



# Tutorial

## 1. esProc Description

### 1.1 Description of esProc Functions

esProc users can complete batch data computing with the following basic functions:

- Data Analysis and Structured Computation
- Free Access to Database and Online Data Analysis

### 1.2 Installation

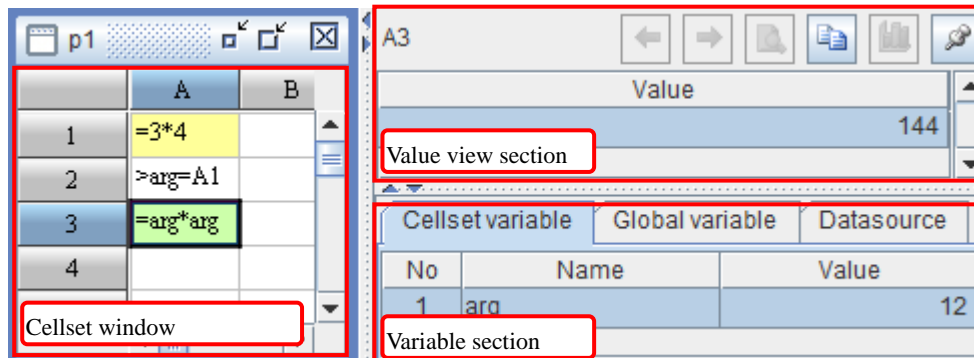
To install esProc, please run the installer according to the instructions below step by step:

- Run installer
- Click Next to continue, accept the agreement
- Select the installation directory, click Install
- Complete the installation

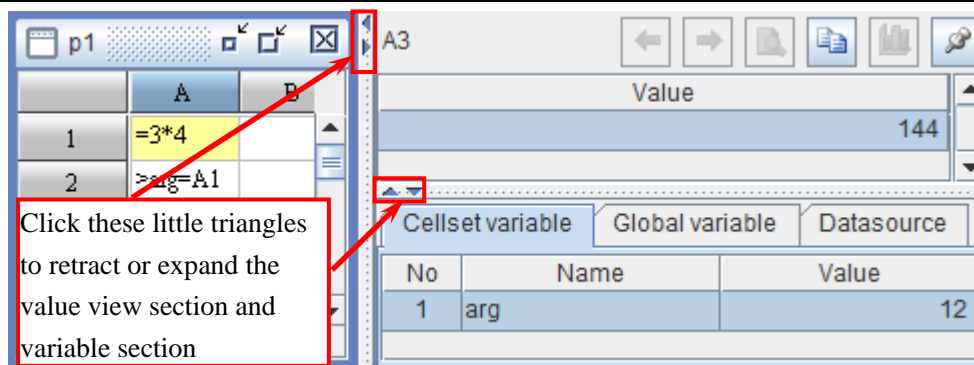
If you are familiar with the configuration of running environment of Java, and JDK1.5 or higher version has been installed locally, then you can also choose to launch the esProc installer that will not install JDK automatically but prompt you to navigate to the directory of JDK in the local machine.

### 1.3 Interface Layout and Cellset File Creation

Run esProc main program and open **esProc standard** and click  button to create the cellset file.

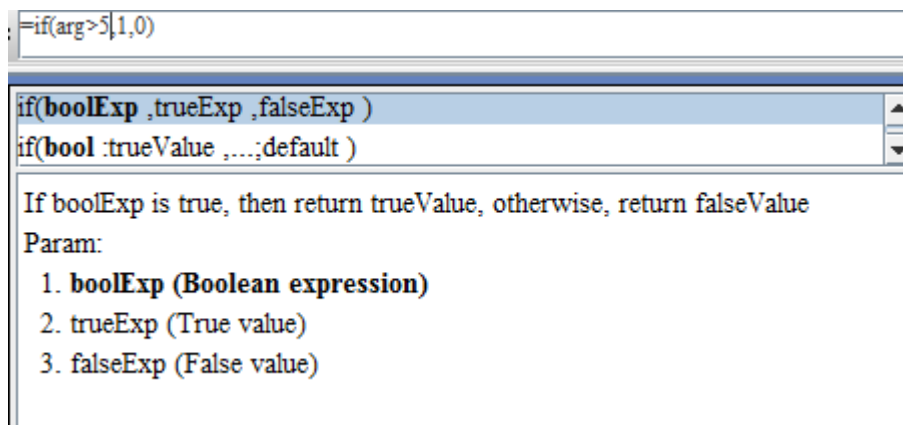


On the left is the **Cellset window**, the active cellset files you are operating are shown here; on the right, the upper area is the **Value View section** and the lower area is the **Variable section**. Variable section and Data View section can be retracted and expanded easily.



## 1.4 Auxiliary Function for Editing Functions in esProc

esProc provides lots of functions. And auxiliary function for editing functions can make code writing in cells more easily. On the aiding interface for editing functions, you can check the function syntax, description, and other information:



You can open/close the auxiliary function for editing functions by pressing Alt+↓.

## 2. Basic Operation

### 2.1 Data Types in esProc

#### 2.1.1 Data Types

In esProc, the following data types are available:

##### Integer

Any integer between  $-2^{31} \sim 2^{31}-1$ , i.e. the value range is -2147483648~2147483647. You can use the type conversion function **int()** to convert other types of data into integer.

##### Long integer

$-2^{63} \sim 2^{63}-1$ , which is a value range greater than that for integer type. You can use the type conversion function **long()** to convert other types of data into long integer.

Particularly, long integer can be represented by appending a capital letter L to the integer. Compared with an integer, a long integer has a bigger value range; strings starting with **0x** can also be used to represent the hexadecimal long integers.

	A	
1	=123456789L*100	Value 12345678900
2	=123456789*100	Value -539222988
3	=0x00FF	Value 255

Since the value range of common integers is  $-2^{31} \sim 2^{31}-1$ , that is, -2147483648~2147483647, the result in A2 is beyond boundary. While by using a long integer in A1, the value range is increased to  $-2^{63} \sim 2^{63}-1$  and correct result can be obtained. It can be noticed that, during performing the operation of a certain step, if one of the operands involved in the operation is a long integer, the result will be a long integer.

### Floating-point number

64-bit floating-point number is the commonest data type for esProc. All real numbers related computations are performed with this data type. You can use the type conversion function **float()** to convert other types of data to floating-point number. Because the floating-point number is used to store data according to the binary rule, there could be some errors in the computation.

### Big decimal

Big decimal can be used to store any real number error-freely, but more memory could be consumed when using big decimal for computation and the computational efficiency is relatively low. The type conversion function **decimal()** can be used to convert other types of data into the big decimal.

54, 43.31, -4.45E13, 3%

### Real number

The real number covers four data types: integer, long integer, floating-point number and big decimal. **number()** is the type conversion function available to change other data to real number.

### Boolean

It includes true/false.

- **true, false**

### String

Double quoted in an expression, and the escape character is \. If the string is defined in a constant cell directly, the double quotation marks are omitted. You can use the function **string()** to convert other types of data to strings. When concatenating two strings *x* and *y* in an expression, you can add the space *x y* in-between.

- **abcd**
- **="US\tChina"**
- **="Beijing" "China"**      Concatenate two strings to get the result "Beijing China".

The escape rule is the same as that of JAVA. For details, please refer to the manual.

### Date / time

In the **yyyy-mm-dd** or **hh:mm:ss** format. Type conversion functions such as **date()**, **time()**, and **datetime()** can be used to convert the string or long integer to date, time, or datetime. **2010-1-3**,

23:04:23

Click the **Tool -> Options** menu item. On the **Environment** tab, set the format of time/date data, the character code, etc.

The screenshot shows the 'Option' dialog box with the 'Environment' tab selected. The fields are as follows:

Log file name	E:\tools\raqsoft\esProc\log\esproc.log	Browse	
Searching path	E:\tools\raqsoft\esProc\demo	Browse	
Main path	E:\tools\raqsoft\Hub\main	Browse	
Temp path	temp	Edit	
Date format	yyyy-MM-dd	Time format	HH:mm:ss
Date time format	yyyy-MM-dd HH:mm:ss	Default charset name	ISO-8859-1
Local host	192.168.0.1	Local port	8282
File buffer(Byte)	65,536		

Buttons: OK, Cancel

Precision loss maybe happen when converting a type of data into another one.

### 2.1.2 Judging the Data Type

In esProc, you can judge the type of data by the following functions:

#### **ifnumber(x)**

Judge if  $x$  is a real number.

#### **ifstring(x)**

Judge if  $x$  is a string.

#### **ifdate(x)**

Judge whether  $x$  is a date or a datetime.

#### **iftime(x)**

Judge if  $x$  is a time.

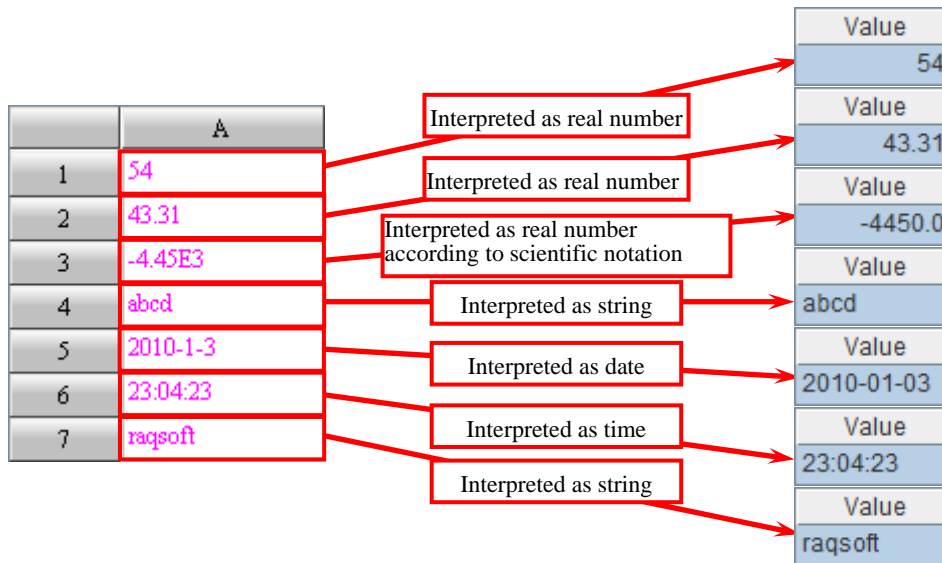
- **=ifnumber(3.5)**      true.
- **=ifstring(now())**      false, **now()** is used to compute the current datetime

## 2.2 Type of Cells

### 2.2.1 Constant Cell

The cell string interpreted as constant is **Constant Cell** by name. Its cell value is the constant.

The constant cell has the following types of values:



The text of the constant cell is in **pink** by default. Depending on the data in cells, it will be interpreted into various data types. For those unrecognized data, they will be taken as the string.

- **Note:** The representation style of **3%** is unacceptable in any expression, but it is acceptable in the constant cell.

Please note the common reserved words below and keep in mind that they are **case-sensitive**!

#### null

Null value; the cell value of any empty cell is always the null

#### true

True

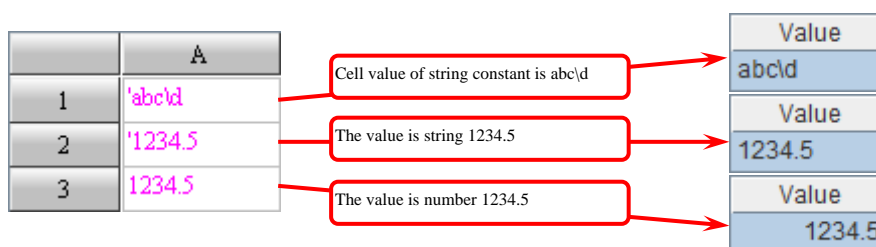
#### false

False

### 2.2.2 String Constant Cell

String Constant Cell is a cell whose cell string starts with single quote ', indicating it is a constant cell with a string value. The cell value is a string composed of those characters after the single quote. In this case, all characters following the single quote will be parsed as string, not requiring any additional quotes, escape characters, etc.

- 'abcd      String abcd
- '1234.5      String 1234.5




### 2.2.3 Computational Cell

A computational cell is the cell whose cell string starts with =, and that is used for computing an expression. Its cell value is the computed result of the expression, which can reference cell

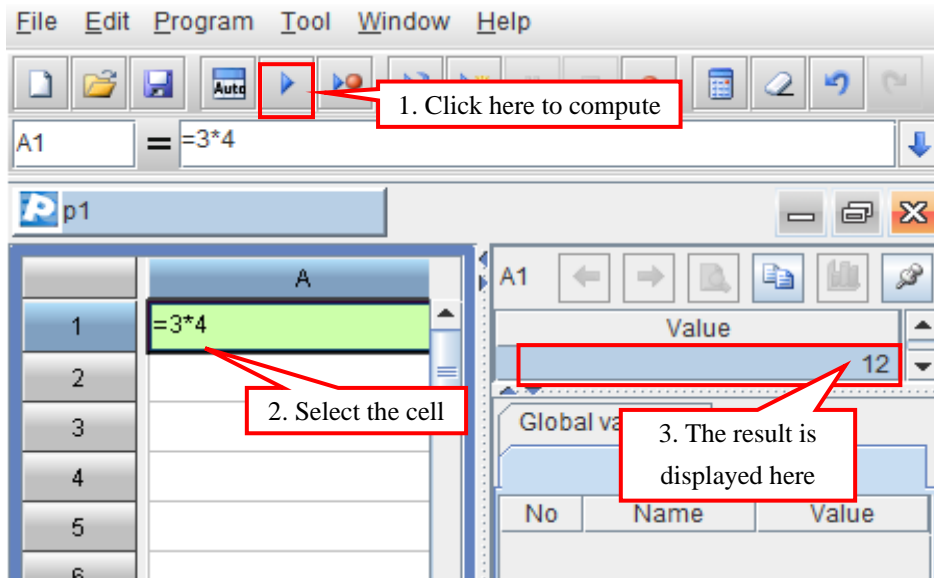
values of other cells  $=5*(3+2)$

- $=A1*3$
- $=A1/B2+B1$

The text of the computational cell is in black by default.

Click  to execute the code and you will find that the background of the computational cell turns to light yellow, indicating its cell value has been computed.

Click the cell with cell value. In the value view section, the current cell value will be displayed for you to view the computed result.



In the expression, you can use various basic operators to compute cell value:

$a+b$   $a-b$   $a*b$   $a/b$

Add, subtract, multiply, and division operations.

- $=3*4$       12
- $=1+2*3$     7

$a\%b$

Seek the remainder of  $a$  divided by  $b$ .

- $=121\%11$     0, which indicates 121 is a multiple of 11, and is divisible by 11
- $=100\%7$      2

$a\backslash b$

Integer division, in which the integer part of  $a$  and  $b$  is kept. The integer part of the result will be returned after division.

- $=11\backslash 4$         2, cut off the decimal part of the result
- $=6.1\backslash 2.9$     3, equal to  $6\backslash 2$ , the result is 3

$s_1+s_2$

Concatenate character string  $s_1$  and  $s_2$ . One thing to note, the string will be ignored if added to a real number.

- `= "Stephen" + " " + "Rolfe"` String Stephen Rolfe
- `= "A" + 3` Integer 3, string "A" will be ignored
- `= "A" + string(3)` String A3. If you do need to concatenate the string and real number, please convert the real number to string first.

#### 2.2.4 Executable Cell

An Executable cell is the cell whose cell string starts with `>`, in which certain actions will be executed. The executable cell has no cell value. If an executable cell starts with one of the reserved words, like **if**, **for**, etc. then `>` can be omitted.

The code in an executable cell can be used to change the value of another cell.

- `>A1=5`
- `>B1=A1+3`

In the executable cell, you can use the cells as variables.

Because the cell itself has no value, both A1 and B2 are the executable cells

Both A2 and B3 are assigned with values. Thus, the cell has value

#### 2.2.5 Comment Cell

A comment cell is the cell whose cell string starts with `/`. Its cell value is null.

Comment cells will be ignored during execution.

The text of comment cells is in **green** by default.

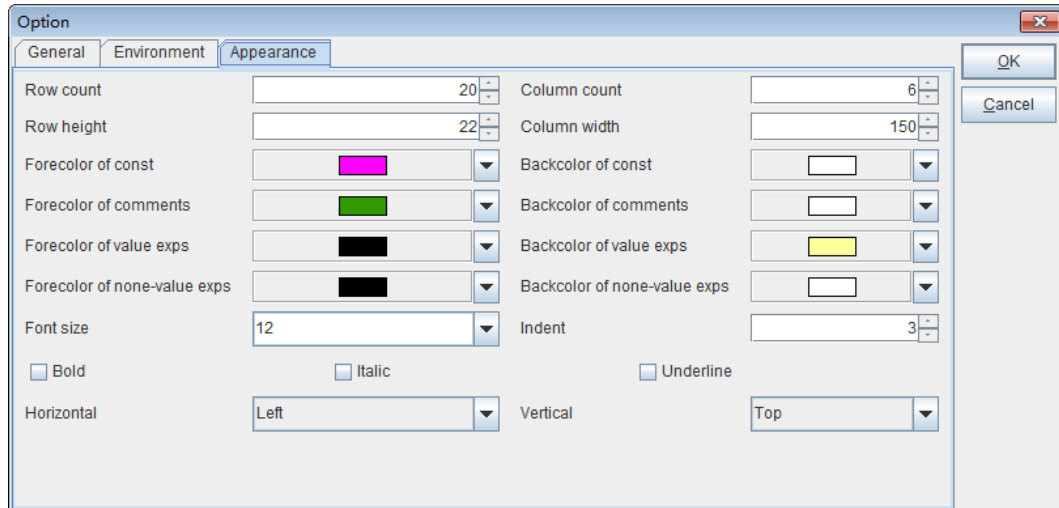
	A	B
1	<code>&gt;a=4</code>	<code>/define variable a</code>
2	<code>5</code>	<code>/define constant 5</code>
3	<code>=a+A2</code>	<code>/calculate a+5</code>

Comment cell

#### 2.2.6 Set the appearance of different types of cells

esProc provides the users with options to set the font styles and the foreground/background color of different types of cells. Click the **Tool -> Options**. On the **Appearance** tab, you can view and modify settings.

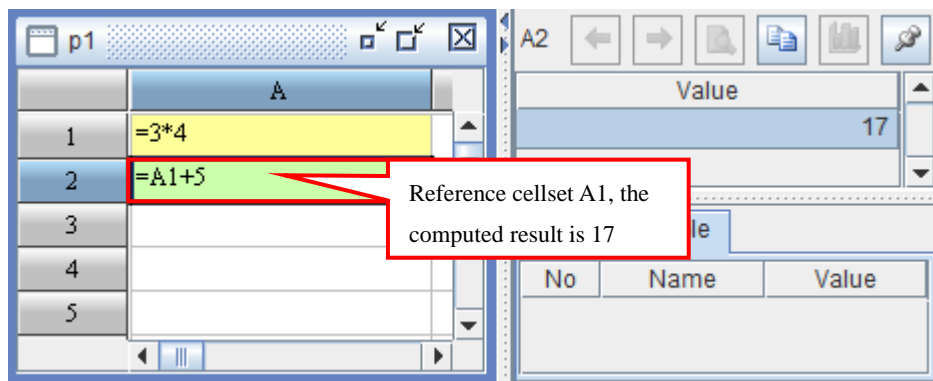




## 2.3 Variables and Parameters in the CellSet

### 2.3.1 Take Cell Values as Parameters

In the examples of 2.2.3 and 2.2.4, we can find that the cell value is used as variable. In esProc, if there is a value in a cell, then the value of this cell can be referenced with the cell name by the computational cell or executable cell. A cell with value can be a constant cell, a computational cell, or a cell assigned with value by an executable cell.



### 2.3.2 Define Cellset Parameters

You can define parameters in cellset files.

Click the **Program** -> **Parameter** to check the parameter settings of a cell:

No.	Name	Value	Remark
1	pi	3.14	
2	arg1	Hello	
3	arg2	[1,4,2,8]	

In the Program Parameter window, you can set the parameters to be used in a cellset file. The cell parameter name should not contain blanks or signs and should not use digits only. The parameter value can be various types of constants, and be parsed automatically into the corresponding data types according to the set value. Please note that the expression cannot be used in the cellset parameters.

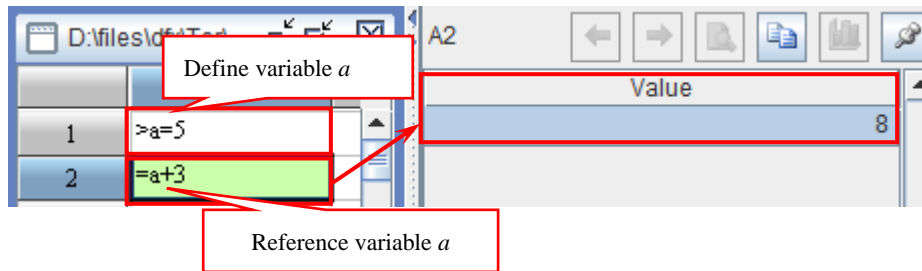
If you check the **Set arguments before run** option, then the Program Parameter window will pop up before computing the cellset.

When using the cellset parameters, you can directly call them with parameter names:

	A		Value
1	=pi*4	→	12.56
2	=arg1+" world!"	→	Hello world!
3	=arg2.(~*~)	→	Member
			1
			16
			4
			64

### 2.3.3 Cellset Variables

You can also use the named variables besides using cell names directly. These variables are called **Cellset Variables**.



You are not required to declare these variables in the first place. They will be auto-generated when assigning values to them, and are valid throughout the whole program cellset. But, referencing variables with no value assigned will cause error.

Cellset variables belong to no data types, and you can change their values at will.

**$a?=x$**

This is the abbreviation for  $a=a?x$ . “?” is an operator. Besides using the expression  $a=x$  to assign value to the variable  $a$ , we can use the abbreviated version  $a?=x$  to change the value of the variable.

**$a+=5$**       5 is added to the variable  $a$ . The expression equals to  $a=a+5$ .

**$-a$**

$a$  and  $-a$  are a pair of opposite number. Their only difference is the signs. Particularly, if the data type of  $a$  is date/time, the symmetric datetime of  $a$  for January 1, 1970 will be used in computing the opposite number.

**$(x_1, x_2, \dots, x_k)$**

Compute expressions  $x_1, x_2, \dots, x_k$  respectively and return the computed result of  $x_k$ .

**$=(a=1, b=a+4, b*b)$**       Result is 25. Meanwhile, set the value of the cellset variable  $a$  as 1, and  $b$  as 5.

### 2.3.4 Judgment and deletion of variables

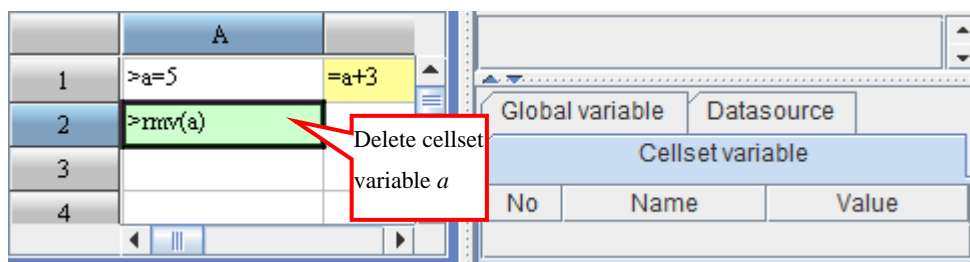
In esProc, when you are done with the variables, then you can use **rmv()** function to remove them.

**ifv( $v$ )**

Judge if  $v$  is a variable.

**rmv( $v_i, \dots$ )**

Delete cell variable  $v_i$ . This variable cannot be referenced in the cellset once it is deleted.



## 2.4 Execution Order of the cellset

The code in the program cellset is executed top-down and left-right.

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				

A computational cell can make reference to the computational cells before it, and the constant cells after it. The constant cell value is valid even before you execute the cell. By comparison, the computational cell must be executed to get valid.

Example:

	A	B	C
1	=A3	=2+1	=B3
2	=B1		
3	3	=3	

**A2** is computed after **B1**. Therefore, it can make reference to the cell **B1**.

**A3** is a constant cell. **B3** is a computational cell whose cell value is the computed result of the expression. Therefore, **A1** cell can make reference to **A3**, but **C1** cannot get the correct **value of cell B3**;

The computed results of **A1**, **A2**, **B1**, and **B3** are all 3, but **C1** is null.

## 2.5 Cell Copy and Paste

Unlike EXCEL, the copy and paste operations on the cell will not perform the adjust-paste for expression automatically.

	A	B	C
1			
2	=3*4		
3	=A2+5		
4			

	A	B	C
1			
2	=3*4		
3	=A2+5	=A2+5	
4			

However, if using Ctrl+Alt+V to paste, the expression will undergo the adjust-paste like what will happen in EXCEL.

	A	B	C
1			
2	=3*4		
3	=A2+5		
4			

1. Select cellset A3 and press Ctrl+C

	A	B
1		
2	=3*4	
3	=A2+5	=B2+5
4		

2. Select cellset B3 and press Ctrl+Alt+V, then A2 will turn into B2 automatically

Like the EXCEL, esProc set it as a rule that the row/column number started with \$ will never get adjust-paste.

	A	B
1		
2	=3*4	
3	=\$A\$2+5	=\$A\$2+5
4		

\$A\$2 is absolute reference, paste won't change the expression even using Ctrl+Alt+V

## 2.6 Insert and Delete like Text-editing

On the cell, press Ctrl+Enter, Ctrl+BackSpace, or Ctrl+Del will result in a similar effect of carriage return, backspace or deleting the current cell. This is similar to pressing Enter, Backspace, or Del keys in a text editor. Pressing Ctrl+Insert will insert a cell in this row.

- 1) Pressing Ctrl+Enter will cause a carriage return.

