Transforming a CSV file in transit between two systems

Sometimes the gap between two systems that need to exchange data is entirely a formatting problem: wrong column order, different delimiter, extra header columns the receiving system does not understand, or a computed field that must be added before import. Rather than standing up a dedicated ETL platform for light transformations, you can express the transformation as a few lines of JavaScript and let AFT! handle both the transfer and the rewriting in a single job. This article walks through a pattern that downloads a CSV file from one VFS, transforms it in memory, and delivers the result to a second VFS.

How It Works

The script exports the source file to a local temporary location, reads and parses it, applies your transformation, serializes the result back to CSV, writes the output to a second temporary file, and then imports that file to the destination. A `try/finally` block ensures both temporary files are always cleaned up, even if an error occurs mid-run.

```
// CSV transform pipeline
// Downloads a CSV file from a source VFS, applies a transformation, and
// uploads the result to a destination VFS. Temporary files are always
// deleted on exit.

var srcVfsName  = Params("src_vfs")  || "source-system";
var srcPath     = Params("src_path") || "/exports/data.csv";
var dstVfsName  = Params("dst_vfs")  || "target-system";
var dstDir      = Params("dst_dir")  || "/imports/";

// Build a timestamped output filename so destination files never collide.
var ts          = FormatDateTime("YYYY-MM-DD_HHmmss");
var dstFileName = "data-" + ts + ".csv";

var tmpIn  = GetTempFileName();       // local temporary file for the downloaded source
var tmpOut = GetTempFileName();       // local temporary file for the transformed output

try {
```

```
// Download
var src = VirtualFSByName(srcVfsName);
var dlResult = src.ExportFile(srcPath, tmpIn);
if (!dlResult.Ok()) {
    Log.Error("download failed: " + dlResult.ErrorMsg());
    Exit(1);
}

// Parse
var raw  = ReadTextFile(tmpIn);
var rows = ParseCSV(raw, ",");    // second argument is the delimiter; omit for comma default

// Transform
// Replace the body of this loop with your own logic.
// This example uppercases column 0 (a name field) and skips the original header row.
var header = ["ID", "NAME_UPPER", "VALUE"];   // write your own output header
var output = [header];
for (var i = 1; i < rows.length; i++) {       // i = 1 to skip the source header
    var row = rows[i];
    if (row.length < 3) continue;             // skip blank or malformed lines
    output.push([
        row[0],                           // ID: pass through unchanged
        row[1].toUpperCase(),                 // NAME: transform applied here
        row[2]                            // VALUE: pass through unchanged
    ]);
}

// Serialize and write
var csv = FormatCSV(output, ",");
WriteTextToFile(tmpOut, csv);

// Upload
var dst = VirtualFSByName(dstVfsName);
var ulResult = dst.ImportFile(tmpOut, dstDir + dstFileName);
if (!ulResult.Ok()) {
    Log.Error("upload failed: " + ulResult.ErrorMsg());
    Exit(1);
}

Log.Info("transformed and delivered " + dstFileName + " (" + (output.length - 1) + " rows)");
```

```
} finally {
    DelFile(tmpIn);
    DelFile(tmpOut);
}
```

`ParseCSV(raw, ",")`: returns a two-dimensional array where each element is a row and each element of a row is a field. The second argument is the delimiter. Omit it and the engine defaults to a comma. Use `"\t"` for tab-separated files.

`FormatCSV(output, ",")`: the inverse of `ParseCSV`. Takes a two-dimensional array and joins it back into a string using the delimiter you supply. Both functions handle quoting and embedded newlines according to RFC 4180.

`var output = [header]; ... output.push([...])`: the transformation loop is the only part you need to change for a different requirement. Deleting columns is as simple as not including them in the pushed array. Adding a computed column means appending a new element. Sorting requires pulling the data rows (indices 1 to n) out into a separate array, sorting it, and merging it back with the header.

The `try/finally` cleanup block is not optional. If the upload fails or the transformation throws an exception, reaching `DelFile` on both temporaries is essential. Leaving temporary files accumulating on the AFT! host is both a security concern and a disk-space concern over time.

Changing the source delimiter

Some partners deliver semicolon-delimited files that are technically not standard CSV. Pass `";"` as the second argument to `ParseCSV` and `FormatCSV` to handle them without any other changes.

Selecting a subset of columns

To drop columns from the output, simply do not include them in the push call. There is no need to filter the input; just address the indexes you want.

Adding a computed column

Append a new element to each pushed row:

```
output.push([
    row[0],
    row[1].toUpperCase(),
    row[2],
    (parseFloat(row[2]) * 1.21).toFixed(2)    // computed tax-inclusive total
]);
```

Remember to add a corresponding header in the `header` array at the top so the receiving system knows what it is getting.

Revision #1
Created 12 April 2026 14:31:25 by DevTeam
Updated 12 April 2026 15:25:43 by DevTeam