

Import and Transform spreadsheets using Espresso Logic SaaS

The Problem

Thousands of businesses run their business using spreadsheets. Each sheet contains information critical to the daily operation and decision support of a company. There are tools today that allow end-users the ability to import spreadsheets directly into a database table. If the table column attributes do not match or the datatypes need conversion (dates, currency, special numbers, special characters) then the problem becomes a bit more complex. Also – if the data in these reports is changing – does the target require appending, replacing, merging values?

The Solution

Espresso Logic offers a REST API service and business logic development services that can connect to an existing database and a dynamic user interface called Live Browser™ that changes with the database model. Using a pre-defined project, the various reports (e.g. Inventory, Orders, Products, Vendore, Pricing, or budget CSV files) can be loaded into a database and business rules apply to the data for transformation and decision support.

The Design

A new SQL model (figure 1) defines each type of CSV file map (*mastermapper*). The model defines the target table, the column mapping (*mastermappercolumns*), and holds the CSV Content and name/value pair details (*mapperdetails*). Each report is loaded in a CSV (comma-separated value) format and uploaded into Espresso using the Live Browser™ (figure 2). The system will parse the file into a set of name/value pairs and generate a column mapping entry for the row header. The column mapper defines the target table (in another Espresso Logic Account/Project) and attribute names and datatypes (figure 3). After the details are loaded, a processing flag is set to transform the name/value pairs into the target table using the mapped attribute name and doing the proper conversions on the values.

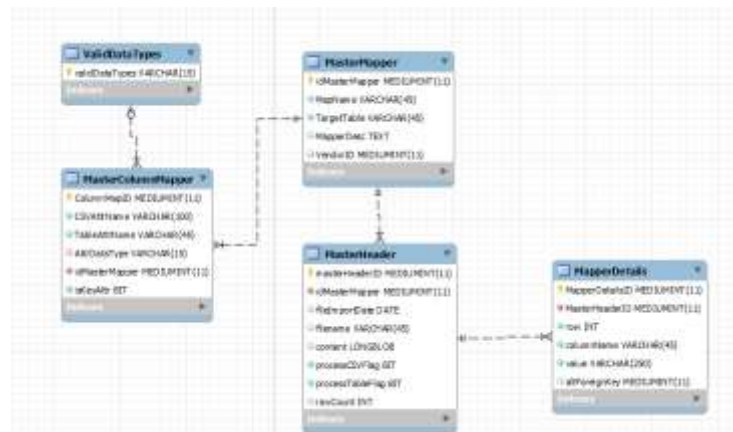


Figure 1 – CSV Data Model

How it works

The data model is loaded into a new database. Then the Espresso Logic Design studio connects to the model using the database credentials. That is it – the REST API is ready for the Live Browser™ and is

ready to run. The first step is define the MasterMapper which holds the name of the map, the target table URL (for POST) and a brief description

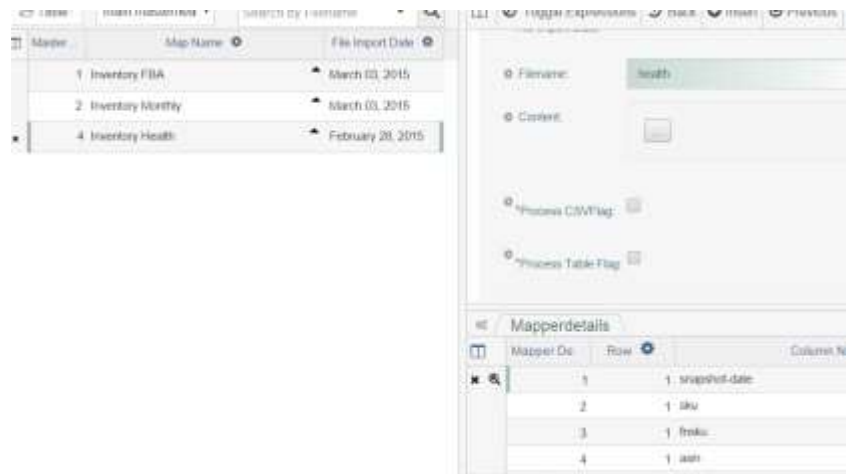


Figure 2 – Live Browser™ Upload Content and process details

The next step is to create a new Header (*masterheader*) which holds the CSV content. This flags will trigger rules to each step of the CSV processing into name/value pairs (*mapperdetails*).

