5

Import CSV Files with File Correcting

This chapter presents a user-friendly CSV file upload program. While the program itself is simple, it uses some interesting procedures to recover records with the most common user errors in a CSV file. In chapter 6, this program is used to send (via email) a Microsoft Excel file with the records that couldn't be recovered, among other things. MS Excel file creation is explained in chapter 7, and the FTP transfer that puts the CSV file in the IFS is in chapter 1. It's advisable to read chapters 1, 2, and 3 before continuing.

In chapters 1 and 2, you learned how to automate file upload to your IBM i system using FTP. However, if the file doesn't arrive via email, someone from the IT department will still have to upload the file to the IFS, because the end-user won't do it or might delete something that shouldn't be deleted.

The program presented here provides you with a template, which will enable you to build your own user-friendly CSV file-upload programs. The program itself is simple, but it establishes some good practices that I really recommend that you follow.

Imagine the following scenario: your company receives a shipment of items, to be logged in your inventory. Along with the shipment comes a CSV file that the end-

user checks and, if necessary, changes by adding or removing records, changing item quantities, and so on. When the CSV and the shipment are in sync, it's time to log the items into the inventory. To do that, the user calls the *INVIMP* sample program, as shown in Figure 5.1.

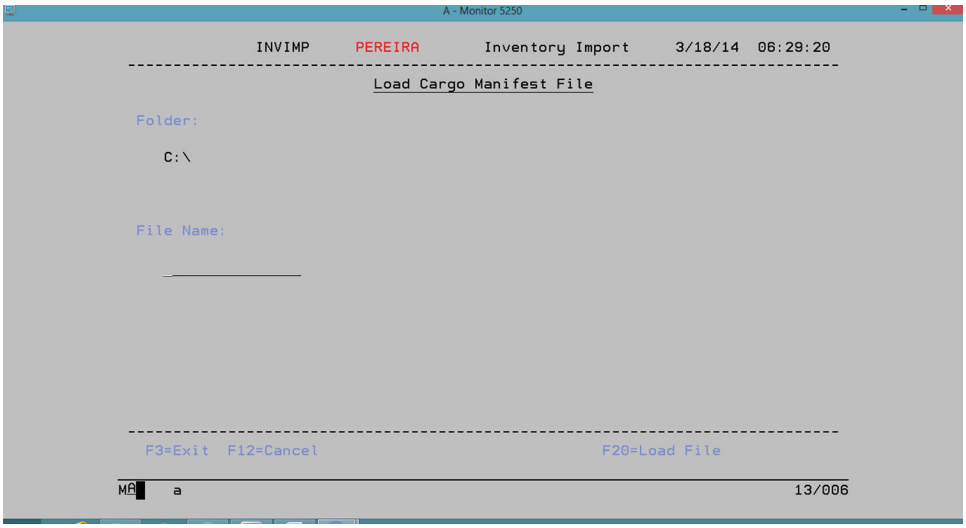

Figure 5.1: The main screen of the INVIMP program for logging inventory.

The user needs to put the CSV file in the folder indicated on the screen (*C:*, in the example in Figure 5.1) and fill the File Name field with the CSV file name. Then, by pressing F20, the CSV file is transferred to the IFS, checked for errors, copied to a temporary physical file, and has its valid records logged into the company's inventory. The invalid records are written to an MS Excel file, with the same structure as the input CSV file and sent, via email, to the user who ran the program. This allows the user to correct the errors, save the file as CSV, and upload it again.

Let's analyze the upload functionality step by step, starting with the program's user interaction. (The email functionality is discussed in the next chapter.)

Main Program Analysis

When the program is called, a few "behind the scenes" things happen. The folder where the CSV file is expected to be placed is dynamic, stored in a data area, and needs to be shown on screen. The folder where the FTP script that will run a bit later

on is also dynamic and—you guessed it—also stored in a data area. So, the first thing to do is to read those data areas and put the CSV file folder on screen. Even though the program is able to handle a long path, up to 207 characters, I advise you to keep it as short as possible. Otherwise, you might have problems with the FTP transfer (more on this later). After this, the program's main cycle can begin and follow its course until the user presses F3 to leave.

