttributeValueTypeCharString,
       &xmlVD.bitRate},
  {xml_element, "transcodage.mp4", element_transcodage},
    {xml_attribute, "codec", attr_codec, recordAttributeValueTypeCharString,
       &xmlVD.codec},
    {xml_attribute, "taux", attr_bitrate, recordAttributeValueTypeCharString,
       &xmlVD.bitRate},
};

void ReadXMLDoc( const char *fName, XMLDoc xmldoc, VODDescription *descr )
{ XMLContent *theContent;
  unsigned short elm;
  elm = 0;
  while( XMLRootElementContentKind( xmldoc, elm ) != xmlContentTypeInvalid ){
    if( XMLContentOfElementOfRootElement( xmldoc, elm, &theContent ) ){
      ReadXMLContent( fName, theContent, xml_design_parser,
           sizeof(xml_design_parser)/sizeof(XML_Record), &elm );
      *descr = xmlVD;
    }
  }
}

/*!
     @class QTVOD selector to read a default VODDescription from the named file,
storing it in the designated variable
```

```
 */
- (ErrCode) ReadDefaultVODDescription:(const char*)fName toDescription:
(VODDescription*)descr
{ ErrCode xmlErr = noErr;
  int errors = 0;
  XMLDoc xmldoc = NULL;
  NSMutableString *errDescr = nil;

  if( !xmlParser ){
    xmlErr = CreateXMLParser( &xmlParser, xml_design_parser,
            sizeof(xml_design_parser)/sizeof(XML_Record), &errors );
  }
  if( errors == 0 ){
    lastSSLogMsg[0] = '\0';
    xmlErr = ParseXMLFile( fName, xmlParser,
             xmlParseFlagAllowUppercase|xmlParseFlagAllowUnquotedAttributeValues
             |elementFlagPreserveWhiteSpace,
             &xmldoc );
    if( xmlErr != noErr ){
      errDescr = [NSMutableString stringWithUTF8String:&lastSSLogMsg[0]];
      if( xmlErr == couldNotResolveDataRef ){
        [errDescr appendString:@" (file does not exist)"];
      }
      else if( lastSSLogMsg[0] ){
        PostMessage( "QTVODosx", [errDescr UTF8String] );
      }
    }
    else{
      if( XMLRootElementID(xmldoc) == element_vodDesign ){
        QTils_LogMsgEx( "Reading VOD parameters from '%s'", fName );
        ReadXMLDoc( fName, xmldoc, descr );
      }
      else{
        QTils_LogMsgEx( "'%s' is valid XML but lacking root element '%s'",
             fName, xml_design_parser[0].xml.element.Tag
        );
      }
    }
  }
  if( xmldoc ){
    DisposeXMLParser( &xmlParser, &xmldoc, 0 );
  }
  return xmlErr;
}
```

Fonctions de support (autres)

```
PROCEDURE QTMovieWindowH_Check( wih : QTMovieWindowH ) : BOOLEAN;
PROCEDURE QTMovieWindowH_isOpen( wih : QTMovieWindowH ) : BOOLEAN;
```

vérifient si *wih* est bien un *QTMovieWindowH* valide ou avec une vidéo affichée dans sa fenêtre. Dans l'API C/C++, ces vérifications sont faites par deux macros du même nom.

```
PROCEDURE LastQTError() : ErrCode;
```

retourne le dernier erreur QuickTime (ou *noErr*).

```
unsigned int PumpMessages(int block);
PROCEDURE PumpMessages( block : Int32 ) : UInt32;
```

Fonction 'minimale' pour actionner la pompe à messages d'événements du système d'exploitation. Si *block* est 0, la fonction retourne toute de suite s'il n'y a pas de messages en attente, sinon elle attend l'arrivé d'un message.

```
PROCEDURE MSWinErrorString( err : INTEGER; VAR errStr : ARRAY OF CHAR ) : UInt32;
```

Obtient une description d'un code erreur système MS Windows (via la routine FormatMessage).

```
PROCEDURE MacErrorString( err: ErrCode; VAR errString ARRAY OF CHAR;
                          VAR errComment: ARRAY OF CHAR) : UInt32 [CDECL];
```

Obtient la représentation canonique du code erreur *err*, et si possible une description. Fonctionne pour les codes erreurs Macintosh (QuickTime; négatifs la plupart du temps) mais aussi les codes erreurs 'standards' (Unix/POSIX). Si le même code est utilisé dans plusieurs contextes, *errComment* contient autant de descriptions. Retourne 0 si la représentation canonique n'est pas disponible (dans ce cas *errString* et *errComment* ne sont pas modifiées); 1 si au moins *errString* a été mise à jour (dans ce cas, *errComment* est vide si la description n'est pas disponible).

