小杉 智昭

株式会社ミガロ.

システム事業部 プロジェクト推進室

Windowsテキストファイル操作ノウハウ

**他システムとの連携役としてのCSVデータ出力、内部統制対応としてのログ出力。
IBM i上のファイルとは一味違う、Windowsのファイル操作を試みる。**

略歴
1973年05月26日生
1996年関西大学工学部卒
2002年03月株式会社ミガロ. 入社
2002年03月RAD事業部配属
2007年04月システム事業部配属

現在の仕事内容
Delphi/400を利用した受託開発とシステム保守、導入支援を担当している。

1.はじめに

最近は利用者向けソフトの普及や機能向上もあり、システム開発側で利用者の求める出力をすべて取り揃えなくても、ベースとなるCSVデータを用意しておけば、PC上で表計算ソフト等を使って自由に編集して利用するユーザーが増えてきた。

また、複雑化したシステムのエラー原因の究明や利用者の不正利用を防止する目的で、ログ出力機能の重要度が増してきている。そのような意図で出力されるログについては、データ容量が非常に大きくなることもあるので、よりディスク単価の安いPCサーバー等を活用することができる。

上記を受け、本稿では、他システムとの連携役としてのCSVデータ出力、および内部統制対応としてのログ出力に注目し、Delphi/400を使ってWindows上のファイル操作を行う、ということを取り上げる。具体的に、Windowsテキストファイルの操作ノウハウやポイントなどを紹介する。

2.ファイル操作の基本

Delphi/400を使ってWindows上のファイル操作を行うには、同じ内容を実現するとしても、さまざまな方法がある。

この章では、ファイル操作にはまずどのような手法があるのか、またそれぞれの手法の特徴はどういったものなのかを、OSに近いものから順にご紹介する。

・Windows APIを利用するファイル操作
データ型：例）THandle
命令群：例）FileOpen / FileClose / FileCreate

OSに一番近い形のファイル操作方法が、「Windows APIを利用したファイル操作」である。

この方法は「ファイルハンドル」と呼ばれるものを使い、ファイル操作を行う。そのため、ハンドルベースで各種APIを利用できるが、個々のAPIそのものは非常に限定された機能しか持たず、またOSバージョン等に依存することがある。

・低レベル命令群を利用するファイル操作
データ型：例）File / TextFile
命令群：例）AssignFile / CloseFile / Append / Reset / Rewrite

Delphi/400で提供されているファイル操作の命令群の中でも、よりOSに近い（これを指して「低レベル」と呼ばれている）基本命令群を利用する。このファイル操作方法が、低レベル命令群を利用するファイル操作である。

それらはDelphi/400におけるファイル操作の基本となっているが、個々の命令でできる機能は非常に限定されているため、ログ出力のような単純な機能で使用される。

・TStream継承クラスによるファイル操作
クラス：例）TFileStream

ソース1

```
1  procedure TfrmCSVSample.btnExportClick(Sender: TObject);
2  var
3    slFile: TStringList;   // 出力ファイルを示すクラス
4    sLine: String;         // CSVデータの一行を示す文字列
5    i: Integer;
6  begin
7    // TStringListクラスの作成
8    slFile := TStringList.Create;
9    try
10     // データの先頭へ移動
11     cdsSample.First;
12
13     // データの終了まで繰返処理
14     while not cdsSample.Eof do
15     begin
16       // 文字列を初期化する
17       sLine := EmptyStr;
18       // 同一レコードを項目の数だけ繰返処理
19       for i := 0 to cdsSample.FieldCount - 1 do
20       begin
21         // 項目属性毎に出力方法を変更する
22         case cdsSample.Fields[i].DataType of
23           // 文字列：     ダブルクォーテーションで囲む
24           ftString:
25             sLine := sLine + AnsiQuotedStr(cdsSample.Fields[i].AsString, '"');
26
27           // 整数・実数：そのまま文字列化する
28           ftInteger, ftFloat:
29             sLine := sLine + cdsSample.Fields[i].AsString;
30
31         else
32           // 上記以外：   エラーとする
33           sLine := sLine + '#ERROR#';
34         end;
35
36         // 項目毎の区切り文字としてカンマを追加する
37         sLine := sLine + ',';
38       end;
39       // カンマが１つ多くついているため、最後のカンマを削除する
40       Delete(sLine, Length(sLine), 1);
41
42       // 出来上がったCSVデータ（１行分）をStringListクラスに追加する
43       slFile.Add(sLine);
44
45       // 次のデータへ移動
46       cdsSample.Next;
47     end;
48
49     // 全レコード分のCSVデータを'Sample.csv'という名前でファイルに保存する
50     slFile.SaveToFile('Sample.csv');
51
52   finally
53     // TStringListクラスの廃棄
54     slFile.Free;
55   end;
56
57 end;
```

