

# Data File Processor

esProc · Issued by Raqsoft



[www.raqsoft.com](http://www.raqsoft.com)



## Often faced with such tasks?

- Do the same summary statistics for more than 500 Excel files
- Remove duplicate lines from text files
- Compare the difference between two CSV files
- Combine dozens of Excel files;  
Split large Excel into several small ones
- Several Excel data needs to be joined with the same column



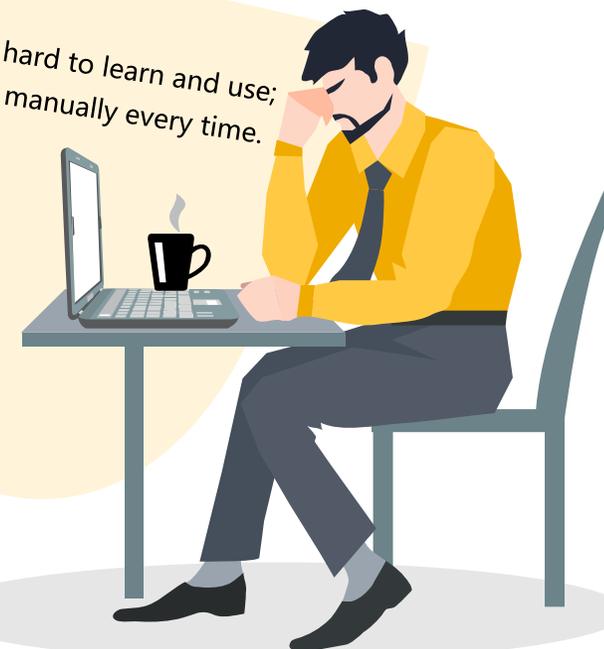
Obsessed with these problems?

File can be handled if loaded into MySQL, but it is too troublesome to install and import

Python installed, but version is not right. It can read Excel2017, but can not recognize Excel2003

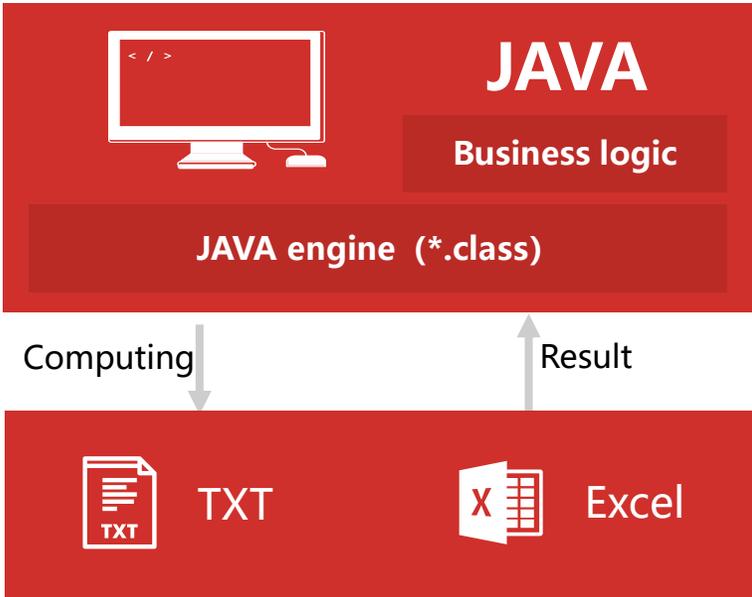
Java is too hard to learn and use;  
Have to do manually every time.