```
PROCEDURE PostMessage( title, message : ARRAY OF CHAR ): CARDINAL;
```

affiche une fenêtre dialogue avec le titre et le message spécifiés, ainsi qu'un bouton "OK".

```
PROCEDURE sprintf( VAR buf : ARRAY OF CHAR; format : ARRAY OF CHAR; ...) : Int32
[RightToLeft, LEAVES, VARIABLE];
PROCEDURE sscanf( source : ARRAY OF CHAR; template : ARRAY OF CHAR; ...) : Int32
[RightToLeft, LEAVES, VARIABLE];
```

versions des fonctions de la librairie C standard du même nom qui permettent d'imprimer dans une chaine de caractères (*sprintf*) et d'en extraire des valeurs, suivant un masque/filtre.

```
PROCEDURE ssprintf( VAR dest: String1kPtr; format : ARRAY OF CHAR; ... ) : Int32
[RightToLeft, LEAVES, VARIABLE];
PROCEDURE ssprintfAppend( VAR dest : String1kPtr; format : ARRAY OF CHAR; ... ) :
Int32 [RightToLeft, LEAVES, VARIABLE];
```

variations de *sprintf*. Le fonctionnement est identique à part pour la cible/destination : elle est allouée en fonction de la longueur de la chaine résultante. Elle doit donc être désallouée après utilisation, avec la fonction *free* du module *QTilsm2*. Le pointeur *dest* doit être initialisée à *NIL* avant l'appel, être le résultat d'un appel précédent à *ssprintf*.

ssprinfAppend permet d'enchainer plusieurs appels pour construire une chaine complexe qui ne peut pas être générée avec un seul appel à *ssprintf*.

```
size_t QTils_LogMsg( const char *msg );
PROCEDURE LogMsg( msg : ARRAY OF CHAR) : UInt32;
```

enregistre un message dans le journal; nécessite la présence de la fonctionnalité 'SS_Log' dans QTils.dll ! (sinon aucun effet) . À noter que la version Modula-2 de cette fonction honore les caractères spéciaux 'C': '\n' pour une nouvelle ligne, '\t' pour un caractère de tabulation, etc.

```
PROCEDURE LogMsgEx( msg : ARRAY OF CHAR; ...) : UInt32 [RightToLeft, LEAVES,
VARIABLE];
```

idem, mais permet la création d'un message formaté/structuré via la fonction *wvsprintf*. Par exemple

LogMsgEx("Le code d'erreur est %d", errCode);

```
lastSSLogMsg : POINTER TO ARRAY OF CHAR
```

Pointeur vers le dernier message enregistré dans le journal. Cette chaine est mise à jour même sans la fonctionnalité 'SS_Log' dans QTils.dll .

```
PROCEDURE free( VAR ptr : ADDRESS );
```

libère (désalloue) la mémoire pointée par *ptr* et allouée dans QTilsM2, avec la fonction correspondante. Le fait d'allouer/désallouer de la mémoire dans des librairies différentes est une mauvaise idée, en particulier sous MS Windows; cette fonction permet de libérer la mémoire dans le même où elle a été allouée.

```
QTILS_ALLOCATORS =
  RECORD
    malloc : PROCEDURE( (*size*) CARDINAL32 ) : ADDRESS [CDECL];
    calloc : PROCEDURE( (*n*) CARDINAL32, (*elemSize*) CARDINAL32 ) : ADDRESS [CDECL];
    realloc : PROCEDURE( (*ptr*) ADDRESS, (*memSize*) CARDINAL32 ) : ADDRESS [CDECL];
    free : PROCEDURE( VAR (*ptr*) ADDRESS ) [CDECL];
END;
QTils_Allocator : POINTER TO QTILS_ALLOCATORS;
```

adresses des routines standard C d'allocation et désallocation de mémoire, utilisées dans QTilsM2 . Remplacez ces adresses par d'autres ***de la même*** famille pour s'assurer que QTilsM2 utilise bien les bonnes fonctions, ou appelez les fonctions par défaut pour allouer/libérer de la mémoire dynamique à utiliser/libérer dans QTilsM2. Par exemple, au tout début du programme :

```
  QTils.QTils_Allocator^.malloc := POSIX.malloc;
  QTils.QTils_Allocator^.calloc := POSIX.calloc;
  QTils.QTils_Allocator^.realloc := POSIX.realloc;
  QTils.QTils_Allocator^.free := POSIX.free;
```

Notez bien que la fonction *free* reçoit un pointeur vers la variable qui contient l'adresse de la mémoire à libérer!