＜コード例①＞

Delphi/400では、さまざまなデータの集合を取り扱うクラスとして、「ストリーム」と呼ばれるクラスが提供されている。その中でも、ファイル操作に特化したのがTFileStreamというクラスになる。

このクラスを使用することで、ファイルのオープン・クローズ制御やファイル作成、モード指定等の煩雑な処理を簡単に実現できる。

また、他のストリーム系クラスとの共通機能を利用することで、メモリ上に展開したデータのファイル保存や、読み込んだファイル内容を通信データにのせてダウンロードする機能を提供するといったことに利用されている。

・TStrings継承クラスによるテキストファイル操作

クラス：例）TStringList

TStrings継承クラスは、どちらかというとファイル操作用ではなく、文字列操作用のクラスであり、それにファイル出力機能が備わっているものである。

しかしながら、CSVデータの編集機能を備えており、利用頻度が高い。そのため、TStrings継承クラスによるテキストファイル操作が行われている。

なお、この具体的な使用方法は、次章でご紹介する。

3.ファイル操作の具体例

前章で挙げたファイル操手法のいくつかを使って、実際にファイル出力する例をご紹介する。

CSVデータ出力の具体例

CSVとはデータの内容をテキスト化し、順番に出力したものである。出力する際、項目ごとの区切りはカンマで、レコードごとの区切りは改行で行う。

古くから利用されているにもかかわらず、詳細な仕様が決定されたのは、2005年と比較的新しいため、仕様確定前から使われていたルールが非常に多く存在する。

例えば、IBM i Client Access等でCSV出力を行うと、文字項目がダブルクォーテーションで囲われる。これをExcelで保存し直すと、通常の文字項目はダブルクォーテーションが外されてしまう。これは仕様の違いによる問題である。

今回は、基本的なCSVのルールに従うが、より互換性を高めるために、文字列を出力する場合は必ずダブルクォーテーションで囲うルールとして実装を考える。

CSVデータ出力の基本的な処理手順は以下のようになる。

【CSVデータ出力の処理手順】
・データの先頭に移動
・データを最終まで繰返処理
**　○項目ごとの繰返処理**
**　　◎項目の種類ごとに異なるルールでテキスト化**
**　　◎カンマを追加**
**　○出力用ワークに1レコード分のCSVデータを追加**
**　○次のデータへ移動**
・全データ分のCSVデータをファイル出力

では実際に、cdsSampleというTClientDataSetのデータを、TStringListクラスを使ってSample.csvファイルに出力する。この例を、ソース1として示す。【ソース1】

forを使った内側の繰り返し処理は、同一レコードの全項目に対する処理である。whileを使った外側の繰り返し処理は、全レコードに対する処理になる。また、網掛けの箇所でCSVファイルを保存している。

CSVファイルの場合、既存のCSVファイルが存在しても、追記せずに既存ファイルを破棄して新規にファイルを作成することが一般的である。そのため、TStringListクラスを使うのが適している。

しかし、すべてのデータをメモリ上に展開するため、あまりに膨大なデータを処理するのは向かない点に注意が必要だ。大量のデータを処理するなら、TFileStream等を利用するほうが適している。

