田村 洋一郎 / 宮坂 優大 / 都地 奈津美

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400]

IBM iデータベースへのFTP送信手法の紹介

1. はじめに

外部システムのデータやユーザーが作成したデータを CSV ファイルに保存し、IBM i データベースに更新する際、SQL や RPG で更新処理を実装しようとすると、更新先のデータベースごとにプログラムを開発する必要がある。【図 1】

また iSeries Access for Windows や PCOMM のデータ転送機能を利用して更新する際も、データベースごとに設定ファイルを作成する必要がある。

本稿では、データベースごとにプログラムを開発することなく汎用的にデータを更新する手法として、FTP 送信処理を紹介する。FTP 処理の手法については、『ミガロ . テクニカルレポート 2014』の「大量データ処理テクニック ～ FTP を利用したデータ転送～」にも紹介されているので、ぜひこちらも参考にしていただきたい。

本稿では、まず第 2 章で IBM i データベースへダイレクトに FTP 送信する方法を紹介する。次に第 3 章では、第 2 章で作成した送信処理をもとに送信先のファイルレイアウトを考慮し、汎用的に送信する手法について紹介する。【図 2】

2. IBM iデータベースへのFTP送受信処理

2-1. FTP送受信処理について

ここでは「Indy」を使用し、TIdFTP コンポーネントを用いて FTP サーバーとの通信を行う方法、並びに FTP サーバーに IBM i データベースを設定し、CSV ファイルを送信する手法を紹介する。

「Indy」とは、Delphi/400 で使用できるオープンソースのネットワーク関連コンポーネント群のことであり、Delphi/400 に標準で付属している。また FTP（File Transfer Protocol、ファイル転送プロトコル）とはサーバー間、またはサーバー・クライアント間でファイル送受信を行う際に利用される手法の 1 つである。

略歴 田村 洋一郎
1983 年 9 月 27 日生まれ
2006 年 3 月 近畿大学 理工学部卒業
2006 年 4 月 株式会社ミガロ. 入社
2006 年 4 月 システム事業部配属

現在の仕事内容
RPG や Delphi/400 などの開発経験を経て、現在は要件定義から安定稼働フォローまで、システム開発全般に携わっている。

略歴 宮坂 優大
1982 年 11 月 19 日生まれ
2006 年 3 月 近畿大学 理工学部卒業
2006 年 4 月 株式会社ミガロ. 入社
2006 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を利用したシステムの受託開発をメインに担当。Delphi および Delphi/400 のスペシャリストを目指して精進する日々である。

略歴 都地 奈津美
1989 年 8 月 19 日生まれ
2012 年 3 月 関西学院大学 理工学部卒業
2012 年 4 月 株式会社ミガロ. 入社
2012 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を使用したシステム受託開発とシステム保守を担当。開発スキルの向上を目指し、日々精進している。

図1　IBM iデータベースへの取込例1(データベース毎にプログラム開発)

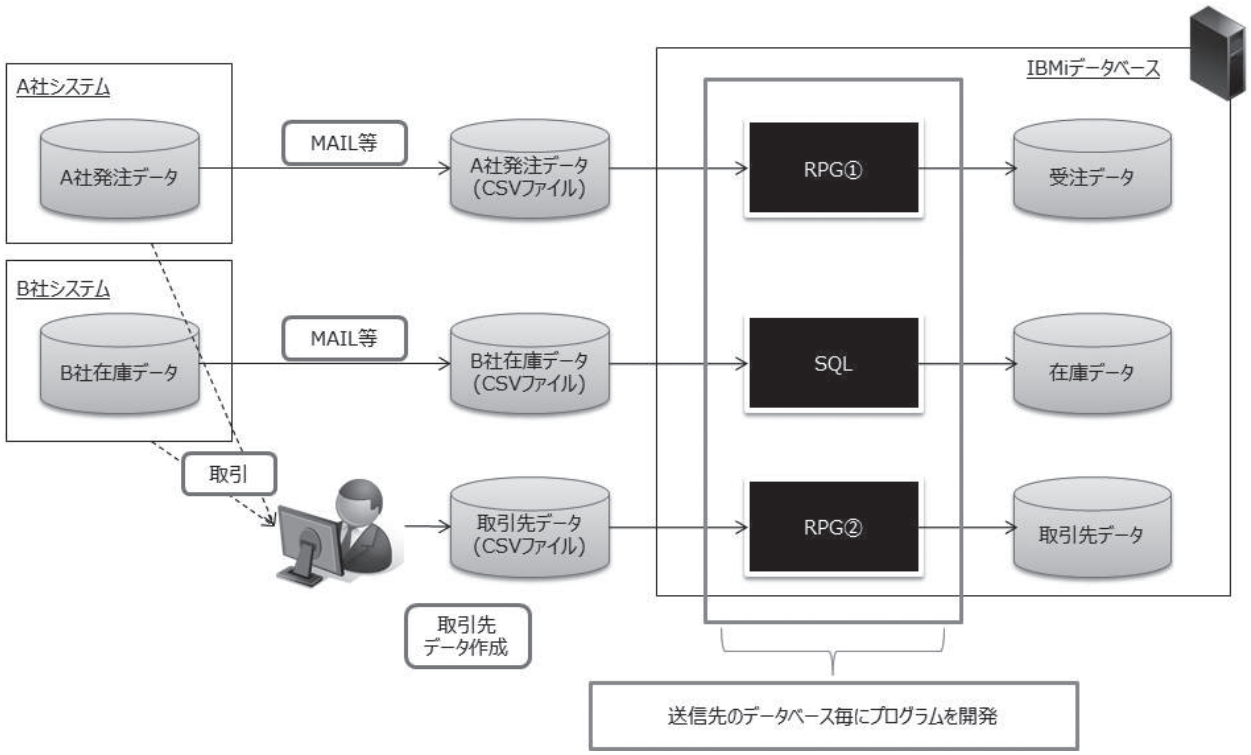
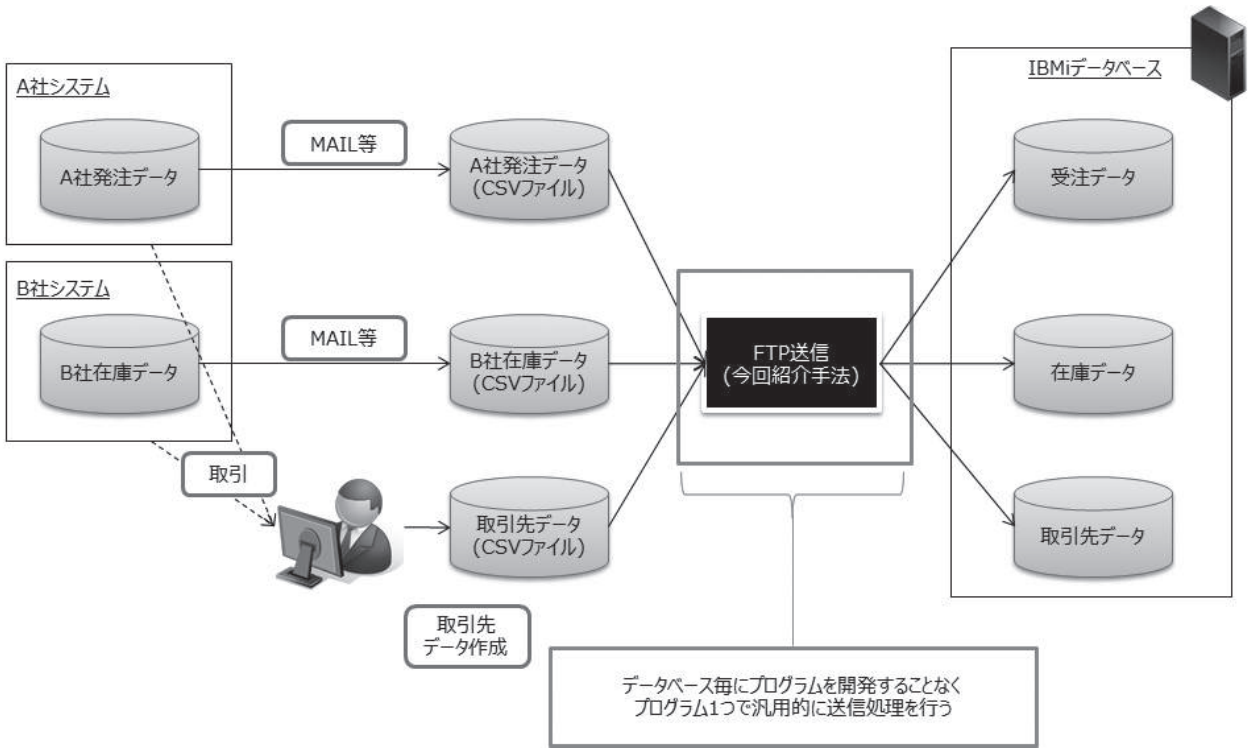