	A	B	C
1	1	2	3
2	4	5	6
3			
4	7	8	9

Select the cellset when it is in non-editing

Press Ctrl+Enter

	A	B	C
1			
2	4	5	6
3			
4	7	8	9

Carriage return

- 2) Pressing Ctrl+BackSpace will cause a forward delete

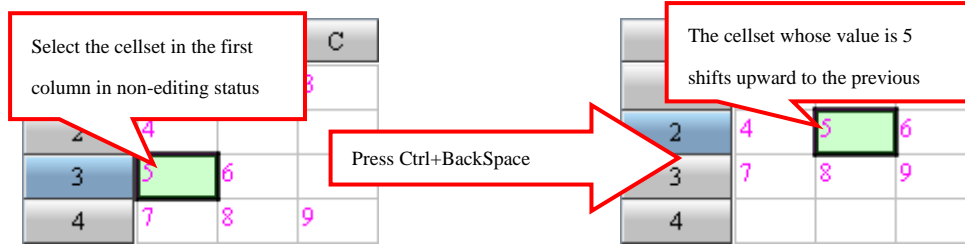
	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9
4			

Select the cellset when it is in non-editing status

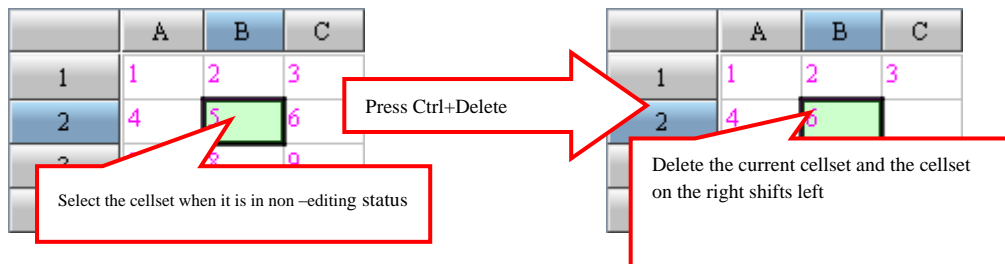
Press Ctrl+BackSpace

	A	B	C
1			
2		5	6
3	7	8	9
4			

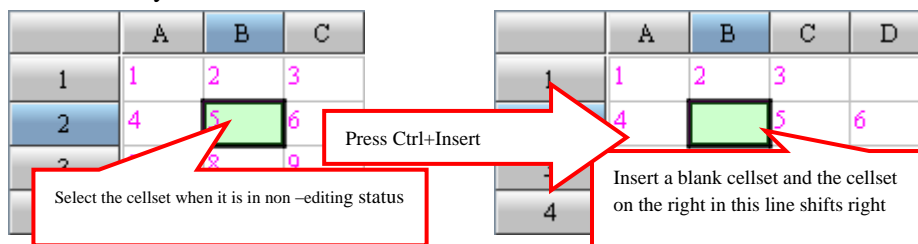
Delete the cellset whose value is 4, then the cellsets whose values are 5 and 6 respectively



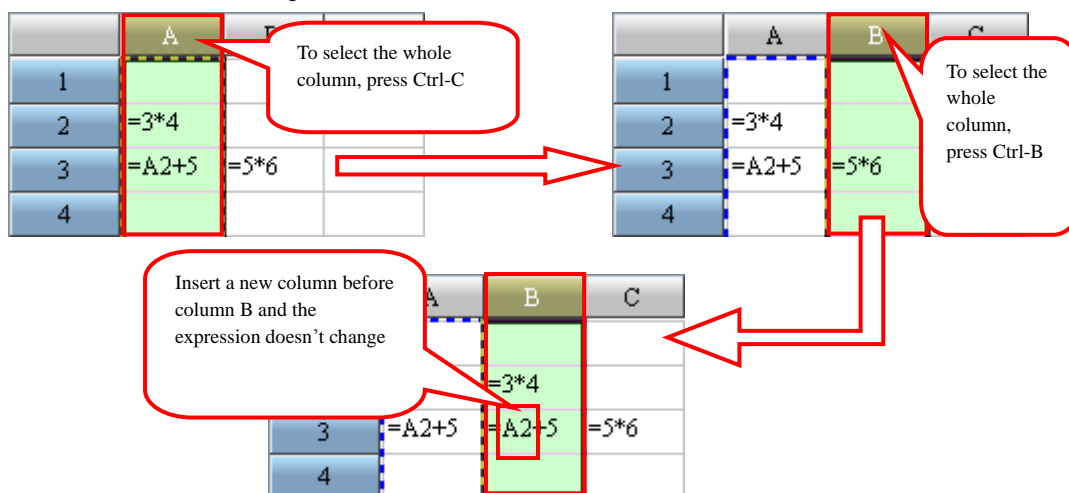
3) Pressing Ctrl+Delete will delete the current cell. The following cells will move leftward to fill the space.



4) Pressing Ctrl+Insert will insert a cell, and the following cells will move rightward to make room for the newly inserted.



5) When the copy area is a whole row/column, pressing Ctrl+B will automatically insert the row/column first and then paste the content.



If pressing Ctrl+Alt+B, the expression will be adjust-pasted in a way similar to pressing Ctrl+Alt+C.

## 2.7 Function and Function Options

### 2.7.1 Normal Functions

In an expression, various functions can be used. The basic format of function is  $f(\dots)$ , in



which "..." represents the parameters of the function.

#### **if(*a*,*b*,*c*)**

Compute the formula *a*. If true, compute *b* and return the result. Otherwise, compute *c* and return the result. If both *b* and *c* are omitted, then return true or false directly.

- **=if(*a*>0,*a*,*-a*)** Compute the absolute value of *a*.

#### **if(*x*<sub>1</sub>:*y*<sub>1</sub>,*x*<sub>2</sub>:*y*<sub>2</sub>,...,*x*<sub>*k*</sub>:*y*<sub>*k*</sub>:*y*)**

Compute the formula *x*<sub>1</sub>. If true, then compute *y*<sub>1</sub> and return the result, otherwise, proceed with *x*<sub>2</sub>, and compute *y*<sub>2</sub> on true and return the result; .....; if all *x*<sub>*i*</sub> is false, then compute *y* and return the result. It equals to **if(*x*<sub>1</sub>,*y*<sub>1</sub>,if(*x*<sub>2</sub>,*y*<sub>2</sub>,...,if(*x*<sub>*k*</sub>,*y*<sub>*k*</sub>,*y*)))**. When using it, you can simply write **if(*x*<sub>1</sub>,*y*<sub>1</sub>,*x*<sub>2</sub>,*y*<sub>2</sub>,...,*x*<sub>*k*</sub>,*y*<sub>*k*</sub>,*y*)**.

- **=if(*a*>1000:*a*\*0.8,*a*>500:*a*\*0.9;*a*)** Compute the purchase value, 10% discount for 500 yuan and above purchase, and 20% discount for 1000 and above purchase.

	A	
1	66.8	
2	=if(A1>=80,"Heavyweight","Others")	Value Others
3	=if(A1>=80,"Heavyweight",A1>=68,"Middleweight",A1>=58,"Lightweight","Flyweight")	Value Lightweight

#### **case(*x*,*x*<sub>1</sub>:*y*<sub>1</sub>,*x*<sub>2</sub>:*y*<sub>2</sub>,...,*x*<sub>*k*</sub>:*y*<sub>*k*</sub>:*y*)**

Compute the expression *x*. If the value is *x*<sub>1</sub>, then compute *y*<sub>1</sub> and return the result; if the value is *x*<sub>2</sub>, then compute *y*<sub>2</sub> and return the result;.....; if not equal to all the *x*<sub>*i*</sub>, then compute *y* and return the result. It equals to **if(*x*==*x*<sub>1</sub>:*y*<sub>1</sub>, *x*==*x*<sub>2</sub>:*y*<sub>2</sub>,..., *x*==*x*<sub>*k*</sub>:*y*<sub>*k*</sub>,*y*)**. When using, it can be simply written to **case(*x*,*x*<sub>1</sub>,*y*<sub>1</sub>,*x*<sub>2</sub>,*y*<sub>2</sub>,...,*x*<sub>*k*</sub>,*y*<sub>*k*</sub>,*y*)**.

- **=case(*a*%3,1,"one",2,"two",3,"zero")** Output the remainder of *a* divided by 3, for example, the result is "two" if *a* is 995.

For all function parameters, the three separators from the greatest to the smallest priority are colon :, comma ,, and semicolon ; in processing.

In addition, esProc also provides lots of object-oriented functions in the format of *o.f*(...), in which the *o* is the object that functions act on. In esProc, *o* can be the sequence, TSeq, RSeq, and records. We will further discuss it later.

### **2.7.2 Function options**

In esProc, many functions can use the function options. Because one function may be used for many purposes, we need options to specify each of its uses. The basic format of function options is ***f*@*o***(...), and *o* is the option of function *f*()

- **=interval@m("2011-7-1", "2011-8-1")**

In this expression, **@m** is the option of **interval()** function. With this option, the month will be taken as the unit to compute the timespan between these two dates.



	A	B
1	=interval("2011-7-1","2011-8-1")	=interval@m("2011-7-1","2011-8-1")
2	By default <i>interval</i> function takes day as the unit to compute	Using @m option, <i>interval</i> function takes month as the unit to compute
	Value 31	Value 1

Multiple options which are allowed together can be used simultaneously. There is no restriction regarding the order of these options.

### **in(*x,a:b*)**

Check if *x* is in the range [*a,b*]. In other words, check if  $a \leq x \leq b$ . Return true or false based on the result. If *a* is omitted, then treat it as infinitely small; if *b* is omitted, then treat it as infinitely great.

- ❖ **@b** Return integer based on the result: on the number axis, return 0 if *x* is in the range, return -1 if it is on the left of the range, and return 1 if on the right of the range.
- ❖ **@l** The left side is the open interval and the evaluation expression is  $a < x \leq b$ .
- ❖ **@r** The right side is the open interval and the evaluation expression is  $a \leq x < b$ .

For the specific usage of various functions, please click **Help>Function Reference** to read the related documents.

## **2.8 Read and Write data with File Object**

### **2.8.1 File Objects in esProc**

In esProc, you can use the file path to define the file object, and perform read/write actions on the file through the operation on the file object.

#### **file(*fn: cs*)**

Open a file object with the file name of *fn*. *fn* is the file name to be loaded with an absolute file path. The relative path is relative to the current path (if no current path, then it is relative to the current system path). *cs* is a character set (omissible, and use OS defaults instead).

- ❖ **@s** Specify the order in which the search will be carried out for the file name through non-absolute path. If the current path exists, follow the order of searching the current path first, and then the path list (the **addressing path** set on the Options menu of esProc). Otherwise, search the class path and then the path list. The result returned is a read-only file with name *fn*.

### **2.8.2 Use file object to store/retrieve string.**

With the file object, you can retrieve or modify a file, like storing the string of esProc as file, and retrieve the content in the file in the form of string.

#### **f.exists()**

Verify if the file exists.

#### **f.size()**



Number of bytes of the file.

### ***f.date()***

Last modification time of the file.

### ***movefile(fn, path)***

Move the file *fn* to the specified path *path*. When *path* is omitted, the file will be deleted; when *path* has only the filename but hasn't the path, file *fn* will be renamed.

- ❖ **@c** Copy file and leave the original one unmoved.
- ❖ **@y** Overwrite any file with the same name on the destination path. If no **@y** is in the expression, then cancel the move action.

### ***f.write(s)***

Output the string *s* to the file object *f*. The *s* can also refer to the Blob data.

- ❖ **@a** Append.
- ❖ **@b** Write into a binary file. The efficiency is much higher and it is not allowed to be edited as text.

	A
1	=file("D:/files/txt/test.txt")
2	test
3	>A1.write(A2)

The file is as follows after the string is written into the ...

test

With **@a** option, you can append the data to the file.

	A
1	=file("D:/files/txt/test.txt")
2	test2
3	>A1.write@a(A2)

With @a option, the string is appended to the file:

test  
test2

### ***f.read()***

Retrieve the file object *f* as string.

- ❖ **@b** Retrieve it as binary data.
- ❖ **@n** Return the contents of the file object *f* as a string sequence, each member of which corresponds a row.
- ❖ **@v** Interpret the returned string sequence as the corresponding data type. The combined use of this option and **@n** is acceptable

	A
1	=file("D:/files/txt/test.txt")
2	=A1.read()

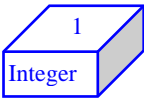
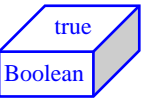
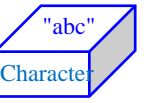
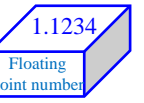
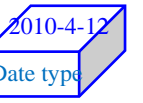
→

Value
test test2

## 3. Sequence

### 3.1 Concept of Sequence

**Sequence** is the ordered set consisting of some data, and the constitutive data of a sequence is **Member** by name. Sequence is equivalent to the array in the high level language. The difference is that the members of a sequence do not have to be of the same type.

No.	1	2	3	4	5	...
Member						...

Members of a sequence are **ordered**. You can locate a member according to its sequence number. Please note that **the sequence number of a sequence starts from 1**.

## 3.2 Generate a Sequence

### 3.2.1 Constant sequence

Members enclosed in a [] form a constant sequence.

- [1,3,4]
- [a,b,c]
- [s,2011-3-14,54]

### 3.2.2 Use an Expression to Define a Sequence

Use an expression to define a sequence directly with the similar format is similar to the constant sequence. You can reference the cell in the expression and use colon to represent a certain area of cells. When getting the values of a certain area of cells, esProc will follow the rule of row first.

- =[1,A4,B1:B4,C1:C3]

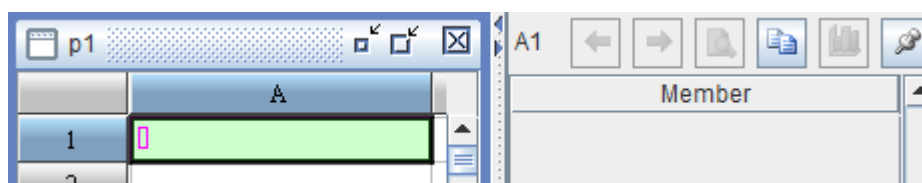
	A	B	C
1	1	2	3
2	4	5	6
3	7	8	9
4	[a,b,c]	[s,2011-3-14,54]	
5	=[1,A4,B1:B4,C1:C3]		

Member
1
[a,b,c]
2
5
8
[s,2011-03-14,54]
3
6
9

### 3.2.3 Empty Sequence

A sequence without member is **Empty Sequence** by name, and you can use [] to define it directly.



## 3.3 Sequence and Members

### 3.3.1 Sequence and its Members

**ifa(x)**

Verify if  $x$  is a sequence.

**A.len()**

Number of members of sequence  $A$ , that is, the **Sequence Length** of  $A$ .

### Empty Sequence and $n$ -Sequence

An empty sequence is the sequence whose length is **0**. An  **$n$ -sequence** is the sequence whose length is  $n$ .

### Unique Member Sequence and Pure Sequence

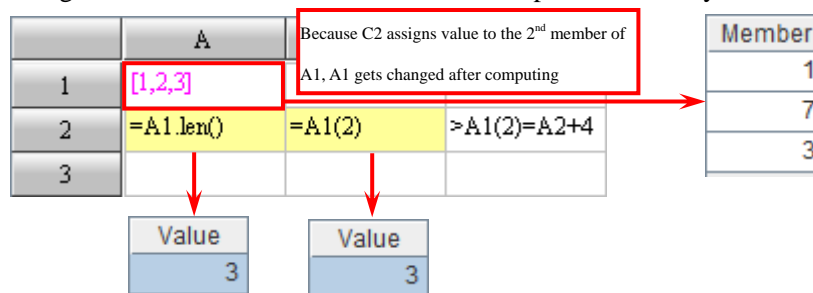
A sequence without duplicate member is called Unique Member Sequence; The **Pure Sequence** is a sequence whose members are of the same data type.

**A(i)**

Get the  $i^{\text{th}}$  member of sequence  $A$ . Note: esProc uses  $A(i)$  instead of  $A[i]$  to get the sequence member.

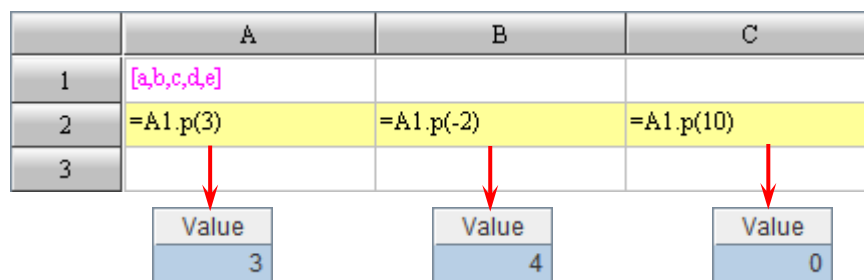
**A(i)=x**

Assign value to the  $i^{\text{th}}$  member, error will be reported if  $i$  is beyond the boundary.



**A.p(i)**

If  $i > 0$ , then return the sequence number of the  $i^{\text{th}}$  member in the sequence  $A$ ; if  $i < 0$ , then the sequence number of the  $i^{\text{th}}$  member from the tail will be returned. If  $i$  is beyond the boundary, then return **0** as the sequence number.



**A.m(i)**

If  $i > 0$ , return the  $i^{\text{th}}$  member of  $A$ ; if  $i < 0$ , return the  $i^{\text{th}}$  member by counting backwards; if  $i$  is beyond the boundary, return null.



	A	B	C
1	[a,b,c,d,e]		
2	=A1.m(3)	=A1.m(-2)	=A1.m(10)
3			

### 3.3.2 Modify Sequence

#### A.insert(k,x)

In the sequence A, insert the new member before the  $k^{\text{th}}$  member. If  $x$  is a sequence, then insert all members of sequence  $x$  as the new members; if  $x$  is not a sequence, insert a member  $x$ .

#### A.delete(k)

$k$  is an integer indicating the  $k^{\text{th}}$  member of the sequence A will be deleted.

#### A.delete(p)

In the sequence A, remove all members from the sequence number sequence  $p$ .

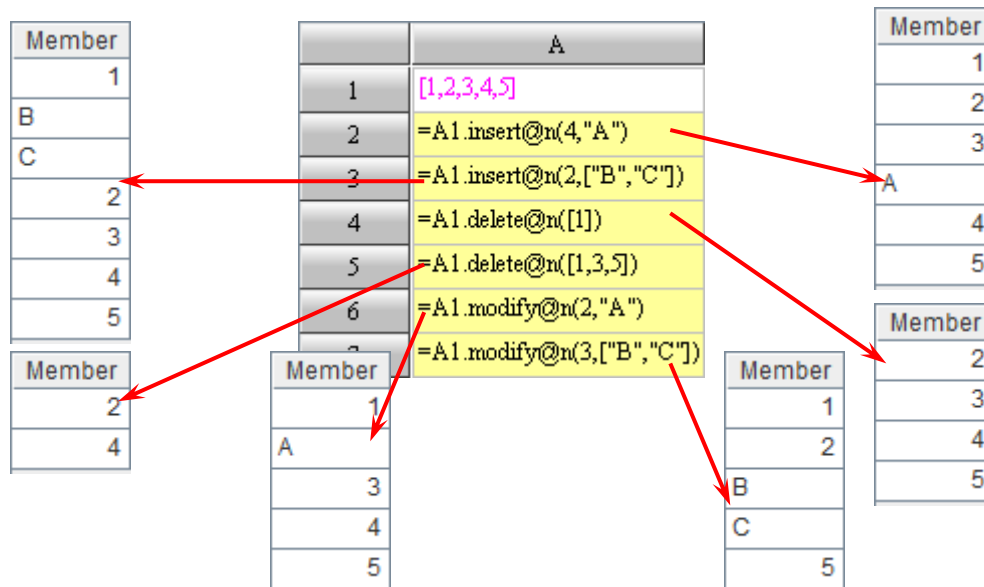
#### A.modify(k,x)

From the  $k^{\text{th}}$  member of the sequence A, modify the values of members of sequence A. If  $x$  is a sequence, then modify their values to each member of sequence  $x$  in turn; If  $x$  is not a sequence, then only modify the  $k^{\text{th}}$  member to  $x$ .

	A	B	C	D
1	[1,2,3,4,5]	>A1.insert(4,"A")	>A1.insert(2,["B","C"])	=A1
2	[1,2,3,4,5]	>A2.delete([1])		=A2
3	[1,2,3,4,5]	>A3.delete([1,3,5])		=A3
4	[1,2,3,4,5]	>A4.modify(2,"A")	>A4.modify(3,["B","C"])	=A4

In every function for modifying sequences, you can use the @n option:

- ❖ @n When it is executed, a new sequence will be returned and sequence A is kept unchanged.



### 3.4 Integer sequence

The sequence made up of integers is **Integer Sequence** or **ISeq** for short.

#### 3.4.1 Concept Relating to Integer Sequence

##### *n* Permutation

If the integer sequence is composed of all the  $n$  integers of  $1,2,...,n$  (no matter the order), it is the  $n$  permutation by name.

- The integer sequences of  $[1,4,3,2,5]$ ,  $[1,2,3,4,5]$  and  $[5,4,3,2,1]$  are all 5 permutation.

#### 3.4.2 to Function

##### **to(a,b)**

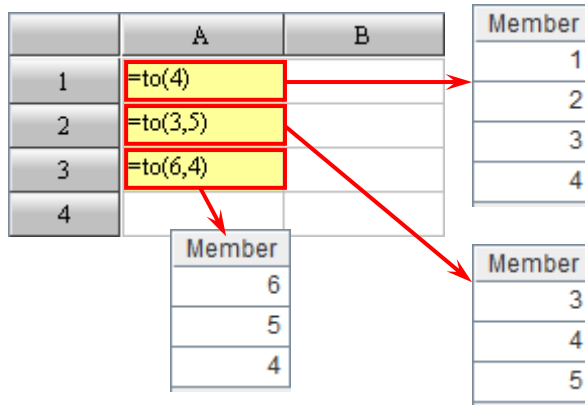
Generate an Integer Sequence made up of all the integers from  $a$  to  $b$  ( $a,b$  inclusive). Both  $a < b$  and  $a > b$  are acceptable.

- ❖ **@s**  $b$  will indicate the number of integers in the result. The ISeq (integer sequence) will increase by 1 when  $b > 0$ , and decrease by 1 when  $b < 0$ .

- `=to@s(10,3)`, the result is ISeq  $[10,11,12]$
- `=to@s(10,-3)`, the result is ISeq  $[10,9,8]$

##### **to(n)**

The abbreviated version of **to(1,n)**.

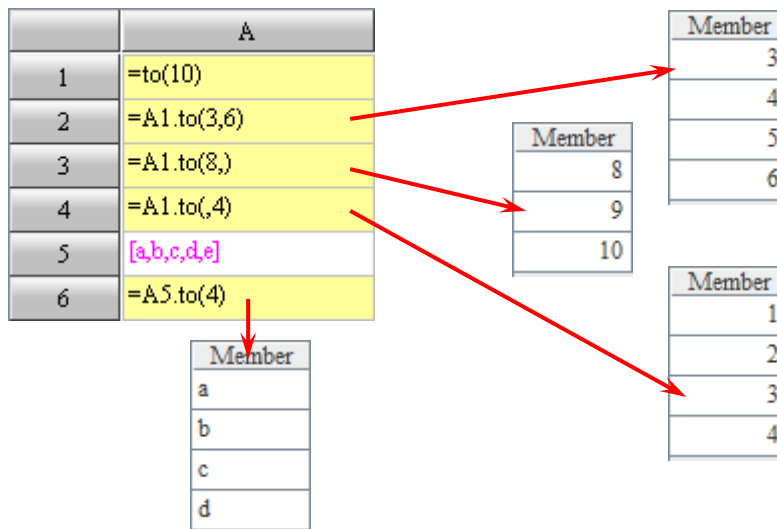


### **A.to(a,b)**

It represents the sequence composed of members from the  $a^{\text{th}}$  to the  $b^{\text{th}}$  in the sequence A. It is the short form of **A(to(a, b))**. If  $a$  is omitted, then the default value is 1; If  $b$  is omitted, then the default value is **A.len()**, note that the comma cannot be omitted here.

### **A.to(a)**

A sequence made up of the members from 1 to  $a$  in Sequence A. It is equivalent to **A.to(1, a)**.



## **3.5 Sub-sequence**

A sequence consisting of the members of A is called the **Sub-sequence** of A.

### **3.5.1 Generate sub-sequence using functions**

#### **A(p)**

[**A(p(1))**,**A(p(2))**,...], in which the position ISeq  $p$  is used to get the sub-sequence of A.

#### **A.p(p)**

[**A.p(p(1))**,**A.p(p(2))**,...] to get the corresponding actual positions of the position ISeq  $p$  in the A. Negative number is allowed in the  $p$ .

#### **A.m(p)**

**A(A.p(p))**, the position ISeq  $p$  is used to get the sub-sequence of A. The negative number is

allowed to exist in  $p$ .

**A.dup()**

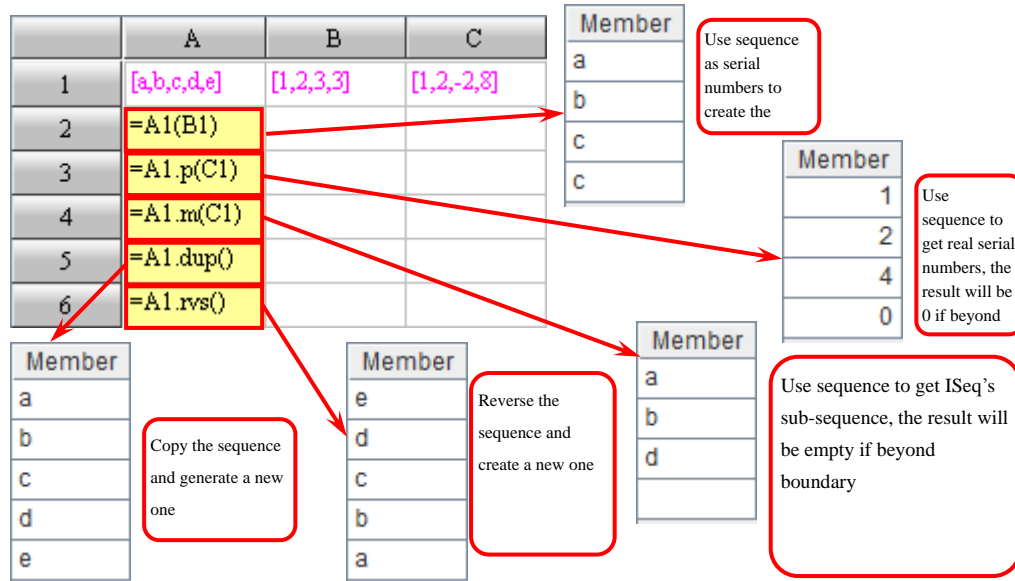
$A(\text{to}(A.\text{len}()))$ , which is equal to copy the A.