```
TYPE
  OSType = UInt32;
  OSTypeStr = POINTER TO ARRAY[0..4] OF CHAR;
PROCEDURE FOUR_CHAR_CODE(code : OSTypeStr) : OSType;
PROCEDURE OSTStr(otype : OSType) : OSTypeStr;
```

QuickTime utilise des *OSType* pour préciser des types et/ou des codes d'erreur de façon facilement compréhensible: une chaine de 4 caractères ASCII entre guillemets (ex. 'TVOD', le type d'un movie QuickTime) simples équivaut un entier de 4 octets en C. Les fonctions *FOUR_CHAR_CODE()* et *OSTStr()* permettent de convertir entre ces 2 représentations en Modula-2.

QTMovieSinks

QTMovieSinks a été incorporé dans QTils et on accède donc à sa fonctionnalité comme décrit ci-dessus. Comme le reste de la librairie, QTMovieSinks encapsule certaines fonctionnalités de QuickTime avec un niveau d'abstraction assez élevé. Le but de cette partie de QTils est la création (en ligne) de vidéos à partir de flux d'images, par exemple des captures d'écran ou des images de synthèse.

C/C++

```
typedef union QTAPixel {

   struct {
      unsigned char
#if TARGET_OS_MAC
         alpha, red, green, blue;

#else
         red, green, blue, alpha;
#endif
   } ciChannel;

   struct {
      unsigned char
#if TARGET_OS_MAC
         alpha, red, green, blue;

#else
         blue, green, red, alpha;
#endif
   } icmChannel;


   unsigned long value;
} QTAPixel;
```

Modula-2

```
QTAPixel      = RECORD
   CASE : QTPixelAddressingClass OF
      QTCompressImage :
         ciChannel : RECORD
%IF TARGET_OS_MAC %THEN
            alpha, red, green, blue :
                                    UInt8;
%ELSE
            red, green, blue, alpha :
                                    UInt8;
%END
            END;
        | QTImageCompressionSession :
         icmChannel : RECORD
%IF TARGET_OS_MAC %THEN
            alpha, red, green, blue :
                                    UInt8;
%ELSE
               blue, green, red, alpha :
                                    UInt8;
%END
            END;
        | QTRawBits :
         value : UInt32;
     END;
END;
QTAPixelPtr   = POINTER TO QTAPixel;
QTFrameBuffer= POINTER TO ARRAY
       [0..1920*1080-1] OF QTAPixel;
QTFrameBuffers= POINTER TO ARRAY
       [0..255] OF QTFrameBuffer;
```

C/C++

```
typedef struct QTMovieSinks {
   struct QTMovieSinkQTStuff *privQT;
   const char *theURL;
   unsigned short useICM;
   unsigned short Width, Height;
   unsigned short hasAlpha;
   short dealloc_qms, dealloc_imageFrame;

   short AddFrame_RT;
   QTAPixel **imageFrame;
   unsigned short currentFrame,
      frameBuffers;
   ErrCode lastErr;
} QTMovieSinks;


typedef struct QTCompressionCodecs {
   unsigned long
      Video
      , JPEG, JPEGA
      , Animation
      , HD720p, HD1080i60
      , MPEG4, H264
      , Raw
      , Cinepak
#if TARGET_OS_MAC
      , ApplePixlet
#endif
      ;
} QTCompressionCodecs;

typedef struct QTCompressionQualities {
   unsigned long
      Lossless,
      Max, Min,
      Low, Normal, High;
} QTCompressionQualities;
```

Modula-2

```
QTMovieSinkQTStuff = RECORD (* … *) END;
QTMovieSinks = RECORD
   privQT : POINTER TO QTMovieSinkQTStuff;
   theURL : String1kPtr;
   useICM : UInt16;
   Width, Height : UInt16;
   hasAlpha : UInt16;
   dealloc_qms, dealloc_imageFrame :
                                      Int16;
   AddFrame_RT : Int16;
   imageFrame : QTFrameBuffers;
   currentFrame,
      frameBuffers : UInt16;
   lastErr : ErrCode;
END;
QTMovieSinkPtr  = POINTER TO QTMovieSinks;

QTCompressionCodecs  = RECORD

      Video
         , JPEG, JPEGA
         , Animation
         , HD720p, HD1080i60
         , MPEG4, H264
         , Raw
         , Cinepak
%IF TARGET_OS_MAC %THEN
         , ApplePixlet
%END
                           : UInt32;
END;

