Concatenate CSV Files Selected by User

Ingredients:

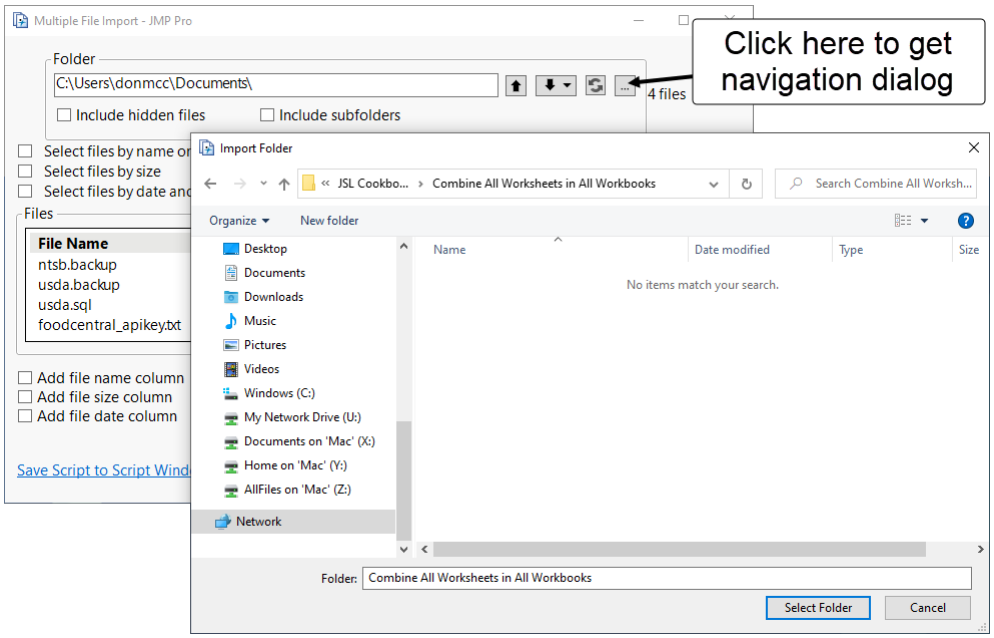
- File Input/Output
- User Input
- String manipulation
- Lists

Other Files: Directory of CSV files
Difficulty – Medium

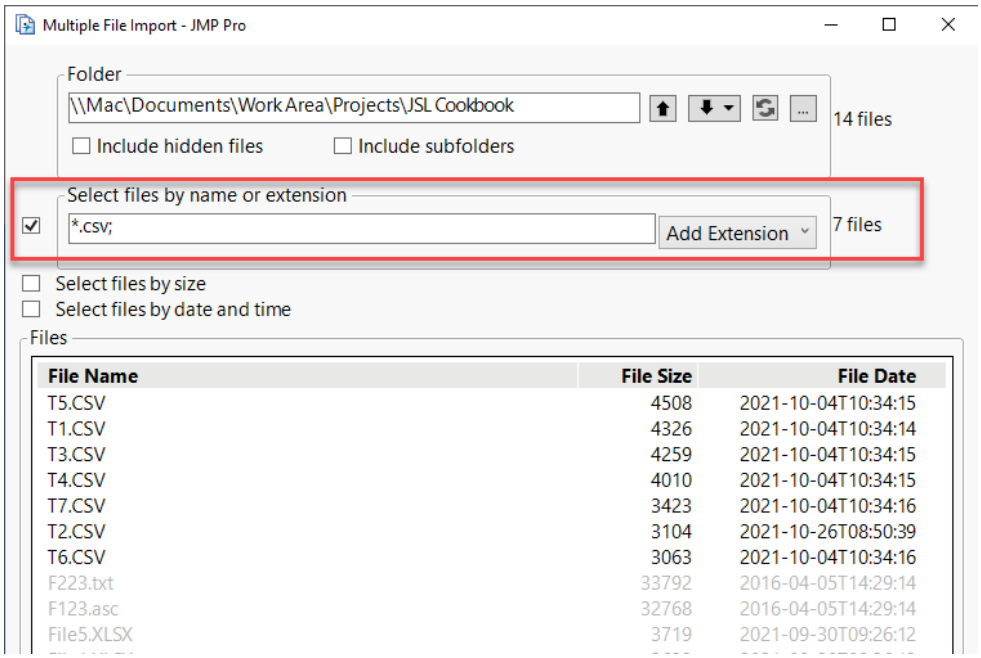
For this recipe, we'll keep things simple. It lets us adapt Multiple File Import so files can be selected at runtime. The Enhanced Log will be used to capture the code associated with opening CSV files. User input comes via the `Pick File` function, returning a list of file paths. We will parse these results to extract the directory containing the files and the file names. These results will let us make two simple changes to the `Import Multiple Files` function to get the results we want. For this to work correctly, we will assume all the files have the same number of columns.