**A.rvs()**

$A(\text{to}(A.\text{len}(), 1))$ , that is, reverse the A.

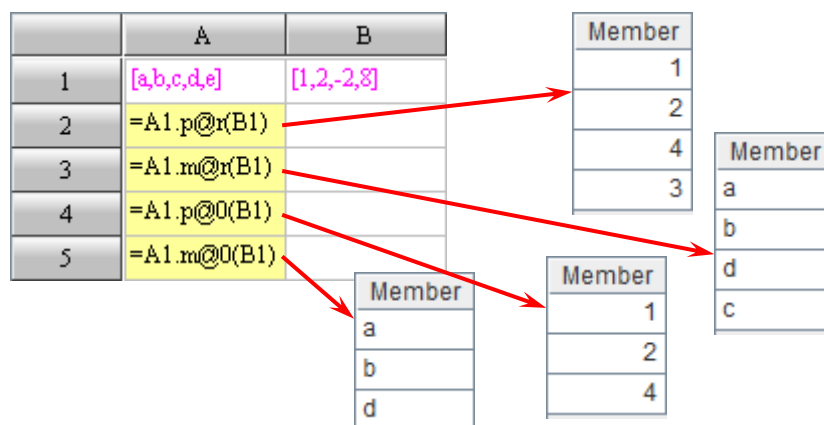
**A.step(m,k)**

$A([k, k+m, k+2*m, \dots])$ , that is, return the sub-sequence of A. The position ISeq starts from  $k$  and the step value is  $m$ .



### 3.5.2 Options in A.p(p) and A.m(p)

In **A.p(p)** and **A.m(p)**, you can use the option **@r** and **@0**. If using the option **@r**, on the sequence number out of range in the ISeq, perform the turn-around operation and retrieve them by loop. When using the **@0** option, the sequence number indicating the out-of-range will be ignored, **0** or the null value will not appear in the resulting sequence.



### 3.5.3 Related Functions of Sub-Sequence

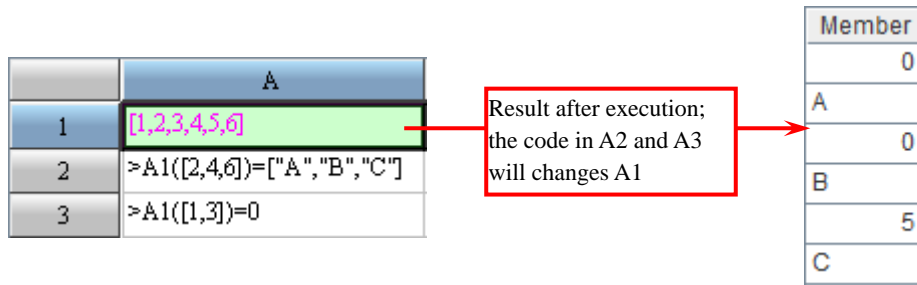
**A(p)=X**

Assign the members of sequence  $X$  as values to the sub-sequence  $A(p)$  of  $A$  in order, that is,  $A(p(1))=X(1)$ ,  $A(p(2))=X(2)$ , .... The length of sequence  $X$  is greater or equal to the length of

sequence  $p$ .

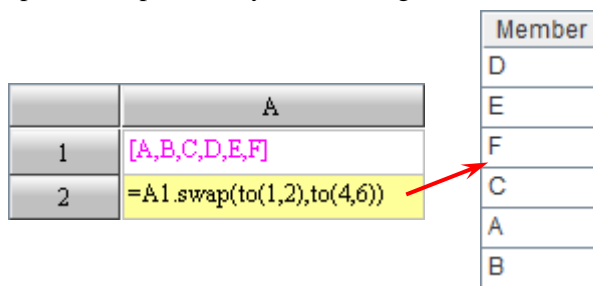
$A(p)=x$

Assign a single value  $x$  to every member of  $A(p)$  which is a sub-sequence of  $A$ . That can be represented like this:  $A(p(1))=x, A(p(2))=x, \dots$



**A.swap(p,q)**

The  $p$  and  $q$  are both the consecutive ISeq range with no intersection. Swap the members of  $A$  at the positions specified by the two ranges and return the new sequence.



The **swap()** function will not change the original sequence.

### 3.6 Sequence and String

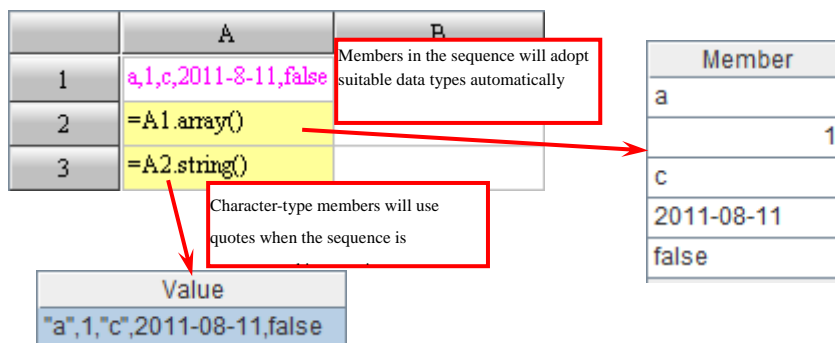
With **s.array()** and **A.string()** functions, sequence and string can be transformed to each other conveniently.

**6) s.array(d)**

Split the string  $s$  into sequences with the delimiter  $d$ ; parse the data type automatically. The  $d$  is comma by default.

**7) A.string(d)**

Concatenate the sequence  $A$  into a string with the delimiter  $d$  and process the data type automatically; the  $d$  is comma by default.



**8) s.regex(rs)**





Match the regular expression *rs* with string *s*, and return the sequence of matching result; if the two don't match up with each other, null will be returned.

- ❖ @i Not case-sensitive.
- ❖ @u Match by using unicode.

The simplest usage of *regex* function is to judge if a certain string matches with a specified regular expression. Let's first look at the situation where a digit string is involved:

	A	B	C
1	'4	'12	'546201.05
2	=A1.regex("[0-9]")	=A1.regex("\\d")	=A1.regex("[0-3]")
3	=B1.regex("[0-9][0-9]")	=B1.regex("[0-9]+")	=B1.regex("[0-9]*")
4	=C1.regex("[0-9]*")	=C1.regex("\\d*")	=C1.regex("\\d*\\.\\d*")

As *s.regex(rs)* is used to judge string *s*, the single quotes ' are added before every digit in the first row to represent a string constant. In a regular expression, some signs can be used to match with digits. For instance, the [0-9] in A2 represents a digit and can match with the string in A1. To enclose the sign with a parenthesis can return the matching result:

Member
4

It can be seen that *regex* function returns a sequence of matching result. In B2, *\d* is also the representation of a digit, but because the right slash \ represents escape character in strings, *\d* is written as *\\d*. The matching result is the same as that in A2. In C2, [0-3] represents a digit between 0 and 3. This doesn't match with "4", so the result is null.

In a regular expression, + or \* is also used to represent multiple characters. For example, [0-9][0-9] in A3 represents two arbitrary digits, [0-9]+ in B3 represents one or more arbitrary digits and [0-9]\* in C3 represents zero or more arbitrary digits. As A3, B3 and C3 all match with the string in B1, the returned result is:

Member
12

Both regular expressions in A4 and B4 represent zero or more arbitrary digits; the string in C1 contains decimal point, so the matching result only uses the digit before the decimal point. The result in A4 and B4 is as follows:

Member
546201

In the regular expression in C4, *\\d\*\\.\\d\** represents a mode of multiple digits followed by a decimal point which is again followed by multiple digits. The matching result is as follows:

Member
546201.05

By using *regex* function, common strings can also be matched with a specified regular expression. For example:



	A	B	C
1	x	Max	Adam Stuart
2	=A1.regex("[a-z]")	=A1.regex("[^0-9]")	=A1.regex@u("(\u0078)")
3	=B1.regex("[a-z]{2}")	=B1.regex("[a-z]*")	=B1.regex@i("[a-z]{2}")
4	=C1.regex("[ASadmrtu]*")	=C1.regex("\S* \S*")	=C1.regex("[A/S][a-z]* [A/S][a-z]*")

In A2, [a-z] represents a matching character between a and z; [^0-9] in B2 represents a matching character that falls out of the range from 0 to 9, i.e. a non-digit character; in C2, the value of unicode \u0078, that is, the x, is used to make a match. The matching result of A2, B2 and C2 is as follows:

Member
x

In A3, {2} represents finding two matching characters that exist between a and z. Because of the capital letter M which makes the string fall out of the range, the matching result is null.

In B3, \* is used to represent zero or more characters between a and z. In this case, zero matching character is allowed. So even the first character M doesn't meet the criterion, the matching result is empty instead of null:

Member

In C3, option @i is added, requiring two matching characters regardless of the case. Matching result is as follows:

Member
Ma

In A4, find matching strings composed of characters ASadmrtu and blanks. In B4, \S represents one non-blank character, that is, a non-blank tab, line break, form feed, carriage return, and the like; the whole regular expression represents a mode of multiple non-blank characters followed by a blank which is again followed by multiple non-blank character. In C4, [A/S] represents A or S, and the whole regular expression represents the mode of words starting with A or S followed by a blank which is again followed by words starting with A or S. The matching result in A4, B4 and C4 are the same as follows:

Member
Adam Stuart

Through these illustrations, we have acquired some preliminary knowledge about using regular expressions to make matches. In fact, regex function in esProc is used more to split strings into sequences according to specified criterions. For example:

	A	B	C
1	Max	Adam Stuart	Adam,Bob,Charlotte,Doug
2	=A1.regex@i("[a-z])([a-z])([a-z]")		
3	=B1.regex("\S*")		
4	=C1.regex@i("[a-z]*"),([a-z]*"),([a-z]*"),([a-z]*")		

A2 splits the string in A1 into a sequence consisting of letters. The splitting is not case-sensitive. Result is as follows:



Member
M
a
x

A3 splits the string in B1 into two parts, consisting of consecutive non-blank characters and separated by a blank. The result is as follows:

Member
Adam
Stuart

A4 splits the string in C4 into four words with commas as the separator. The splitting is not case-sensitive. The result is as follows:

Member
Adam
Bob
Charlotte
Doug

It can be seen that, when strings are split into sequences using *regex* function, multiple expressions enclosed in parenthesis are needed to match in order with members of the resulting sequences. The expressions in A3 and A4 that can split a string into a sequence containing many words are frequently used. Please note that if there is in the string a member that cannot match with the regular expression, the result returned by the function will be null.

## 3.7 Basic Operations on Sequence

### 3.7.1 Binary Operation

#### $A|B$

Concatenate: Concatenate the two sequences straightforwardly. Members of  $B$  are appended behind members of  $A$ . If  $A$  or  $B$  or both  $A$  and  $B$  are single-value instead of sequences, then it will be handled as a single member sequence.

#### $A\&B$

Union: It will create a new sequence by putting members of  $B$  after members of  $A$  and removing members of  $B$  that already exist in  $A$ . If one of  $A$  and  $B$  or both are single-value instead of sequences, then it will be handled as a single member sequence.

#### $A^{\wedge}B$

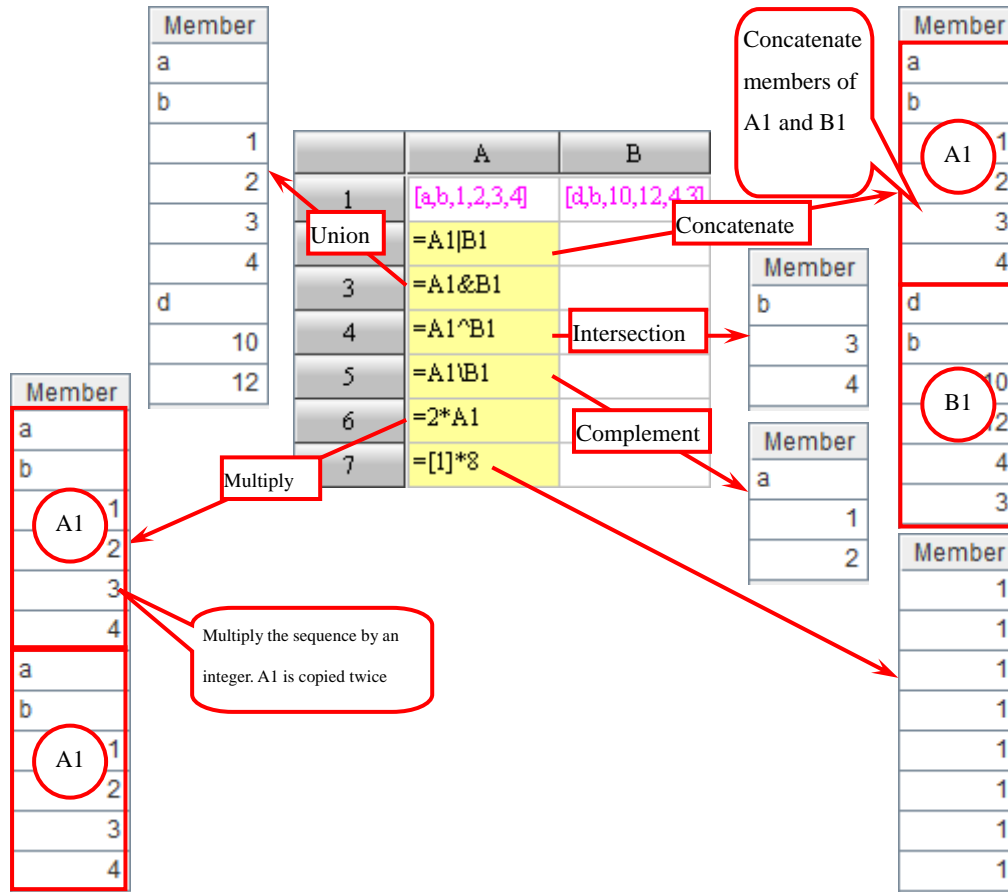
Intersection: Intersection set of  $A$  and  $B$ . Get a sequence composed of members that not only exist in  $A$  but also exist in  $B$ .

#### $A\backslash B$

Complement: Members existing in  $A$  but not in  $B$ . If  $B$  is not a sequence, then treat it as the single member sequence.

#### $k*A$

$A|A|\dots|A$ , concatenate  $k$  sequences of  $A$ , i.e. copy  $A$  for  $k$  times. The positions of  $k$  and  $A$  can be exchanged.



### 3.7.2 Alignment Operation on Sequences

Two sequences which have the same length and are composed of numerical values can perform the alignment computation on members, and return a new sequence.

**$A++B$**

$[A(1)+B(1), A(2)+B(2), \dots]$

**$A--B$**

$[A(1)-B(1), A(2)-B(2), \dots]$

**$A**B$**

$[A(1)*B(1), A(2)*B(2), \dots]$

**$A//B$**

$[A(1)/B(1), A(2)/B(2), \dots]$

**$A\%\%B$**

$[A(1)\%B(1), A(2)\%B(2), \dots]$ , the % here is the Mod computation.



Member		A	B	Member
5	1	[1,2,3]	[4,1,2]	-3
3	2	=A1++B1		1
5	3	=A1--B1		1
	4	=A1/B1		1
0.25	5	=A1%%B1		0
2.0				1
1.5				

### 3.7.3 Sequence Comparison

The function **cmp(x,y)** can be used to compare the result of expression  $x$  and  $y$ . The function **cmp(A,B)** can be used to compare the sequence  $A$  and  $B$ .

#### cmp(x,y)

Compare the computed result of expression  $x$  with that of  $y$  and return 1 if  $x>y$ , return -1 if  $x<y$ , or return 0 if  $x=y$ . Error will be reported if  $x$  and  $y$  are incomparable.

	A	B	Value
1	=cmp("Alex", "Allen")	"Alex"<"Allen"	-1
2	=cmp(0.4*3, 1)	1.2>1	1
3	=cmp(2+4, 6)	6==6	0
4	/=cmp("abc", 15)	/=cmp(15, "abc")	0

Error will be reported if compare a string with an integer

In esProc, the following expressions are used to represent the comparing relationship between  $a$  and  $b$ :

$a>b$	$a$ is greater than $b$ , i.e. $\text{cmp}(a,b)>0$
$a<b$	$a$ is less than $b$ , i.e. $\text{cmp}(a,b)<0$
$a==b$	$a$ and $b$ are equal, i.e. $\text{cmp}(a,b)=0$ . Two equal signs are used to differentiate from the execution of value assigning
$a!=b$	$a$ and $b$ are not equal, i.e. $\text{cmp}(a,b)\neq 0$
$a>=b$	$a$ is greater than or equal to $b$ , i.e. $\text{cmp}(a,b)\geq 0$
$a<=b$	$a$ is less than or equal to $b$ , i.e. $\text{cmp}(a,b)\leq 0$

In the sequence  $A$ , if each member is not less than its previous member, i.e.  $A(i)\geq A(i-1)$ , or each member is not greater than its previous one, i.e.  $A(i)\leq A(i-1)$ , then the sequence  $A$  can be regarded as the **Ordered Sequence** in both cases. In which, the former sequence is referred to as an **Increasing Sequence**.

#### cmp(A,B)

Compare the member values of two sequences in alignment, and return 1 or -1 when encountering the first unequal member; if  $A$  is identical to  $B$ , then return 0. Specifically, **cmp(A)** or **cmp(A,0)** will compare  $A$  with the sequence of the same length whose members are all 0, i.e. **cmp(A,[0,0,...,0])**.

	A		Value
1	=cmp(["a","b","c"],["a","b","c"])	Equal	0
2	=cmp([1,3,5,7],[1,3,7,5])	5<7	-1
3	=cmp([7,6,5,4],[7,6,4,10,11])	5>4	1

In order to differentiate from comparing normal values, the expression to compare two sequences cannot be written in a shortened form.

### 3.7.4 Aggregate Operation

In esProc, functions can be used to perform various aggregate operations on sequence A that contains  $n$  members.

#### A.count(x)

Compute the number of A's members that make the result of expression  $x$  non-null and is not false. When  $x$  is omitted, count up based on the members of A. This is different to A.len().

#### A.ifn()

Get the first non-null member of A.

#### A.sum(x)

Compute all members in A with expression  $x$ , and seek the summation of the results. If  $x$  is omitted, seek the summation of members of A.

#### A.avg(x)

Compute all members in A with expression  $x$ , and seek the average value of the results. If  $x$  is omitted, seek the average value of members of A, excluding null members.

	A	B	C	D	E
1	[null,4,6,,2,4,,5]				
2	=A1.count()	=A1.count@b(~>3)	=A1.ifn()	=A1.sum()	=A1.avg()

Value	Value	Value	Value	Value
5	4	4	21	4.2
Number of non-null	Number of members that are greater than 3	The first non-null	Summate	Average. Null values are excluded in

#### A.min(x)

Compute all members of A with expression  $x$  seek the minimum value of the results. If  $x$  is omitted, seek the minimum value of members of A.

#### A.max(x)

Compute all members of A with expression  $x$  seek the maximum value of the results. If  $x$  is omitted, seek the maximum value of members of A.

#### A.variance()

Seek Variance. The null member will not be counted when computing variance.

#### A.rank(y)

Seek the ranking of  $y$ ' value in the sequence.  $y$  is not required to be the value of members in

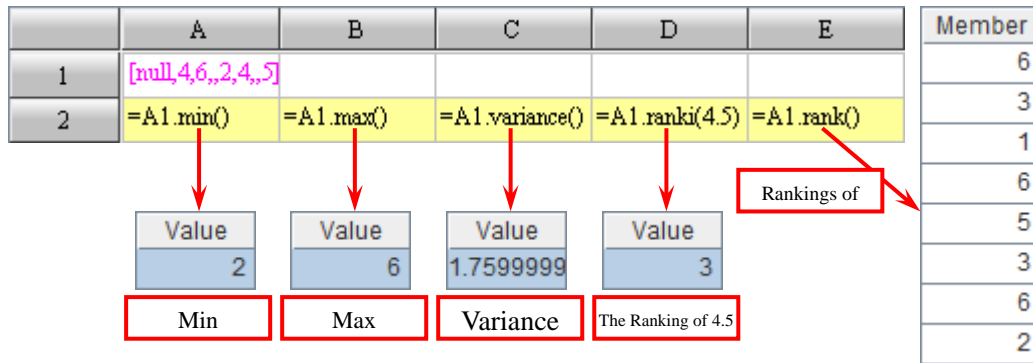
the sequence.

- ❖ @z Sort in ascending order.

### A.rank()

Seek each member's ranking in the sequence.

- ❖ @z Sort in ascending order.



### A.conj()

Concatenate all members in the sequence A. If any member in A is not a sequence, then process it as 1-sequence.

### A.union()

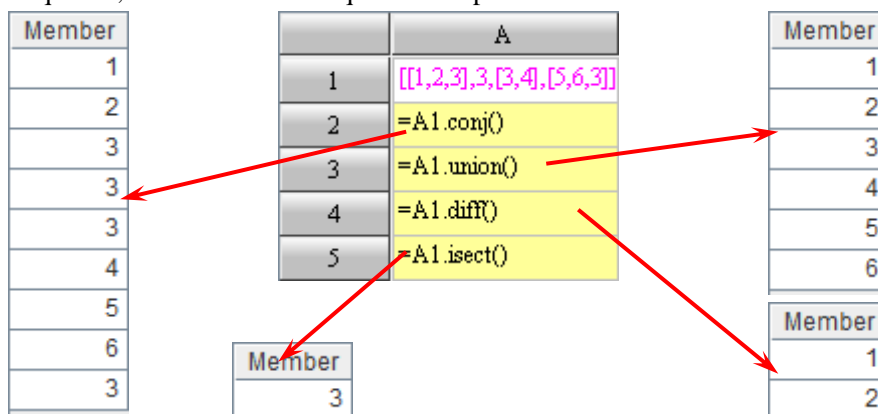
Join all members in the sequence A. If any member in the A is not a sequence, then take it as 1-sequence and process (similar to the role of **UNION** operator in SQL, and no duplicate will appear in the resulting sequence)

### A.diff()

Perform the complementation operation on members in sequence A. If any member in A is not a sequence, then process it as 1-sequence.

### A.isect()

Compute the intersection sequence for all members in the sequence A. If any member of A is not a sequence, then take it as 1-sequence and process.

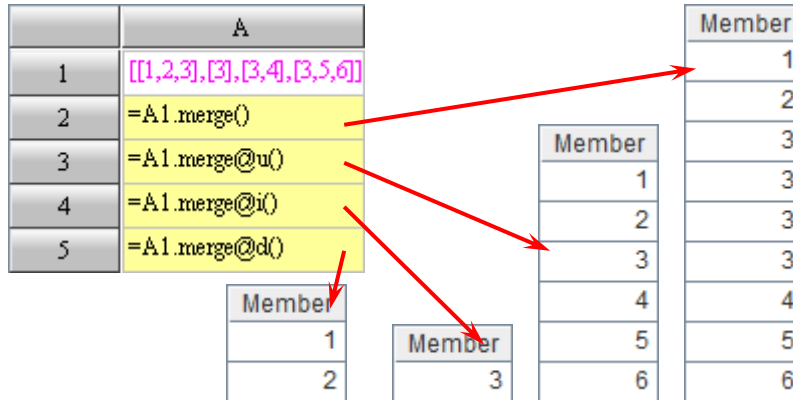


### A.merge(x<sub>i</sub>,...)

Merge operation. A is a sequence of sub-sequences, and every sequence A(i) is ordered by the [x<sub>i</sub>,...]. Merge and combine all members of A, and the resulting sequence after the combination shall be still in order for [x<sub>i</sub>,...]. If x<sub>i</sub> is omitted, members of A(i) shall be ordered, and we get a

sequence with ordered members after the combination.

- ❖ @u Seek the union sequence when merging
- ❖ @i Seek the intersection sequence when merging
- ❖ @d Seek the subtraction sequence when merging



Note: the function **merge** requires all members of A are in order.

## 3.8 Cases of Sequence

### 3.8.1 Name List Statistics

Take the winners of 100 m dash, 200 m dash and long jump events (top 8) in a sports game as arguments (name list is fields of names separated with comma) to compute:

Athletes winning 2 or more medals.

Determine whether the winners of 100m dash get the same good results in the 200m dash.

**Program parameter** ✕

☐ Set arguments before run

No.	Name	Value	Remark
1	100Meters	Bray,Jacob,Michael,John,David,Jam...	
2	200Meters	Joshua,Michael,Ethan,Bray,Joseph,J...	
3	longjump	Michael,Alexander,Bray,Zachary,Bran...	