図2　IBM iデータベースへの取込例2(汎用的に送信処理を開発)



Indy のコンポーネントの 1 つである TIdFTP を利用することで、IBM i データベースやファイルサーバー等とクライアントの間で、FTP 通信を利用したファイルの送受信を行える。

なお、本章で作成しているプログラムは、Delphi/400 10.2Tokyo を使用している。

2-2. FTP送受信処理の前提条件

まず、IBM i データベースに対して FTP を利用したファイルの送受信を行うための前提条件として、IBM i データベースが FTP 送受信を許可しているかを確認する必要がある。

5250 画面で「NETSTAT *CNN」コマンドを実行すると、IPV4 接続状況の一覧が表示される。

接続状況一覧の中で、「ftp-con」または 21 番のローカルポートが「接続待機」になっていれば、FTP 接続が許可されている。【図 3】

存在しない場合は、5250 画面で「STRTCPSVR SERVER(*FTP)」コマンドを実行すると、FTP 接続を許可できる。また、IBM i で接続待機になっていても FTP 通信ができない場合、クライアント側のファイアウォールの設定で IBM i との FTP 通信が許可されているかを確認する。

次に、送信する CSV ファイルの書式について解説する。

CSV ファイルの項目の並び順および桁数は、送信先となるデータベースファイルのフィールド順と揃える必要がある。

今回使用する送信先のファイルのレイアウトを確認する。送信先のファイルは、次のように定義している。【図 4】

・半角文字フィールド（A タイプ 10 桁）
・全角文字フィールド（O タイプ 20 桁）
・数値フィールド（S タイプ 10 桁）

送信する CSV ファイルは、【図 5】のように作成した。

2-3. IBM i データベースへのアップロード処理

2-2. の準備事項が確認できたら、基本的な FTP 送信を行うプログラムを実際に作成していく。

(1) コンポーネントの配置

接続先情報と送信元ファイルを設定するための TEdit、FTP 通信を行うための TIdFTP、並びに TLabel や TButton をそれぞれ画面に配置する。【図 6】

送信元となる CSV ファイルは、画面上の「転送元 CSV ファイル」で設定する。また、送信先は「転送先ライブラリ名」「転送先ファイル名」に設定する。

(2) FTP 送信処理の実装

配置した「FTP 送信開始」ボタンの OnClick 処理を作成したら、実際に FTP 送信するソースコードを記述する。

次に、TIdFTP コンポーネントが持つプロパティや、今回の処理で行っているメソッドについて解説する。

①接続設定と接続処理

Host、Username、Password の各プロパティに、FTP 通信を行うための値を設定する。【ソース 1-①】

Host には接続先 IBM i データベースの IP アドレスを、Username と Password には IBM i データベースサインオン時のユーザー名とパスワードを設定する。

接続設定が完了したら、Connect メソッドで接続する。接続後は、try ～ finally で処理を囲み、処理終了後は Disconnect メソッドで接続を終了する。

② Passive プロパティ

パッシブモードで送信するため、Passive プロパティは True を設定する。【ソース 1-②】

③ TransferType プロパティ

ftASCII、ftBinary の 2 種類が存在し、ファイルの送受信をテキストファイル形式、バイナリ形式のどちらで行うかを設定できる。IBM i データベースは EBCDIC のため、ftBinary を設定する。なお、ftBinary は IdFTPCommon.pas で定義されているため、uses に「IdFTPCommon」を追加する。【ソース 1-③】

④送信元 CSV ファイルの文字列成型（CreateString メソッド）

FTP 送信する CSV ファイルは、カンマ区切りで構成されている。そのまま FTP 送信するとカンマ自体もデータとして送信されるので、文字列成型を行う。文字列の成型方法については以下、④-1. ～④-3. に記載する。

④-1. 送信元 CSV ファイルの読み込み

TStringList（slReadCSV）を生成し、LoadFromFile メソッドにて IBM i データベースへ FTP 送信する CSV ファイルを設定し、ファイルを読み込む。読み込んだ CSV ファイルをさらに 1 行ごとに読み込み、以下④-2. の処理で文字列成型を行う。【ソース 2-④-1】

④-2. 読み込んだファイルの文字列成型

TStringList（slRowCSV）の Comma Text に対して、④-1. で読み込んだデータ（slReadCSV）を代入する。slRowCSV の各項目を文字列として接続し、String 型変数（sCVSText）へ代入し、カンマを除去する。カンマを除去した sCSVText を、ファイル保存用 TStringList（slSaveCSV）に追加する。【ソース 2-④-2】

