Integrated Custom File Generation

This feature is available in Data Import as of version 1.0.1.

As described in Preprocessing CSV Files, Data Import is not a general purpose programming language. Some complex file preparation may simply require the power of a general purpose programming language prior to bringing data into Data Import, leaving Data Import to focus on the task of transforming rows of data into ODS Resources. Thankfully, though, *most* custom CSV file cleanup *can* be integrated into Data Import's Transform/Load process without the need for a separate preprocessing of CSV files. This page documents the feature and walks through a representative example.

Applying Custom File Generation

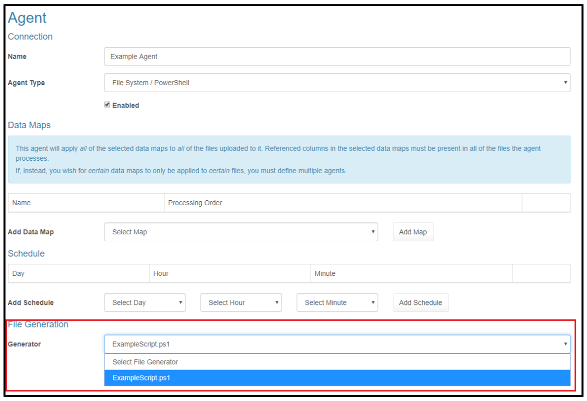
When custom PowerShell file generation scripts are properly developed, reviewed, and placed by a trusted system administrator, they will become available for selection within the Data Import *Agent* definition screen. When defining your data-processing *Agent*, you must select the "File System / PowerShell" Agent Type. You can then select which custom script you'd like to use to generate the CSV file.

Later, when the *Agent* runs the custom file generation PowerShell script, the Transform/Load process will reference the output CSV file for loading into the ODS, and only then will it be transformed by your *Data Map* to be POSTed to the ODS. You must also remember to set a schedule within the agent, similar to how you would set it up for FTP agents.

Instructions for System Administrators

The Data Import web application and Transform/Load process each rely on a "ShareName" config setting. This indicates the location where files are stored when uploaded for processing. For instance, if the config setting is set to "C:\Temp", then an uploaded CSV file would appear in the subfolders of C:\Temp\DataImport\... File processing then looks here for files to process.

Similarly, Data Import looks within this folder structure for custom file generation scripts. An administrator can choose to place custom PowerShell scripts within this folder structure. For instance, if the ShareName config setting is set to "C:\Temp" and the administrator has a custom script named ExampleScript.ps1, they should save it to "C:\Temp\DataImport\FileGenerators\ExampleScript.ps1". When setting this up for the first time, the administrator is free to create the folder structure "DataImport\FileGenerators\" beneath their configured "ShareName" path so that they can place their *.ps1 files within. Be sure that files are truly saved with the ".ps1" extension (if file extensions are hidden in Windows Explorer, you might accidentally save as ".ps1.txt", which would not be recognized as a valid script). Once present, that option will become selectable on the *Add/Edit Agent* screen:



Agent

Connection

Name Example Agent

Agent Type File System / PowerShell

☑ **Enabled**

Data Maps

This agent will apply *all* of the selected data maps to *all* of the files uploaded to it. Referenced columns in the selected data maps must be present in all of the files the agent processes.

If, instead, you wish for *certain* data maps to only be applied to *certain* files, you must define multiple agents.

Name	Processing Order	

Add Data Map Select Map

Schedule

Day	Hour	Minute	

Add Schedule Select Day Select Hour Select Minute

File Generation

Generator ExampleScript.ps1

Writing Custom File Generation Scripts

Scripts need to be written in the PowerShell programming language. The script can take in input data from any source, but it is up to the PowerShell script to read in that data, manipulate the data, and then save that data in an output CSV file. The only requirement is that the script returns the full path of that output file.