OK

Cancel

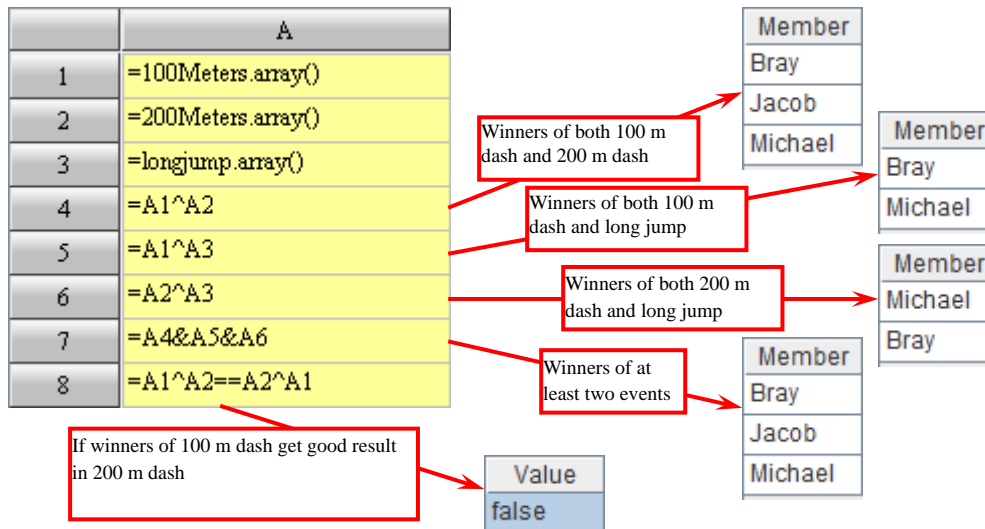
Add

Delete

Up

Down





### 3.8.2 Judge Scoring (1)

The scoring in a gymnastics competition. Compute the average scores after removing the highest and the lowest scores.

	A	B	C	D	E	F	G	H	I
1	9	9.1	8.5	9.8	9.4	9.5	8.8	9.3	9.0
2	=[A1:I1]	Sequence of scores							
3	=round((A2.sum()-A2.min()-A2.max())/(A2.len()-2),2)	Value							
4	=round((A2\A2.min()\A2.max()).avg(),2)	9.16							

### 3.8.3 Judge Scoring (2)

For different levels of judges, weighted index varies. Compute the weighted average.

	A	B	C	D	E	F	G	H	I
1	9	9.1	8.5	9.8	9.4	9.5	8.8	9.3	9.0
2	0.9	0.8	1.0	0.95	1	0.85	0.9	0.95	0.90
3	=[A1:I1]	Sequence of scores							
4	=[A2:I2]	Sequence of weighted index							
5	=round((A3**A4).avg(),2)	Value							
		8.39							

## 4. Code Block and Program Flow Control

### 4.1 Code Block

In the program cellset, a certain area of indented cells is called as **Code Block**. The starting cell is the **Master Cell** of the code block.

The green area, in which cells are all blank, is called as Code Block

	A	B	C	D	E
1					
2		>Code Begin			
3					
4					
5	Not Null				

As shown in the above, if the green area is blank, the 2<sup>nd</sup> to the 4<sup>th</sup> row is the code block of B2, the master cell; if any cell of A5 and B5 in the red area contains code, constant and even note, or is used for computing and executing the code, the 5<sup>th</sup> row is no longer the code block of B2.

The master cell of code block is usually the **Statement Cell**, that is, the cell string is the statement code starting with a reserved word.

Unlike JAVA and other high level languages, esProc uses the straightforward format of code block to specify the working range of statements, instead of symbols like {} or reserved words (BEGIN/END) to enclose the working range.

## 4.2 if/else Statement

The branch statement **if/else** has the following formats:

### if x ... else ... in the same line

If condition *x* is true, then execute the statement after it. Otherwise execute the statement after **else**. The **else** part can be omitted. **else** and **if** must be in the same row. After execution, the value of the cell in which **if** is located is the computed result of *x*.

	A	B	Value	D
1	12	21	false	
2	if A1>B1	>A3=A1 else >A3=B1		
3			Output the bigger number between A1 and B1	

Value: 21

### if x ... else ... in the code block

If *x* is true, then execute the code block of **if**; otherwise, execute the code block of **else**. You can certainly ignore the **else** part. **else** and **if** must be in the same column.

	A	B
1	UnitPrice	Quantity
2	67.00	15
3	39.80	8
4	if B2+B3>20	=(A2*B2+A3*B3)*(1-0.15)
5		>B8=round(B4,2)
6	else	=(A2*B2+A3*B3)*(1-0.1)
7		>B8=round(B6,2)
8	Total:	

Value: true

Master cell of the code block of if; if value is true, ...

Master cell of the code block of ...

Output the total amount of discounted goods purchased

Value: 1124.89

The code block of if

The code block of

### Multiple blocks of if x ... else if y ...

The statement can be written continuously like this. **else if** must be written in the same cell. Stress again that there is no corresponding **end if** statement. esProc uses the range of the code

block to determine where the **if** statement ends.

	A	B
1	66.8	
2	if A1>=80	Heavyweight level when weight is equal to and greater than 80kg
3		>B10="Heavyweight"
4	else if A1>=68	Middleweight level when weight is between 68kg and 80kg
5		>B10="Middleweight"
6	else if A1>=58	Lightweight level when weight is between 58kg and 68kg
7		>B10="Lightweight"
8	else	Flyweight level when weight is less than 58kg
9		>B10="Flyweight"
10	Weight Class	It is lightweight level according to the weight in A1

Value
Lightweight

The following logical connectors can be used in the judgment statements:

**a&&b**

For the “*a* and *b*”, the result is true only on condition that both *a* and *b* are true.

**a||b**

For the “*a* or *b*”, the result is true only on condition that either *a* or *b* is true.

**!a**

For the “not *a*”, the result is true only on condition that *a* is **false**.

### 4.3 for Loop and next/break Statement

Loop statement **for** will repeatedly execute the code cells with itself as the master cell in the following formats:

**for**

Endless loop, and the cell value is the current loop counts one by one.

**for n**

Cycle for *n* times, and the cell value is the current loop counts one by one.

**for A**

Cycle members of sequence *A*, and the cell value is the current members of *A* respectively.

**for x**

Cycle when *x* is true, and the cell value is the computed value of *x*.

In the loop body:

**#C**

The current cyclic counter, and *C* is the master cell of the loop body, that is, the cell where **for** is located.

**next**

Skip the remaining code of the innermost loop body, and enter directly to the next loop.

### next C

Enter the next cycle of which the loop body takes *C* as the master cell.

### break

Jump out of the current innermost loop body, and execute the codes in the cells following the code block.

### break C

Jump out the loop body, which takes *C* as its master cell.

- Sum up 1 to 100:

	A	B	C	Value
1	0			5050
2	for			
3		>A1=A1+#A2		
4		if #A2==100 break		

Final result output by A1 is 5050

Loop continuously until program control system intervenes

Exit loop when the number of loops reaches 100

	A	B	Value
1	0		5050
2	for 100		
3		>A1=A1+#A2	

Final result output by A1 is 5050

Set the number of loops as 100

- Count the number of students whose scores are higher than 90:

	A	B	C	Value
1	[99,98,87,76,94,78,95,87,85,99,69,78,88,89,94]			6
2	0			
3	for A1			
4		if A3<90	next	
5		>A2=A2+1		

Final result output by A2

Loop sequence A1; cell values will be members of A1 in order

Count the number of students whose scores are higher than 90

- Compute the number at which the summation is over 10,000 during summing up from 1 to n:

	A	B	Value
1	10000		141
2	for A1>0		
3		>A1=A1-#A2	
4		=#A2	

Loop continuously if the result of A1>0 is true

Final result output by B4 is 141

## 4.4 func C, arg/func/return/end Statement

### func

Define the subroutine, of which the code is a code block with **func** as its master cell. The parameters used for calling the subroutine will be copied in the cell where **func** is located and that on the right.

## return x

Return the result  $x$  from the subroutine and terminate the running of subroutine, return null if no return value.

## func(C, $x_i, \dots$ )

With the parameter  $x_i, \dots$ , you can invoke subroutine starting from cell pC, and return the result to fill in this cell, or be written in the function mode  $\text{func}(C, x_i, \dots)$ . When calling subroutine, multiple parameters will be filled in the cells from left to right in order.

During the cellset is running, the code block of subroutine will only be executed when the subroutine is called:

	A	B		Member
1	func		Parameters will be copied to A1 when A5 is calling the A1	1
2		return A1.sum()		5
3	func		Sub routine in A3 is not invoked. The code block with A3 as the master cell will not be executed	3
4		return A3.avg()		4
5	func A1,[1,5,3,4,2,3]		Invoke the sub routine in A1 to sum up	2
			Value	3
			18	

The return value does not necessarily exist for the subroutine. If no **return** command in the code block of subroutine, then the statement in the code block will be executed in order when calling.

	A	B	C	Member
1	[23,113,56,23,1,7]		After running, the member in A1 will be sorted descendingly	113
2	func			56
3		=A1(A2(1))	The sub routine will not return any value. In A1 sequence, the two members located by the parameters are exchanged.	23
4		>A1(A2(1))=A1(A2(2))		23
5		>A1(A2(2))=B3		7
6	for A1.len()-1			1
7		for A1.len()-A6		
8			if A1(B7)<A1(B7+1) func A2,[B7,B7+1]	

There is no local variable available in esProc. All variables can be called in the range of the cellset. Therefore no real recursion program can be developed by principle. However, esProc does not forbid the recursive call actively.

	A	B	C	D	E
1	func				
2		if A1<0	return -1		
3		else if	return A5		
4		else	>A5=A5*A1	func A1,A1-	return A5
5	1				
6	func A1,8		Compute the factorial of 8		Value
					40320

## 4.5 fork

➤ fork  $A_i, \dots$

It means executing the code block of this cell in multithreads.  $A_i, \dots$  represents sequence parameters. The length of sequence of each parameter should be equal. The single value parameters will be regarded and used as sequences that have identical members. The current state of cellset will be duplicated to every thread to be executed respectively. Parameters will be split into many and each will be set into a thread. The result in the code block returned by *result* will be combined into a sequence and then returned to the main thread.

Different from the common *for* loop, each loop in the fork loop is independent and won't affect others, which is similar to multithreads' call of the subroutine. For example:

	A	B	C
1	fork [2,4,6],2,[5,3,1]		
2		if A1(1)>A1(2)	>A1=A1.swap([1],[2])
3		if A1(2)>A1(3)	>A1=A1.swap([2],[3])
4		if A1(1)>A1(2)	>A1=A1.swap([1],[2])
5		result A1	

The instruction in A1 is the computations of three threads that use 2,2,5; 4,2,3; 6,2,1 respectively as their parameters. In the code block, sequences are sorted in ascending order and returned. The result of A1 is as follows:

Member
[2,2,5]
[2,3,4]
[1,2,6]

## 4.6 Execution of Cellset Program and Debug

esProc provides the common program and control debug functionalities, such as breakpoint setup, and single step execution. All functionalities can be accessed from the Program menu and the tool bar:

Run



Run the whole cellset program.

Run and Debug



Start debug and run to the next breakpoint.

Single-step



Run to the next cell which contains an expression

Run to cursor



Run to the cell where the cursor is located

Pause



Pause the current execution, and you can click the Run button to resume.

Terminate



Terminate the current execution, and you can restart the computation from the very beginning by clicking the Run button.

Set/Clear breakpoint



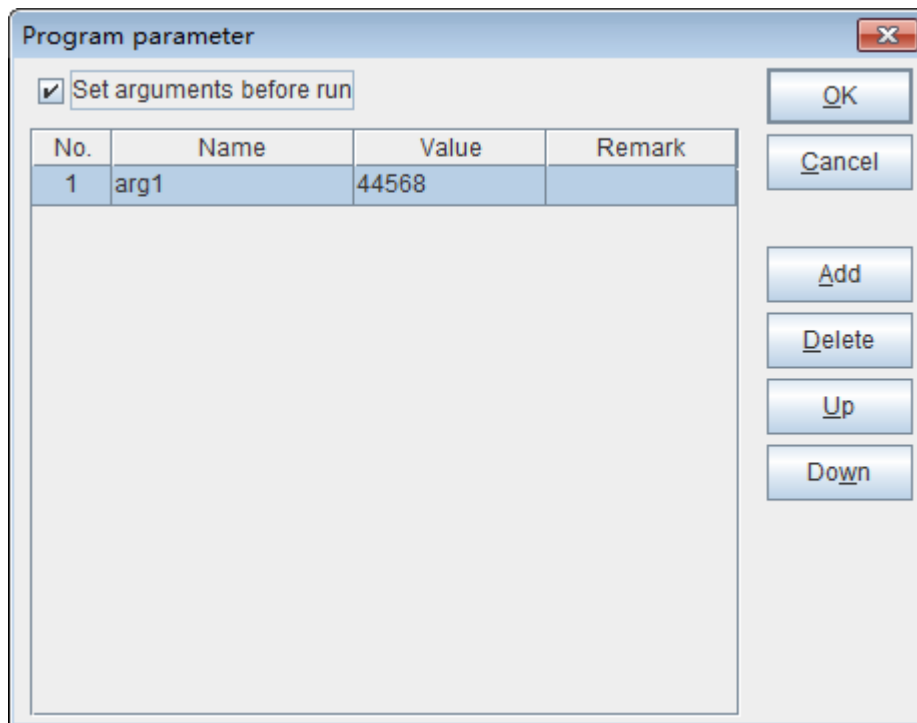
Set or cancel the break point at the cell where the cursor is located.

The **func** statement can be executed in a single step.

## 4.7 Cases of Judgment/Loop Function

### 4.7.1 Prime Factoring

Take the natural numbers as input parameters. Factor it and write to a certain sequence.



Program parameter

☒ Set arguments before run

No.	Name	Value	Remark
1	arg1	44568	

OK  
Cancel  
Add  
Delete  
Up  
Down

	A	B		Member
1	0	=arg1	A1 stores the result of prime factorization	2
2	2		A2 stores the minimal prime factor	2
3	for A2*A2<B1		Judge if number to be factored is a prime	2
4		if B1%A2==0	Exact division means a prime factor is found	3
5			>A1=A1 A2	3
6			>B1=int(B1/A2)	619
7		else	>A2=A2+1	
8	>A1=A1 B1		Add the last prime factor to the sequence of	

## 5. RSeq and TSeq

### 5.1 Table Sequence and Record

esProc inherits the data table concept of relational database, and defines it as **Table Sequence** or **TSeq** for short. Consistent with the concept of relational database, in esProc, every Table Sequence also has its own **Data Structure**, which consists of several **Fields**. Members of a table sequence are also called as **Records**.

The key differences between a table sequence and a relational database table are as follows:

A table sequence is itself a sequence. Therefore, **its members have a certain order** and that is why we call it "table sequence".

The field of table sequence is of no data type. The data types of the values of the same field from different records are allowed to be different. If a certain field of all records has the same data type, the field can be called a **pure field** of the table sequence.

You are not required to name fields of the table sequence, and **you can access them with their sequence numbers**.

Essentially different from the relational database table, **records of a table sequence can be retrieved as the data objects for reference**, and the table sequence can be regarded as the sequence consisting of records. You can access its members just as dealing with the normal sequences.

- **T(1)** the first record of table sequence *T*.
- **T(to(3))** the sequence consisting of the first 3 records of table sequence *T*.

But, in order to guarantee the consistency of data structure, you are **not allowed** to assign values to members of a table sequence in the same way of assigning value to a normal sequence.

- **T(1)=x** error, forbid to execute

Each record in the TSeq will store the reference to the TSeq and share the data structure of this TSeq with each other.

TSeq member modification requires specific functions to be introduced. For details, please refer to the **6.3 Modify Data**.

### 5.2 Record Sequence





A sequence consisting of records from a table sequence is called a **Record Sequence** or **RSeq** for short (such as the above mentioned  $T(\text{to}(3))$ ). As a sequence, the members of a record sequence can be assigned values with the above mentioned method.

The members of a record sequence are **not required from a same table sequence**. The record sequence whose members are from a same table sequence is called as **Pure Record Sequence**.

Though both record sequence and table sequence are sequences consisting of records in appearance, they are different virtually:

The table sequence stores the values of records virtually. Every record must belong to and only belong to one table sequence. Records will not exist without a table sequence.

The record sequence stores the references to the records rather than the actual values.

Therefore, a same record can belong to multiple record sequences or show up in a same record sequence repeatedly.

The record sequence is a concept introduced for the convenience of explanation. There is no specific computation for record sequence in esProc, and a record sequence will not be validated, that is to say, you can assign a value not from any record to a member of the record sequence.

## 5.3 Create TSeq

### 5.3.1 Retrieve and Write TSeq from File

When retrieving and writing the TSeq, you can use many types of common file formats, such as the TXT file, the XML file and CSV file, etc.

***f.import( $F_i;type_i,...;s;b:e$ )***

From the file  $f$ , retrieve the TSeq and return. With default parameters, *import* function will use line break character (\n) to separate the rows, and *Tab* (\t) to separate the columns in the file. Each row corresponds to one record, and each column corresponds to one field. Define the data scope for retrieval with  $b:e$ , that is, from the  $b^{\text{th}}$  byte to the  $e^{\text{th}}$  byte. Define the fields for retrieval with  $F_i$ , and you can set the data type of the field as  $type_i$ . Without the parameter  $F_{i...}$ , all fields of  $A$  will be retrieved. Define the column separator with  $s$ . When *type* is omitted, data type of the first row will be the first choice.

By default, **import()** will return the TSeq with no field name. You can get the TSeq with field name with options:

- ❖ **@t** Take the first row of the file as the field name of the Tseq to be returned.
- ❖ **@b** Retrieve TSeq from exported binary file. In this case option **@t** is invalid and parameters *type*, *s* will be ignored.
- ❖ **@z** Retrieve data from file by block. In this case, *b* and *e* are respectively the block number and total blocks.
- ❖ **@s** Without splitting fields, retrieve the single-field string to form a TSeq. In this case, the parameters will be omitted.

***f.write(A)***

Write the string sequence  $A$  to the file  $f$ . Each member of  $A$  will occupy a row in the file.

- ❖ **@a** Append.



StateId	Name	Population	ShortName	Area	Capital
1	Alabama	4779736	AL	52419	Montgomery
2	Alaska	710231	AK	663267	Juneau
3	Arizona	6392017	AZ	113998	Phoenix
4	Arkansas	2915918	AR	52897	Little Rock
5	California	37253956	CA	163700	Sacramento

	A
1	=file("D:/files/txt/states1.txt")
2	=A1.import()
3	=A1.import@t()

_1	_2	_3	_4	_5	_6
StateId	Name	Popul	Short	Area	Capita
1	Alaba	47797	AL	52419	Montg
2	Alaska	71023	AK	66326	Junea
3	Arizon	63920	AZ	11399	Phoer
4	Arkans	29159	AR	52897	Little F
5	Califo	37253	CA	16370	Sacra

StateId	Name	Popu...	Short...	Area	Capi...
1	Alaba	4,779,7	AL	52419	Montg
2	Alaska	710,23	AK	66326	Junea
3	Arizon	6,392,0	AZ	11399	Phoer
4	Arkans	2,915,9	AR	52897	Little F
5	Califo	37,253	CA	16370	Sacra

Not using @t option, TSeq data structure hasn't field names

Using @t option, the first row of the file object will be the field names in TSeq data structure

Similar to the retrieval of TSeq, esProc can export the TSeq to a specified file.

#### **f.export(A, x:F,...;s)**

It computes the expression  $x$  as fields  $F, \dots$  based on  $A$  and write them to the file  $f$ . Without parameter  $x, \dots$ , it will write out all fields of  $A$ . But still, the file format is that rows are delimited with *Enter* and columns delimited with *Tab*. The **f.export()** function will export *txt* file by default.  $s$  is the column delimiter, and the default delimiter is *Tab*.

- ❖ **@t** Write the field name of first record to the file as the title.
- ❖ **@a** Append. Its omission means overwriting the original file. This option cannot co-exist with option **@t**.
- ❖ **@b** Export as binary compressed file, which is a more efficient way for exporting. This option cannot co-exist with option **@t** and **@a**.
  - ❖ **@bg** When exporting as the binary file, adopt the rule of dividing blocks by group.

The **export()** function is a little different to the **import()** function. The **export()** function can only be used to process those fields which can be converted to strings. The field values which cannot be converted to strings, such as record and RSeq, will be ignored.

#### **5.3.2 Create TSeqs Directly**

The blank table sequence is the one with data structure definition but without records. You can use the following functions to create a blank TSeq directly.

##### **create( $F_i, \dots$ )**

Create a blank table sequence with fields of  $F_i, \dots$

##### **T.create()**

Create a blank table sequence with the same data structure of table sequence  $T$ .



	A
1	=file("D:/files/txt/states1.txt").import@t()
2	=A1.create()
3	=create(StateId,Name,Population)

Create a new TSeq with directly specified field names

Create a new TSeq by copying another's data

StateId	Name	Population

StateId	Name	Population	ShortName	Area	Capital

### **T.record(A,k)**

Fill every fields of the TSeq with members of sequence A in order to form records. When  $k$  is default or  $k == 0$ , append new records to all records of TSeq; If  $k != 0$ , then modify the data in  $T$  from the record at the specified position.

- ❖ **@i** If  $k != 0$ , then insert (not modify) new data before the specified record.

The **create()** function together with the **record()** function can organize the constants in the cellset into a TSeq for future use.

	A	B	C
1	=create(StateId,Name,Population).record(to(6))		
2	1	Alabama	4779736
3	2	Alaska	710231
	3	Arizona	6392017
	=create(StateId,Name,Population).record([A2:C4])		

Fill the blank TSeq with sequences

Organize constants in the cellset into a TSeq for future

StateId	Name	Population
1	2	3
4	5	6

StateId	Name	Population
1	Alabama	4779736
2	Alaska	710231
3	Arizona	6392017

### **5.3.3 Retrieve TSeqs from Databases**

Because a TSeq has the similar structure to that of a data table in the database, you can use the data from the database to form a TSeq.

	A
1	=demo.query("select * from LIQUORS")

LID	NAME	TYPE	PRODUCTION	STOCK
1	42Below Vodka	Vodka	New Zealand	301
2	Absolut Vodka	Vodka	Sweden	95
3	Appleton Estate	Rum	Jamaica	202
4	Bacardi Superior	Rum	Puerto Rico	741
5	Ball's Irish Whiskey	Whiskey	Ireland	124

For details on TSeq retrieval from the database, please refer to the Section **8.2.1 Retrieve TSeq with SQL**.

## 5.4 How to View TSeq or RSeq

Click the cell where a TSeq is located. You can check the value of the TSeq in the Data View section: (In this case, this is the TSeq in cell **A1**)

	A
1	=demo.query("select * from STATES")

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgomery	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
4	Arkansas	2915918	AR	52897	Little Rock	7
5	California	37252956	CA	163700	Sacramento	0

Right click on the field in which records are selected to view the TSeq. You can also **set the display format of this field in the TSeq**.

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgomery	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
4	Arkansas	2915918	AR	52897	Little Rock	7
5	California	37252956	CA	163700	Sacramento	0

Set the display format in the popup Edit Format window:

Edit format

Format

Type

Number  
Currency  
Date  
Time  
Date time  
Fractions  
Scientific notation

#0.00  
#.00  
##  
#0.000  
#.000  
###0.00  
###.00  
###.#  
###0.000

OK

Cancel

Once you set the format, the TSeq will be displayed in the set display format:

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4,779,736	AL	52419	Montgomery	6
2	Alaska	710,231	AK	663267	Juneau	9
3	Arizona	6,392,017	AZ	113998	Phoenix	8
4	Arkansas	2,915,918	AR	52897	Little Rock	7
5	California	37,252,956	CA	163700	Sacramento	0

Viewing RSeqs is similar to viewing TSeqs. However, in an RSeq, the data structure of each record is not always the same. When rendered, the records will be displayed according to the data



structure of the first record. For the records of various data structures, only values of the fields with the same name will be displayed. When viewing an RSeq, you **are not allowed** to set the display formats of fields.

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgome	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
32	New York	8084316				
5	Los Angeles	3798981				
13	Chicago	2886251				

## 5.5 Access Records and TSeqs

### 5.5.1 Access Records

$T(i)$

A TSeq or an RSeq is the sequence composed of records which you can directly access by position.

	A	B	C
1	Black Heart Rum	Rum	New Zealand
2	Bombay Sapphire	Gin	England
3	Canadian Club Sherry Cask	Whisky	Canada
4	Canterbury Cream	Cordials	New Zealand
5	=create(liquor,Type,Production).record([A1:C4])		
6	=A5(2)		

liquor	Type	Production
Bombay Sapp	Gin	England

Access the 2<sup>nd</sup> record

### 5.5.2 Record Field Access and Assignment

Like most structured programming language, the "." operator is adopted for field access.

$r.F$

Return the value of record  $r$ 's field  $F$ .

$r.F=x$

Assign  $x$  to the record  $r$ 's field  $F$ .

You can also access the field with its sequence number. Obviously, fields without a name can only be accessed with this method:

$r.\#i$

Return the value of the  $i^{\text{th}}$  field of record  $r$ .

$r.\#i=x$

Assign  $x$  to the  $i^{\text{th}}$  field of record  $r$ .

$r.\text{field}(i,x)$

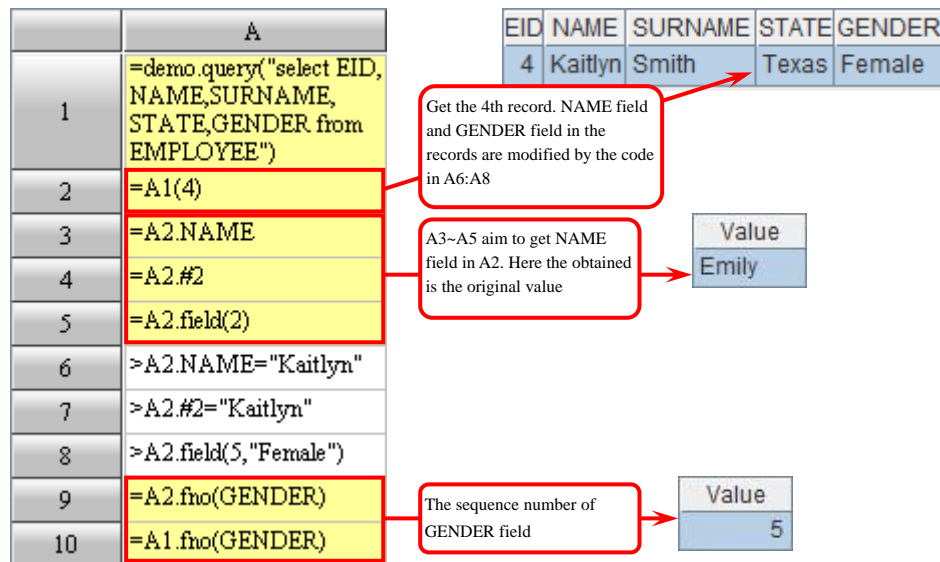
Assign the  $i^{\text{th}}$  field of the record  $r$  with the value  $x$ .