④-3. 文字列成型後の CSV ファイルを保存

カンマを除去した文字列を保管した TStringList（slSaveCSV） を、SaveToFile メソッドにて保存する。【ソース 2-④-3】

⑤ Put メソッド

各プロパティの設定および送信元 CSV ファイルの成型が完了したら、Put メソッドを呼び出し、FTP 送信処理を行う。引数で設定した条件で、ファイルを FTP 送信先にアップロードする。第 2 以降の引数は省略可能である。

第 1 引数：

アップロード元ファイルのフルパスを設定する。(1) で配置した「転送元 CSV ファイル」を設定する。

第 2 引数：

アップロード先のファイル名を設定する。(1) で配置した「転送先ライブラリ名」「転送先ファイル名」を設定する。ブランク設定時または省略時は、第 1 引

図3　IBM iのFTP送受信許可の確認

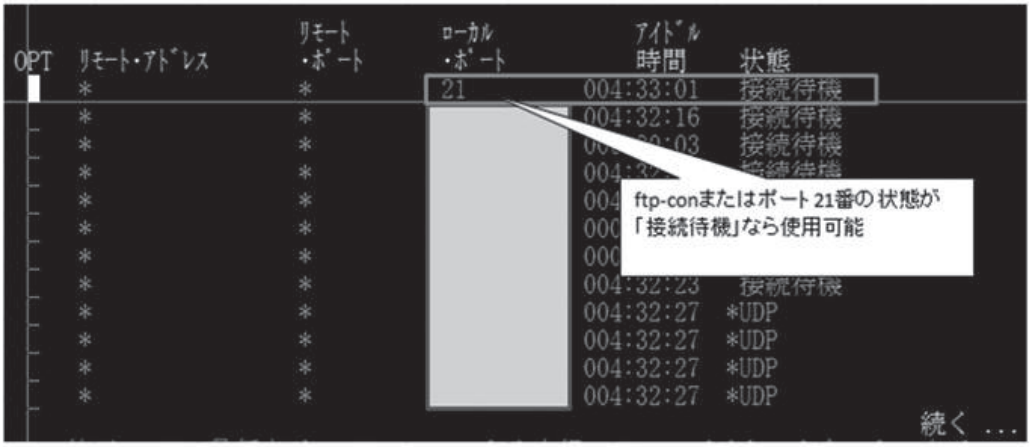


図4　送信先ファイルレイアウト

```
A*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
********** データの始め **********************************************
A*-------------------------------------------------------------------*
A*       ファイル ID: FTPTEST
A*       ファイル名 :  サンプルファイル
A*-------------------------------------------------------------------*
A          R TESTR                        TEXT(' テスト ')
A            DATA1          10A          COLHDG(' 半角フィールド ')
A            DATA2          200          COLHDG(' 全角フィールド ')
A            DATA3          10S 0        COLHDG(' 数値フィールド ')
A*
********** データの終わり ********************************************
```

図5　送信元CSVファイル

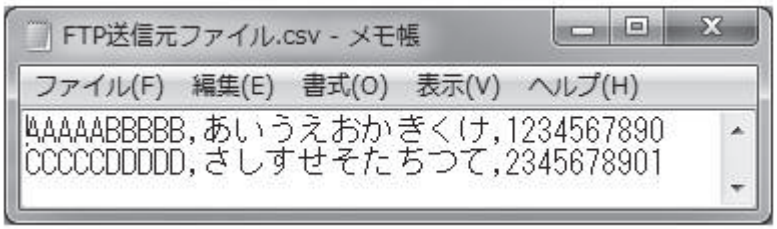
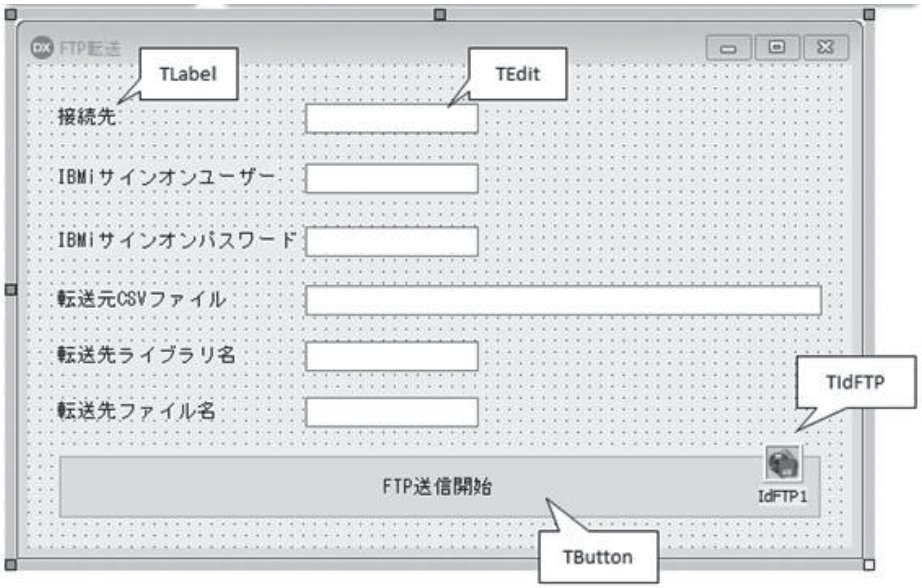


図6　コンポーネントの配置



数と同じファイル名になる。
第3引数：
ファイルダウンロード時、上書き保存するかどうかを設定する。Trueを設定すると、上書き保存する。省略時はFalse。

上記プログラムを実行し、「FTP送信開始」ボタンを押した結果、送信されたデータを5250画面の「RUN QUERY」コマンドで確認する。【図7】
これを見ると、FTP送信処理は正しく完了していたが、すべてのフィールドで文字化けが発生している。文字化けの原因はFTP送信時に、Windowsの文字コードとIBM iデータベースの文字コードのマッピングが正しく行われていないためである。
この文字化けを回避するには、マッピングテーブルを合わせる必要がある。FTP送信時に使用するマッピングテーブルは、「Quote」メソッドを使って設定できる。文字化けを発生させずに正しく送信するため、「TYPE C 943」を設定する。【ソース3】
修正したプログラムを実行し、再度「FTP送信開始」ボタンを押した結果、送信されたデータを5250画面の「RUN QUERY」コマンドで確認する。【図8】

このようにQuoteメソッドを使用することで、文字コードのマッピングが正しく行われ、送信元CSVファイルの内容がそのままIBM iデータベース上のファイルにFTP送信される。