Here's the code for data area loading and preparing, and for the main cycle:

```
// Load the folder paths
ExSR Load_Paths;

// Main Loop
DoW *InKC <> *On;

  // Show Main screen
  ExFmt INVIMP01;

  // Initialize Indicators/Messages
  ExSR Init_Ind_Msg;

  // Main process
  Select;
  When *InKC = *On;
    // Dummy - Exit Program

  When *InKL = *On;
    // Cancel user input (clear screen fields)
    ExSR Cancel_Input;

  When *InKU = *On;
    // Validate user input
    ExSr Vld_Screen;
    If (W_Screen = *Off);
      // Validate and Load file
      ExSr Vld_Load_File;
    EndIf;
```

Continued

```
  Other;
    // Initialize Indicators/Messages
    ExSR Init_Ind_Msg;
    // Validate user input
    ExSr Vld_Screen;
  EndSL;

EndDo;

// End Program
*InLR = *On;
```

Don't worry, I'll explain all the subroutines mentioned here as we follow the program's cycle. Before the cycle begins, the aforementioned data area reading and preparing takes place in subroutine *Load_Paths*:

```
// ----------------------------------------------------------------------
// Subroutine: Load the folder paths
// ----------------------------------------------------------------------
BegSR Load_Paths;

  // Load Data Area CSV Path Data Area
  IN *LOCK W_DAInvFld;
  W_LoadPath = W_DAInvFld;
  OUT W_DAInvFld;

  // Load FTP Scripts Path Data Area
  IN *LOCK W_DAScpFld;
  W_ScriptFolder = W_DAScpFld;
  OUT W_DAScpFld;

  // Process Data Area to show on screen
  ExSR Process_Path;

EndSR;
```

I'm using data areas to store configuration data simply because it's easier from a programming point of view. However, it's not always the best solution. From a maintenance and user-friendliness point of view, a flexible configurations file is more advisable. (Feel free to change the code and adapt it to whatever your company's best practice is for configuration data.) This subroutine then calls *Process_Path* to transform the big string with the CSV file folder into the screen fields:

```
// ----------------------------------------------------------------------
// Subroutine: Process Data Area to show on screen
// ----------------------------------------------------------------------
BegSR Process_Path;

  // Load long variable used to process the transfer path
  W_PathScript = W_LoadPath;

  // Path size
  W_SizPath = %Len(%Trim(W_LoadPath));

  // Screen's folder path first line (PathFile1) size
  W_SizPathF1 = %Len(PathFile1);
  // Screen's folder path second line (PathFile2) size
  W_SizPathF2 = %Len(PathFile2);
  // Screen's folder path third line (PathFile3) size
  W_SizPathF3 = %Len(PathFile3);

  // Fill first line on screen (field PathFile1)
  If W_SizPath > W_SizPathF1;

    PathFile1 = %Subst(%Trim(W_LoadPath): 1: W_SizPathF1);
    W_SizPath = W_SizPath - W_SizPathF1;

    // Fill second line on screen (field PathFile2)
    If W_SizPath > W_SizPathF1;
      PathFile2 = %SubSt(%Trim(W_LoadPath):
                         W_SizPathF1 + 1  :
                         W_SizPathF2      );
      W_SizPath = W_SizPath - W_SizPathF2;
```

Continued

```
      // Fill third line on screen (field PathFile3)
      PathFile3 = %SubSt(%Trim(W_LoadPath)             :
                         W_SizPathF1 + W_SizPathF2 + 1 :
                         W_SizPath                     );

    Else;
      PathFile2 = %SubSt(%Trim(W_LoadPath) :
                         W_SizPathF1 + 1   :
                         W_SizPath         );
    EndIf;

  Else;
    PathFile1 = %SubSt(%Trim(W_LoadPath): 1: W_SizPath);
  EndIf;

EndSR;
```