$r.\text{fno}(F), T.\text{fno}(F)$

It is the sequence number of field  $F$  in the record  $r$  or TSeq  $T$ . If  $F$  is omitted, then return the number of TSeq fields.

### **$r.\text{field}(i)$**

It is the value of the  $i^{\text{th}}$  field in the record  $r$ .

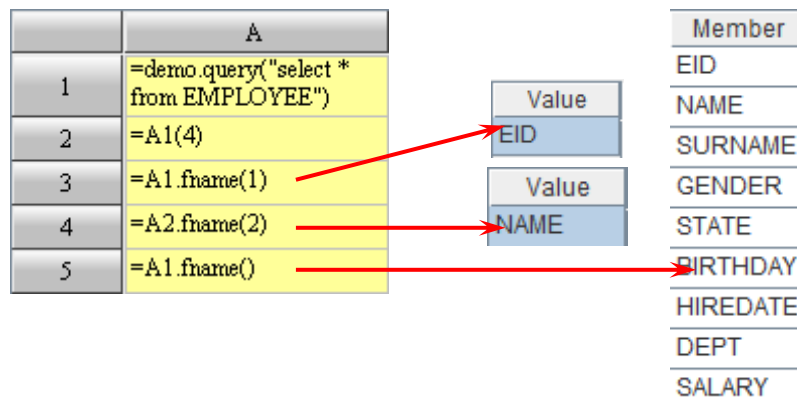


### **$T.\text{fname}(i)$**

It is the name of the  $i^{\text{th}}$  field in TSeq  $T$  and returned as a string.

### **$r.\text{fname}(i)$**

It is the name of the  $i^{\text{th}}$  field in record  $r$  and returned as a string.



## **5.5.3 Judgment of Record/TSeq**

### **ifr(x)**

Judge if  $x$  is the record and return true/false.

### **ift(x)**

Judge if  $x$  is TSeq, and return true/false.

- **ifr(A1)** Judge if the result in A1 is record.
- **ift(A3)** Check if the result in A3 is TSeq.

## 5.6 Loop

### 5.6.1 Expression Convention in the Sequence Loop

Members of a sequence can be referenced in the loop function parameters and the rules are defined as given below:

~

Current sequence member, also called as **Base Member**.

#

Current member's sequence number.

- **A.sum(~\*~)** Compute the sum of squares of the members in A.
- **A.max(if(#%2==0,~,0))** Find out the max value of members in the even positions.

**A[i], ~[i]**

Relative reference. Return the member whose sequence number differs by *i* from that of the current, looped member. The *i* could be a negative number.

- **A. (~-~[-1])** Compute the sequence composed of the differences of each member subtracting the previous member in A.

### 5.6.2 Loop Function of Sequences

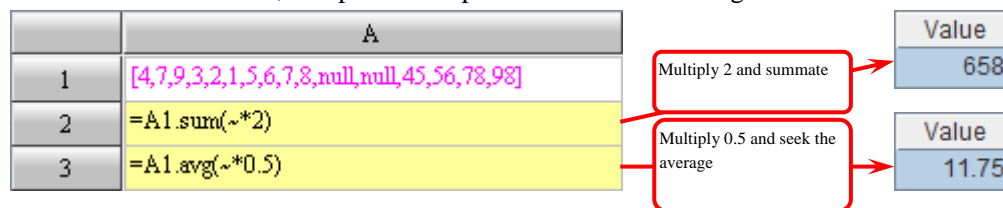
The function to perform certain computations on every member of a sequence is called **Loop Function**, of which the usual form is **A.f(x)**. In fact, converge operation of sequences can also be implemented with the loop function. For example, the summation function **A.sum()** introduced in **3.7.4 Converge Computation**. The computational behavior of the loop function is determined by the function name, such as **sum** indicating sum up, and **avg** indicating average.

**A.sum(x)**

With each member of A, compute the expression *x* and sum up the results.

**A.avg(x)**

With each member of A, compute the expression *x* and the average of the results.



Return the relevant sequence:

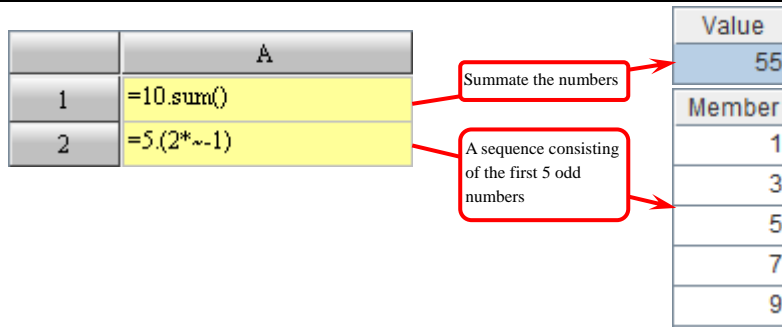
**A.(x)**

Return the sequence after computing *x* for every member in the A.

Integer Loop:

**n.f(x)**

The abbreviated form of **to(n),f(x)**.



### 5.6.3 Expression Convention in the RSeq Loop Computation

***F***

It represents referencing field *F* in an expression. All records in the RSeq must have this field when this field is referenced in the expression.

***r.F, ~.F, A.F***

In the expression, reference fields of the specified records and fields of base members, or fields of specified RSeq.

**The following deserves special attention:** If there is any variable in memory whose name is the same as the field name, please do not use the abbreviation of this field in the loop function. You must use the full name like *~.F* or *A.F*. The *F* alone in coding will be interpreted as the variable of the same name.

The loop function also provides the reference scheme for the neighboring members:

***A[i], ~[i]***

Reference the member which is *i* records away from the current member in the expression. The *i* could be negative value, and return null if the referenced member is out of range.

***A{a:b}, ~{a:b}***

The expression is to reference an RSeq composed of records from the *a*<sup>th</sup> to the *b*<sup>th</sup> record after the current record. *a* or *b* can be negative value. A default *a* will start from the first record in *A*; a default *b* will end at the last record.

***F[i]***

This is equal to *A[i].F*, that is, referencing the field *F* of record *i* records away from the current record.

***F{ ... }***

This is equal to *A(...).(F)*. This expression is to build a sequence composed of records of *F* field after getting records from *A*.





	A	B	C	Name	OBP	SLG
1	Ryan Johnson	0.411	0.529	Ryan Joh	0.411	0.529
2	Jacob Moore	0.370	0.529	Jacob Mo	0.37	0.529
3	Matthew Johnson	0.354	0.464	Matthew J	0.354	0.464
4	Christopher Hernandez	0.34	0.37	Christoph	0.34	0.37
5	=create(Name,OBP,SLG).record([A1:C4])					
6	=A5.(round(OBP+A5.SLG,3))					
7	=A5.(round(OBP-A5[1].OBP,3))					
8	=A5.(round(OBP-OBP[1],3))					
9	=A5.(A5{1:2}.(OBP))					
10	=A5.(OBP{1:2})					

Member
0.94
0.899
0.818
0.71

Member
[0.37,0.35]
[0.354,0.3]
[0.34]

Member
0.041
0.016
0.014
0.34

#### A.field(i)

To RSeq A, return a sequence which is composed of the  $i^{\text{th}}$  field of all records.

#### A.field(i,x)

Typically, the computational result of expression  $x$  is a sequence. Assign the values of the sequence members to the  $i^{\text{th}}$  field of each record in the RSeq A one by one.

	A	Member
1	=demo.query("select * from STUDENTS")	Emily
2	[Alex,Ben,Cathy,Danny,Eric,Frank,Gaby]	Elizabeth
3	=A1.field(2)	Sean
4	>A1.field(2,A2)	Lauren
5	=A1	Michael
		John
		Nicholas

ID	NAME	GENDER	AGE
1	Alex	F	17
2	Ben	F	16
3	Cathy	M	17
4	Danny	F	15
5	Eric	M	16
6	Frank	M	13
7	Gaby	M	16

### 5.6.4 The run function

#### A.run(x)

Cycle the function to compute the  $x$ , but still return A.

	A	Member
1	[1,2,3,4]	2
2	=A1.run(~=~*2)	4
		6
		8

The **run** function can be used for records to simplify the coding when referencing the field for multiple times.



### ***r.run(x)***

For record *r*, compute *x* and return *r*.

- ***r.run(amount=price\*quantity)*** Equal to ***r.amount =r.price \*r.quantity***.

### ***r.(x)***

For record *r*, compute *x* and return *x*.

- ***r.(price\*quantity)*** Equal to ***r.price \*r.quantity***.

In most cases, **run()** function is used to assign values to the records of a record sequence although you can still use it to handle a normal sequence. When the **run()** function is used for record assignment, esProc provides a much clearer way of coding:

### ***A.run( x<sub>i</sub>:F<sub>i</sub>,... )***

Assign value *x<sub>i</sub>* to the field *F<sub>i</sub>*. This is equivalent to ***A.run(F<sub>i</sub>=x<sub>i</sub>,...)***.

- ***A.run(age+1:age)*** Equivalent to ***A.run(age=age+1)***.

The **run** function in this format can also be used to assign values to multiple fields simultaneously. You can use it by simply putting it in proper order:

- ***A.run(age+1:age, age-entryage:schoolage)***

	A	B
1	42-Inch 1080p LCD TV - Black	1110.99
2	50-Inch 1080p Plasma HDTV	1099.95
3	42-Inch 1080p Plasma HDTV	799.95
4	51-Inch 1080p 600Hz 3D Plasma HDTV	1299.99
5	=create(TV,Original,Off,Price).record([A1:D4])	
6	>A5.run(Off=round(Original*0.15,2), Price=Original-Off)	>A5.run(round(Original*0.15,2):Off, Original-Off:Price)

The code in A6 equals to that in B6. Use run() function to compute the deduction and the discounted price after 15% discount.

## 5.6.5 Iteration

The loop function can be used iteratively, that is, use loop function again in the computation expression.

### ***A<sub>1</sub>.f<sub>1</sub>(A<sub>2</sub>,f<sub>2</sub>(x))***

In the loop computation, the symbol in the *x* will be explained preferentially as belonging to *A<sub>2</sub>*.

- ***10.(#.sum(~\*~))*** A sequence of the accumulated quadratic sum from 1 to 10.

In the nested loop function, the **~** and **#** will be interpreted as the current member and the sequence number of the inner sequence. In referencing the outer sequence, the sequence name must be added, like ***A.~*** and ***A.#***.

- ***A.max(B.count(A.~==~))*** Max occurrence of members of *A* showing up in *B*.



	A	B	C	D
1	Nicholas Johnson	47.8	Light	48
2	Jacob Williams	55.8	Bantamweight	54
3	Michael Harris	56.1	Featherweight	57
4	Tyler Anderson	68		
5	=create(Name,Weight).record([A1:B4])			
6	=create(Class,Weight,Count).record([C1:E3])			
7	>A6.run(Count=A5.count(Weight<=A6. Weight && (Weight>A6[-1].Weight)))			

In the expression of A7, if the Weight is not clearly indicated as A.F, then it will be interpreted as belonging to A5 firstly. The A7 is to compute the total number of athletes at various classes. The computation of A6 is as follows:

Class	Weight	Count
Light flyweight	48	1
Bantamweight	54	0
Featherweight	57	2

### 5.6.6 Summary Loop

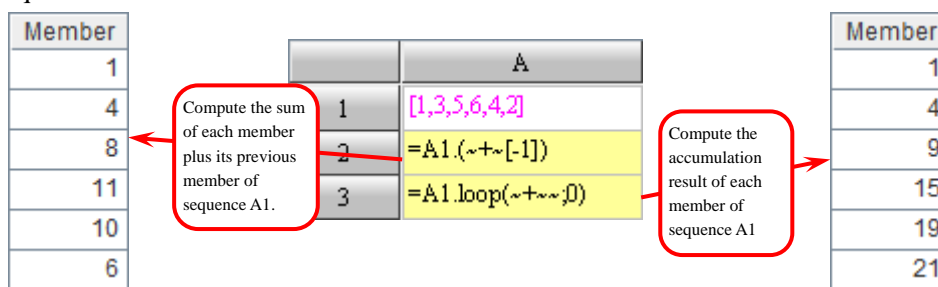
~~

Used to perform calling in the summary loop statement, then return the result of computing expression  $x$  last time.

#### A.loop(x;a)

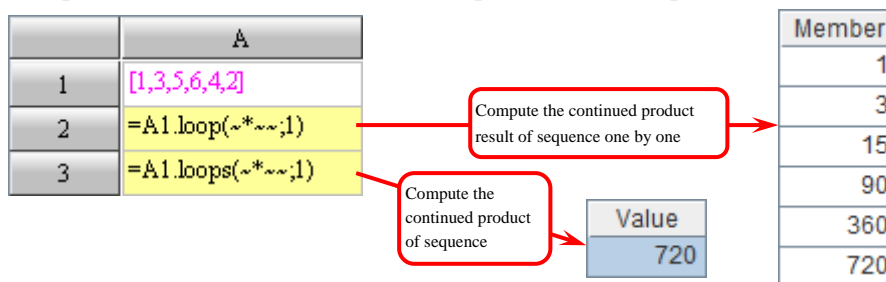
A summary loop, meaning executing loop on members in sequence A, and the results of computing expression  $x$  will form a sequence. In the expression  $x$ , you can use  $~~$  to reference the previous result of the expression. The initial value of  $~~$  is  $a$ . **loop()** function can handle loop computations, such as accumulation and summarizing.

If  $~~$  is used to make a reference, the previous computed result of expression  $x$  will be returned. This differs from the case of relative reference.  $~[-1]$  will return the previous member in the sequence A.



#### A.loops(x;a)

Compute A.loop(x;a). Return the last computed result of expression  $x$ .



You can use A.loops(x;a) to compute the final result of summary loops.

## 5.7 Basic Operations on RSeqs

### 5.7.1 Locate Function

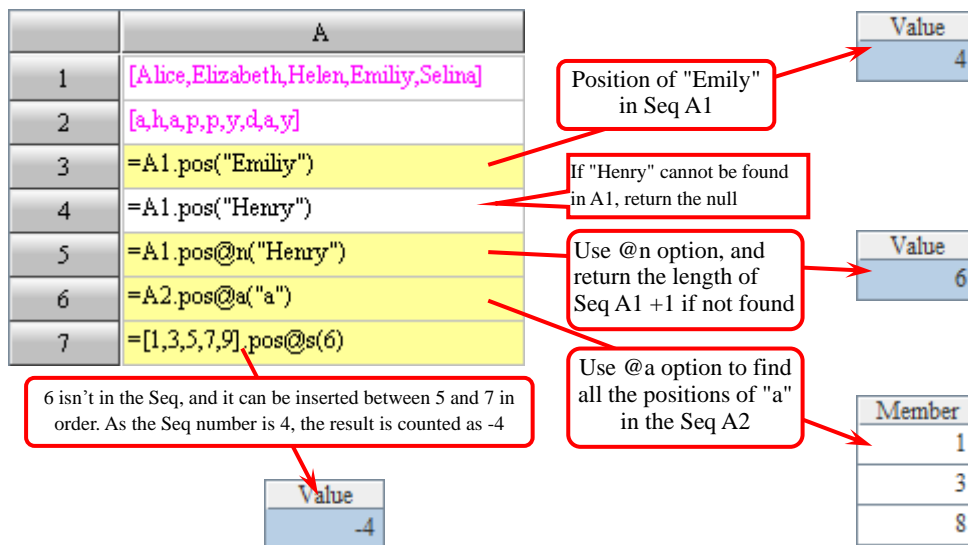
Position functions can be used to locate the specified members or members meeting certain conditions in a sequence or an RSeq. The locate function usually has the following two common options:

- ❖ **@a** Return the position sequence of all members meeting the conditions.  
Otherwise, only return the position of the first member.
- ❖ **@z** Locate from back to front.

#### A.pos(x)

Return the sequence number of member  $x$  in sequence  $A$ , in which  $x$  is not a sequence; when  $x$  is not found, return null.

- ❖ **@b** Members of  $A$  are ordered. Binary search is used for locating desired members.
- ❖ **@s** Members of  $A$  are ordered. With the binary search, the position of  $x$  will be returned if  $x$  is a member of  $A$ ; otherwise, return the opposite number of the sequence number at which position the  $x$  can be inserted orderly.
- ❖ **@n** If  $x$  is not found, return  $A.len()+1$  instead of null, it is mutually exclusive with **@a** option.



#### A.pmin(x)

Return the sequence number of the member in  $A$  which can produce a min value for expression  $x$ .

- **A.pmin(UnitPrice\*Quantities)** The sequence number of the item of the lowest total price.

#### A.pmax(x)

Return the sequence number in  $A$  which can produce a max value for expression  $x$ .

- **A.pmax@a(Win\*3+Draw)** To compute the sequence numbers of all football teams with the highest points in the league football match.

#### A.pselect(x)



Return the sequence number of the first member meeting the condition  $x$  in  $A$ . When  $x$  is set by default, return a sequence composed of the sequence numbers of all members of  $A$ .

- ❖ **@b** Members of  $A$  are ordered for the expression  $x$ . Use binary search to locate the sequence numbers of members in  $A$  which meets the condition  $x==0$ , and then return the sequence number.
- ❖ **@s** Members of  $A$  are ordered for the expression  $x$ . With the binary search, if none of the members in  $A$  can make the expression  $x$  generate a result of 0, then return the opposite number of the sequence number of the position at which the number meeting the condition can be inserted.
- ❖ **@n** If no member in  $A$  meets the condition  $x$ , return  $A.len()+1$ . The **@a** option and it are mutually exclusive.
- ❖
- **A.pselect( ~>5 )**
- **[1,2,3,4,5].pselect@bs(cmp(~,3.5))** The result is -4. With **@bs** option, the negative value will be returned if all members in the sequence cannot meet the condition of the expression.

#### **A.pselect( $x_k:y_k,\dots$ )**

Return the sequence number of the first member in  $A$  that meets the condition  $x_k==y_k\dots$

- ❖ **@b** All members of  $A$  are sorted in ascending/descending order based on expression  $x_k$ . Locate needed members with binary search.
- ❖ **@s** All members of  $A$  are in descending/ascending order based on expression  $x_k$ . With binary search, if no members meeting the condition are found, then return the opposite number of the sequence number of the position at which the member meeting the condition can be inserted.
- ❖ **@n** If no member in  $A$  meets the condition, return  $A.len()+1$ . The **@a** option and it are mutually exclusive.
- **A.pselect( Gender:"M",left(Name):"C")** Locate the sequence number of the first male employee whose initial is C.

	A		
1	=demo.query("select STATEID, NAME, POPULATION, AREA from STATES")	STATEID	NAME
2	[Colorado, Idaho, Texas, Ohio]	1	Alabama
3	=A2.pos("Texas")	2	Alaska
4	=A1.pmax(POPULATION)	3	Arizona
5	=A1.pmin(POPULATION/AREA)	4	Arkansa
6	=A1.pselect(left(NAME,1)=="C")	5	California
7	=A1.pselect@a(left(NAME,1)=="C", POPULATION>5000000:true)	6	Colorado

Value	
3	Serial number of Texas in A2
5	Serial number of state with the highest population
2	Serial number of state with the lowest population density
5	Serial number of the first state whose name starts with C

Member	
5	Serial numbers of all states whose name starts with C and population exceeds 5 million.
6	

When using the above position functions, you can specify the starting position for searching:

**A.f(x,k)**

Search from the  $k^{\text{th}}$  member; and you need to write it as **A.f(k)** if  $x$  is set by default.

- **A.pos(1,5)** Locate the position of 1, starting from the 5<sup>th</sup> member.

Also, you can return positions of multiple members at the same time using function **pos()** with parameters of sequence.

**A.pos(x)**

In it,  $x$  is a sequence. Return the ISeq  $p$  to make  $A(p)=x$ . In searching, for each member of  $x$ , only locate the first position.

- ❖ **@i** Return the unique member increasing ISeq  $p$  to make  $A(p)=x$ .
- ❖ **@b**  $A$  is ordered. Use binary search to locate members.
- ❖ **@c** Find their first positions in which members of  $x$  appear in  $A$  in **proper order**.

	A	
1	[3,2,1,9,6,9,1,2,8]	
2	=A1.pos([3,1,1,1])	
3	=A1.pos@i([3,1,1,1])	
4	=A1.pos@i([2,1,9,1])	
5	=A1.pos@c([1,9,6,9])	

Member	
1	
3	
3	
3	

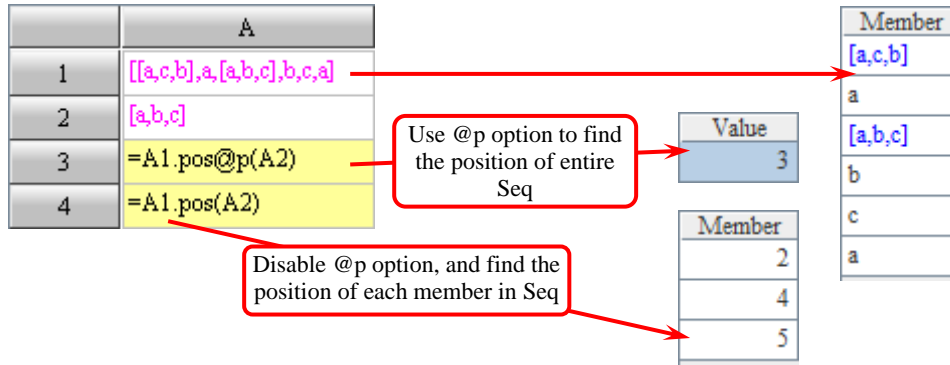
Member	
2	
3	
4	
7	

Value	
3	

In particular, the member  $x$  can also be located in  $A$  if  $x$  is a sequence. In this case,  $A$  is usually a sequence of another sequence.

- ❖ **A.pos@p(x)** Return the position of sequence member  $p$  in  $A$ .



### A.in(B)

All members of sequence A are held in the sequence B, that is, **B.posi@p(A) != null**.

Let's discuss some combined functions below. They will search the desired position according to a certain condition, and then perform computing in the specified sequence according to this position.

### A.lookup(A<sub>i</sub>:x<sub>i</sub>,...)

Find the first  $k$  that satisfies the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Return the  $k^{\text{th}}$  member of the sequence A.

❖ @a Return the sub-sequence of A according to all positions satisfying the condition.

### A.sumif(A<sub>i</sub>:x<sub>i</sub>,...)

Find all  $ks$  that satisfy the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Return the sum of values of the members in these positions of the sequence A.

### A.countif(A<sub>i</sub>:x<sub>i</sub>,...)

Find all  $ks$  that satisfy the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Count the members of the sequence A in these positions and return the result.

### A.avgif(A<sub>i</sub>:x<sub>i</sub>,...)

Find all  $ks$  that satisfy the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Return the average of values of the members in these positions of the sequence A.

### A.minif(A<sub>i</sub>:x<sub>i</sub>,...)

Find all  $ks$  that satisfy the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Return the minimum value of the members in these positions of the sequence A.

### A.maxif(A<sub>i</sub>:x<sub>i</sub>,...)

Find all  $ks$  that satisfy the condition, making the  $k^{\text{th}}$  member of the sequence  $A_i, \dots$  equal to  $x_i, \dots$  respectively. Return the maximum value of the members in these positions of the sequence A.

These combined functions are usually used when data are stored in the cellsets directly. The desired data can be searched on the specified conditions. For example:



		A	B	C	D
	1	1	New York	8084316	NY
	2	2	Los Angeles	3798981	CA
	3	3	Chicago	2886251	IL
	4	4	Houston	2009834	TX
	5	5	Philadelphia	1492231	PA
	6	6	Phoenix	1371960	AZ
	7	7	San Diego	1259532	CA
	8	8	Dallas	1211467	TX
	9	9	San Antonio	1194222	TX
	10	10	Detroit	925051	MI
Member	11	=[A1:A10]	=[B1:B10]	=[C1:C10]	=[D1:D10]
Los Angeles	12	=B11.lookup(A11:5)			Value
San Diego	13	=B11.lookup@a(D11:"CA")			Philadelphia
Value	14	=B11.lookup(D11:"CA",B11.(left(~,1)):"S")			Value
4415523	15	=C11.sumif(D11:"TX")			San Diego
Value	16	=C11.countif(D11:"TX")			Value
1471841.0	17	=C11.avgif(D11:"TX")			3
Value	18	=C11.minif(D11:"TX")			Value
2009834	19	=C11.maxif(D11:"TX")			1194222

The data in the cellsets are about the top 10 cities ranked by population in the USA. A12 is to find the name of the city ranked the 5<sup>th</sup>; A13 is to find the names of all the cities in California State; A14 is to find the city whose initial is S and located in California State; A15 is to compute the total urban population of Texas State; A16 is to compute the total cities of Texas State; A17~A19 is to compute the average, minimum, and maximum urban population of Texas State.

In this case, the sequence for computation is directly built with cellsets. When using the sequence, you can compute it through TSeq.