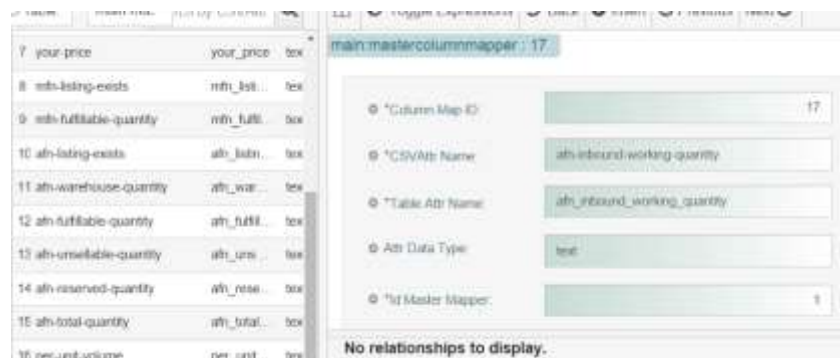


Figure 3 – Attribute Name and Data Type Mapping

The final step is to define the target table attribute (column) names and attribute datatypes (text, money, number, yyyyMMdd, ddMMyyyy, date). These names will be used to read the name/value pairs (stored as text strings) and convert them to rows to POST to the target table. Espresso Logic JavaScript can call user defined methods in libraries that are added using the Design Studio. The CSVParserToJSON is based on the GitHub branch library (<https://github.com/tylerm007/jackson-dataformat-csv/tree/EspressoLogicJSONMapper>)

Notes: When mapping CSV files, make sure there are no special characters, double quotes or extra commas in the name. Also, try to have dates in a standard format for the parser to convert (dashes and slashes will be removed before conversion). Finally, make sure your target table exists and that the column attribute names match the name and datatype in the column mapper (number is INTEGER, and money will convert to DECIMAL).

Figure 4 – UPDATE Event Rule to process CSV File (Server JavaScript)

```
var parms = null;
var settings = {
  headers: {"Authorization": "Espresso WUyXwrLEgvhMxhUYYHhI:123"}
};
var csvParser = new com.espressologic.file.csv.CSVParserToJSON("idMasterMapper");
var header = csvParser.convertFileHeaderToJSON(row.idMasterMapper, 1, row.content);
var columnMap = SysUtility.getResource("columnMap",{filter: {idMasterMapper: row.idMasterMapper}});
if(Array.isArray(columnMap) && columnMap.length <= 0){
  log.debug("insert header to master column mapper");
  var headerData = JSON.parse(header);
  var urlh = req.baseUrl+"v1/main:mastercolumnmapper";
  var postHeader = SysUtility.restPost(urlh,parms,settings,headerData);
  log.debug(postHeader);
  columnMap = SysUtility.getResource("columnMap",{filter: {idMasterMapper: row.idMasterMapper}});
}
csvParser = new com.espressologic.file.csv.CSVParserToJSON("MasterHeaderID","altForeignKey");
if(Array.isArray(columnMap) && columnMap.length > 0){
  log.debug("use map column headers to transform to real column names");
  for (var i = 0; i < columnMap.length; i++) {
    csvParser.addColumnMap(columnMap[i].CSVAttrName ,columnMap[i].TableAttrName );
  }
}

var result = csvParser.convertFileToJSON(row.masterHeaderID,1,row.content);
// this can be very large
// log.debug(result);
var json = JSON.parse(result);

//we could break the JSON into array[] and post smaller sets
//if(Array.isArray(json) && json.length > 1){
//  //data = json[i]
//}
var url = req.baseUrl+"v1/main:mapperdetails";
log.debug(url);
//POST is async - that is - it is not part of this transaction
var post = SysUtility.restPost(url,parms,settings,json);
row.content = null;//no need to store the large CSV file in the database
log.debug(post);
```