3. ファイルレイアウトを考慮したFTP送信処理

3-1. ファイルレイアウトを考慮する理由

前述の第2章で、データベースへダイレクトにFTP送信する方法を紹介した。ただ前述の送信処理では、CSVファイルのフィールド長と送信先のフィールド長が一致していない場合、桁ずれを起こすことになる。【図9】のように、CSVファイルのフィールド長がデータベースのフィールド長よりも長い場合、FTP送信すると、この事象が発生する。
このようなケースに対応させるため、ここでは送信先のファイルレイアウトのフィールド情報を取得し、桁ずれを起こさないよう考慮した送信手法について紹介する。
この手法を採用することで、送信先データベースのファイルレイアウトが異なる場合でも、プログラムを個別に開発する必要がなく、汎用的な送信処理を実現できる。

3-2. ファイルレイアウトを考慮した送信処理

第2章で作成したサンプルプログラムを利用し、ファイルレイアウトを考慮したFTP送信を行うプログラムを作成していく。

(1) コンポーネントの配置

まず、第2章で作成したサンプルプログラムに対し、ファイルレイアウトを取得するための情報を入力するTEditならびにTLabelをそれぞれ画面に配置する。【図10】

(2) ファイルレイアウトの取得

FTP受信処理にて、ファイルレイアウトを取得する。第2章で作成したサンプルプログラムの、「FTP送信開始」ボタンのOnClick処理の最初に、ファイルレイアウト取得を行うよう処理を組み込む。

以下に、ファイルレイアウトを取得するのに必要なTIdFTPコンポーネントが持つプロパティや、今回の処理で行っているメソッドについて解説する。なお、第2章で解説しているプロパティ、メソッドについての解説は本稿では割愛する。また、DDSライブラリ・DDSオブジェクト・DDSファイルについて【図11】に補足する。

①接続設定と接続処理
第2章2-3の（2）FTP送信処理の実装を参考に、接続処理を行う。【ソース4-①】

②Getメソッド
各プロパティの設定が完了したら、Getメソッドを呼び出し、ファイルレイアウトの取得（FTP受信処理）を行う。引数で指定した条件で、ファイルをFTP受信先にダウンロードする。第3引数は省略可能である。【ソース4-②】

第1引数：
ダウンロード元のファイル名を設定する。(1) で配置した「DDSライブラリ名」「DDSオブジェクト名」「DDSファイル名」を設定する。
第2引数：
ダウンロード先ファイルのフルパスを設定する。(1) で配置した「DDS保管テキストファイル」を設定する。
第3引数：
ファイルダウンロード時、上書き保存するかどうかを設定する。Trueを設定すると上書き保存する。省略時はFalse。

以上のサンプルプログラムを実行した結果、FTP受信したファイルレイアウトのテキストファイルを確認する。今回は、第2章で使用したファイルのファイルレイアウトを取得する。【図12】

(3) 送信元CSVファイルの文字列成型

次に、(2) で受信したファイルレイアウトをもとに、送信元CSVファイルの文字列成型を行う。送信元CSVファイルは【図13】のように作成した。
送信元CSVファイルについて、第2章では桁数やシフト・コード文字（シフトイン文字、シフトアウト文字）を考慮した文字列を作成していたが、本章ではそれらを考慮せずに文字列を作成する。
まず、作成した送信元CSVファイルを、文字列成型せずにFTP送信する。IBM iデータベースへのFTP送信処理は、第2章で作成したサンプルプログラムで実行し、送信されたデータを5250画面の「RUN QUERY」コマンドで確認する。【図14】
送信結果を確認すると、ファイルレイアウトを考慮せずに作成したCSVファイルの場合、FTP送信結果に文字化けや桁ずれが発生していることがわかる。そこでFTP送信前に、送信元CSVファイルをファイルレイアウトに合わせて文字列成型を行い、FTP送信結果に文字化けや桁ずれが発生しないようにする。
文字列の成型方法については以下、①

ソース1　FTP送信プログラムソース

```
{*************************************************************************
 目的 : FTP送信開始ボタン押下
 引数 :
 戻値 :
*************************************************************************}
procedure TfrmSample.btnTransferFTPClick(Sender: TObject);
begin
  IdFTP1.Host     := edtDataBaseName.Text;  // 接続先
  IdFTP1.Username := edtSignon.Text;        // IBMiサインオンユーザー
  IdFTP1.Password := edtPassword.Text;      // IBMiサインオンパスワード
  IdFTP1.Passive  := True;                  // パッシブモード

  IdFTP1.Connect;                           // FTP接続
  try
    if IdFTP1.Connected then
    begin
      IdFTP1.TransferType := ftBinary;      // FTP転送タイプ

      // 文字列成型処理
      CreateString;

      try
        // FTP送信処理
        IdFTP1.Put(edtText.Text, edtLibrary.Text + '/' + edtFileName.Text);

        // 完了メッセージの表示
        ShowMessage('転送が終了しました');
      except
        // エラーメッセージの表示
        ShowMessage('転送中にエラーが発生しました');
      end;
    end;
  finally
    IdFTP1.Disconnect;  // FTP切断
  end;
end;
```

①接続設定
②Passiveプロパティ
①接続処理
③TransferTypeプロパティ
④送信元CSVファイルの文字列成型
⑤Putメソッド

ソース2　文字列成型メソッド

```
{*************************************************************************
 目的 : 文字列成型メソッド
 引数 :
 戻値 :
*************************************************************************}
procedure TfrmSample.CreateString;
var
  slReadCSV: TStringList;
  slSaveCSV: TStringList;
  slRowCSV : TStringList;
  sCSVText: String;
  iReadCSV, iRowCSV: Integer;
begin
  // 転送元ファイル成型用StringListの生成
  slReadCSV := TStringList.Create;
  slSaveCSV := TStringList.Create;
  slRowCSV  := TStringList.Create;
  try

    // CSVファイルの読み込み
    slReadCSV.LoadFromFile(edtText.Text);

    // 読み込んだデータをslRowCSVに保管
    for iReadCSV := 0 to slReadCSV.Count - 1 do
    begin
      // n行目のデータをCommaTextで取得
      slRowCSV.CommaText := slReadCSV[iReadCSV];

      // 成型後文字列保管用変数の初期化
      sCSVText := '';

      // カンマを取り除いた文字列に成型
      for iRowCSV := 0 to slRowCSV.Count - 1 do
      begin
        sCSVText := sCSVText + slRowCSV[iRowCSV];
      end;

      // カンマを取り除いた文字列を保存用StringListに保存
      slSaveCSV.Add(sCSVText);
    end;

    // 成型した転送元ファイルの保存
    slSaveCSV.SaveToFile(edtText.Text);

  finally
    // 内部生成したStringListの解放
    FreeAndNil(slRowCSV);
    FreeAndNil(slReadCSV);
    FreeAndNil(slSaveCSV);
  end;
end;
```