The file is too large, memory overflowed after reading

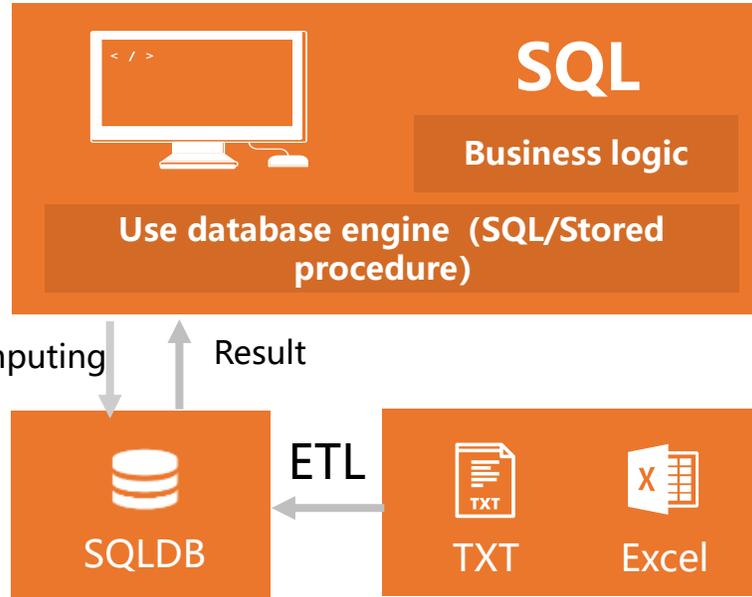




## > The trouble of data file calculation



- ### JAVA
- Need professional programmers, high technical requirements
  - Computing class libraries are few and the code is tedious
  - Complex configuration of development environment



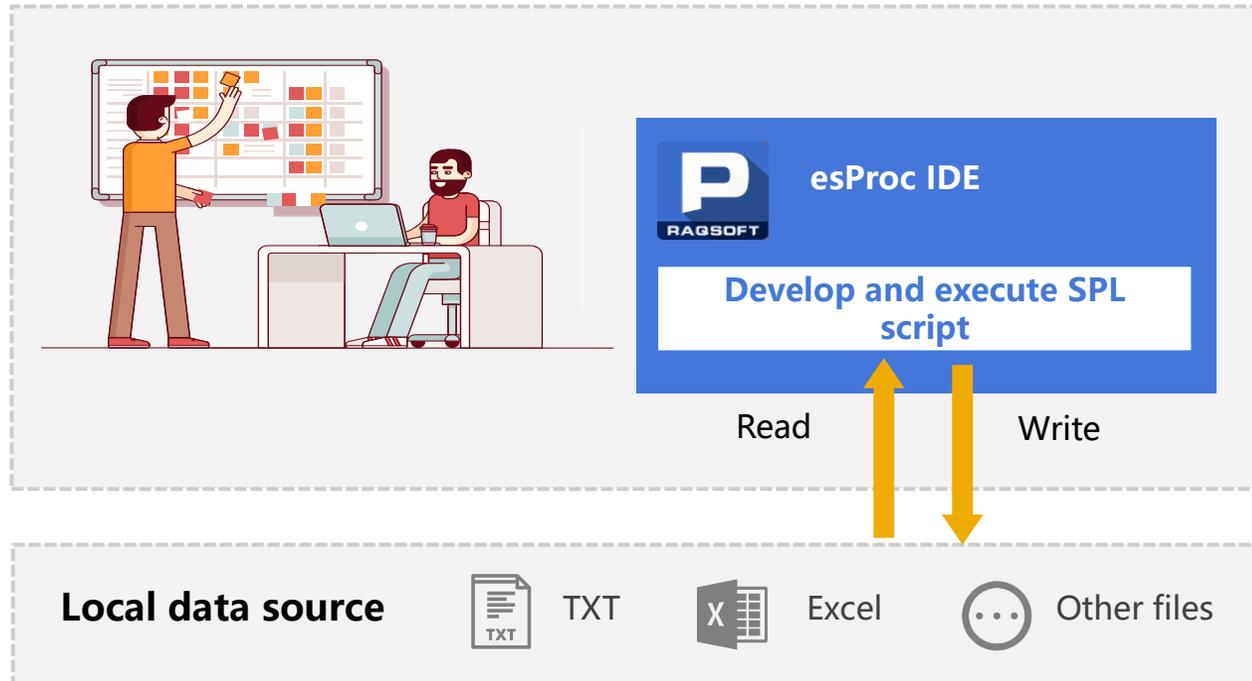
- ### SQL
- File needs to be loaded into database before calculation, very troublesome
  - Complex deployment, configuration and algorithm
  - IDE is unfriendly, debugging is difficult



- ### Python
- Syntax is not designed specifically for structured data computing
  - Complex computing is not easy to develop
  - Incompatible between versions
  - Troublesome installation and configuration of peripheral open source package



## Data file calculation using esProc



With basic programming knowledge, you can calculate files freely, self-service and quickly

Desktop level tool, ready to use, simple environment configuration!