QTCompressionQualities  = RECORD

      Lossless,
      Max, Min,
      Low, Normal, High  : UInt32;
END;
```

La création de vidéos se fait en trois phases: initialisation; ajout des images (trames); finalisation (fermeture).

Initialisation

La création d'une vidéo QuickTime nécessite d'avoir QuickTime initialisé et ensuite de choisir le codec et la qualité utilisés pour compression, ainsi que l'algorithme interne. À l'heure actuelle, QuickTime fournit deux méthodes pour la compression: une plus ancienne (basée sur CompressImage), et une récente (basée sur ICM Compression Sessions). Cette dernière méthode est un peu plus flexible mais certainement pas plus performante en termes de vitesse.

Il est possible d'utiliser toutes les codecs connus par QuickTime qui sont compatibles avec

des pixels RGB/RGBA[1] (à l'exclusion probable des codecs fournis par des *Components* de tierce partie). Les codecs les plus à même d'être utilisé sont disponible via la structure *QTCompressionCodecs*. La qualité de l'encodage est spécifiée via une des constantes dans la structure *QTCompressionQualities*. À noter que *LossLess* (sans perte) peut être combiné avec la plupart des codecs, mais ne porte correctement son nom qu'avec les codecs *Animation* et *Raw*. (Pour le *JPEG*, *LossLess* met la qualité de compression à 100%.)

```
QTCompressionCodecs *QTCompressionCodec();
QTCompressionQualities *QTCompressionQuality();

VAR
  QTCompressionCodec : QTCompressionCodecs;
  QTCompressionQuality : QTCompressionQualities;
```

C'est également au moment de l'initialisation que la taille des images est précisée, et si elles contiennent de la transparence (canal alpha).
NB: la mémoire pour recevoir les trames individuelles (*imageFrame*) est (ou doit être) allouée avec 4 octets par pixel, et le résultat de l'encodage est en général mieux (et plus prévisible) quand il y a un canal alpha. Dans tous les cas, *imageFrame* doit être un vecteur de *frameBuffers* de matrices de taille *Width * Height*. Si les *ICM Compression Sessions* sont utilisées (*useICM = TRUE*), jusqu'à 255 de ces trames sont supportées; pour *CompressImage* (*useICM = FALSE*), une seule.

Les fonctions d'initialisation:

```
QTMovieSinks *open_QTMovieSink( QTMovieSinks *qms, const char *theURL,
                    unsigned short Width, unsigned short Height, int hasAlpha,
                    unsigned short frameBuffers,
                    unsigned long codec, unsigned long quality, int useICM,
                    int openQT, ErrCode *errReturn );
QTMovieSinks *open_QTMovieSinkWithData( QTMovieSinks *qms, const char *theURL,
                    QTAPixel **imageFrame,
                    unsigned short Width, unsigned short Height, int hasAlpha,
                    unsigned short frameBuffers,
                    unsigned long codec, unsigned long quality, int useICM,
                    int openQT, ErrCode *errReturn );
```

Les noms de fonction et la convention d'appel sont légèrement différents en Modula-2:

```
PROCEDURE OpenQTMovieSink( VAR qms: QTMovieSinks; theURL: ARRAY OF CHAR;
                    Width: UInt16; Height: UInt16; hasAlpha: INTEGER;
                    frameBuffers: UInt16;
                    codec: UInt32; quality: UInt32;
                    useICM: INTEGER;  openQT: INTEGER ) : ErrCode [CDECL];
PROCEDURE OpenQTMovieSinkWithData( VAR qms: QTMovieSinks; theURL: ARRAY OF CHAR;
                    imageFrame : QTFrameBuffers;
                    Width: UInt16; Height UInt16; hasAlpha: INTEGER;
                    frameBuffers: UInt16;
                    codec: UInt32; quality UInt32;
                    useICM: INTEGER; openQT: INTEGER ) : ErrCode [CDECL];
```

1 il est également possible d'utiliser des pixels non-RGB, comme les différents types de YUV. QTMovieSinks ne supporte que les pixels RGB pour l'instant, le type le plus courant dans les images de synthèse.

L'argument *qms* pointe vers une structure *QTMovieSinks* à initialiser et peut être *NULL* en C pour faire créer une structure. Il est possible d'allouer *imageFrame* par ses propres soins, ou par la fonction.
Les fonctions retournent un code d'erreur (ou *noErr*) en Modula-2; un pointer vers *qms* (ou *NULL*) ainsi que le code d'erreur *errReturn* en C/C++.
Finalement, l'initialisation de QuickTime peut être demandé en précisant *openQT = TRUE* , sinon un appel préalable à *OpenQT()* doit être fait.

L'ajout d'images

Une fois que le descripteur *qms* de la vidéo a été initialisé, il est possible d'ajouter des trames. Toutefois, il faut d'abord préparer l'image: les deux fonctions d'ajout décrites ci-dessous l'attendent dans *qms->imageFrame[qms->currentFrame]* , c-a-d la trame actuelle. Le processus d'encodage actuel est strictement sériel et synchrone pour la méthode *CompressImage*, mais potentiellement asynchrone avec la méthode *ICM* qui en plus supporte de changer l'ordre des trames avec des codecs comme MPEG4 et H.264 . Il s'agit là de la raison pour laquelle il peut y avoir plusieurs trames dans *qms->imageFrame* . Il ne faut donc pas, au moins pour les codecs concernés, varier *qms->currentFrame* soi-même; les fonctions d'ajout d'image le font en fonction de la disponibilité des trames internes[1]. Voir plus loin pour des fonctions de support pour accéder à cette mémoire en Modula-2 .