④-1.送信元CSVファイルの読み込み
④-2.読み込んだファイルの文字列成型
④-3.文字列成型後のCSVファイルを保存

～③に記載する。

① IBM i データベースの項目属性の取得

TStringList を生成し、LoadFromFile メソッドにて（2）で保存したテキストファイルを指定し、ファイルを読み込む。読み込んだテキストファイルを 1 行ごとに読み込み、各フィールドの項目属性を取得する。

ファイルレイアウトの構文規則として、30 ～ 34 桁目に桁数、35 桁目に文字タイプが保管されている。また、2 桁目に「*」が指定されている場合、コメント行となっている。これらの構文規則をもとに、フィールドの桁数と文字タイプを取得する。【ソース 5】

取得した桁数と文字タイプをもとに、以下②～③にて送信元 CSV ファイルの文字列成型を行う。【ソース 6 ～ 11】

文字列成型処理は、第 2 章サンプルプログラムの「CreateString」手続き内で、送信元ファイルのカンマ（,）を排除する処理（【ソース 2-④ -2】）の代わりに実装する。

② O タイプフィールド以外の文字列成型

O タイプフィールド以外の場合、ファイルレイアウトの桁数に対して桁数超過している場合、送信元文字列をファイルレイアウトの桁数に合わせてカットする。【ソース 7 ～ 8】

またファイルレイアウトの桁数に満たない場合、不足桁数分、S タイプフィールドの場合は 0 を送信元文字列の前方に付与【ソース 7】、A タイプフィールドの場合は半角スペースを送信元文字列の後方に付与する。【ソース 8】

③ O タイプフィールドの文字列成型

O タイプフィールドの場合、シフト・コード文字を考慮した文字列成型を行う。【ソース 9 ～ 11】

シフトアウト文字は 2 バイト文字の始まり、シフトイン文字は 2 バイト文字の終わりを表すので、1 バイト文字の直後に 2 バイト文字が発生した場合、または 2 バイト文字の直後に 1 バイト文字が発生した場合に、一時的に半角スペースを埋め込むことで、シフト・コード文字を考慮した文字列を成型し【ソース 10】、ファイルレイアウトの桁数に合わせて文字列をカットする。

FTP 送信する際は、一時的に埋め込んだシフト・コード文字用の半角スペースは不要なので削除する。【ソース 11】

このとき、送信元の文字列はバイト単位で処理する必要があるので、送信元文字列を AnsiString として扱うことに注意する。

④文字列成型した送信元ファイルの保存

①～③で文字列成型を行った TString List を、SaveToFile メソッドにより保存することで、ファイルレイアウトを考慮した FTP 送信用の送信元 CSV ファイルが作成できる。【図 15】

（4）FTP 送信処理

第 2 章 2-3（2）の⑤ Put メソッドを参考に、(3) で保存した送信元 CSV ファイルを IBM i データベースへ FTP 送信する。

上記プログラムを実行し、「FTP 送信開始」ボタンを押した結果、送信されたデータを 5250 画面の「RUN QUERY」コマンドで確認する。【図 16】

文字列成型前の送信では文字化けや桁ずれが発生していたが、シフト・コード文字を考慮した文字列成型を行うことにより、正常に FTP 送信できていることが確認できる。

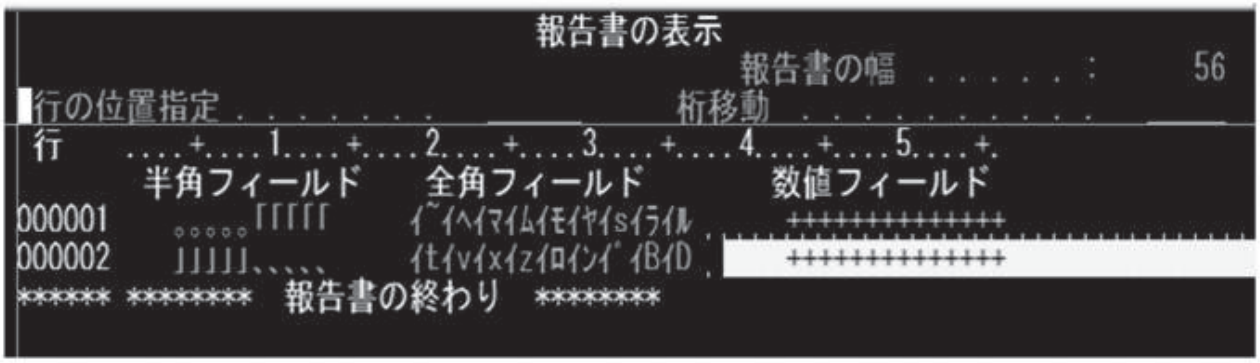
4. さいごに

以上本稿では、CSV ファイルのデータを IBM i データベースに送信する手法を紹介した。FTP 送信は処理自体が高速なので、速いレスポンスが要求されるアプリケーション開発でも効果が期待できる手法である。

また今回は IBM i データベースへの送信処理を紹介したが、IBM i データベースからの受信処理も行えるので、IBM i データベースとの転送手段の 1 つとして今回の手法を役立てていただければ幸いである。

M

図7　送信データの確認



ソース3　Quoteメソッドの追加

```
{*************************************************************************************
 目的 : FTP送信開始ボタン押下
 引数 :
 戻値 :
*************************************************************************************}
procedure TfrmSample.btnTransferFTPClick(Sender: TObject);
begin
  IdFTP1.Host     := edtDataBaseName.Text;  // 接続先
  IdFTP1.Username := edtSignon.Text;        // IBMiサインオンユーザー
  IdFTP1.Password := edtPassword.Text;      // IBMiサインオンパスワード
  IdFTP1.Passive  := True;                  // パッシブモード

  IdFTP1.Connect;                           // FTP接続
  try
    if IdFTP1.Connected then
    begin
      IdFTP1.TransferType := ftBinary;      // FTP転送タイプ
      IdFTP1.Quote('TYPE C 943');           // Quoteタイプの指定

      // 文字列成型処理
      CreateString;

      try
        // FTP送信処理
        IdFTP1.Put(edtText.Text, edtLibrary.Text + '/' + edtFileName.Text);

        // 完了メッセージの表示
        ShowMessage('転送が終了しました');
      except
        // エラーメッセージの表示
        ShowMessage('転送中にエラーが発生しました');
      end;
    end;
  finally
    IdFTP1.Disconnect;  // FTP切断
  end;
end;
```