### 5.7.2 Pickout Function

Pickout functions can be used to select out a member or record at the specified location or meeting certain conditions from a sequence or an RSeq. The pickout functions usually have the following four common options:

- ❖ **@1** Return the first member meeting specific conditions. Please note that the **@1** is integer 1 instead of character l. In fact, there is no such character option as l in esProc.
- ❖ **@a** Return a sequence composed of all members meeting conditions.
- ❖ **@z** Locate from back to front.

#### A.minp(x)

Return the member of A resulting in the min value for expression x. The default option of the function is **@1**. If requiring the function to return all members meeting the conditions, then you can use **@a** option.

- **A.minp(UnitPrice\*Quantities)** The goods with the lowest total price.



### A.maxp(x)

Return the member of A resulting in the max value for expression  $x$ . The default option of this function is @1. To return all members meeting the conditions, you may use @a option.

- **A.maxp@a(Win\*3+Draw)** Compute all football teams with the highest points in the league football match.

### A.select(x)

Return the sequence consisting of members satisfying the condition  $x$  in the A. Return all members of A when  $x$  is omitted.

- ❖ **@b** Binary search is used to locate desired members, the filtering condition is  $x==0$ , and the A must be ordered for  $x$ .

- **A.select( ~>5 )** Select all the members greater than 5 in the sequence A.

Unlike **pselect**, the **select** function will return all members satisfying the conditions by default. By comparison, the **pselect** will only return the first match. **A.select(x)** can be defined as **A(A.pselect@a(x))**. The @1 option can be used if the first member meeting the conditions is to returned only.

	A	
1	=demo.query("select * from SOCCERSTAT")	The team with the highest points
2	=A1.maxp(W*3+D)	All the teams with the highest of matches ended in draw
3	=A1.minp@a(D)	
4	=A1.select(F-A>30)	All teams with a goal difference over 30

ID	TEAM	W	D	L	F	A
1	Red Devils	23	11	4	78	37

ID	TEAM	W	D	L	F	A
6	Reds	17	7	14	59	44
13	Potters	13	7	18	46	48
17	Wolves	11	7	20	46	66

ID	TEAM	W	D	L	F	A
1	Red Devils	23	11	4	78	37
2	Pensioners	21	8	9	69	33

## 5.7.3 Sorting

### A.psort(x)

Sort the members in A to arrange the results of expression  $x$  in ascending order, and return a sequence composed of sequence number of every member in A. If  $x$  is omitted, then sort the members in A and return the sequence of sequence numbers.

### A.psort(x<sub>i</sub>:d<sub>i</sub>,...)

Sort members of A in the order of the results of expression  $x_i$ , and return a sequence composed of the sequence number of each member of A. For processing, the  $d_i$  is the direction of each time of sorting, 1 or omission represents the ascending, -1 represents the descending, and 0 represents the original order.

- ❖ **@i** Return **A.psort(...).inv()**. For details on Inverse ISeq, please refer to the Section **5.8.2 Inverse ISeq and Inverse Sequence**.

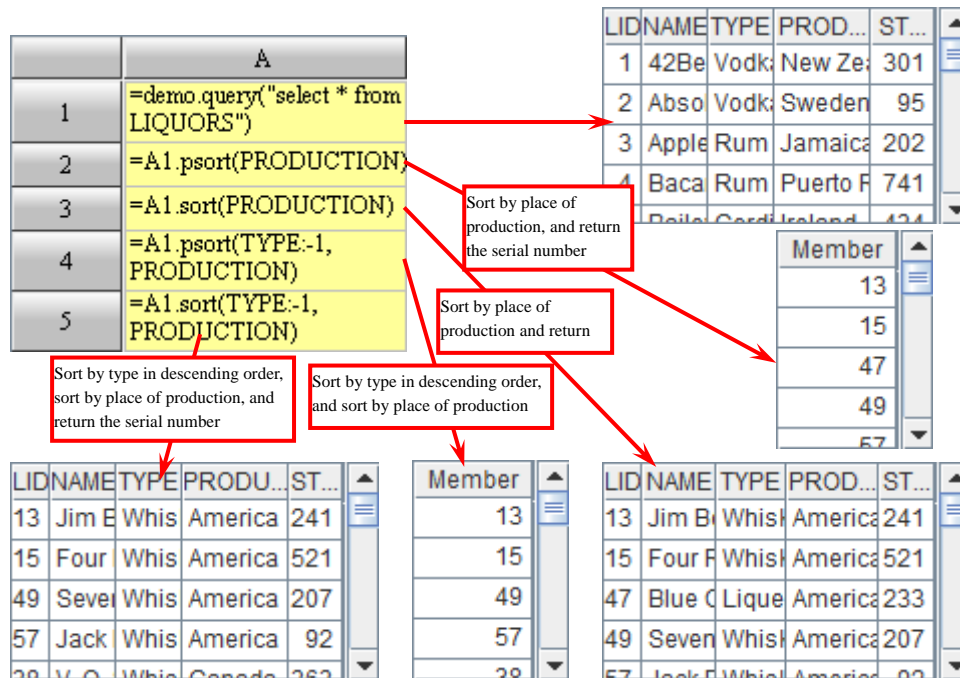
### A.sort(x)

Sort the members in A and then return the sequence, so as to arrange the results of expression  $x$  in ascending order. If  $x$  is omitted, then you sort the members in A.



### **A.sort( $x_i; d_i, \dots$ )**

Sort the members in  $A$  by the results of expression  $x_i$  and then return the sequence. During sorting,  $d_i$  is the direction for sorting at each expression, 1 or omission represents ascending, -1 represents the descending, and 0 represents the original order.



### **A.sort(...;loc)**

Sort with the specified local language, and  $loc$  is the language name.

- **A.sort(;"zh")** Sort  $A$  in Chinese. Besides the language name, the  $loc$  can be used to indicate the region, for example, "en", "zh\_TW", and "en\_GB".

### **A.ptop( $x...;n$ )**

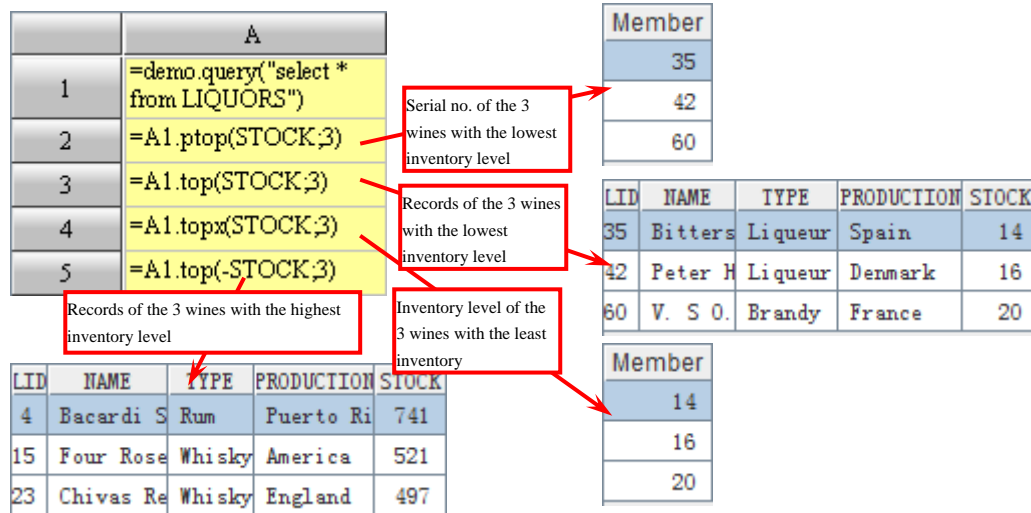
Sort the members in  $A$ , and return a sequence of sequence numbers of  $n$  members that result in the smallest results of expression  $x$ .

### **A.top( $x...;n$ )**

Sort the members in  $A$ , and return  $n$  members that result in the smallest results of expression  $x$ .

### **A.topx( $x...;n$ )**

Sort the members in  $A$ , and return the  $n$  smallest results of expression  $x$ .



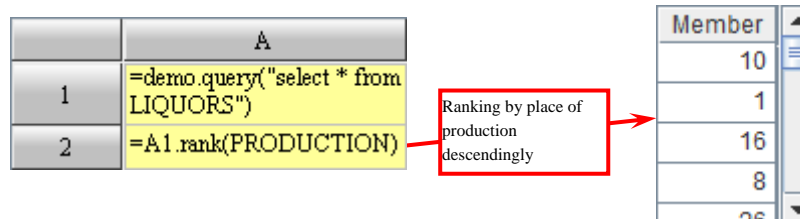
### 5.7.4 Ranking

In the sequences, you can use **A.rank(y)** and **A.rank()** functions of converge computation to compute the rankings. Similar computations can also be performed in the RSeqs.

#### A.rank(x)

For each member in the RSeq A, compute the ranking of the expression x, that is, **A.(x).rank()**.

❖ @z Sort in ascending order.



## 5.8 Other Operations on RSeq

### 5.8.1 Locating Computation

The locating computation function can be used to compute the expression on the specified members of a sequence and return the result. Although it is required to compute on the sequence, you can still use it to accomplish the computation based on an RSeq.

#### A.calc(k,x)

At the  $k^{\text{th}}$  record of the RSeq A, compute the expression x and return the result.

#### A.calc(p,x)

For the RSeq A, compute the expression x on every record whose sequence number is in the ISeq p, and return a sequence which is composed of the results of the expression.

EID	NAME	SURNAME
1	Rebecca	Moore
2	Ashley	Wilson
3	Rachel	Johnson
4	Emily	Smith
5	Ashley	Smith

A
1 =demo.query("select EID,NAME,SURNAME from EMPLOYEE where EID<20")
2 =A1.calc(4,NAME+" "+SURNAME)
3 =A1.calc(A1.pselect(NAME=="Emily"), NAME+" "+SURNAME)
4 =A1.calc([2,4,5,8],NAME+" "+SURNAME)

Value
Emily Smith
Value
Emily Smith
Member
Ashley Wilson
Emily Smith
Ashley Smith
Megan Wilson

### 5.8.2 Inverse ISeq and Inverse Sequence

Suppose that there is an ISeq  $p'$  with a length of  $k$  and an initial value of 0. Perform loop on the ISeq  $p$ , and assign value to  $p'$ . If the member at the  $i^{\text{th}}$  position in  $p$  is  $p(i)$ , then the member in  $p'$  at the  $p(i)^{\text{th}}$  position get assigned as  $i$ . The resulting  $p'$  is the **Inverse ISeq** of  $p$ .

In the sequence  $A$ , take the inverse ISeq  $p'$  that is as long as the ISeq  $p$  as the position ISeq. The resulting sub-sequence of  $A$  is called the inverse sequence of  $A$  regarding  $p$ .

#### $p.\text{inv}(k)$

Return the inverse ISeq of  $p$  with a length of  $k$ . If omitting  $k$ , then return an inverse ISeq that is as long as  $p$ .

#### $A.\text{inv}(p)$

Return the inverse sequence of  $A$  regarding  $p$ . Please note that if ISeq  $p$  is an  $n$  permutation, then  $p$  and  $p.\text{inv}()$  are inverse ISeqs of each other, and  $p.\text{inv}(p)$  will obtain  $[1,2,3,\dots,n]$ .

A
1 [2,3,1,5,4]
2 =A1.inv()
3 =A1.inv(A1)

Member
3
1
2
5
4

Member
1
2
3
4
5

## 5.9 Application of RSeqs

In a sports game, the scores of female all-around gymnastics scores are stored in the database table GYMNASTICSWOMEN:



ID	NAME	COUNTRY	VAULT	UNEVENBARS	BALANCEBEAM	FLOOR
1	Ana Ailva	BRA	14.175	14.175	14.175	14.35
2	Anna Pavlor	RUS	15.275	14.525	15.975	15.05
3	Ariella Kask	SUI	15.35	14.275	14.425	13.95
4	Barbara Keesling	SUI	14.525	14.125	15.075	14.3
5	Becky Down	GBR	15.025	15.625	14.7	14.1
6	Beth Tweddle	GBR	14.4	15.675	14.8	15.1

esProc will be used to collect the statistics on scores below.

### 5.9.1 Score Statistics 1

Sort the scores of athletes by nationality and vault scores respectively:

	A	ID	NAME	COU...	VAULT	UNE...	BALA...	FLOOR
1	=demo.query("select * from GYMNASTICSWOMEN")	14	Dicks	AUS	14.075	14.875	14.85	14.475
2	=A1.sort(COUNTRY)	18	Euler	AUS	14.425	15.275	14.525	14.275
3	=A1.sort(VAULT:-1)	20	Gaelle	BEL	14.0	12.875	13.1	13.975
		1	Ana Ail	BRA	14.175	14.175	14.175	14.35
		15	Diogo	BRA	12.05	14.075	15.05	14.725
		21	Hong	PRK	15.525	13.5	14.25	13.675
		3	Ariella	SUI	15.35	14.275	14.425	13.95
		2	Anna F	RUS	15.275	14.525	15.975	15.05
		5	Becky	GBR	15.025	15.625	14.7	14.1
		26	Okon	GBR	15.025	14.8	14.85	14.4

### 5.9.2 Score Statistics 2

List of athletes who are ranked in Top 10 in both the vault and the uneven bars events:

	A	Member
1	=demo.query("select * from GYMNASTICSWOMEN")	Becky Down
2	=A1.sort(VAULT:-1)	Chellsie Memmel
3	=A1.sort(UNEVENBARS:-1)	Elyse Hopf
4	=A2(to(1,10))^A3(to(1,10))	
5	=A4.sort(ID).(NAME)	

### 5.9.3 Score Statistics 3

Compute the total scores for each athlete to generate a ranking list of total scores, and the ranking list of total scores of athletes from USA and China:



	A	
1	=demo.query("select * from GYMNASTICSWOMEN")	
2	=A1.(round(VAULT+UNEVENBARS+ BALANCEBEAM+FLOOR,2))	
3	=A1(A2.psort(~:-1))	
4	=A3.(NAME)	
5	=A3.select(COUNTRY=="USA"   COUNTRY=="CHN").(NAME)	
6	=A1.select(COUNTRY=="USA"   COUNTRY=="CHN").sort(VAULT +UNEVENBARS+ BALANCEBEAM +FLOOR:-1).(NAME)	

Ranking list of total scores

Member
Chellsie Memmel
Ferrari
Anna Pavlor
Bigorre
Ruth Tweddle

The ranking list of athletes of China and USA. The results in A5 and A6 are the same.

Member
Chellsie Memmel
Bigorre
Zhou Zhuoru
Pang Panpan

## 6. Table Sequence Maintenance and Modification

### 6.1 Computed Fields

You can add a computed field to a TSeq or an RSeq, and return a new TSeq.

**A.derive( $x_i:F_i, \dots$ )**

Add field  $F_i, \dots$  to RSeq A with the value of  $x_i, \dots$  in which  $x$  is an expression related to A, and return a new TSeq. You can use the fields of A in this expression, and get null value if omitting the fields. If  $F$  is omitted, an unnamed field will be added.

**A.derive(UnitPrice\*Quantity:Amount)** Add the "Amount" field to TSeq A and return the newly created TSeq

	A	
1	=demo.query("select * from RECEIPT")	
2	=A1.derive(round(UNITPRICE* QUANTITY,2):Total)	

The derive() functions will not change the original TSeq. A new TSeq will be returned.

NAME	UNITPRICE	UNIT	QUANTITY	Total
Apple	1.69	LB	1.2	2.03
Banana	0.69	LB	3.23	2.23
Cucur	0.77	EACH	3.0	2.31
Onion	0.99	LB	1.33	1.32
Orange	4.99	BAG	1.0	4.99
Red Grape	0.99	LB	2.87	2.84

### 6.2 Record Reference

#### 6.2.1 Record Reference

The fields of records in a TSeq are of no data type, which means that you can assign any value to them. If assigning another record to it, then you can conveniently implement the foreign key references.

Suppose A is a Department table, and B is a Staff table, if you perform the following code:

- **A.run(Manager=B.select@1(ID:A.Manager))**

The field Manager of the table sequence *A* will become the records of the table sequence *B*.  
At this point, you can use:

- **A.select(Manager.Gender:"Female")** Get the department whose manager is Female.

This is to avoid the multiple table associations that frequently occur in SQL. The coding will thus be much clearer and the computation will be much faster.

A	
1	=demo.query("select CID,NAME, POPULATION,STATEID as STATE from CITIES")
2	=demo.query("select STATEID,NAME, ABBR,CAPITAL from STATES")
3	=A1.run(STATE=A2.select@1 (STATEID==STATE))

Set value of STATE field in cities information as the

CID	NAME	POPULATION	STATE
1	New York	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402234	20

Double click to view the value of STATE field. The value is a

STATEID	NAME	ABBR	CAPITAL
13	Illinois	IL	Springfield

### 6.2.2 RSeq Reference

You can also assign an RSeq to the field since there is no restriction on the field values.

Suppose there is a Department table *D* and an Employee table *E*, run the following program:

- **D.derive(E.select(DeptNumber:D.Serial Number):Employee)**

You can add an Employee field in *D*, whose value is a record sequence consisting of the records from the Employee table, to create a new TSeq.

- **D.select@1(Dept:"Sales").Employee.avg(Age)**

Then, you can compute the average age of employees in Sales department.

A	
1	=demo.query("select CID,NAME, POPULATION,STATEID as STATE from CITIES")
2	=demo.query("select STATEID, NAME,ABBR,CAPITAL from STATES")
3	=A1.run(STATE=A2.select@1 (STATEID==STATE))
4	>A2.primary(ABBR)
5	=A2.derive(A1.select(STATE:A2.~).CITIES)

STATEID	NAME	ABBR	CAPITAL	CITIES
1	Alabama	AL	Montgom	[90,104]
2	Alaska	AK	Juneau	[74]
3	Arizona	AZ	Phoenix	[6,27,39]
4	Arkansa	AR	Little Ro	
5	Californ	CA	Sacram	[9,7,14]

Double click to view the Cities field value that is an RSeq

CID	NAME	POPULATION	STATE
6	Phoenix	1371960	AZ
27	Tucson	503151	AZ
39	Mesa	426841	AZ
81	Glendale	246531	AZ
85	Glendale	246531	AZ

The State field is still the record reference, but the display differs to the above example. The record will display the value of its primary key.

Similar to record reference, join operation in SQL can also be avoided for record sequence reference, so as to write clearly and compute quickly. The difference is that record reference usually refers to referencing primary table records in subtable, while on the contrary, record sequence reference usually refers to referencing subtable records in the primary table.

## 6.3 Modify Data

### 6.3.1 Insert/Delete Records

***T.insert( $k, x; F_i, \dots$ )***

Insert a record before the  $k^{\text{th}}$  record in a table sequence, and assign value  $x$  to field  $F$ . If  $k$  is 0, then the record will be inserted at the end of the TSeq; omitting  $F$  means assigning value one after another; that both  $x$  and  $F$  are omitted represents inserting an empty record.

- ***T.insert(0, "David":Name, 28:Age, "M":Gender)***
- ***T.insert(2, "Julia", "F", 25)***

***T.delete( $k$ )***

Delete the  $k^{\text{th}}$  record in the TSeq  $T$ .

***T.delete( $p$ )***

Delete records of TSeq  $T$  whose sequence numbers are in ISeq  $p$ .

***T.delete( $A$ )***

Delete the records of RSeq  $A$  from TSeq  $T$ .

- ***T.delete(3)*** Delete the 3<sup>rd</sup> record.
- ***T.delete(to(2,4))*** Delete the records from the 2<sup>nd</sup> to the 4<sup>th</sup>.

In the following example, in order to check the result of insert/delete operation, you can click



the button continuously to use the stepwise execution of the debug function, and check the execution result of each cell one by one:

A				
1	=demo.query("select * from STUDENTS")			
2	=A1.insert(2,25:ID,"William":NAME,20:AGE,"M":GENDER)			
3	=A2.insert(0,111,"Jessica","F",18)			
4	=A3.delete(1)			
5	=A4.delete(to(3,5))			

ID	NAME	GENDER	AGE
1	Emily	F	17
25	William	M	20
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

ID	NAME	GENDER	AGE
25	William	M	20
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16
111	Jessica	F	18

ID	NAME	GENDER	AGE
25	William	M	20
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16
111	Jessica	F	18

For TSeqs, you have to use **insert** and **delete** functions to insert and delete its members. This

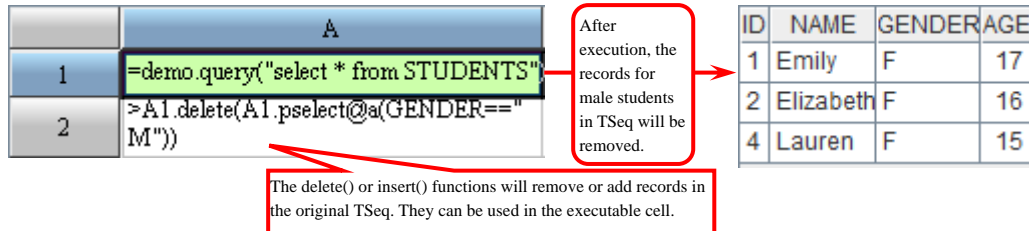




is different with common sequences because the latter can be reassigned with values to make any change.

esProc does not provide the counterpart of deleting on conditions function as given in SQL though, you can implement this function by combining with the **pselect**:

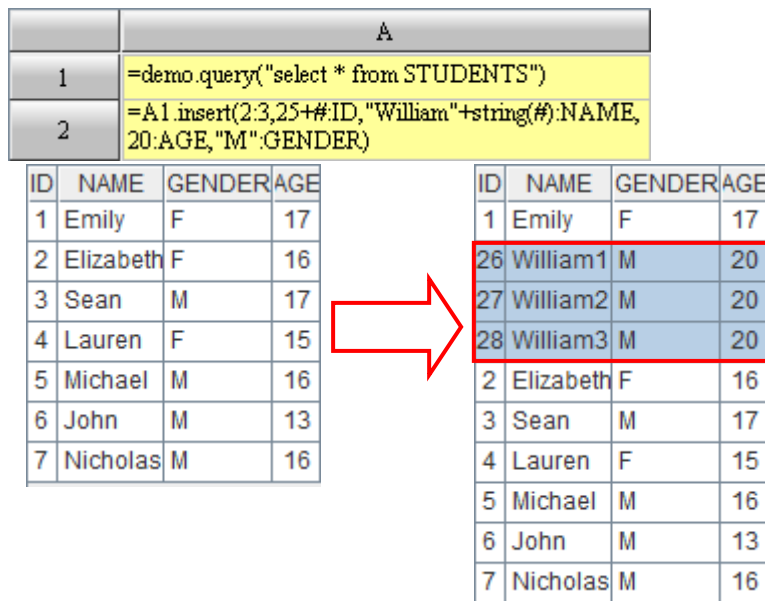
– **T.delete(T.pselect@a(Age>=30))**



The following methods can be adopted to insert records in batches:

**T.insert(k:A,x<sub>i</sub>:F<sub>i</sub>,...)**

Operation against each member of *A* through looping execution. Insert multiple records before the *k*<sup>th</sup> record of TSeq *T*. Field *F* gets assigned with *x*. If *k* is 0, then insert at the ending, and omitting *F* indicates the assignment will be done one by one. For *A*, the integer *n* can be used to represent the sequence to(*n*).



### 6.3.2 Modify Records

**r.modify(x<sub>i</sub>:F<sub>i</sub>,...)**

Modify record *r*, and assign field *F* with value *x*. If *F* is omitted, then assign values one after another in proper order.

**T.modify(k,x<sub>i</sub>:F<sub>i</sub>,...)**

**T(k).modify(x<sub>i</sub>:F<sub>i</sub>,...)**, modify the *k*<sup>th</sup> record in TSeq *T*.

- **T.modify(1,"Tyler":Name,28:Age,"M":Gender)**
- **T.modify(2,"Alexis","F":Gender,25)**

With the students table in the 6.3.1 again, the record modification is illustrated below:



A		ID	NAME	GENDER	AGE
1	=demo.query("select * from STUDENTS")	11	Matthew	M	17
2	>A1.modify(1,11:ID,"Matthew":NAME,"M":GENDER)	2	Elizabeth	F	16
3	>A1(4).modify(22,"Alyssa","F",18)	3	Sean	M	17
		22	Alyssa	F	18
		5	Michael	M	16
		6	John	M	13
		7	Nicholas	M	16

Again, we emphasize that it is forbidden to assign value to the members of table sequence by replacing them, for example,  $T(2)=r$  is illegal. In fact, there is no way to change the members of table sequence as a whole. You can only reassign values to its fields.

### ***T.modify(k:A,x<sub>i</sub>:F<sub>i</sub>,...)***

Operate against each member of *A* through looping execution to modify multiple records since the  $k^{\text{th}}$  record in *T* in batches.

A		ID	Name	Gender	Age
1	=esProc.query("select * from students")	1	Emily	F	17
2	>A1.modify(2:3, ID+10:ID)	12	Elizabeth	F	16
		13	Sean	M	17
		14	Lauren	F	15
		5	Michael	M	16
		6	John	M	13
		7	Nicholas	M	16

### **6.3.3 Reset a table sequence**

#### ***T.reset()***

Empty table sequence *T* while keeping its data structure.

### **6.3.4 Paste**

#### ***r.paste(r')***

Change the field value of the record *r* to the field value of the record *r'* according to its position.

- ❖ **@n** When modifying the record, only modify the values of fields in record *r* that have the same names as those in record *r'*. The case is sensitive when judging if the fields are of the same name.