```
ErrCode QTMovieSink_AddFrame( QTMovieSinks *qms,
                      double frameDuration );
ErrCode QTMovieSink_AddFrameWithTime( QTMovieSinks *qms,
                      double frameTime );

PROCEDURE AddFrameToQTMovieSink( VAR qms: QTMovieSinks;
                      frameDuration: Real64 ) : ErrCode [CDECL];
PROCEDURE AddFrameToQTMovieSinkWithTime( VAR qms: QTMovieSinks;
                      frameTime: Real64 ) : ErrCode [CDECL];
```

Si la méthode *ICM* est utilisée, il est possible de préciser le temps en secondes relatif au début de la vidéo quand la trame doit être affichée lors de la lecture de la vidéo (ces temps doivent être successifs et incrémentaux). Il est possible de spécifier une durée de présentation (en secondes) pour les deux méthodes. Les fonctions retournent *noErr* en cas de succès (-codecCantQueueErr, 8975, dans les rares cas que ICM n'a plus de trames disponibles, ce qui est également compté comme une trame perdue).

À noter que sur MS Windows, les fonctions d'ajout d'image passent en mode temps réel lors de leur exécution, si *qms->AddFrame_RT* est non-nulle.

Finalisation

La création se termine par la fermeture du descripteur *qms* . La vidéo ne sera disponible qu'après cette étape, car la consolidation des images ajoutées vers le fichier spécifié au moment de l'ouverture se fait à la fin. La fonction de fermeture s'en occupe, ainsi que de rendre toute mémoire allouée par QTMovieSinks lors du processus. Si souhaité, une piste TimeCode est ajoutée à la vidéo. Il s'agit d'une piste simple qui ne sera véridique que pour les vidéos avec un nombre de trames par seconde constant. Enfin, si *stats* pointe vers une structure QTMSEncodingStats, cette structure est initialisée avec un appel de *get_QTMovieSink_EncodingStats* (*GetQTMovieSinkEncodingStats*) juste avant la fermeture du *movie*.

[1] En jargon QuickTime; QTMovieSinks utilise un *CVPixelBufferPoolRef* de *frameBuffers CVPixelBufferRef* .

Finalement, il est possible de fermer QuickTime comme ultime opération.

```
ErrCode close_QTMovieSink( QTMovieSinks **qms, int addTCTrack,
                    QTMSEncodingStats *stats, int closeQT );
PROCEDURE CloseQTMovieSink( VAR qms: QTMovieSinks; addTCTrack: INTEGER;
                    stats: POINTER TO QTMSEncodingStats;
                    closeQT: INTEGER ) : ErrCode [CDECL];
```

Fonctions de support

```
PROCEDURE QTMSFrameBuffer( VAR qms: QTMovieSinks;
                    frameBuffer: UInt16 ) : QTFrameBuffer [CDECL];
```

retourne le pointeur vers la trame *frameBuffer*; *frameBuffer* est un entier >= 0 et < *qms.frameBuffers* .
En C: *qms->imageFrame[frameBuffer]*

```
PROCEDURE QTMSCurrentFrameBuffer( VAR qms: QTMovieSinks ) : QTFrameBuffer [CDECL];
```

retourne le pointer vers la trame image actuel.
En C: *qms->imageFrame[qms->currentFrame]* .

