LES FICHIERS

Définition

Les fichiers sont constitués d'une suite d'octets.

Ils sont soit de type BINAIRE soit de type TEXTE.

Pour les fichiers de type TEXTE, la plupart des octets représentent des caractères dits imprimables.

Des octets, non visibles, sont utilisés pour l'indentation (TAB) ou pour marquer la fin d'un ligne :

- CR/LF pour le format Windows
- CR pour le format Mac
- LF pour le format Unix

Les caractères d'un fichier TEXTE peuvent être codés de différentes manières :

- ASCII Windows Français
- ASCII Windows US
- ...
- UNICODE UTF-8 sans BOM (entête)
- UNICODE UTF-8 avec BOM
- UNICODE UTF-16 sans entête
- UNICODE UTF-16BE (Big-Endian)
- UNICODE UTF-16LE (Little-Endian)
- ...

Pour les modifier, on utilise des éditeurs de texte.

La plupart d'eux comprennent ces différentes formats et changent automatiquement la séquence de fin de ligne à l'enregistrement.

LECTURE d'un fichier TEXTE ou BINAIRE en C++ avec Qt

```
QByteArray BUFFER;
QString    LINE;

QFile Fd_R( PATH_Item );

bool b_OpenR = Fd_R.open( QIODevice::ReadOnly );

if ( ! b_OpenR ) {
  //-- Traitement Erreur Ouverture
  return;
}

qint64 NbR = Fd_R.read( BUFFER, Q_MAX_BUFFER );

while ( NbR > 0 ) {
  //-- Traitement des donnees
  NbR = Fd_R.read( PtrB, Q_MAX_BUFFER );
  if ( NbR == 0 ) {
    //-- Soit une Erreur, soit plus de données à lire
  }
}
Fd_W.close();
```

Ce programme utilise un tableau de BYTE qui récupère les octets du fichier.

Il ne permet pas de compter les caractères « é », par exemple, contenus dans le fichier, car il ne tient pas compte du format du fichier.

LECTURE d'un fichier TEXTE codé ISO-8859-1 (Latin1)

```
QFile Fd_R( PATH_Item );

if ( ! Fd_R.open( QIODevice::ReadOnly ) ) {
  //-- Traitement Erreur Ouverture
  return;
}

QString TEXT;

QTextStream qTS_R( &Fd_R );

qTS_R.setCodec( QTextCodec::codecForName( "ISO-8859-1" ) );

bool b_atEND = qTS_R.atEnd();

while( ! b_atEND ) {
  QString LINE = qTS_R.readLine().trimmed();
  LINE.replace( QChar( '\t' ), QChar( ' ' ) );

  TEXT += LINE + Qstring( "\n" );
  b_atEND = qTS_R.atEnd();
}
Fd_R.close();
```

Pour les fichiers codés en UNICODE, la détection du format est automatique.

Il n'est pas besoin de déclarer :

```
qTS_R.setCodec( QTextCodec::codecForName( "UTF-8" ) );
ou
qTS_R.setCodec( QTextCodec::codecForName( "UTF-16BE" ) );
ou ...
```

Écriture dans un fichier TEXTE au format UTF-8

```
QFile Fd_W( PATH_Item );

if ( ! Fd_W.open( QIODevice::WriteOnly ) ) {
  //-- Traitement Erreur Ouverture
  return;
}

QTextStream qTS_W( &Fd_W );
qTS_W.setCodec( QTextCodec::codecForName( "UTF-8" ) )

try {
  qTS_W << qTXT;
} catch (...) {
  //-- Traitement ERREUR
}
Fd_W.close();
```

Écriture dans un fichier TEXTE dans le format local :

```
qTS_W.setCodec( QTextCodec::codecForLocale() );
```

Différentes méthodes pour LIRE ou ÉCRIRE un fichier TEXTE

- Lecture Caractère par Caractère
- Lecture Bloc par Bloc (avec un Buffer Utilisateur)
- Lecture Ligne par Ligne
- Lecture Totale

Différentes méthodes pour LIRE ou ÉCRIRE un fichier BINAIRE

- Positionnement
- Accès direct par HASH CODING
- Accès direct à l'aide d'une bibliothèques d'Entrées / Sorties
- Lecture Totale

Exemples de Fichiers TEXTE codés en BINAIRE

- Anciennes versions de WinWord
- WordPad
- Anciennes versions de Libre Office
- PDF
- ...