STUDENTS			
ID	NAME	GENDER	AGE
1	Emily	F	17
2	Elizabeth	F	16
3	Sean	M	17
4	Lauren	F	15
5	Michael	M	16
6	John	M	13
7	Nicholas	M	16

SELLERS			
GROUPID	ID	NAME	GENDER
1	1	Ashley Williams	F
1	2	Andrew Miller	M
2	1	Sarah Johnson	F
2	2	Grace Johnson	F
2	3	Christina Williams	F

	A									
1	=demo.query("select * from STUDENTS")	<table><tr><th>ID</th><th>NAME</th><th>GENDER</th><th>AGE</th></tr><tr><td>2</td><td>2</td><td>Grace Johnson</td><td>F</td></tr></table>	ID	NAME	GENDER	AGE	2	2	Grace Johnson	F
ID	NAME	GENDER	AGE							
2	2	Grace Johnson	F							
2	=demo.query("select * from SELLERS")	<table><tr><th>ID</th><th>NAME</th><th>GENDER</th><th>AGE</th></tr><tr><td>2</td><td>Grace Johnson</td><td>F</td><td>17</td></tr></table>	ID	NAME	GENDER	AGE	2	Grace Johnson	F	17
ID	NAME	GENDER	AGE							
2	Grace Johnson	F	17							
3	=A1(1).paste(A2(4))									
4	=A1(3).paste@n(A2(4))									
5	=A1(5).paste(["p1","p2","p3"])	<table><tr><th>ID</th><th>NAME</th><th>GENDER</th><th>AGE</th></tr><tr><td>p1</td><td>p2</td><td>p3</td><td>16</td></tr></table>	ID	NAME	GENDER	AGE	p1	p2	p3	16
ID	NAME	GENDER	AGE							
p1	p2	p3	16							

Values of fields in red box are changed

### **P.paste(A)**

With the members of sequence A, paste the records in RSeq P in order. In most cases, A is an RSeq (not a must) and the members of A can also be normal sequences.

- ❖ **@n** Modify every record of P with the **@n** option.

	A													
1	=demo.query("select * from STUDENTS")	<table><tr><th>ID</th><th>NAME</th><th>GENDER</th><th>AGE</th></tr><tr><td>1</td><td>1</td><td>Ashley Williams</td><td>F</td></tr></table>	ID	NAME	GENDER	AGE	1	1	Ashley Williams	F				
ID	NAME	GENDER	AGE											
1	1	Ashley Williams	F											
2	=demo.query("select * from SELLERS")	<table><tr><th>ID</th><th>NAME</th><th>GENDER</th><th>AGE</th></tr><tr><td>1</td><td>2</td><td>Andrew Miller</td><td>M</td></tr><tr><td>2</td><td>1</td><td>Sarah Johnson</td><td>F</td></tr></table>	ID	NAME	GENDER	AGE	1	2	Andrew Miller	M	2	1	Sarah Johnson	F
ID	NAME	GENDER	AGE											
1	2	Andrew Miller	M											
2	1	Sarah Johnson	F											
3	=A1(to(1,3)).paste(A2)													
4	=A1(to(4,6)).paste@n(A2)													

ID	NAME	GENDER	AGE
1	Ashley William	F	15
2	Andrew Miller	M	16
1	Sarah Johnson	F	13

## **6.4 Primary Key**

### **6.4.1 Set Primary Key**

One or certain fields can be set as the primary key(s) of a TSeq. To simplify the code, some specific functions are provided to conduct the search on the basis of the primary key(s). By principle, **it is assumed** that the value(s) of the primary key(s) in the record of the TSeq is **unique**, but no compulsory check is set to validate it. No error will be reported even if any primary key values are repetitive. The primary key can be left unset, and in this case the primary key will be the first field. When *T.create()* is used to generate a new TSeq, the setting of *T*'s primary key(s) will be copied.

### **T.primary( *F<sub>i</sub>*,... )**

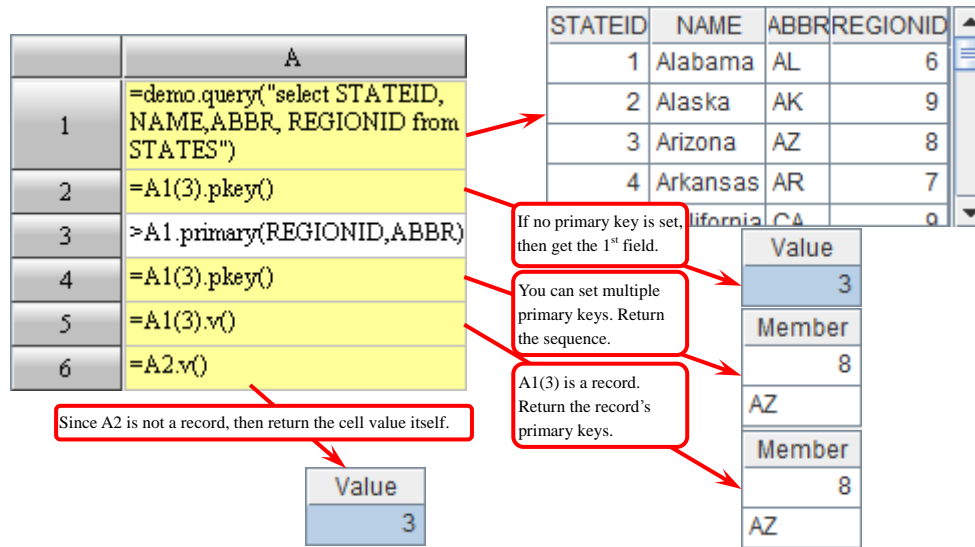
Set the primary keys of the TSeq *T* as *F<sub>i</sub>*,.... When *T.create()* is used to create an empty TSeq, what is about *T*'s primary keys will be copied.

### **r.pkey()**

Return a sequence composed of all primary key fields in the record. If the member of a primary key field is still a record, then this record will continue to execute **pkey()** and insert the results to the sequence. If merely one primary key exists, then only return the field member instead of the sequence.

### **v.v()**

If *v* is a record, then return field value of the record's primary key, or a sequence composed of the field values of primary keys. Otherwise, return *v* itself.



### T.index(n)

For the primary key of TSeq  $T$ , create an index table with the length of  $n$ . Without  $n$ , the length will be assigned automatically. If the primary-key-based search should be performed for several times, the efficiency will be improved with the index table created.

### 6.4.2 Usage of Primary Key

You can perform locating and pickout operations on the corresponding records with the primary key values.

#### A.pfind(v)

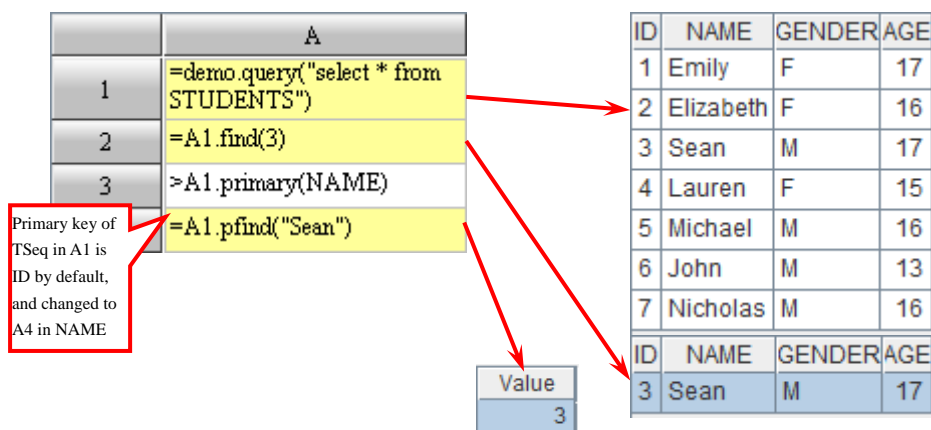
Return the position of record member in the sequence  $A$  whose primary key value is  $v$ , or the positions of non-record member in  $A$  whose value is  $v$ . If there are multiple primary keys, then  $v$  is a Sequence.

- ❖ @b If the members of  $A$  are ordered based on its primary key, then use the binary search to locate them.
- ❖ @bs The members of  $A$  are ordered based on its primary key. With binary search, the position will be returned if  $v$  can be found; If  $v$  is not the primary key value of members of  $A$  or the member value, return the opposite number of the sequence number at which position the  $v$  can be inserted orderly.

#### A.find(v)

Return the record member whose primary key value is  $v$  or the non-record member whose value is  $v$  in the sequence  $A$ . If there are multiple primary keys, then  $v$  is an RSeq. If there is an index table for  $A$ , then use it for searching.

- ❖ @b If the member of  $A$  is ordered against its primary key, then use the binary search and ignore the index table of  $A$ .



With the primary key, the role of some functions can be affected, such as **append@p**. For details, please refer to the descriptions in the Section 7.1.3 **Append Record**.

### 6.4.3 Record Conversion and switch() Function

With **switch()** function, you can swap it with the record  $r$ , and convert the value of one or several fields in  $r$  to the record reference in RSeq  $P$  with these field values as the primary keys.

#### **P.switch( $F_i, A_i; x; \dots$ )**

Assign the fields  $F_i$  of each record of RSeq  $P$  with  $A_i$ . **select@1( $x:F_i$ )**, which indicates the references of the record whose primary key is the  $F_i$  in  $A_i$ . If there is an index table for  $A$ , then use it for searching. When  $x$  is omitted, perform searching according to  $A_i$ 's primary key; When  $A_i$  is omitted too, field  $F_i$  gets assigned with value  $F_i.v()$ . Specifically, use the sequence number to locate if  $x$  is #. If no corresponding records are found in  $A_i$ , assign the related field with null.

- ❖ **@i** In computing, if no corresponding value in  $F_i$  is found for each  $A_i$  against a record in  $P$ , then remove the record.

With **switch()** function, you can implement the record reference and assignment much easier:

- **A.primary(ID)**
- **B.switch(Salesperson,A)**



STATEID	NAME	ABBR
1	Alabama	AL
2	Alaska	AK
3	Arizona	AZ
4	Arkansas	AR
5	California	CA

CID	NAME	POPULATION	STATE
1	New York	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

A
=demo.query("select STATEID, NAME, ABBR from STATES")
=demo.query("select CID, NAME, POPULATION, STATEID as STATE from CITIES")
>A2.switch(STATE,A1)

CID	NAME	POPULATION	STATE
1	New York	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

STATEID	NAME	ABBR
13	Illinois	IL

After executing the switch function in A3

Double click to view the records

With **switch()** function, you can handle some rather complicated situations, such as changing field value simultaneously in multiple fields:

STATEID	NAME	ABBR
1	Alabama	AL
2	Alaska	AK
3	Arizona	AZ
4	Arkansas	AR
5	California	CA

ID	UNIT	DETAIL
1	LB	Pound
2	BAG	Bag
3	EACH	Each
4	KG	Kilogram
5	CRATE	Crate

GROUPID	ID	NAME	GENDER
1	1	Ashley	F
1	2	Andrew	M
2	1	Sarah	F
2	2	Grace	F
2	3	Christina	F

A	B	C
=demo.query("select STATEID, NAME, ABBR from STATES")	=demo.query("select * from UNIT")	=demo.query("select * from SELLERS")
	>B1.primary(UNIT)	>C1.primary(NAME)
=demo.query("select * from FOODS")		
>A3.switch(UNIT, B1:ID; PLACE, A1:ABBR; SELLERS, C1)		

ID	NAME	PLACE	UNIT	UNITP...	SELLERS
1	Apple	22	LB	1.69	Ashley W
2	Banana	9	LB	0.69	Andrew M
3	Cucumber	5	EACH	0.77	Christina
4	Onion	5	LB	0.99	Christina
5	Orange	9	BAG	4.99	Ashley W

STATEID	NAME	ABBR
5	California	CA

ID	UNIT	DETAIL
1	LB	Pound

GROUPID	ID	NAME	GENDER
2	3	Christina	F

After executing switch function in A4

## 7. Data Association

### 7.1 Data Generation

#### 7.1.1 Create a New TSeq

➤ **A.new( $x_i; F_i, \dots$ )**

A is a sequence or an RSeq. Firstly, create a TSeq whose field is  $F_i, \dots$ . Then, loop every member in A through the expression  $x_i$  and insert the new records to the TSeq.

	A		Value	Square	Cube
1	=to(5).new(~:Value,~*:Square,~*:~*:Cube)		1	1	1
			2	4	8
			3	9	27
			4	16	64
			5	25	125

With **A.new()** function, you can use the existing RSeq data to generate a new TSeq:

STATEID	NAME	POPULATION	ABBR	AREA	CAPITAL	REGIONID
1	Alabama	4779736	AL	52419	Montgomery	6
2	Alaska	710231	AK	663267	Juneau	9
3	Arizona	6392017	AZ	113998	Phoenix	8
4	Arkansas	2915918	AR	52897	Little Rock	7
5	California	37253956	CA	163700	Sacramento	9
6	Colorado	5029196	CO	104185	Denver	8

	A		ID	NAME	POPULATION	AREA
1	=demo.query("select * from STATES")		CA	California	37253956	163700
2	=A1.sort(POPULATION:-1)		TX	Texas	25145561	268820
3	=A2(to(1,5)).new(ABBR:ID, NAME,POPULATION,AREA)		NY	New York	19378102	54556
			FL	Florida	18801310	65755
			IL	Illinois	12830632	54826

Get information of states whose population to be ranked in the top 5, and create a new TSeq

➤ **A.regex( $rs, F_i, \dots$ )**

Execute **regex( $rs$ )** on each member of string sequence A, and combine the matching results into a new TSeq whose field name is  $F_i, \dots$

- ❖ @i Not case-sensitive.
- ❖ @u Use unicode to match.

Using **A.regex()** function, you can split a string sequence to create a new TSeq. For example:

	A	B	C	D
1	Oliver Twist	Noah Claypole	Rose Maylie	Toby Crackit
2	=A1:D1			
3	=A2.regex("(\\S*) (\\S*)",FirstName,LastName)			

In A3, the name sequence is split into FirstName and LastName which are separated by a blank. The computed results in A2 and A3 are as follows:



Member	FirstName	LastName
Oliver Twist	Oliver	Twist
Noah Claypole	Noah	Claypole
Rose Maylie	Rose	Maylie
Toby Crackit	Toby	Crackit

When a regular expression cannot match with a certain string, records won't be created. According to this rule, we can filter sequences and select specified strings to create a TSeq. For example:

	A
1	=demo.query("select NAME,STATE,SALARY from EMPLOYEE")
2	=A1.(~.string())
3	=A2.regex("(B\\S*),(\\S*),\\S*",Name,State,Salary)
4	=A2.regex("(\\S*),([C T].*),\\S*",Name,State,Salary)

The string sequence created in A2 is as follows:

Member
Rebecca,California,7000
Ashley,New York,11000
Rachel,New Mexico,9000
Emily,Texas,7000
Ashley,Texas,16000
Matthew,California,11000
Alexis,Illinois,9000

In A3, names of employee, states where the employees are and their salary are matched according to strings. The point . in the regular expression represents a character which is neither a line break nor a carriage return. The matching criterion requires employee names starting with B. The result of A3 is as follows:

Name	State	Salary
Brandon	Pennsylvania	7000
Brandon	New York	10000
Brianna	Georgia	8000
Bryan	Illinois	12000
Brandon	Pennsylvania	6500
Benjamin	Florida	5000
Rebecca	New York	12000

Still, in A4, names of employee, states where the employees are and their salary are matched in order to create a TSeq. The matching criterion requires that the states being matched start with C or T. The computed result is as follows:

Name	State	Salary
Rebecca	California	7000
Emily	Texas	7000
Ashley	Texas	16000
Matthew	California	11000
Megan	California	11000
Victoria	Texas	3000
Joseph	Texas	12000



### 7.1.2 TSeq Duplication

As we mentioned in the Section 3.5.1 **Generate the subsequence**, the **A.dup()** function can be used to copy a sequence. A TSeq is actually a sequence. When copying a sequence, you need to note several points.

#### **T.dup()**

Take and copy *T* as a sequence. Please note that the result is an RSeq composed of records in *T*, instead of a TSeq.

- ❖ **@t** When using **dup()** function in the TSeq, you can use the **@t** option to generate a TSeq. The records from the original TSeq will be copied to the new TSeq one by one and new records in the new Tseq will be generated.

	A	
1	=demo.query("select * from CITIES")	
2	=A1.dup()	
3	=A1.dup@t()	
4	=ift(A2)	
5	=ift(A3)	

CID	NAME	POPULATION	STATEID
1	New York	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

Value
false
Value
true

**Note:** If the **T.dup()** is used without **@t** option, then the record of new RSeq is only the reference of the record of the original TSeq. The record itself is not copied.

	A	
1	=demo.query("select * from CITIES")	
2	=A1.dup()	
3	=A1.dup@t()	
4	>A2(1).modify("DAVID1".NAME)	
5	>A3(2).modify("DAVID2".NAME)	
6	=A1	

In A2, since **T.dup()** is used alone without **@t** option, only record references occur in A2. When A2 is changed in A4, then the record in the original TSeq is also changed.

CID	NAME	POPULATION	STATEID
1	DAVID1	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

CID	NAME	POPULATION	STATEID
1	New York	8084316	32
2	DAVID2	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

CID	NAME	POPULATION	STATEID
1	DAVID1	8084316	32
2	Los Angeles	3798981	5
3	Chicago	2886251	13
4	Houston	2009834	43
5	Philadelphia	1402221	28

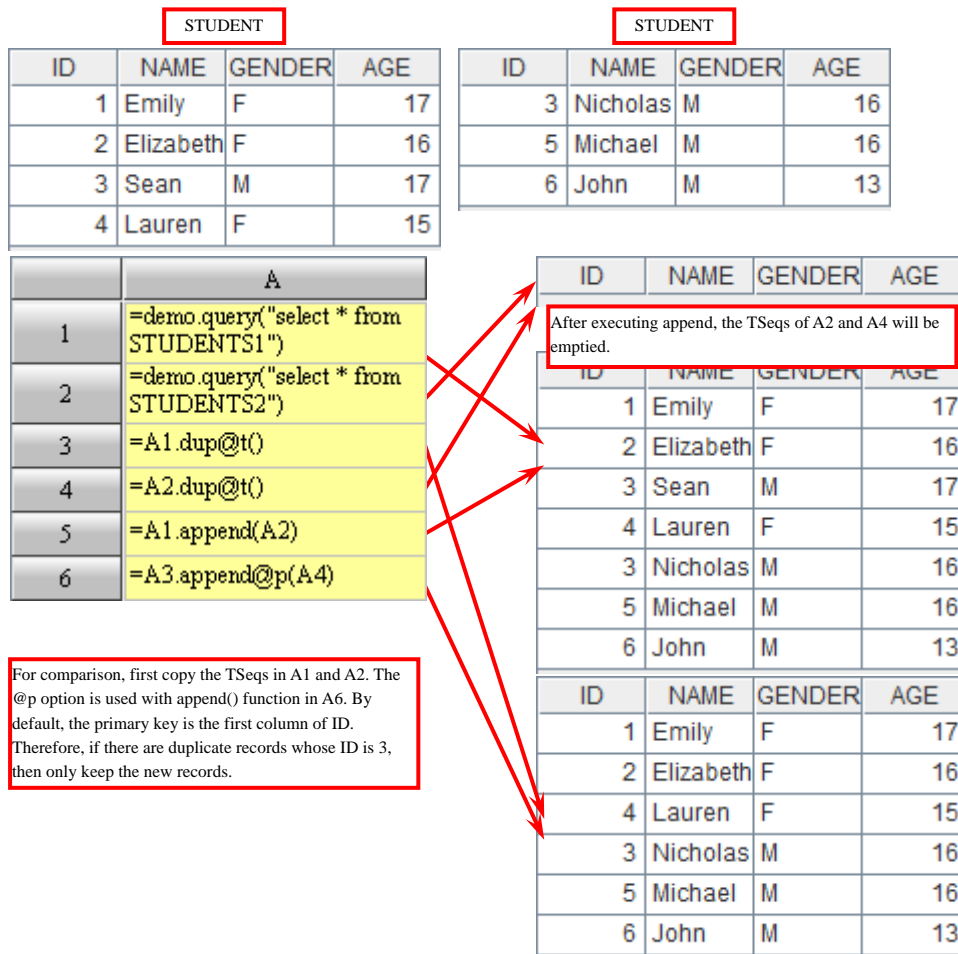
### 7.1.3 Append Records

#### **T.append(T<sub>i</sub>,...)**

*T* and *T<sub>i</sub>,...* are the TSeqs with the same number of fields. Append the records from *T<sub>i</sub>,...* to *T*, and empty all records in the TSeq *T<sub>i</sub>,...*



❖ **@p** During appending records with the **@p** option, if there are records to be appended having the same primary key as that of some of the original records, then these original records will be replaced by those to-be-appended ones.



### 7.1.4 Split Record

#### A.news(...)

Based on the expression ..., the operation can be performed that each record in the RSeq or TSeq A is split into a RSeq or a Sequence, then return the merged members in all the Sequences or merged records in all the RSeqs. In the end, the returned result is usually a Sequence or an RSeq.

For example, the scores of a women's individual gymnastics all-around are stored in a database table named GYMNASTICSWOMEN:

ID	NAME	COUNTRY	VAULT	UNEVENBARS	BALANCEBEAM	FLOOR
1	Ana Ailva	BRA	14.175	14.175	14.175	14.35
2	Anna Pavlor	RUS	15.275	14.525	15.975	15.05
3	Ariella Kask	SUI	15.35	14.275	14.425	13.95
4	Barbara Keesling	SUI	14.525	14.125	15.075	14.3
5	Becky Down	GBR	15.025	15.625	14.7	14.1
6	Beth Tweddle	GBR	14.4	15.675	14.8	15.1

Every athlete's score for each game is computed as below:

A		成员	
1	\$select * from GYMNASTICSWOMEN		
2	=A1.news([NAME,"Vault",VAULT],[NAME,"UnevenBars",UNEVENBARS],[NAME,"BalanceBeam",BALANCEBEAM],[NAME,"Floor",FLOOR]))	[Becky Down, Vault, 15.025]	
		[Becky Down, UnevenBars, 15.625]	
		[Becky Down, BalanceBeam, 14.7]	
		[Becky Down, Floor, 14.1]	
		[Beth Tweddle, Vault, 14.4]	
		[Beth Tweddle, UnevenBars, 15.675]	
		[Beth Tweddle, BalanceBeam, 14.8]	

You can see that in the parameters of **A.news()** function, the expression splits each record of the original TSeq into a Sequence, and stores in each sequence the score for each event. The final returned result is a sequence of sequence in which any event score of every athlete is shown.

To list the results in a clear way, a TSeq will be generated for each record:

A				
1	\$select * from GYMNASTICSWOMEN			
2	=A1.news(create(Name,Event,Score).record([NAME,"Vault",VAULT,NAME,"UnevenBars",UNEVENBARS,NAME,"BalanceBeam",BALANCEBEAM,NAME,"Floor",FLOOR]))	Name	Event	Score
		Becky Down	Vault	15.025
		Becky Down	UnevenBars	15.625
		Becky Down	BalanceBeam	14.7
		Becky Down	Floor	14.1
		Beth Tweddle	Vault	14.4
		Beth Tweddle	UnevenBars	15.675
		Beth Tweddle	BalanceBeam	14.8

At this point, when **A.news()** function is performed, the records in all TSeqs will be merged, and then return a RSeq.

## 7.2 Group and Summarize Data

### 7.2.1 Compute Unique Value

#### **A.id(x)**

Compute expression *x* on every member in the sequence *A*, sort the sequence composed of the results, keep only one of the neighboring members with the same value, and then return the sequence. If *x* is omitted, then sort the members of *A*, keep only one of the neighboring members with the same value, and return the sequence.

- ❖ **@o** No sorting. Keep only one of the neighboring members of the same value directly, and there could be members of the same value in the result.

The score sheet of a gymnastics competition is as follows:

ID	NAME	EVENT	SCORE
1	Ana Silva	Vault	14.175
2	Berky Downie	Vault	15.025
3	Ariella Kaslin	Vault	15.35
4	Ana Silva	UnevenBars	14.175
5	Ariella Kaslin	UnevenBars	14.275
6	Berky Downie	UnevenBars	15.625
7	Berky Downie	BalanceBeam	14.7
8	Ana Silva	BalanceBeam	14.175
9	Ariella Kaslin	BalanceBeam	14.425
10	Ariella Kaslin	Floor	13.95
11	Ana Silva	Floor	14.35
12	Berky Downie	Floor	14.1

Let's check the execution of **id()** function:

	A
1	=demo.query("select * from GYMSCORE")
2	=A1.id(NAME)
3	=A1.id@o(NAME)

Compute the unique value of name.  
If no option, then the result will be sorted in ascending order.

@o option is to remove the neighboring member with the same value, and duplicate may exist.

Member
Ana Silva
Ariella Kaslin
Berky Downie

Member
Ana Silva
Berky Downie
Ariella Kaslin
Ana Silva
Ariella Kaslin

The working of **id** function is similar to the **distinct** operation of SQL.

### A.rank@i(y,x)

Compute the expression  $x$  on each member of the sequence  $A$ , remove the duplicate results and then compute the ranking of  $y$ .

### A.rank@i(x)

Compute the distinct ranking sequence of expression  $x$  on each member of sequence  $A$ .