```
PROCEDURE QTMSPixelAddress( VAR qms: QTMovieSinks;
                    frameBuffer: UInt16; pixnr: UInt32 ) : QTAPixelPtr [CDECL];
PROCEDURE QTMSPixelAddressInCurrentFrameBuffer( VAR qms: QTMovieSinks;
                    pixnr: UInt32 ) : QTAPixelPtr [CDECL];
```

retourne un pointeur vers le pixel *pixnr* dans la trame *frameBuffer* ou actuelle. *pixnr* est un entier >= 0 et < *qms.Width * qms.Height* .
en C: *qms->imageFrame[frameBuffer][pixnr]* ou
qms->imageFrame[qms->currentFrame][pixnr].

```
Movie get_QTMovieSink_Movie( QTMovieSinks *qms );
PROCEDURE GetQTMovieSinkMovie( VAR qms: QTMovieSinks ) : Movie [CDECL];
```

retourne l'object *Movie* par exemple pour ajouter des méta données

```
int get_QTMovieSink_EncodingStats( QTMovieSinks *qms,
                    QTMSEncodingStats *stats );
PROCEDURE GetQTMovieSinkEncodingStats( VAR qms: QTMovieSinks;
                    VAR stats: QTMSEncodingStats ) : INTEGER [CDECL];
```

retourne une structure avec la durée et le nombre de trames par secondes actuels et le cas échéant quelques statistiques du processus d'encodage ICM.
Retourne *TRUE* en cas de succès. Les statistiques ICM comprennent le nombre de trames encodées jusqu'à maintenant ainsi que le nombre de trames perdues, combinées (*merged*), dont le temps a du être changé et par lesquelles de la mémoire a du être copiée (ce qui est un facteur de ralentissement mais inévitable pour certains codecs).

Exemple

Voici un exemple de création d'un *Movie* en Modula-2 (fragments):

```
FROM QTilsM2 IMPORT *;

VAR
   qmsErr : ErrCode;
```

```
   qms : QTMovieSinks;
   stats : QTMSEncodingStats;

   PROCEDURE generateImage( VAR qms : QTMovieSinks );
   VAR
      len, p : UInt32;
      fb : QTFrameBuffer;
   BEGIN
      len := VAL(UInt32, qms.Width) * VAL(UInt32, qms.Height);
      WITH QTils DO
         fb := QTMSCurrentFrameBuffer(qms);
         (*
            initialisation du pixel avec lequel on peindra
            l'image pour ce pas:
         *)
         p := 0;
         WHILE p < len DO
            IF qms.useICM = 0 THEN
                  fb^[p].ciChannel.red := currentRed();
                  fb^[p].ciChannel.green := currentGreen();
                  fb^[p].ciChannel.blue := currentBlue();
               ELSE
                  fb^[p].icmChannel.red := currentRed();
                  fb^[p].icmChannel.green := currentGreen();
                  fb^[p].icmChannel.blue := currentBlue();
            END;
            INC(p);
         END;
      END;
   END generateImage;

   PROCEDURE captureImage( VAR qms : QTMovieSinks );
   BEGIN
      WITH QTils DO
         glReadBuffer(GL_FRONT);
         glReadPixels(0, 0, qms.Width, qms.Height,
            GL_BGRA, GL_UNSIGNED_INT_8_8_8_8,
            QTMSCurrentFrameBuffer(qms)
         );
      END;
   END captureImage;

BEGIN
   IF NOT QTilsAvailable() THEN
      HALT;
   END;

   OpenQT();

   qmsErr := OpenQTMovieSink( qms, " test.mov", width, height, 1, fBuffers,
         QTCompressionCodec.JPEG, QTCompressionQuality.High,
         useICM, 0 );
   IF qmsErr = noErr THEN
         WHILE ( NotYetTheEND() AND qmsErr = noErr ) DO
            generateImage(qms);
            IF qms.useICM = 1 THEN
                  qmsErr := AddFrameToQTMovieSinkWithTime( qms, currentTime() );
```