# Convenient development environment



Install and use immediately, with perfect debugging function

Execute/Debug/Step

Set breakpoint

The screenshot displays a development environment with a menu bar (File, Edit, Program, Tool, Window, Help) and a toolbar. The main workspace shows a code editor with the following code:

```
A2 = 1 =file("../demo\zh\bt\Sale.txt").import@t().select(month(Datetime)==6)
2 =file("../demo\zh\bt\Stock.txt").import@t().select(month(Datetime)==6)
3 =file("../demo\zh\bt\Storage.txt").import@t().select(month(Date)==5)
4 =file("../demo\zh\bt\Commodity.txt").import@t()
5 '08:00:00 '21:30:00
6 =periods@d(date("2009-6-1"), date("2009-6-30"), 1)
7 =A1.align@a(A6:~,date(Datetime))
8 =A2.align@a(A6:~,date(Datetime))
9 =A4.new(ID:Commodity,0:Stock,0:OosTime,0:TotalOosTime)
10 >A9.keys(Commodity)
11 =A3.run(A9.find(Commodity) Stock=Stock)
13 =A9.select(Stock<=0).run(OosTime=string(A12)+" "+A:
14 = =A9.find(B14.Commodity)
15 >C14.run(Stock=Stock-B14.Volume)
16 =C14.Stock=0 >C14.OosTime=B14.Dat
```

On the left, a console window shows system information:

```
File Console
System... Copy Clean
Log level:WARNING
Log level:INFO
```

On the right, a data table is displayed:

Index	Datetime	Commodity	Volume
1	2009-06-01 08:05:00	20077	28
2	2009-06-01 08:11:40	20056	47
3	2009-06-01 08:18:20	20094	34
4	2009-06-01 08:21:40	20020	19
5	2009-06-01 08:41:40	20013	42
6	2009-06-01 08:45:00	20077	1
7	2009-06-01 08:51:40	20069	19
8	2009-06-01 09:05:00	20011	22
9	2009-06-01 09:08:20	20007	22
10	2009-06-01 09:11:40	20005	39
11	2009-06-01 09:18:20	20085	31
12	2009-06-01 09:21:40	20054	8

At the bottom, a variable declaration table is visible:

No.	Name	Value
	Cellset variable	
	Task variable	
	Global variable	

Real-time system info output

WYSIWYG-style interface that enables easy debugging and convenient intermediate result reference

Simple syntax, natural & intuitive computing logic

## Agile syntax



Count the longest consecutively rising trading days for a stock.

```
1 import pandas as pd
2 def iterate(col):
3     prev = 0;
4     res = 0;
5     val = 0;
6     for curr in col:
7         if curr - prev > 0:
8             res += 1;
9         else:
10            res = 0;
11            prev = curr;
12            if val < res:
13                val = res;
14            return val;
15 data = pd.read_excel( 'D:/Stock.xlsx' ,sheet_name=0).
    sort_values( 'Date' ).groupby( 'Company' )[ 'Price' ].apply(iterate);
```

Python

	A
1	=file( "D:/Stock.xlsx" ).xlsimport@t().sort(Date).group(Company)
2	=0
3	=A1.max(A2=if(Price>Price[-1],A2+1,0))

SPL

Professional syntax in the field of set operation,  
code is more concise for the same process!



## Direct SQL syntax



Use SQL directly for files, and there is no need to import data into database.

	A
1	<b>\$SELECT * FROM D:/Stock.xlsx WHERE Company='0001' ORDER BY date</b>
2	<b>\$SELECT Company,max(price),min(price) from D:/Stock.xls WHERE month(date)=1 GROUP BY Company</b>
3	<b>\$SELECT Company.Name,Stock.date,Stock.price FROM D:/Stock.xls Stock LEFT JOIN D:/Company.txt Company ON Stock.company=Company.ID</b>