Organisation de Fichiers BINAIRES

- Fichiers à enregistrements logiques
- Fichiers à accès direct : exemple B-TREE
- Exécutables binaires
- Archives (zip, rar, tar)
- Bibliothèques (library) de modules object
- Données aux formats binaires (entiers 32 ou 64 bits, réels 32 ou 64 bits, caractères...)
- Codage des entités en XDR (eXternal Data Representation)

Fichiers spéciaux

- Raccourcis sous Windows
- Liens symboliques ou physiques sous UNIX
- Fichier /proc/cpuinfo
- Fichier /proc/meminfo
- Fichier /proc/bus/usb/devices

Dossiers spéciaux

- System Volume Information
- $RECYCLE.BIN
- /dev

Attributs de fichiers

- Nom du fichier
- Propriétaire
- Date de Création (naissance)
- Date de Création (copie dans le dossier)
- Date de Modification (dernier accès en écriture)
- Date de dernier accès en lecture (par des utilisateurs)
- Taille en octets
- Indication de présence sur disque dur ou de migration
- Droits (utilisateur, groupe, public, système...)
- Mots de passe (lecture, écriture, maintenance, …)
- Date de dernier archivage par l'utilisateur, pas le système
- Attributs SYSTEM, HIDDEN, etc
- …

```
//
// <<<< TL_Get_LastStatusChange
//-- http://www.cplusplus.com/reference/ctime/strftime/
//
QString TL_Get_LastStatusChange( QFileInfo * p_FI )
{
  QString LastStatusChange;
  QString PathName = p_FI->absoluteFilePath();

  struct stat stat_buf;
  char buffer [80];
  struct tm * timeinfo;

  if ( stat( PathName.toLocal8Bit(), &stat_buf ) != -1 ) {
    timeinfo = localtime( &stat_buf.st_ctime );
    strftime( buffer, 80, "%d.%m.%Y %H.%M.%S", timeinfo ); //-- 03.05.2016 13.15.08
    LastStatusChange = buffer;
    //-- LastStatusChange = ctime( &stat_buf.st_ctime );
  }
  return LastStatusChange;
}
// >>>> TL_Get_LastStatusChange
```

Autres Attributs de fichiers

- Durée pour les fichiers MULTIMEDIA
- Auteur, Titre, etc
- Caractéristiques : qualité, etc

Exemple - Fichier au format WAV

L'en-tête d'un fichier WAV commence dès le premier octet (*offset* 0).
Il a une taille de 44 octets, et est constitué des champs suivants (listés dans l'ordre) :

[**Bloc de déclaration d'un fichier au format WAVE**]
   FileTypeBlocID (4 octets) : Constante «RIFF» *(0x52,0x49,0x46,0x46)*
   FileSize   (4 octets) : Taille du fichier moins 8 octets
   FileFormatID   (4 octets) : Format = «WAVE» *(0x57,0x41,0x56,0x45)*

[**Bloc décrivant le format audio**]
   FormatBlocID   (4 octets) : Identifiant «fmt » *(0x66,0x6D, 0x74,0x20)*
   BlocSize   (4 octets) : Nombre d'octets du bloc - 16 *(0x10)*

   AudioFormat   (2 octets) : Format du stockage dans le fichier *(1: PCM, ...)*
   NbrCanaux   (2 octets) : Nombre de canaux *(de 1 à 6, cf. ci-dessous)*
   Frequence   (4 octets) : Fréquence d'échantillonnage (en hertz)
                                [Valeurs standardisées : 11025, 22050, 44100 et éventuellement 48000 et 96000]

   BytePerSec   (4 octets) : Nombre d'octets à lire par seconde *(i.e., Frequence * BytePerBloc)*.
   BytePerBloc   (2 octets) : Nombre d'octets par bloc d'échantillonnage
                                *(i.e., tous canaux confondus : NbrCanaux * BitsPerSample/8)*.
   BitsPerSample   (2 octets) : Nombre de bits utilisés pour le codage de chaque échantillon (8, 16, 24)

[**Bloc des données**]
   DataBlocID   (4 octets) : Constante «data» *(0x64,0x61,0x74,0x61)*
   DataSize   (4 octets) : Nombre d'octets des données *(i.e. "Data[]",*
                                i.e. taille_du_fichier - taille_de_l'entête (qui fait 44 octets normalement).
   DATAS[] : [Octets du **Sample 1** du *Canal 1*] [Octets du **Sample 1** du *Canal 2*]
                                [Octets du **Sample 2** du *Canal 1*] [Octets du **Sample 2** du *Canal 2*]

   * Les Canaux :
      1 pour mono,
      2 pour stéréo
      3 pour gauche, droit et centre
      4 pour face gauche, face droit, arrière gauche, arrière droit
      5 pour gauche, centre, droit, surround (ambiant)
      6 pour centre gauche, gauche, centre, centre droit, droit, surround (ambiant)

NOTES IMPORTANTES : Les octets des mots sont stockés sous la forme *(i.e., en « little endian ») → octet de poids le plus faible en premier*