ログ出力の具体例

ログ出力は、何らかの操作やエラー、警告といったものを時系列で絶えず出力していくことが肝要となる。また、複数の機能から同じログファイルに出力したり、前回の続きに出力することが行われる。そのため、処理開始時に既存のログ内容をメモリ上に展開して、処理終了時に編集されたログ内容を出力するといった使い方は避けるべきである。

今回は、「年月日時分秒ミリ秒 + 区切り文字 + 出力メッセージ」という出力フォーマットでログを出力する。

ログ出力の基本的な処理手順は以下のようになる。

【ログ出力の処理手順】
・ログファイルが存在するか確認する
**　○ログファイルが既存であれば追記モードでファイルを開く**
**　○ログファイルが存在しなければ新規作成する**
・出力フォーマットに合わせたログデータを作成する
・ログデータをファイルに出力する
・ログファイルを閉じる

では実際に、低レベルなファイル操作命令群を使ってSample.logファイルに出力する。この例を、ソース2として示す。【ソース2】

出力前にログファイルが既存かどうかを調べ、追記するか、新規作成するかを判断している。また、網掛けの箇所でログファイルに対する出力を行っている。

ログ出力を行う場合、ファイルに対する操作が単純であるため、低レベル命令群でも記述しやすい。また、ここで使った命令は、既存ファイルの読み込みやメモリ上への展開といったことを一切行っていないため、ファイルサイズの影響をほとんど受けない。

以上、CSVデータ出力とログ出力といった2つの機能を、TStrings継承クラスと低レベル命令群を使った手法で実装した。

このサンプルコードは、ファイル名が固定になっている等、紹介用に若干機能を割愛しているが、多少の手直しで十分利用していただけるであろう。

ソース2

```
1  procedure LogOutput(sMsg: String);
2  var
3    logFile: TextFile;
4    sLogMsg: String;
5  begin
6
7    // ファイル名とファイル変数を関連付ける
8    AssignFile(logFile, 'Sample.log');
9
10   // Sample.logファイルが存在するか確認する
11   if FileExists('Sample.log') then
12
13     // Sample.logファイルが存在する場合、追記モードでファイルを開く
14     Append(logFile)
15
16   else
17
18     // Sample.logファイルが存在しない場合、新規作成する
19     Rewrite(logFile);
20
21
22   // 出力フォーマットに合わせた形にメッセージを加工する
23   // 書式：yy/mm/dd hh:nn:ss.zzz> (Message)
24   sLogMsg := FormatDateTime('yyyy/mm/dd hh:nn:ss.zzz', Now) + '> ' + sMsg;
25
26   // 加工済みのメッセージをログファイルに出力する
27   WriteLn(logFile, sMsg);
28
29   // ログファイルを閉じる
30   CloseFile(logFile);
31
32 end;
```

＜コード例②＞

4.ユニコードテキストの出力

Delphi/400は2009以降のバージョンで、ユニコード対応が行われた。しかし、ファイル出力機能は、Delphi/400のバージョンにかかわらずANSIベースで行われる。

つまり、前章で紹介した例を使ってファイル出力すると、ユニコードにしか存在しない文字は「?」に変換されてしまう。

IBM iでは専用のCCSIDを指定する必要があるため、ユニコードが使われない場合もあるが、他システムとの連動等を考慮し、ユニコード対応する例も紹介しておこう。

ユニコードテキストの種類

そもそも「ユニコード」とは一体何だろうか。

この問いに正確に答えることは意外と難しい。非常に乱暴な表現となるが、“これまでコンピュータ上で扱えなかったたくさんの文字の集まりとその扱い方をまとめたルール”これを指してユニコードと呼ばれることが多いようである。

しかし、上記したように文字の集まりとその扱い方ルールという2種類を指しており、また、それぞれが複数の種類に分かれている。UCS-2やUTF-16といった用語を聞いたことはないだろうか。それらが、ユニコードと呼ばれるものの正体に近い。