Quoteメソッドの追加

図8　Quoteメソッド追加後の送信データの確認

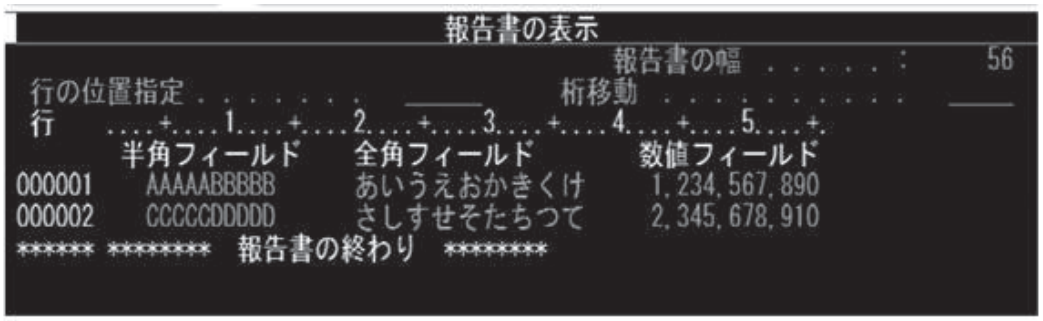


図9　フィールド長の違いによる桁ずれのイメージ

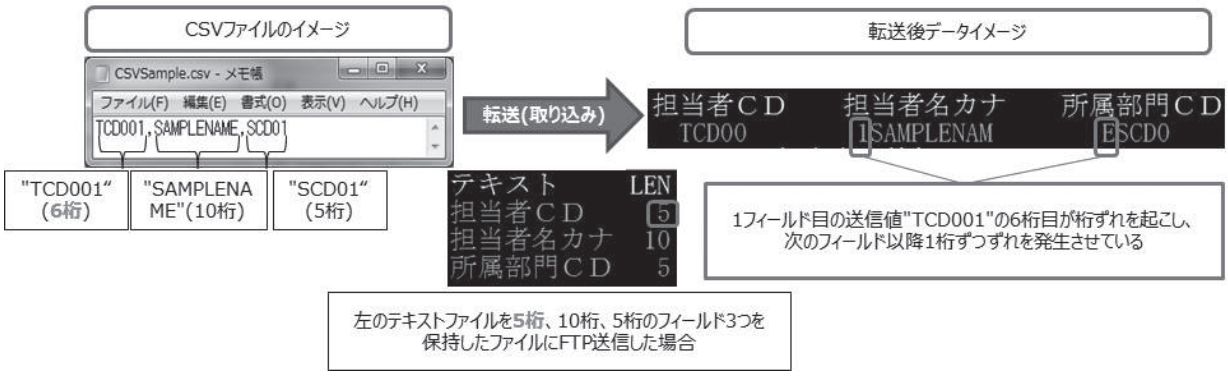


図10　コンポーネントの配置

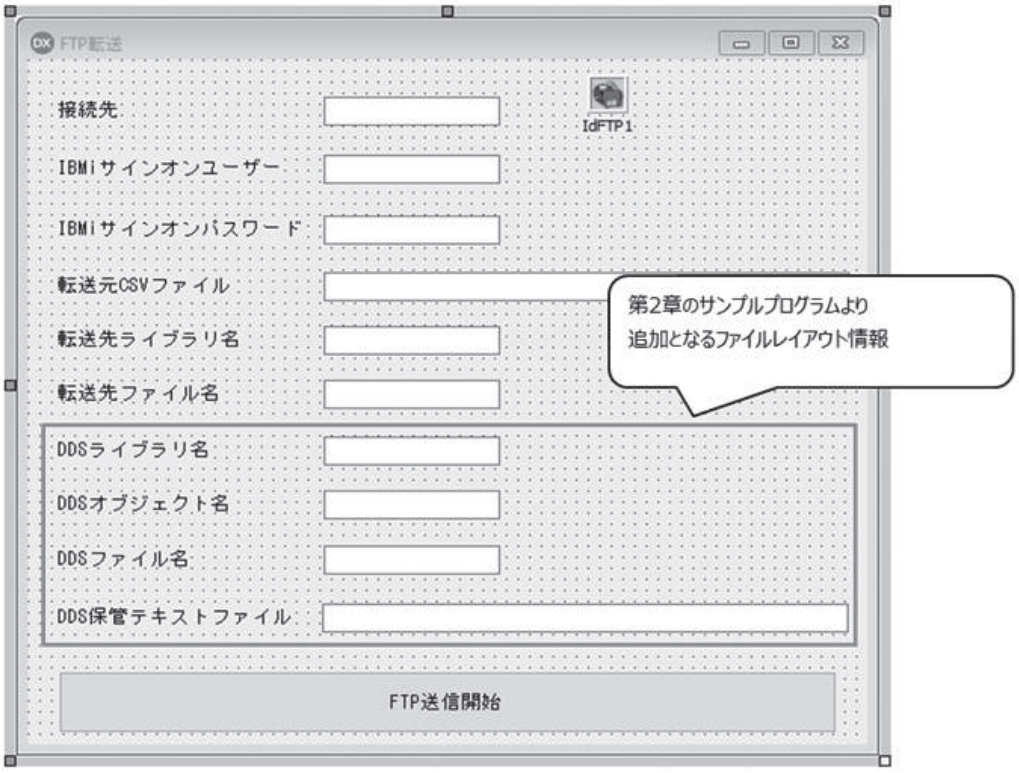
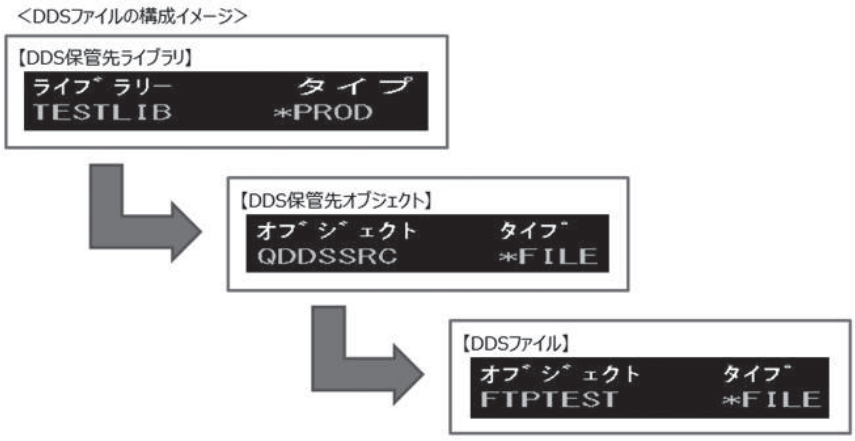