Steps:

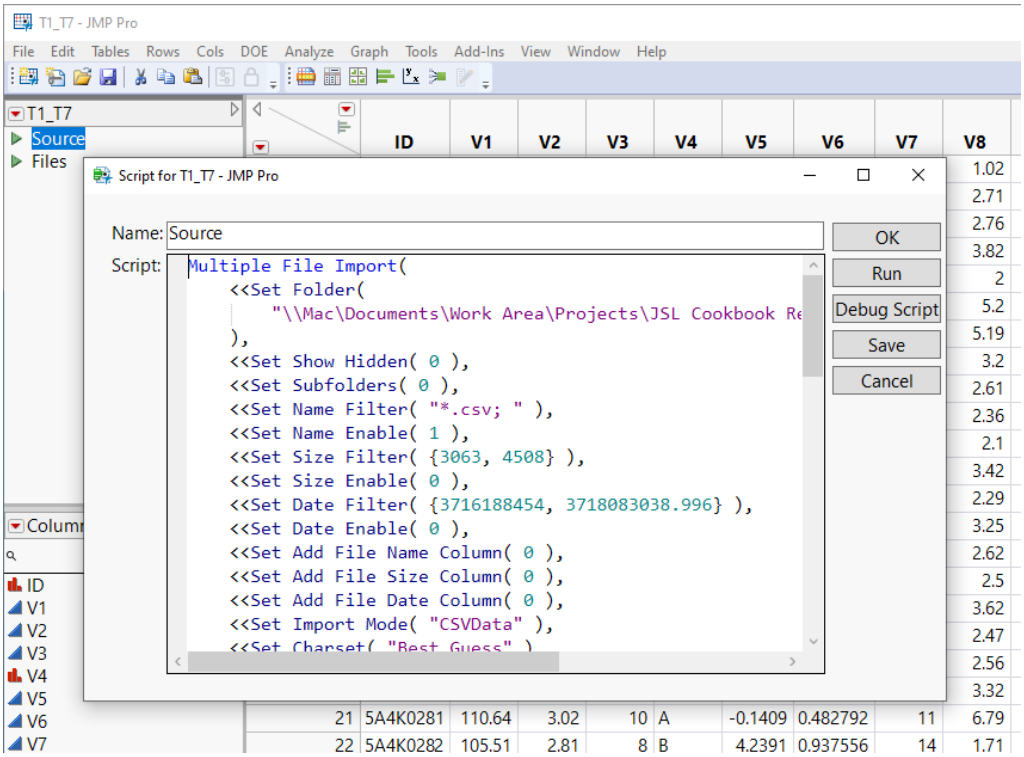
1. Using File > Import Multiple Files, navigate to the directory that contains the files. Click the button at the top right to get the dialog allowing directory navigation and selection.



Check the first filter and select or type in CSV. The resulting list will grey the files not matching the filters. We won't be able to select individual files here, it's all or nothing.



2. Expose the code by right clicking the Source table variable in the resulting table and selecting Edit. Copy it to a script window.



3. We will add a variable reference to access the resulting table and change two of the function arguments. The top of the function should look similar to this:

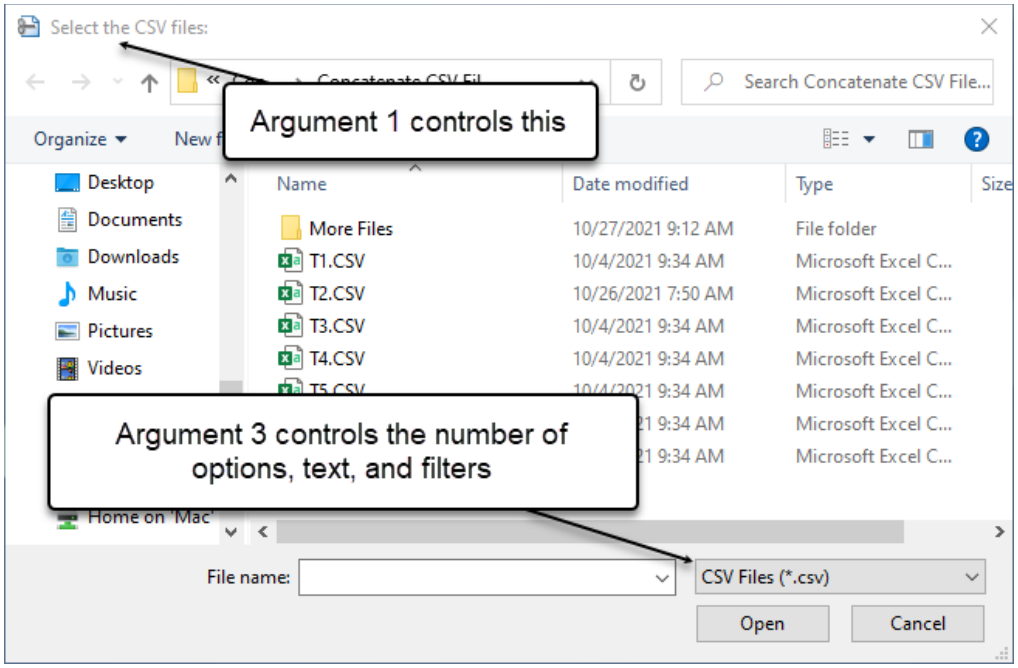
```
dtFinalTbl = Multiple File Import(
    <<Set Folder(mainFolder),
    <<Set Show Hidden( 0 ),
```

```
    <<Set Subfolders( 0 ),
    <<Set Name Filter( fileList),
```

Only the first and fourth arguments have been changed, the hard coded string values are replaced with variables.