Figure 5 – Process Name/Value pairs into target table

```
if(row.processTableFlag == 1 && oldRow.processTableFlag != 1){
  //STEP 1 - Get the Mapping Rows
  var rowNum = 1;
  var tableArray = [];
  var batchSize = 100;
  var pgsize = batchSize;
  var offset = 0;
  var jsonRow = {};
  var parms = null;
  var url = req.baseUrl;
  url = url +"v1/main:"+row.fk_MasterHeader_MasterMapper1.TargetTable;
  var settings = {
    headers: {"Authorization": "Espresso demoapikey:123"}
  };
};
```

```

var mapRows = SysUtility.getResource("columnMap",
    {filter: "idMasterMapper = "+row.idMasterMapper,
    pagesize: 50,
    offset: 0});
//STEP 2 - get the map detail list rows 40 at a time
var detailRows = SysUtility.getResource("mapDetails",
    {filter: "MasterHeaderID = "+row.masterHeaderID,
    order: "row asc",
    pagesize: pgsz,
    offset: offset});

//STEP 3 - convert each row set into a JSON insert for target tablet - only select the
mapRows
while(Array.isArray(detailRows) && detailRows.length > 0){

    for (var i = 0; i < detailRows.length; i++){
        if(rowNum !== detailRows[i].row && detailRows[i].row > 1 ){
            rowNum = detailRows[i].row;
            tableArray.push(jsonRow);
            jsonRow = {};
        }

        for (var j = 0; j < mapRows.length; j++){
            if(detailRows[i].columnName === mapRows[j].TableAttrName
            || detailRows[i].columnName === mapRows[j].CSVAttrName){
                try{
                    var value = detailRows[i].value; //text
//replace this with eval() JS and regex
                    if(mapRows[j].AttrDataType == 'money'){
                        value = value.replace("$", "");
                        value = value.replace("%", "");
                        value = parseFloat(value);
                    } else {
                        if(mapRows[j].AttrDataType == 'number'){
                            Number(value);
                        } else {
                            if(mapRows[j].AttrDataType == 'date'){
                                value = new java.text.SimpleDateFormat("dd/MM/YYYY").parse(value);
                            }
                        }
                    }
                } catch(e){
                    log.debug(e.message);
                    if(mapRows[j].AttrDataType == 'money' || mapRows[j].AttrDataType ==
'number'){
                        value = 0;
                    } else {
                        value = null;
                    }
                }
                if(value !== null){
                    jsonRow[mapRows[j].TableAttrName] = value;
                }
            }
        }
    }
}

//STEP 4 - post the new array to target table
if(Array.isArray(tableArray) && tableArray.length > 0){

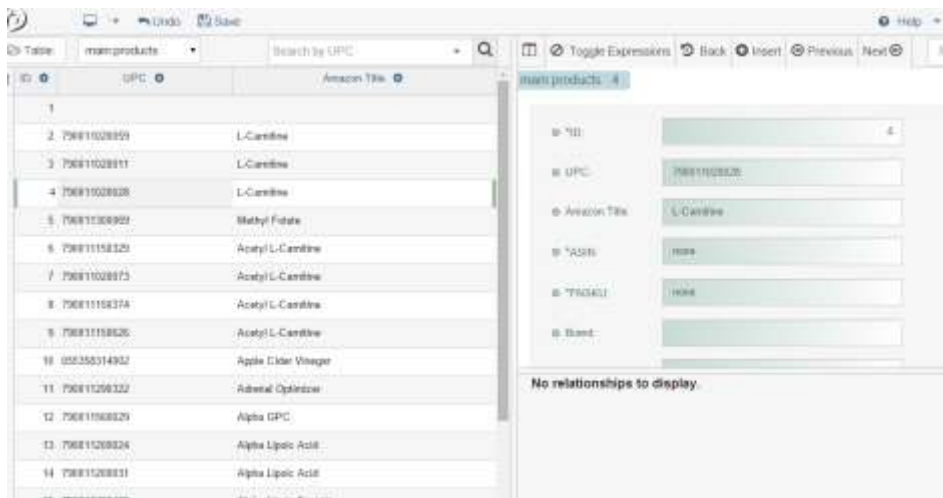
```

```

log.debug(url);
var post = SysUtility.restPost(url,parms,settings,tableArray);
log.debug(post);
}
offset = offset + pgsize;
//Step 5 – read the next batch and loop until done
detailRows = SysUtility.getResource("mapDetails",
    {filter: "MasterHeaderID = "+row.masterHeaderID,
    order: "row asc",
    pagesize: pgsize,
    offset: offset});
tableArray = [];
} //while loop
}

```

Once the processing has completed – the target table (Figure 6) will have the content of the CSV file ready for review.



ID	UPC	Amazon Title
1		
2	79681028901	L-Cardine
3	79681028911	L-Cardine
4	79681028928	L-Cardine
5	79681028929	Methyl Forte
6	79681118329	Acetyl L-Carnitine
7	79681028973	Acetyl L-Carnitine
8	79681118374	Acetyl L-Carnitine
9	79681118326	Acetyl L-Carnitine
10	052350314902	Apple Elder Vinegar
11	79681028932	Adrenal Optimize
12	79681118329	Alpha GPC
13	79681028924	Alpha Lipic Acid
14	79681028931	Alpha Lipic Acid

Figure 6 – The target table fully populated from the imported CSV

Summary

Espresso Logic generic sample applies to many different types of CSV files. The ability to create new target database tables, import CSV files, and map/transform them gives developers a new tool to get a complete picture of their business using Live Browser. The ability to run this service on-premise or in the cloud gives tremendous flexibility. Sales reports, pricing, orders, inventory, budgets, and external analysis tools all can be easily loaded into your database using Espresso Logic REST API services.

Future Features

1. Break the parsed details into smaller arrays and post in batches
2. Change the way the conversions are handled to use JavaScript eval() and regex
3. Better date formatting and handling
4. Make the process async using the logic rules
5. Better error handling
6. Add ApiKey to mastermapper to pass with target URL