図11　DDSファイルの構成イメージ



<上記イメージのファイルレイアウトを取得するサンプル画面のTEdit指定例>

ソース4　FTP受信によるファイルレイアウトの取得

```
// DDSの取得
IdFTP1.Host     := edtDataBaseName.Text;  // 接続先
IdFTP1.Username := edtSignon.Text;        // IBMiサインオンユーザー
IdFTP1.Password := edtPassword.Text;      // IBMiサインオンパスワード
IdFTP1.Passive  := True;                  // パッシブモード

IdFTP1.Connect;                           // FTP接続
try
  if IdFTP1.Connected then
  begin
    IdFTP1.TransferType := ftBinary;      // FTP転送タイプ
    IdFTP1.Quote('TYPE C 943');           // Quoteタイプの指定

    try
      // FTP受信処理
      IdFTP1.Get(edtDDSLibrary.Text + '/' +  // 転送元：ライブラリ名
                 edtDDSObject.Text  + '.' +  //         オブジェクト名
                 edtDDSFileName.Text,        //         ファイル名
                 edtDDSText.Text,            // 転送先：テキストファイル名
                 True);                      // 上書き保存
    except
      // エラーメッセージの表示
      ShowMessage('DDS取得中にエラーが発生しました');
    end;
  end;
finally
  IdFTP1.Disconnect;  // FTP切断
end;
```

①接続設定と接続処理

②Getメソッド

図12　ファイルレイアウトの取得結果

図13　IBM iへFTP送信するテキストファイル

(Ctrl)

【送信元CSVの作成内容】
1行目：全項目、ファイルレイアウトに準じた桁数の1バイト文字を入力
2行目：半角フィールド（1項目目）のみ、桁数を超過した1バイト文字を入力
3行目：全角フィールド（2項目目）のみ、桁数を超過した2バイト文字を入力
4行目：全角フィールド（2項目目）のみ、桁数を超過した1バイト文字と2バイト文字を混合入力

図14　文字列成型前のFTP送信結果

	半角フィールド	全角フィールド	数値フィールド
000001	1234567890	12345678901234567890	1,234,567,890
000002	1234567890	AAA12345678901234567	8,901,234,567
000003	ABCDEFGHIJ	++++++++++++++++++++	0
000004	1234567890	++++++++++++++++++++	0

【FTP送信結果】
1行目：正常に転送完了
2行目：半角フィールドで超過桁数分、全角フィールド・数値フィールドにて桁ずれが発生
3行目：全角フィールドにて文字化けが発生、数値フィールドにて桁ずれが発生
4行目：全角フィールドにて文字化けが発生、数値フィールドにて桁ずれが発生

ソース5　ファイルレイアウトから項目属性を取得

```
// ファイルレイアウトの読み込み用StringListの生成
slDDS := TStringList.Create;

try
  // ファイルレイアウトの読み込み
  slDDS.LoadFromFile(edtDDSText.Text);

  // フィールドの開始桁
  iStrLen := 1;

  // ファイルレイアウトの行数分処理を繰り返す
  for iDDS := 0 to slDDS.Count - 1 do
  begin
    // コメント行の場合、次の処理へ
    if (Copy(slDDS[iDDS], 7, 1) = '*') then
    begin
      Continue;
    end;

    // 文字タイプ
    sType := Copy(slDDS[iDDS], 35, 1);

    // 桁数
    iKeta := StrToIntDef(Trim(Copy(slDDS[iDDS], 30, 5)),0);

    // 桁数が取得できていない場合、次の処理へ
    if (iKeta = 0) then
    begin
      Continue;
    end;

    // ※※ 以下、(3)-③～④(文字列成型)の処理を記述 ※※



    // フィールドの開始桁
    iStrLen := iStrLen + iKeta;
  end;

  // 成型した転送元ファイルの保存
  slSaveCSV.SaveToFile(edtText.Text);
finally
  FreeAndNil(slDDS);
end;
```

ソース6　文字列成型①処理の流れ

```
// 成型後の文字列の初期化
sCSVText_After := '';

// ファイルレイアウトを基に転送元ファイルを成型
for iReadCSV := 0 to slReadCSV.Count - 1 do
begin
  slRowCSV.CommaText := slReadCSV[iReadCSV];

  // 初期化
  iRowCSV        := 0;    // CSV行情報の読み込みフィールド番号
  sCSVText_After := '';   // 成型後の文字列

  // ファイルレイアウトの行数分処理を繰り返す
  for iDDS := 0 to slDDS.Count - 1 do
  begin
    // ※※ 以下、(3)-①(IBMiデータベースの項目属性の取得)の処理を記述 ※※


    // 処理対象のフィールドの文字列を取得
    sCSVText := slRowCSV[iRowCSV];

    // ※※ 以下、(3)-②～③の処理を記述 ※※



    // CSV行情報の読み込みフィールド番号
    iRowCSV := iRowCSV + 1;
  end;

  // 成型した文字列を保存用StringListに保存
  slSaveCSV.Add(sCSVText_After);
end;
```

sCSVTextは、バイト単位での処理を行う為、
AnsiString型の変数として定義する

ソース7　文字列成型②Sタイプフィールド

```
// Sタイプフィールドの場合
if (sType = 'S') then
begin
  // 桁数に満たない場合、不足桁分の0を文字列前方に付与
  if (Length(sCSVText) < iKeta) then
  begin
    for i := Length(sCSVText) + 1 to iKeta do
    begin
      sCSVText := '0' + sCSVText;
    end;
  end
  else
  // 桁数を超える場合、超過分の文字を削除
  if (Length(sCSVText) > iKeta) then
  begin
    sCSVText := Copy(sCSVText, 1, iKeta);
  end;

  // 成型した文字列を保管
  sCSVText_After := sCSVText_After + sCSVText;
end;
```

桁数不足

桁数超過

ソース8　文字列成型③Aタイプフィールド

```
// Aタイプフィールドの場合
if (sType = 'A') then
begin
  // 桁数に満たない場合、不足桁分の半角スペースを文字列後方に付与
  if (Length(sCSVText) < iKeta) then
  begin
    for i := Length(sCSVText) + 1 to iKeta do
    begin
      sCSVText := sCSVText + ' ';
    end;
  end
  else
  // 桁数を超える場合、超過分の文字を削除
  if (Length(sCSVText) > iKeta) then
  begin
    sCSVText := Copy(sCSVText, 1, iKeta);
  end;

  // 成型した文字列を保管
  sCSVText_After := sCSVText_After + sCSVText;
end
```