The code is dynamic enough to handle any path length, filling the necessary screen fields. After this, the screen is shown and the user can interact with the program. When F20 is pressed, subroutine *Vld_Screen* is invoked:

```
// ----------------------------------------------------------------------
// Subroutine: Validate user input
// ----------------------------------------------------------------------
BegSR Vld_Screen;

  W_Screen = *Off;

  // If the file name has been filled, show F20 function key
  If FileName = *Blanks;
    W_Screen = *On;
    *In21 = *On;
  Else;
    *In21 = *Off;
  EndIf;
```

Continued

```
  // The file extension must be .CSV
  If %Scan('.CSV' : FileName) = *Zeros;
    W_Screen = *On;
    *In23 = *On;
  Else;
    *In23 = *Off;
  EndIf;

EndSR;
```

This subroutine checks if the file name was in fact filled in and if it ends with .CSV, because the file name length is important for the FTP transfer. The errors are handled "old school" with indicators that are linked to error messages on the display file. If everything is OK, subroutine *Vld_Load_File* is called to handle the CSV file's validation and transfer to the IFS:

```
// ----------------------------------------------------------------------
// Subroutine: Validate and Load file
// ----------------------------------------------------------------------
BegSR Vld_Load_File;

  // Check if the process can continue
  If W_Load_File = *On;
    // Retrieve the file load delay
    ExSr Rtv_Delay;

    If W_Load_File = *On;
      // Transfer the file for the CSV input folder to the IFS
      ExSr Transfer_File;

      If W_Load_File = *On;
        // Wait for the FTP transfer to end before trying to process
        ExSr Delay_FTP;

        // The delay required this workaround to prevent the screen
        // from going black until the end of the process
        WRITE INVIMPW04;
                                                                    Continued
```

```
        // Copy the data from the CSV to the temporary physical file
        ExSR COPY_CSV_PF;

        IF W_Load_File = *On;
        // The delay required this workaround to prevent the screen
        // from going black until the end of the process
          WRITE INVIMPW05;

          // Load data to the inventory file
          ExSr Load_Inv_PF;

          If W_Load_File = *On;
            // Delete the CSV file from the IFS
            ExSr Del_CSV;
            // Show success screen
            // with a warning note in case one or more records couldn't
            // be processed
            If W_TotErr > *Zeros;
              TotErrMsg1 = 'However, ' + %Char(W_TotErr) + ' errors ' +
                           'were found.';
              TotErrMsg2 = 'Check the log file sent to your mailbox.';
              TotErrMsg3 = *Blanks;

            Else;
              TotErrMsg1 = *Blanks;
              TotErrMsg2 = *Blanks;
              TotErrMsg3 = *Blanks;
            EndIf;
            ExFmt INVIMPW02;
            // Initialize screen fields
            ExSr Init_Scrn_Fields;
          EndIf;

        Else;
          // Process screen errors
          ExSr Prc_Scrn_Errors;
        EndIf;
```

Continued

```
      Else;
        // Process screen errors
        ExSr Prc_Scrn_Errors;
      EndIf;

    Else;
      // Process screen errors
      ExSr Prc_Scrn_Errors;
    EndIf;
  EndIf;

EndSR;
```

The first task it performs is transferring the file to the IFS, using FTP. Subroutine *Transfer_File* takes care of that:

```
// ----------------------------------------------------------------------
// Subroutine: Transfer the file for the CSV input folder to the IFS
// ----------------------------------------------------------------------
BegSR Transfer_File;

  // Compose transfer path
  W_PathScript = %Trim(PathFile1) + %Trim(PathFile2) + %Trim(PathFile3);


  // Determine the system type (UAT or PRD) in order to set the correct
  // transfer details
  If ProductionSys;
    P_Cmdlin = 'STRPCCMD PCCMD(' + '''' +  %Trim(W_ScriptFolder) +
               '\INVIMP\PRD_Transf_INV.BAT ' + %TRIM(FileName) +
               ' ' + '"' + %TRIM(W_PathScript) + '"' + '''' + ') +
               PAUSE(*NO)';
                                                                    Continued
```