コラム的に少しだけ細かい説明をしたいと思う。プログラムには直接関係がなく、若干複雑な内容であるため、UTF-8とUTF-16、エンディアンという3つの用語だけ覚えておけば、以下の2段落（※）は読み飛ばしてもかまわない。

※その1

まず、文字の集まりとしてのユニコードだが、厳密にはUnicode、UCS-2、UCS-4といったものがある。このうちUnicodeは、2013年8月現在でバージョン6.2まで拡張されている。現時点では、取り扱い可能な文字の多さで並べれば、UCS-2 ＜ Unicode ＜ UCS-4となるが、主流はUnicodeである。

※その2

続いて、文字の取り扱いルールとしてのユニコードだが、厳密にはUTF-8、UTF-16、UTF-32といったものがある。これらは本来、ユニコードを取り扱うルールであって、ユニコードそのものではない。ただし、実際のプログラムにはこちらの影響が大きく、開発者サイドには馴染みのあるものになっているだろう。

インターネットでは、アルファベット等でASCIIと互換性のあるUTF-8が主流であるが、漢字や仮名をUTF-8で取り扱うとほとんどが3バイト必要になるため、従来のShift_JISに比べても効率が悪くなってしまうことが多い。

そのためか、ユニコード対応版のDelphi/400は、文字データを基本的にUTF-16で取り扱っている。

UTF-16の場合、かなり特殊な文字以外はすべて2バイトで表現するようになっている。多バイトデータになるため、これをファイルに保存する際にはCPUアーキテクチャの影響を受けることになる。いわゆるエンディアンの問題である。エンディアンを明示的に指定しなければ、リトルエンディアンとなる。

上記のように、一口にユニコードと言ってもいろいろな種類があり、テキスト保存の際にはどういった形式で保存すればいいのか気を付ける必要がある。しかし、よく使われる形式は、UTF-8またはUTF-16のリトルエンディアンであろう。

ちなみに、Windows標準のメモ帳でUnicodeを選択すると、UTF-16のリトルエンディアンになる。

では、次から、前章のデータ出力例のユニコード対応、具体的にはUTF-16のリトルエンディアンで保存するケースを示していこう。

なお、ユニコード対応が行われたDelphi/400 2009以降のバージョンを前提としている。

CSVデータ出力例をユニコード対応する

TStringListは、エンコーディングを指定するだけでユニコード対応が可能である。修正したコード例を、ソース3として示す。【ソース3】

コード例に示したように、TStringListクラスのSaveToFileメソッドにパラメーターを追加するだけである。

もし、UTF-8で保存したければ、50行目にあるSaveToFileメソッドの第2パラメーターをTEncoding.UTF8に変更するだけである。

ログ出力例をユニコード対応する

低レベル命令群でユニコードの出力を行うのは若干面倒なことと、TStream継承クラスを使ったサンプルプログラムも示しておきたいので、こちらのユニコード対応はTFileStreamクラスを使用するように変更する。

基本的な処理手順は、前述の低レベル命令群を使ったものと同じである。ただし、ユニコード化の影響として、以下のポイントに気を付ける必要がある。

- **UTF-16のテキストには、BOM（バイトオーダーマーク）が必要となる。**
- **出力データを、バイト配列に変換してから出力する。**

TFileStreamクラスを使用したコード例をソース4として示す。【ソース4】

低レベル命令群からのTFileStreamへの置き換えとユニコード対応を同時に行ったため、大きく変わってしまったが、ボリュームは全体でも50行未満とそれほどでもないことがわかる。

ポイントとしては、TEncodingを使い、ログファイルを新規作成する場合にBOMを付加することと、出力したいメッセージを予め変換しておくことである。また、網掛けの箇所でメッセージの変換と出力を行っている。