4. We will use the `Pick File` function for user input. It has 7 arguments, all optional. We want to specify the user prompt text (1), selectable file types (3), and allow the user to choose multiple files (7). We need to use the default values for the other arguments to get the function to work properly. Everything can go on one line, but it's shown like this, so the arguments stand out.

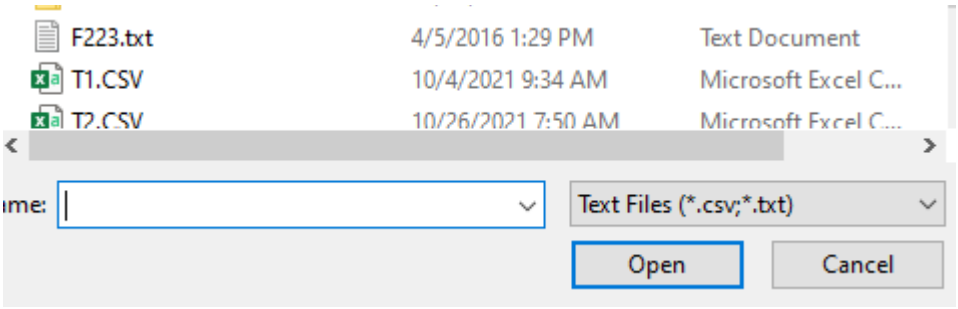
```
csvFiles = Pick File(
      "Select the CSV files: ",
      "",
      {"CSV Files|csv"},
      1 ,
      0 ,
      "",
      Multiple
);
```



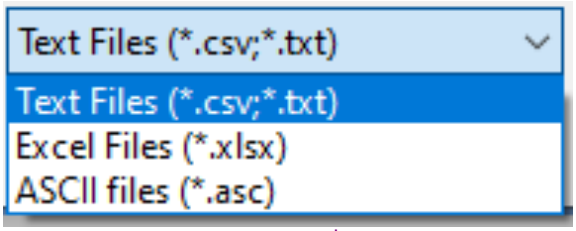
Argument 3 is a list with one or more strings taking the format:

"fileTypeText | fileTypeExtension"

fileTypeText only effects Windows (but is still required to be supplied for Macs). It supplies the text shown in the combo box at the lower right that filters the file types. *fileTypeExtension* specifies the extensions of the selectable file types in the dialog.




```
{"Text Files|csv;txt}
```




```
{"Text Files|csv;txt","Excel Files|xlsx","ASCII files|asc"}
```

5. `Pick File` returns a list of strings when the `Multiple` option is used. Each string is the full (absolute) path to a file the user selected. A list is a composite data type, containing zero or more items that can be of any other data type (including lists). It starts and ends with curly brackets. Items are separated by commas. Before going on, we should check that `csvFiles` contains something, if not, we can stop.

```
If(N Items(csvFiles) == 0,
    Print("No files selected.");
    Return()
);
```

`N Items` counts the number of items in the list. If there are none, `"No files selected."` is printed to the Log and the script terminates. (Technically, control would be passed back to the script that executes this script, but since there isn't one, processing ends.)

6. We will need to extract the file names from each string along with the directory from which the files came. Since all files are in the same directory, we only need to do this for one of the items in the list. The following two lines will first locate the last backslash (i.e., the separator of the last directory in the path) then find the path associated with that directory

```
lastBackslash = Contains(csvFiles[1],"/",-1);
mainFolder    = Substr(csvFiles[1],1,lastBackslash);
```

List items are identified by position using the syntax above. The `-1` argument to `Contains` indicates to search starting at the end of the string. The second and third arguments to `Substr` correspond to starting position and string length, respectively.

7. To extract the file names, we'll use a loop.

```
fileList = "";
For Each({nextFileName},csvFiles,
   fileList ||= Substr(nextFileName,lastBackslash+1) ||";";
);
```

`For Each` first appeared in JMP 16 and can be used for Lists, Matrices, and Associative Arrays. It is more compact and faster than its alternative, `For`. Its first argument appears in curly brackets and can have 0 – 2 variable names. The first of these is used to hold sequential values from the list starting with the first. If a second is given, it contains the iteration number. The scope of these variables is limited to the body of the loop, an additional benefit. The second argument is the list to be iterated over given as a variable name or explicitly. The body of the function is used to process the items. In our case we are building a string of file names separated by semicolons. The `Substr` function returns the file name from the complete path by extracting a substring from one character beyond the last backslash.

To loop using `For`, the code below can be substituted

```
fileList = "";
For(i=1,i<=N Items(csvFiles),i++,
   nextFileName = csvFiles[i],
   fileList ||= Substr(nextFileName,lastBackslash+1) ||";";
);
```

Hints for Success:

- The Source table script contains JSL for recreating a table it appears when a non JMP file is opened or when table operations create a new table.
- `Contains` will search from the end of a string.
- `For Each` is a faster, more compact, and more robust version of `For`.