Representative Example: Cleaning Up Extra Character from Column

In this example, we'll experience a realistic obstacle involving a CSV file in need of file cleanup. We have a CSV file that contains an extra character for all elements in the "sasid" column and we know the ODS will reject that value in that format. We'll account for this problem by introducing a custom script that cleans up the entire file before performing any mappings, ultimately resulting in the successful loading of the ODS.

Consider a CSV file from a third-party system which contains Student Assessment data. In this case, the third-party system produces data with an extra pound sign in one of the columns we are interested in (sasid):

StudentAssessmentsWithExtraPoundSigns.csv

```
adminyear,DistrictNumber,DistrictName,SchoolNumber,SchoolName,sasid,"
listen,ingss_adj",speakingss_adj,readingss_adj,writingss_adj,
comprehensionss_adj,oralss_adj,literacyss_adj,Overallss_adj
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604825,"
333,444",349,270,246,289,341,258,283
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604826,
303,392,100,100,161,348,100,174
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604835,
363,230,152,202,215,297,177,213
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604864,
294,262,251,263,264,278,257,263
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604870,
209,237,269,277,251,223,273,258
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604888,
270,237,296,251,288,254,274,268
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604890,
270,262,289,242,283,266,266,266
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604904,
934,948,932,926,933,941,929,933
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604902,
939,939,928,930,931,939,929,932
2018,255901,Grand Bend ISD,255901107,Grand Bend Elementary School,#604876,
938,925,929,916,932,932,923,925
```

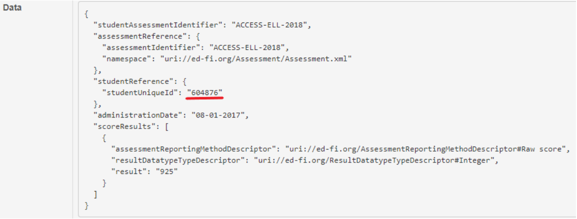
A custom script to "trim" these excess pound signs is fairly straightforward:

GenerateAssessments.ps1

```
$inputCsvFile = "C:
\Temp\FileGenerationWorkingFolder\StudentAssessmentsWithExtraPoundSigns.
csv"
$outputCsvFile = "C:
\Temp\FileGenerationWorkingFolder\StudentAssessmentsFixed.csv"

function TransformCsv($inputCsvFile, $outputCsvFile, $alterRow) {
    Import-Csv $inputCsvFile | ForEach-Object { Invoke-Command $alterRow -
ArgumentList $_ } | Export-Csv $outputCsvFile -NoTypeInformation
}

TransformCsv $inputCsvFile $outputCsvFile {
    param ($row)

    $row.'sasid' = $row.'sasid'.Replace("#", "")

    # Output the modified row.
    return $row
}

return $outputCsvFile
```

In short, there is a raw file at an arbitrary location that we need to clean up. We read in that file, remove the pound signs from the "sasid" column, and save it in a new file at any arbitrary location. The file path of the new file is then returned.

After placing this PowerShell file within the Share as described above, and updating our *Agent* to use this script by using the "File System / PowerShell" Agent Type, we run Transform/Load and encounter success. The *Logs \ Ingestion* screen shows that cleaned-up values were POSTed to the ODS:



Data

```
{
  "studentAssessmentIdentifier": "ACCESS-ELL-2018",
  "assessmentReference": {
    "assessmentIdentifier": "ACCESS-ELL-2018",
    "namespace": "uri://ed-fi.org/Assessment/Assessment.xml"
  },
  "studentReference": {
    "studentUniqueId": "604876"
  },
  "administrationDate": "08-01-2017",
  "scoreResults": [
    {
      "assessmentReportingMethodDescriptor": "uri://ed-fi.org/AssessmentReportingMethodDescriptor#Raw score",
      "resultDatatypeTypeDescriptor": "uri://ed-fi.org/ResultDatatypeTypeDescriptor#Integer",
      "result": "925"
    }
  ]
}
```