なお、こちらの例をUTF-8に対応させるには、12行目のENC変数への代入をTEncoding.UTF8にするだけである。

5.まとめ

今回は、Windows上のテキストファイルを取り扱う代表的なパターンを2つ紹介した。さまざまなシステムが入り乱れる昨今、他システムとの連携役として

ソース3

```
procedure TfrmCSVSample.btnExportClick(Sender: TObject);
var
  slFile: TStringList;   // 出力ファイルを示すクラス
  sLine: String;         // CSVデータの一行を示す文字列
  i: Integer;
begin
  // TStringListクラスの作成
  slFile := TStringList.Create;
  try
    // データの先頭へ移動
    cdsSample.First;

    // データの終了まで繰返処理
    while not cdsSample.Eof do
    begin
      // 文字列を初期化する
      sLine := EmptyStr;
      // 同一レコードを項目の数だけ繰返処理
      for i := 0 to cdsSample.FieldCount - 1 do
      begin
        // 項目属性毎に出力方法を変更する
        case cdsSample.Fields[i].DataType of
          // 文字列：    ダブルクォーテーションで囲む
          ftString:
            sLine := sLine + AnsiQuotedStr(cdsSample.Fields[i].AsString, '"');

          // 整数・実数：そのまま文字列化する
          ftInteger, ftFloat:
            sLine := sLine + cdsSample.Fields[i].AsString;

        else
          // 上記以外：  エラーとする
          sLine := sLine + '#ERROR#';
        end;

        // 項目毎の区切り文字としてカンマを追加する
        sLine := sLine + ',';
      end;
      // カンマが１つ多くついているため、最後のカンマを削除する
      Delete(sLine, Length(sLine), 1);

      // 出来上がったCSVデータ（１行分）をStringListクラスに追加する
      slFile.Add(sLine);

      // 次のデータへ移動
      cdsSample.Next;
    end;

    // 全レコード分のCSVデータを'Sample.csv'という名前でファイルに保存する
    slFile.SaveToFile('Sample.csv', TEncoding.Unicode);

  finally
    // TStringListクラスの廃棄
    slFile.Free;
  end;

end;
```

＜コード例③＞

の CSV データ出力、および内部統制対応としてのログ出力の需要や重要度はますます高まってくると思われる。

また、Delphi/400 では、上記以外の固定長テキストや ini ファイルといったテキスト形式のファイルだけでなく、バイナリ形式のファイルも簡単に取り扱うことが可能である。

ぜひこの機会に、IBM i 上のファイルとは一味違う、Windows のテキストファイル操作を試していただければ幸いである。

M

ソース4

```
procedure LogOutput(sMsg: String);
const
  cCRLF = #$000D + #$000A;
var
  ENC: TEncoding;
  fslog: TFileStream;
  Preamble, Buffer: TBytes;
  sLogMsg: String;
begin

  // エンコーディングを変数化する(UTF-16 リトルエンディアン)
  ENC := TEncoding.Unicode;

  // Sample.logファイルが存在するか確認する
  if FileExists('Sample.log') then
  begin

    // Sample.logファイルが存在する場合、ファイルを開き、最後尾に位置付けする
    fslog := TFileStream.Create('Sample.log', fmOpenReadWrite);
    fslog.Position := fslog.Size;

  end
  else
  begin

    // Sample.logファイルが存在しない場合、新規作成し、BOMを付加する
    fslog := TFileStream.Create('Sample.log', fmCreate);
    Preamble := ENC.GetPreamble;
    if Length(Preamble) > 0 then
      fslog.WriteBuffer(Preamble[0], Length(Preamble));

  end;

  // 出力フォーマットに合わせた形にメッセージを加工する
  // 書式：yy/mm/dd hh:nn:ss.zzz> (Message)
  sLogMsg := FormatDateTime('yyyy/mm/dd hh:nn:ss.zzz', Now) + '> ' + sMsg;

  // 加工済みのメッセージをログファイルに出力する
  Buffer := ENC.GetBytes(sLogMsg + cCRLF);
  fslog.WriteBuffer(Buffer[0], Length(Buffer));

  // ログファイルを閉じる
  fslog.Free;;

end;
```

＜コード例④＞