```
            ELSE
               qmsErr := AddFrameToQTMovieSink( qms, frameDuration );
            END;
         END;
         (*
            ici on pourrait ajouter quelques méta-données, par exemple
            AddMetaDataStringToMovie( GetQTMovieSinkMovie(qms), MetaData.akComment,
                 "Movie généré avec QTMovieSinks pour Modula-2", "fr_FR" );
         *)
         qmsCloseErr := CloseQTMovieSink( qms, 1, ADR(stats), 0 );
         LogMsgEx( 'Enregistré %gs et %ld trames dans "%s" => %gfps; err=%d/%d\n',
                     stats.Duration, stats.Total,
                     qms.theURL, stats.frameRate, qmsErr, qmsCloseErr );
         IF qmsErr = noErr THEN
               qmsErr := qmsCloseErr;
         END;
   END;
   CloseQT();
END;
```

POSIXm2

POSIXm2 est un module pour Modula-2 qui fournit un petit nombre de fonctions de type 'POSIX' et/ou les différentes autres librairies standard de **C**, fournies par POSIXm2.dll . (Certaines de ces fonctions existent d'ailleurs également dans QTils pour limiter le nombre de dépendances à des DLLs externes.)

Initialisation:

```
FROM POSIXm2 IMPORT
   POSIX, POSIXAvailable;

(* … *)
   IF NOT POSIXAvailable()
      THEN
         HALT;
   END;
```

Après un initialisation réussie, les fonctions suivantes sont disponibles dans le tableau *POSIX* :

```
PROCEDURE sprintf( VAR buf : ARRAY OF CHAR;
   format : ARRAY OF CHAR; ...) : Int32 [RightToLeft, LEAVES, VARIABLE];
PROCEDURE sscanf( source : ARRAY OF CHAR;
   template : ARRAY OF CHAR; ...) : Int32 [RightToLeft, LEAVES, VARIABLE];
```

Interfaces vers les fonctions **C** du même nom.

```
PROCEDURE LogLocation( fName: ARRAY OF CHAR; lineNr: INTEGER32 ) [CDECL];
```

enregistre un nom de fichier (code source) et numéro de ligne valable pour le prochain appel à *LogMsg* ou *LogMsgEx*. Avec le compilateur Stony Brook, on peut utiliser l'appel *POSIX.LogLocation(SOURCEFILE,SOURCELINE)*.

```
PROCEDURE LogMsg( msg : ARRAY OF CHAR) : UInt32;
```

enregistre un message dans le journal; nécessite la présence de la fonctionnalité 'SS_Log' dans QTils.dll ! (sinon aucun effet) . À noter que la version Modula-2 de cette fonction honore les caractères spéciaux 'C': '\n' pour une nouvelle ligne, '\t' pour un caractère de tabulation, etc.

Le journal de POSIXm2 est distinct de celui de QTils, et la fenêtre de sortie ne s'ouvre qu'après le premier message enregistré.

```
PROCEDURE LogMsgEx( msg : ARRAY OF CHAR; ...) : UInt32 [RightToLeft,LEAVES, VARIABLE];
```

idem, mais permet la création d'un message formaté/structuré via la fonction wvsprintf. Par exemple *POSIX.LogMsgEx("Le code d'erreur est %d", errCode);*

```
lastSSLogMsg : POINTER TO ARRAY OF CHAR
```

Pointeur vers le dernier message enregistré dans le journal. Cette chaine est mise à jour même sans la fonctionnalité 'SS_Log' dans QTils.dll .

```
TYPE
   String1024 = ARRAY[0..1023] OF CHAR;
   String1024Ptr = POINTER TO String1024;
   ArgString  = String1024;
   ArgStringPtr = String1024Ptr;
   ArgVector = POINTER TO ARRAY[0..255] OF ArgStringPtr;
argc : CARDINAL16;
argv : ArgVector;
PROCEDURE Arg( arg: CARDINAL16 ) : ArgString [CDECL];
```

argc représente le nombre d'arguments avec lequel l'application a été invoquée, plus

1 (le nom de l'application, l'argument 0). Il s'agit de la convention de la fonction *main* de **C:** ***main(int argc, char *argv[])***. Les arguments peuvent être obtenus soit avec *POSIX.argv^[i]^*, soit *POSIX.Arg(i)*. Le découpage de la ligne de commande est fait par un algorithme qui préserve les espaces dans les arguments individuels s'ils ont été proprement entourés par des parenthèses - par exemple "*c:\Program Files\QTVODm2\Documentation\QTilities-v1.1.pdf*".
NB: il semblerait que cette fonctionnalité n'est pas disponible si POSIXm2.dll a été compilée en 'mode debug' !