桁数不足

桁数超過

ソース9　文字列成型④Oタイプフィールド

```
// Oタイプフィールドの場合、シフト・コード文字を考慮した文字列の成型
if (sType = 'O') then
begin
  // 桁数に満たない場合、不足桁分の半角スペースを文字列後方に付与
  if (Length(sCSVText) < iKeta) then
  begin
    for i := Length(sCSVText) + 1 to iKeta do
    begin
      sCSVText := sCSVText + ' ';
    end;
  end;

  // シフト・コード文字を考慮した文字列に変換
  sCSVText := AddSISO(sCSVText);

  // 文字列成型結果文字列を桁数分でカット
  sCSVText := Copy(sCSVText, 1, iKeta);

  // 最終桁が全角の1バイト目の場合、半角スペースに置き換える
  if (ByteType(sCSVText, iKeta) = mbLeadByte) then
  begin
    sCSVText := Copy(sCSVText, 1, iKeta - 1) + ' ';
  end
  // 最終桁が全角2バイト目の場合、シフト・コード文字部分があふれるため
  // 半角スペースに置き換える
  else
  if (ByteType(sCSVText, iKeta) = mbTrailByte) then
  begin
    sCSVText := Copy(sCSVText, 1, iKeta - 2) + '  ';
  end;

  // シフト・コード文字を考慮した文字列に変換
  sCSVText := RmvSISO(sCSVText);

  // 成型した文字列を保管
  sCSVText_After := sCSVText_After + sCSVText;
end
```

ソース10

ソース11

ソース10　文字列成型⑤Oタイプフィールド(シフト・コード文字の付与)

```
{*******************************************************************************
 目的：文字列に対して、シフト・コード文字の位置にダミーの半角スペースをセット
 引数：AStr - 元の文字列
 戻値：成型された文字列
*******************************************************************************}
function TfrmSample.AddSISO(AStr: AnsiString): String;
var
  i: Integer;
  S: AnsiString;
begin
  // 初期化
  // 1バイト目が全角の場合
  if (ByteType(AStr, 1) = mbLeadByte) then
  begin
    S := ' ';
  end
  // 1バイト目が半角の場合
  else
  begin
    S := '';
  end;

  // 文字列を確認し、全角と半角の切替ポイントにダミーの半角スペースをセット
  for i := 1 to Length(AStr) do
  begin
    // 1文字ずつ保管
    S := S + AStr[i];

    // 現在の文字が半角 かつ 1文字先が全角の1バイト目の場合
    // シフト・コード文字の代わりに半角スペースを付与
    if (ByteType(AStr, i) = mbSingleByte) then
    begin
      if (ByteType(AStr, i + 1) = mbLeadByte) then
      begin
        S := S + ' ';
      end;
    end;

    // 現在の全角2バイト目 かつ 1文字先が半角の場合
    // シフト・コード文字の代わりに半角スペースを付与
    if (ByteType(AStr, i) = mbTrailByte) then
    begin
      if (ByteType(AStr, i + 1) = mbSingleByte) then
      begin
        S := S + ' ';
      end;
    end;
  end;

  // 結果をセット
  Result := String(S);
end;
```

ソース11　文字列成型⑥Oタイプフィールド(シフト・コード文字の削除)

```
{*******************************************************************************
 目的：文字列に対して、シフト・コード文字の位置の半角スペースをカット
 引数：AStr - 元の文字列(※シフト・コード文字考慮済みの文字列)
 戻値：成型された文字列
*******************************************************************************}
function TfrmSample.RmvSISO(AStr: WideString): String;
var
  i: Integer;
  bSI: Boolean;
begin
  // 初期化
  Result := '';
  bSI    := False;

  // 文字単位で後ろからカウント
  for i := Length(AStr) downto 2 do
  begin
    // 半角スペース(シフト・コード文字)の場合、セットしない
    if (bSI) then
    begin
      bSI := False;
    end
    else
    // 半角スペース(シフト・コード文字) かつ 1文字前が全角の場合、セットしない
    if (AStr[i] = ' ') and (Length(AnsiString(AStr[i - 1])) = 2) then
    begin
      // 何もしない
    end
    // 上記以外の場合、文字をセット
    else
    begin
      Result := AStr[i] + Result;

      // 全角かつ1文字前が半角スペース(シフト・コード文字)のとき、フラグセット
      if (Length(AnsiString(AStr[i])) = 2) and (AStr[i - 1] = ' ') then
      begin
        bSI := True;
      end;
    end;
  end;

  // 全角始まりでない場合は1文字目(シフト・コード文字でない)を最後に足す
  if (not bSI) then
  begin
    Result := AStr[1] + Result;
  end;
end;
```

図15　文字列成型後の送信元ファイル

＜文字列成型前の送信元CSVファイル＞

＜文字列成型後の送信元CSVファイル＞

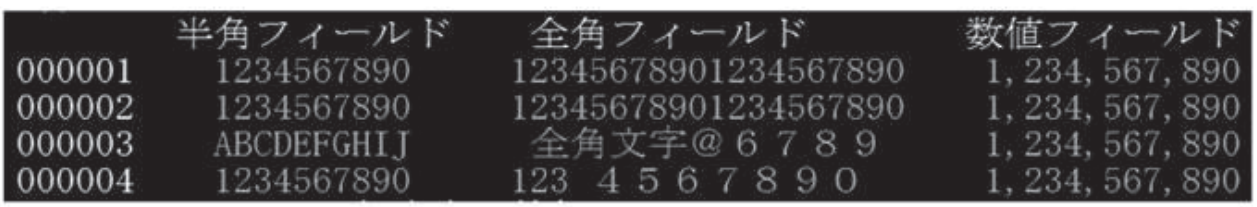
【送信元CSVファイルの文字列成型内容】
1行目：文字列成型前からファイルレイアウトに準じているため、文字列成型前と同じ文字列
2行目：半角フィールド（1項目目）のみ、桁数を超過した文字列を削除
3行目：全角フィールド（2項目目）のみ、シフト・コード文字を考慮＆桁数を超過した文字列を削除
4行目：全角フィールド（2項目目）のみ、シフト・コード文字を考慮＆桁数を超過した文字列を削除

図16　文字列成型後のFTP送信結果

＜文字列成型前のFTP送信結果＞

	半角フィールド	全角フィールド	数値フィールド
000001	1234567890	123456789012345678901	1,234,567,890
000002	1234567890	AAA123456789012345678	8,901,234,567
000003	ABCDEFGHIJ	++++++++++++++++++++	0
000004	1234567890	++++++++++++++++++++	0

＜文字列成型後のFTP送信結果＞



	半角フィールド	全角フィールド	数値フィールド
000001	1234567890	123456789012345678901	1,234,567,890
000002	1234567890	123456789012345678901	1,234,567,890
000003	ABCDEFGHIJ	全角文字@６７８９	1,234,567,890
000004	1234567890	123 ４５６７８９０	1,234,567,890