Simple SQL in SPL



# Procedure-oriented computing

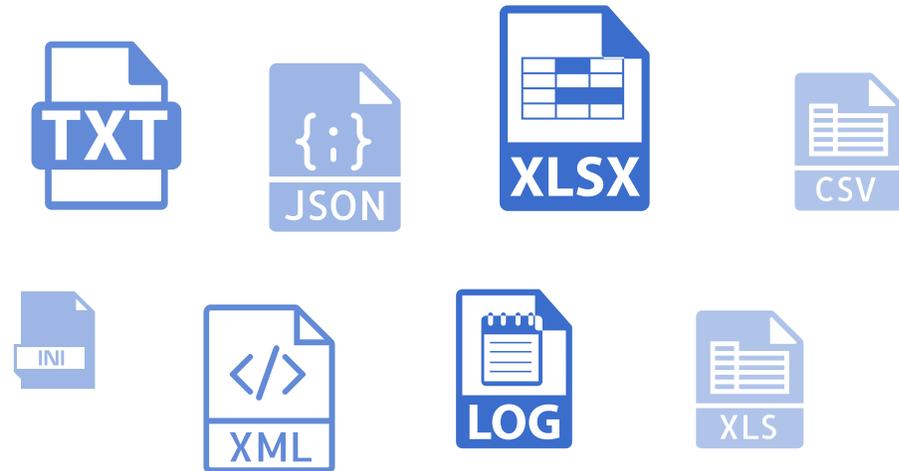


## Reliable loop branch control

	A	B	C	D	E	F
1	=esProc.query("SELECT orderID as contract, orderDate as date, customer, amount, empID as salesman FROM sales where year(orderDate)=? OR year(orderDate)					
2	=esProc.query(select * from employeeInfo")					
3	>A1.run(salesman=A2.select@1(ID:A1.salesman))	/field value is record				
4	>A1.group(salesman)					
5	=create(salesman, thisyearAmount, lastyearAmount, growthRate, custNumber, bigCustNumber, bigCustProportion)					
6	for A4	=A6(1).salesman.name				
7		=A6.select(year(date)==year).sum(amount)				
8		=A6.select(year(date)==year-1).sum(amount)				
9		=B8/B7-1	/growth rate			
10		=A6.group(customer).(~.sum(amount))				
11		=B10.count()	/number of customer			
12		=B10.count(~>=10000)	/number of big customer			
<b>Natural &amp; clean step-by-step computation, direct reference of cell name without specifically defining a variable</b>						
14		=A5.insert(0,B6,B7,B8,B9,B11,B12,B13)				
15	result A5					

## Multiple file formats

- ✓ Txt, csv, log, ini
- ✓ Xls, xlsx
- ✓ Excel versions 2003...2013...2019
- ✓ Xml, json





## Example: Differences between texts

### Algorithm

Relative to old.csv, find the new records in new.csv, where username and date jointly determine a record.

### File

	Old.csv	New.csv
	userName,date,saleValue,saleCount	userName,date,saleValue,saleCount
1	Rachel,2015-03-01,4500,9	Rachel,2015-03-01,4500,9
2	Rachel,2015-03-03,8700,4	Rachel,2015-03-02,5000,5
3	Tom,2015-03-02,3000,8	Ashley,2015-03-01,6000,5
4	Tom,2015-03-03,5000,7	Rachel,2015-03-03,11700,4
5	Tom,2015-03-04,6000,12	Tom,2015-03-03,5000,7
6		John,2015-03-04,4800,4

### Code

	A	B	C
1	=file("d:\\old.csv").import@ct()	=file("d:\\new.csv").import@ct()	/Comma separated text
2	=new=[B1,A1].merge@od()		/Difference set



## > Example: Group aggregation of multiple excel and multiple sheets

### Algorithm

There are multiple excel files in the same directory, and each excel file has multiple sheets. The format of the sheets is the same. Merge all the sheets of all excel files, group by month and calculate the sum and average of amount.

### File

Customer ID	Customer Name	Invoice Number	Amount	Purchase Date
1234	John Smith	100-0002	\$1,200.00	2013/1/1
2345	Mary Harrison	100-0003	\$1,425.00	2013/1/6
3456	Lucy Gomez	100-0004	\$1,390.00	2013/1/11
4567	Rupert Jones	100-0005	\$1,257.00	2013/1/18
5678	Jenny Walters	100-0006	\$1,725.00	2013/1/24
6789	Samantha Donaldson	100-0007	\$1,995.00	2013/1/31

### Code

	A	B	C
1	for directory@p(" d:/excel/*.xlsx ")	=file(A1).xlsopen()	/Loop every excel in the directory
2		=B1.conj(B1.xlsimport@t(;~.stname))	/Loop every sheet
3		=@ B2	/Merge sheets in turn
4	=A4.groups (month('Purchase Date'):Month;sum(Amount):Total,avg(Amount):Average)		/Group aggregation



## Resource link



- SPL codes of common calculation <http://doc.raqsoft.com/>
- Structured Text Computing <http://c.raqsoft.com/article/1571711703952>
- SPL parsing and exporting Excel <http://c.raqsoft.com/article/1571712027010>
- JSON data calculation and importing into database <http://c.raqsoft.com/article/1576466956518>
- Installation and free authorization <http://c.raqsoft.com/article/1573787506233>