```
PROCEDURE new_jmp_buf() : jmp_buf [CDECL];
PROCEDURE dispose_jmp_buf( VAR env: jmp_buf ) [CDECL];
PROCEDURE longjmp( env: jmp_buf; val: INTEGER ) [CDECL];
PROCEDURE setjmp( env: ARRAY OF CARDINAL ) : INTEGER [CDECL];
```

longjmp fait un saut programmé vers un endroit marqué dans le *jmp_buf* avec le code *val* qui ne peut pas être 0. L'endroit a du être marqué avec un appel préalable à *setjmp*; cette fonction retourne soit 0 (et aura alors marqué l'endroit et le contexte d'exécution actuelle), soit *val* (et on retourne alors d'un *longjmp*). Les fonctions *new_jpm_buf* et *dispose_jmp_buf* ne sont pas nécessaires en **C**.

```
PROCEDURE calloc( n: CARDINAL32; elemSize: CARDINAL32 ) : ADDRESS [CDECL];
PROCEDURE realloc( ptr: ADDRESS; memSize: CARDINAL32 ) : ADDRESS [CDECL];
PROCEDURE free( ptr: ADDRESS ) [CDECL];
PROCEDURE memset( VAR mem: ARRAY OF BYTE, val: BYTE,
            size: CARDINAL32 ) [CDECL];
PROCEDURE strstr( str: ARRAY OF CHAR;
            pattern: ARRAY OF CHAR ) : String1024Ptr [CDECL];
PROCEDURE strrstr( str: ARRAY OF CHAR;
            pattern: ARRAY OF CHAR ) : String1024Ptr [CDECL];
```

Interfaces vers les fonctions **C** du même nom.

Notes

Pour installer QTils, il suffit de copier QTils.dll et POSIXm2.dll dans un répertoire où le système la trouvera (le répertoire du logiciel et c:\windows étant les choix surs). Il est également possible d'utiliser l'installeur du lecteur QTVODm2 (QTVODm2-vXX-dev.exe) qui contient les codes source et librairies nécessaires pour le développement en Modula-2, ainsi que la documentation disponible en PDF et CHM (HTML compilé).

L'interface Modula-2 charge QTils.dll au démarrage du logiciel, et obtient ensuite les adresses de quelques fonctions, dont celle qui exportera les adresses des fonctions 'publiques' dans le tableau ('*RECORD*') QTils. L'initialisation de QuickTime reste à la charge de l'utilisateur et n'est pas obligatoire pour accéder aux types et constantes.
NB: Si la version QTils-dev.dll (POSIXm2-dev.dll) existe au lieu ou à côté de QTils.dll (POSIXm2.dll), c'est elle qui est chargée. Ceci permet d'exploiter la version *Develop* ponctuellement sans devoir recompiler QTilsM2.mod .

QTils.dll (POSIXm2.dll) est compilée avec MS Visual C++ 2008 ou 2010 (l'Express Edition suffit); projet QTils.sln (QTils-vs2010.sln) dans le répertoire QTils dans QTilities. Le SDK QuickTime doit être installé à l'endroit par défaut (C:\Program Files) pour compiler QTils. Les modes Debug et Develop (nommée *QTils-dev.dll*) utilisent *SS_Log* pour afficher des messages de suivi (log). QTils dépend également de *google-sparsehash*.
[http://www.codeproject.com/KB/macros/ss_log.aspx]
[http://code.google.com/p/google-sparsehash/]
L'interface Modula-2 est dans le répertoire Mod2 sous QTils (ou sous le répertoire d'installation choisi via l'installeur).
Code source :

```
\\pandore\USBShare\msis\rjvb\Libs
\\pandore\USBShare\msis\rjvb\QuickTimeStuff\QTilities
ftp://anonymous@ftp.inrets.fr/incoming/FtpSimU/logiciels/RJVB/QTilities.tar.bz2
ftp://anonymous@ftp.inrets.fr/incoming/FtpSimU/logiciels/RJVB/QuickTime-Installers/
QuickTimeSDK-73.zip
```

QTils.dll exporte ses fonctions non pas via un fichier *.def*, mais via les directives de compilation *__declspec(dllexport)* et *__declspec(dllimport)* :

```
#if defined(_WIN32) || defined(__WIN32__)
# ifdef _QTILS_C
#  define QTLSext __declspec(dllexport)
# else
#  define QTLSext __declspec(dllimport)
# endif
#else
# define QTLSext /**/
#endif
```

Sous Mac OS X, QTils vient en 2 formes: *QTils.framework* qui s'installe dans /Library/Frameworks ou $HOME/Library/Frameworks et *QTils-embed.framework* qui peut être inclut dans un *application bundle* (application 'Cocoa'). Ici, la librairie exploite les APIs *QTKit* et *Cocoa* le plus possible. Il n'y a pas d'interface Modula-2 sous Mac OS X.