	A
1	=demo.query("select * from GYMSCORE")
2	=A1.rank@i(15.5,SCORE)
3	=A1.rank@i(SCORE)

Ranking of 15.5points

Compute the ranking of score in each record; note that there are 3 14.175 points ranking 8th, and the 14.1 point still ranks 9th

值
2

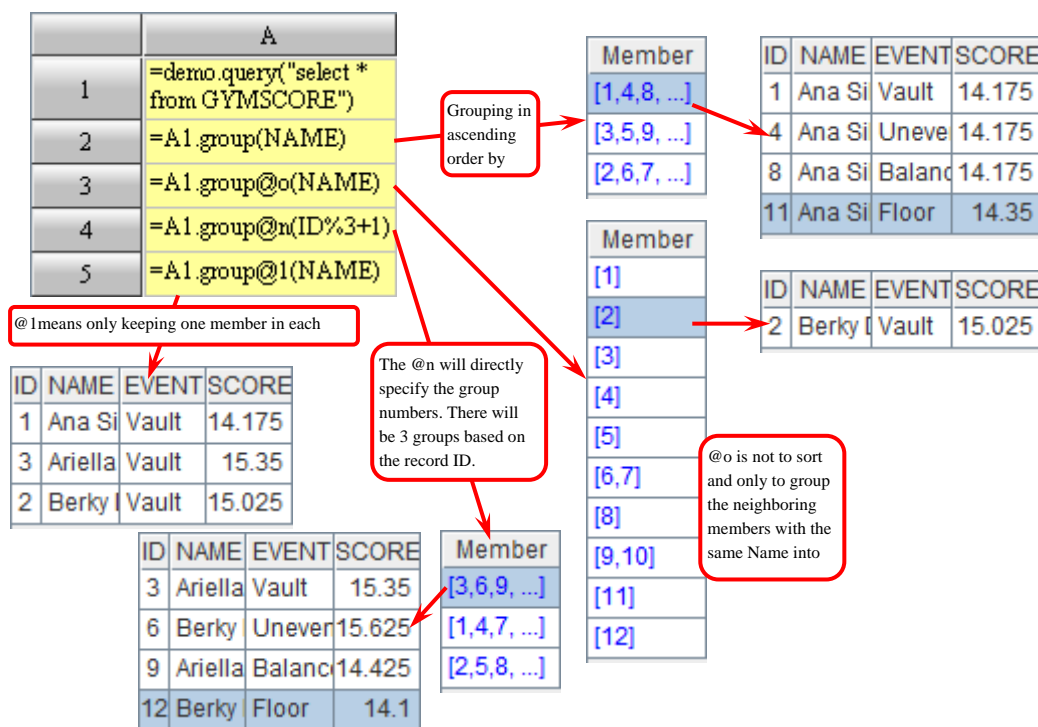
成员
8
3
2
8
7
1
4
8
5
10
6
9

## 7.2.2 Equal Group

### **A.group( $x_i...$ )**

Group sequence A by the result of expression  $x_i...$  in ascending order, and return the sequence composed of each group.

- ❖ **@o** No sorting on values of  $x_i...$ . Collect the neighboring  $x_i...$  members of the same value to a group.
- ❖ **@n** The result of expression  $x_i...$  is an integer; as a group number, it is for direct locating. This option and **@o** are mutually exclusive.
- ❖ **@1** Only keep the first member for each group.



The **group** function works as the **GROUP BY** operation in SQL. The return value of **group** function is a sequence consisting of subgroups. Each subgroup is a sub-sequence of the original sequence (or the position ISeq in the original sequence). The final result is a sequence consisting of sequences. The group summary value can be further computed for many times based on group result.

There is no explicit set data type in SQL and the group result will not be saved. You must compute the group summary value immediately after **GROUP BY** operation. After computation, the group results will be discarded forever.

In esProc, you can summarize the group results after grouping:



	A	
1	=demo.query("select * from GYMSCORE")	NAME Sum
2	=A1.group(NAME)	Ana Silva 56.875
3	=A2.new(NAME,round(~.sum(SCORE),3):Sum)	Ariella Kaslin 58.0
4	=A2.new(NAME,~.max(SCORE):Max,~.min(SCORE):Min)	Berky Downie 59.45
		NAME Max Min
		Ana Silva 14.35 14.175
		Ariella Kaslin 15.35 13.95
		Berky Downie 15.625 14.1

It reveals that cells **A3** and **A4** have made group summary twice towards the same group result in **A2**. The grouping result can be reused, which is one main feature of esProc.

Let's jump to **=A2.new(NAME,round(~.sum(SCORE),3):Sum)**. Every member of **A2** is a set of records (i.e. RSeq) since **A2** is a grouping result. Therefore, when **new()** function is being used to perform loop on **A2**, its internal expression is operating on the RSeq. For example, **NAME** means getting the value of the Name field of current member's first record. **~.sum(SCORE)** means summing up the Score field of the current RSeq.

As the **GROUP BY** in SQL, **group** function also supports grouping more than one field(expression) at the same time:

	A	
1	=demo.query("select * from SELLERS")	GROUPID ID NAME GENDER
2	>A1.primary(NAME)	1 1 Ashley F
3	=A1.group(GROUPID, GENDER)	GROUPID ID NAME GENDER
		1 2 Andrew M
		GROUPID ID NAME GENDER
		2 1 Sarah F
		2 2 Grace F
		2 3 Christi F

### 7.2.3 Group Summarization

esProc also provides functions to directly compute the group summary value:

#### A.groups(x:F,...;y:G,...)

Group and summarize the data in sequence **A** by the computed result of **x,...**. The result will be aggregated into the result set to return. The process equals to **A.group(x,...).new(x:F,...;y:G,...)**. In aggregating, members of **A** will not be sorted. Instead, every member of it will be computed one by one in a loop, and match the result of **x,...** in specified order; You are only allowed to use simple converging functions **sum/count/max/min** in the expression **y**. Once computation is completed, a new TSeq will be returned with the fields **F,...,G,...**, in which field **F,...** is the primary key.

- ❖ **@o** Grouping with @o option.
- ❖ **@n** The computed result of **x** is the sequence number of a group for direct locating in aggregating.



	A		Dept	Count	TotalSalary
1	=file("D:/files/txt/employee.txt")		Administration	4	40000
2	=A1.import@t()		Finance	24	177500
3	=A2.groups(DEPT:Dept;count(~):Count,sum(SALARY):TotalSalary)		HR	19	138000
			Marketing	99	733500
			Production	91	663000
			R&D	29	239000
			Sales	187	1362500
			Technology	47	344000

## 7.3 Alignment and Enumeration Grouping

### 7.3.1 Alignment

#### P.align(A:x,y)

Align members of sequence *P* with members in sequence *A* in a one-to-one relation. The results of expression *y* for members of *P* equal to the results of expression *x* for corresponding members of *A*. Omitting *y* indicates that the member value of *P* itself will be used, and omitting *x* indicates that the member value of *A* itself will be used. By default, only the first member of *P* is reserved for each result.

Similar to equal grouping, alignment grouping usually uses the following options for various settings on grouping:

- ❖ **@a** Alignment grouping. In aligning, retrieve all members from *P* meeting the specified conditions. Like **group** function, the return value of **align@a** function is also a sequence of the grouping sequence.
- ❖ **@b** *A* is ordered. The binary search can be used to increase efficiency.
- ❖ **@p** Return a sequence composed of sequence numbers of members from each group.

	A		Member		ID	NAME	EVENT	SCORE
1	=demo.query("select * from GYMSCORE")		[7,8,9]		10	Ariella	Floor	13.95
2	[BalanceBeam,UnevenBars,Floor,Vault]		[4,5,6]		11	Ana Sil	Floor	14.35
3	=A1.align@a(A2,EVENT)		[10,11,12]		12	Berky	Floor	14.1
4	=A1.align@a(A2:~,EVENT)		[1,2,3]					

Results of A3 and A4 are the same

After alignment grouping is executed, you can associate the grouping result sequence **A3** with **A2** by sequence numbers, and make the statistics, for example:

	A	
1	=demo.query("select * from GYMSCORE")	
2	[BalanceBeam, UnevenBars, Floor, Vault]	
3	=A1.align@a(A2, EVENT)	
4	=A1.align@a(A2:~, EVENT)	
5	=A3.new(A2(#):Event, round(~.avg(SCORE),3):AvgScore)	

Event	AvgScore
BalanceBeam	14.433
UnevenBars	14.692
Floor	14.133
Vault	14.85

Let's talk about expression **=A3.new(A2(#):EVENT, round(avg(SCORE),3):AvgScore)**. It represents performing computation on each member in A3 by loop and generating corresponding records for a new TSeq. Since **A3** and **A2** have the same order, the sequence numbers of their members are completely identical. Therefore, you can get the member of **A2** with the same position through **A2(#)**, here **#** refers to the current sequence numbers of **A3**'s members. The expression is equal to another one: **=A3.new(EVENT:Event, round(~.avg(SCORE),3):AvgScore)**.

- ❖ **@n** **@a** option is used in computation. Among the results, an extra set will be generated, which can be used to store members of *P* that cannot find their counterparts in *A*.
- ❖ **@s** Members of the sequence *P* are sorted according to *A*'s order, where the non-corresponding member with *A* is aligned in the last row.

	A	Member
1	=demo.query("select * from GYMSCORE")	[7,8,9]
2	[BalanceBeam, UnevenBars, Floor]	[4,5,6]
3	=A1.align@a(A2, EVENT)	[10,11,12]
4	=A1.align@a(A2:~, EVENT)	[7,8,9]
5	=A1.align@s(A2, EVENT)	[4,5,6]
		[10,11,12]
		[1,2,3]

A4 uses @n option, and one more group is produced than A3 to store Vault item which doesn't exist in A2

ID	NAME	EVENT	SCORE
1	Ana Silva	Vault	14.175
2	Berky Down	Vault	15.025
3	Ariella Ka	Vault	15.35

With @s option, the athlete's scores are sorted in A5 according to A2's order, namely, the records of each group in A4 are joined in turn

ID	NAME	EVENT	SCORE
7	Berky Downie	BalanceBeam	14.7
8	Ana Silva	BalanceBeam	14.175
9	Ariella Kaslin	BalanceBeam	14.425
4	Ana Silva	UnevenBars	14.175
5	Ariella Kaslin	UnevenBars	14.275
6	Berky Downie	UnevenBars	15.625
10	Ariella Kaslin	Floor	13.95

An easier direct sequence number alignment:

### **P.align(n,y)**

**P.align(to(n),y)**, which means the value of *y* is already the sequence number for alignment grouping.

To use the sequence number to create alignment, the grouping fields should be consecutive integers. In this case, you can use the **P.align(n,y)** to uplift performance. **@a** and **@p** options can



also be used together with sequence number alignment.

- ❖ **@r** The computed result of  $y$  is not a sequence number, but a sequence of sequence numbers. In this case, a record in  $P$  is allowed to appear repeatedly.

A		Member	EID	NAME	GENDER	BIRTHDAY
1	=demo.query("select EID,NAME, GENDER,BIRTHDAY from EMPLOYEE")	[71,84,172, ...]	76	Tyler	M	1972-02-06
2	=A1.align@a(12,month(A1.BIRTHDAY))	[76,85,87, ...]	85	Hailey	F	1977-02-10
		[4,10,18, ...]	87	Sarah	F	1972-02-16
		[8,31,95, ...]	100	Jacob	M	1978-02-12
		[5,12,22, ...]	122	Ashley	F	1987-02-08

Group by month of birth

When aligning with the sequence number, you can also generate consecutive integers with the locate function.

A		Member	ID	NAME	EVENT	SCORE
1	=demo.query("select * from GYMSCORE")	[7,8,9]	7	Berky Dow	BalanceBeam	14.7
2	[BalanceBeam, UnevenBars, Floor, Vault]	[4,5,6]	8	Ana Silva	BalanceBeam	14.175
3	=A1.align@a(4,A2.pos(EVENT))	[10,11,12]	9	Ariella Kas	BalanceBeam	14.425
		[1,2,3]				

### 7.3.2 Enumeration Grouping

**eval(x,a)**

Compute the expression in the string  $x$  with argument  $a$ . In the  $x$ , use the  $i^{\text{th}}$  ? to represent the  $i^{\text{th}}$  parameter or represent the parameter with ? $i$  by pointing out its position. When the number of parameter is less than "?", you can start the loop from the first member.

- **eval("'+3",5)** Return 8
- **eval("'>3",5)** Return true
- **eval("'"+3\*?"',5,7)** Return result of  $5+3*7$ , that is, 26
- **eval("'"+3\*?\*?"',5,7)** Return result of  $5+3*7+5$ , that is, 31

In particular, if **eval(x,a)** returns true, then  $a$  meets the condition  $x$ .

**E.penum(y)**

$E$  is a sequence composed of the condition expression. Compute expression  $y$ , find members in  $E$  requiring condition which the result matches and return the sequence number of the first member that can be found. In particular, the result is regarded as eligible if the condition is null.

A		B	Rank	Score
1	A	?>85	A	?>85
2	B	?>70	B	?>70
3	C	?>60	C	?>60
4	=create(Rank,Score).record([A1:B3])			
5	=A4.(Score).penum(77)			
6	=B1:B3.penum(77)			

Return 2 since 77 meets the condition the

Value
2

- ❖ **@r** Conditions can be overlapping return the ISeq composed of sequence numbers of all members whose condition matches with the result. Only when the result cannot satisfy

conditions of all the members, return the ISeq composed of sequence numbers of the members whose condition is null.

	A	B
1	A	?>85
2	B	?>70
3	C	?>60
4	D	null
5	=create(Rank,Score).record([A1:B4])	
6	=A5.(Score).penum@r(77)	
7	=[B1:B4].penum@r(77)	
8	=[B1:B4].penum@r(55)	

Rank	Score
A	?>85
B	?>70
C	?>60
D	

Member
2
3

Member
4

77 can meet the conditions of 2 and 3, return an ISeq

55 cannot meet the first 3 conditions, return the position of null condition: 4

- ❖ **@n** If any condition is not met, return  $E.len()+1$ . This and **@r** option are mutually exclusive. Now the *null* condition is not required.

	A	B
1	A	?>85
2	B	?>70
3	C	?>60
4	=create(Rank,Score).record([A1:B3])	
5	=[B1:B3].penum@n(55)	

Rank	Score
A	?>85
B	?>70
C	?>60

Value
4

55 can't meet all the 3 conditions, return 4

### **P.enum(E,y)**

Allocate the member of the sequence *P* to several groups in one-to-one corresponding relation to members of RSeq *E*. Group members of sequence *P* and create a one-to-one correspondence between these groups and members of *E*, the condition expression sequence. The results of expression *y* for members of each group shall satisfy the conditions required by the corresponding members of *E*. If omitting *y*, then you use the value of members of *P* itself. By default, every member in *P* will only be placed in the first group meeting the conditions.

Similarly, the return value of **enum** function is also a sequence of sequence.

	A
1	=demo.query("select * from GYMSCORE")
2	[?>15,?>14.5,?>14]
3	=A1.enum(A2,SCORE)

Group by score, for example, Group 1 satisfies the condition that SCORE > 15

Member	ID	NAME	EVENT	SCORE
[2,3,6]	2	Berky Downie	Vault	15.025
[7]	3	Ariella Kaslin	Vault	15.35
[1,4,5, ...]	6	Berky Downie	Unever	15.625

When  $E.m(-1)=null$ , that is, the result of the expression is eligible when compared with the null condition, **enum** function will gather all members from *P* not satisfying any other condition fields in *E* into a group corresponding to  $E.m(-1)$ , which is equivalent to the "others" group.



	A	Member	
1	=demo.query("select * from GYMSCORE")	[2,3,6]	
2	[?>15, ?>14.5, ?>14, null]	[7]	
3	=A1.enum(A2, SCORE)	[1,4,5, ...]	
		[10]	

ID	NAME	EVENT	SCORE
10	Ariella Kaslin	Floor	13.95

Records not meeting any conditions will be placed in the last group corresponding to the null.

By default, **enum** function assumes there are no overlapping groups during the process of computing, that is, the member in *P* won't match the results of two condition expressions at the same time.

- ❖ **@r** Allow overlapping groups, that is, the member in *P* may correspond to multiple members in *E* simultaneously.

	A	Member	
1	=demo.query("select * from GYMSCORE")	[2,3,6]	
2	[?>15, ?>14.5, ?>14, null]	[2,3,6, ...]	
3	=A1.enum@r(A2, SCORE)	[1,2,3, ...]	
		[10]	

ID	NAME	EVENT	SCORE
2	Berky D	Vault	15.025
3	Ariella k	Vault	15.35
6	Berky D	Unever	15.625
7	Berky D	Balance	14.7

Records that meet SCORE>15 also meet the condition of SCORE>14.5. With @r option, the result will also appear in the 2<sup>nd</sup> group.

ID	NAME	EVENT	SCORE
10	Ariella k	Floor	13.95

With @r option, only the records not meeting any conditions will be placed in the group corresponding to null

The following options are often used with the **enum()** function to perform various grouping:

- ❖ **@p** Return a sequence composed of ISeqs of the sequence numbers of members in each group
- ❖ **@n** When all conditions cannot be satisfied, return **E.len()+1**. This and @r option are mutually exclusive. *Null* condition is not required now.

	A	Member	
1	=demo.query("select * from GYMSCORE")	[2,3,6]	
2	[?>15, ?>14.5, ?>14]	[7]	
3	=A1.enum@n(A2, SCORE)	[1,4,5, ...]	
		[10]	

ID	NAME	EVENT	SCORE
2	Berky Downie	Vault	15.025
3	Ariella Kaslin	Vault	15.35
6	Berky Downie	Unever	15.625

ID	NAME	EVENT	SCORE
10	Ariella k	Floor	13.95

With @n option, any record which does not meet any condition is stored in a new group without null condition

## 7.4 Join

esProc also provides the join operation like SQL does, but this kind of operation has become not common since esProc began to provide the reference mechanism for records and the RSeq, as well as **align** and other alike functions.

### 7.4.1 Equal Join

**join**(*A<sub>i</sub>*; *F<sub>i</sub>*; *x<sub>j</sub>*; ...; ...)

Join Sequence *A<sub>i</sub>*, ... on the condition that the corresponding fields (expressions) *x<sub>j</sub>*, ... has the same value, and generate a TSeq with *F<sub>i</sub>*, ... as field to save the results. *F<sub>i</sub>* takes the member of *A<sub>i</sub>* as its value.



A	
1	=demo.query("select * from STATENAME")
2	=demo.query("select * from STATECAPITAL")
3	=demo.query("select * from STATEINFO")
4	=join(A1.Name,STATEID; A2:Capital,STATEID;A3:Info,STATEID)

Name	Capital	Info
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

STATEID	NAME	ABBR
1	Alabama	AL
2	Alaska	AK
3	Arizona	AZ
4	Arkansas	AR
5	California	CA

STATEID	CAPITAL
1	Montgomery
2	Juneau
3	Phoenix
4	Little Rock
5	Sacramento

STATEID	POPULATION	AREA
1	4779736	52419
2	710231	663267
3	6392017	113998
4	2915918	52897
5	27252056	162700

**join** function approximately acts as the following statements in SQL:

- **select  $A_i.*$  as  $F_i...$  from  $A_i...$  where  $A_i.x_j = ...$  and ...**

This is also the most common writing style of join in SQL.

The **join** function has three common options as follows:

- ❖ **@f** Full join. Every member of  $A_i$  appears at least once. Use null if there is no corresponding value.
- ❖ **@1** Left join. Every member of  $A_1$  appears at least once. Use null if there is no corresponding value. Note: the 1 in the option is a digit instead of the letter l. esProc does not use the letter l in an option.

A	
1	=demo.query("select * from STATENAME").step(2,1)
2	=demo.query("select * from STATECAPITAL").step(2,2)
3	=demo.query("select * from STATEINFO").step(3,1)
4	=join@f(A1.Name,STATEID; A2:Capital,STATEID;A3:Info,STATEID)
5	=join@1(A1.Name,STATEID; A2:Capital,STATEID;A3:Info,STATEID)

Name	Capital	Info
1		1
3		
5		
7		7
9		

Name	Capital	Info
1		1
	2	
3		
	4	4
5		

STATEID	NAME	ABBR
1	Alabama	AL
3	Arizona	AZ
5	California	CA
7	Connecticut	CT
9	Florida	FL

STATEID	CAPITAL
2	Juneau
4	Little Rock
6	Denver
8	Dover
10	Atlanta

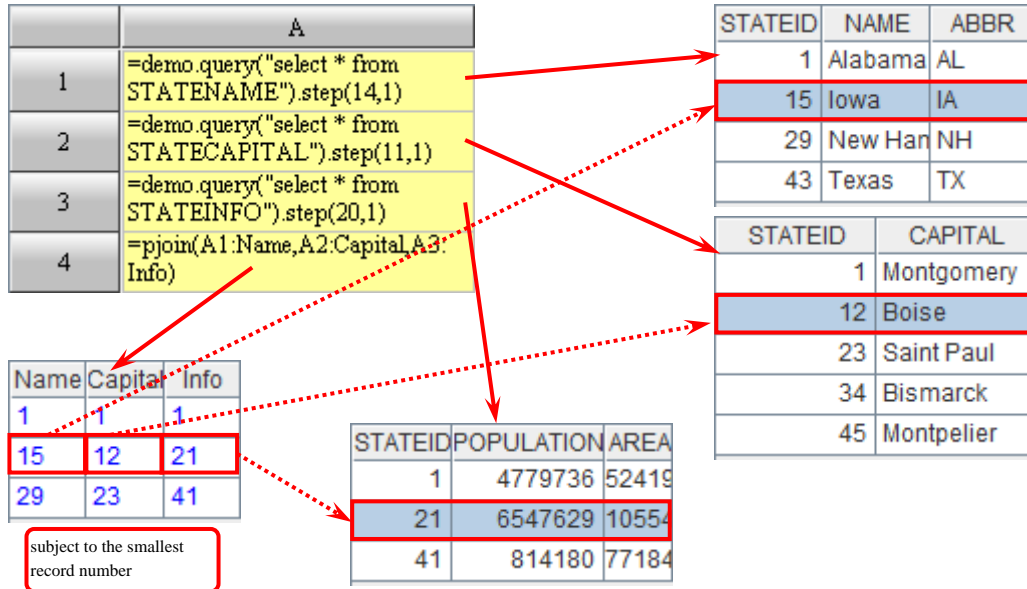
  

STATEID	POPULATION	AREA
1	4779736	52419
4	2915918	52897
7	3574097	5543
10	9687653	59425
12	12920622	54826

### 7.4.2 Apposition Join

**pjoin( $x_i:F_i,\dots$ )**

$x_i$  is a sequence; generate a TSeq with  $F_i,\dots$  as field, in which  $F_i$  of the record  $k^{\text{th}}$  gets assigned with  $x_i(k)$ , that is, joining members of sequence  $x_i,\dots$  in order, with a length subject to the smallest one,  $x_i.\text{len}()$ , and then return the TSeq.



When members of a record sequence are joined in apposition, the joining order conforms to the order of the records and the field stores the records from the original table.

## 7.5 Application of Data Association

The data association is used widely. Regarding data analysis, data preparation, and other data process work, you will need to perform grouping, summarizing, joining, and other operations related to data association.

### 7.5.1 Group and Collect Statistics on Wine Inventory according to Requirements

The LIQUORS table in the database stores various data on liquors, such as the name, type, origin, inventory, and other information:

LID	NAME	TYPE	PRODUCTION	STOCK
1	42Below Vodka	Vodka	New Zealand	301
2	Absolut Vodka	Vodka	Sweden	95
3	Appleton Estate Reserve	Rum	Jamaica	202
4	Bacardi Superior	Rum	Puerto Rico	741
5	Baileys Irish Cream	Cordials	Ireland	434

- Collect statistics on the types of liquors in inventory and the total inventory level of various liquors.

Collect statistics on the inventory level of each kind of liquor. You are only required to group by the TYPE field, and collect statistics respectively:



	A
1	=demo.query("select * from LIQUORS")
2	=A1.group(TYPE)
3	=A2.new(TYPE,~.count():Count,~.sum(STOCK):Stock)

Get the liquor information from A1. In A2, group by the TYPE; In A3, compute the resulting TSeq based on the grouping data in A2.

After cellset computation, you can view the grouping of records in A1 from A2:

Member		LID	NAME	TYPE	PRODUCTION	STOCK
[11,16,21, ...]		12	X O. Hennessy X. O. Cognac	Brandy	France	168
[12,22,30, ...]		22	Martell Medaillon	Brandy	France	88
[5]		30	Martell Medaillon	Brandy	France	216
[14,24,32, ...]		37	Club De Remy Martin	Brandy	France	29
[10,10,17, ...]		44	XO. Courvoisier X.O.	Brandy	France	78

The resulting TSeq in A3 is as follows:

TYPE	Count	Stock
Aperitif	6	800
Brandy	11	1941
Cordials	1	434
Gin	9	2410
Liqueur	13	2369

- Collect statistics on the respective types of Rum, Brandy, Gin, and Whisky in inventory, and their total inventory.

From the above statistics, we can see that the record is sorted by TYPE field first and then group the record of TYPE field if group function is used directly. The **align@a** function is frequently used to group records by a specified order.

	A
1	=demo.query("select * from LIQUORS")
2	[Rum,Brandy,Gin,Whisky]
3	=A1.align@a(A2,TYPE)
4	=A3.new(TYPE,~.count():Count,~.sum(STOCK):Stock)

In A3, group the liquor information in A1 in alignment by the TYPE field against the member values in the corresponding sequence A2. The resulting TSeq in A4 is as follows:

TYPE	Count	Stock
Rum	6	2077
Brandy	11	1941
Gin	9	2410
Whisky	13	3813

- Collect statistics on the liquors of OverStock(>500), LackStock(<100), and NormalStock(enough) respectively, and compute the total inventory level of each group. Enum grouping can be used to group records by certain conditions.



	A	B
1	=demo.query("select * from LIQUORS")	
2	?>500	OverStock
3	?<100	LackStock
4	null	NormalStock
5	=A1.enum([A2:A4],STOCK)	
6	=A5.new([B2:B4](#):Condition,~count():Count,~sum(STOCK):Stock)	

In A5, group the records in A1 by STOCK according to the grouping conditions; In A6, according to the grouping result in A5 and the name of each condition, you compute the resulting TSeq as follows:

Condition	Count	Stock
OverStock	2	1262
LackStock	16	927
NormalStock	50	14211

### 7.5.2 Data Preparation for the Sales Statement

The SALES table in the database stores the sales information over a time. This table stores information including the ORDERID, CLIENT, SELLERID, AMOUNT, and ORDERDATE:

ORDERID	CLIENT	SELLERID	AMOUNT	ORDERDATE
1	UJRN	17	392.0	11-02-2008 15:2
2	SJCH	6	4802.0	11-09-2008 15:2
3	UJRN	16	13500.0	11-05-2008 15:2
4	PWQ	9	26100.0	11-08-2008 15:2
5	PWQ	11	4410.0	11-12-2008 15:2
6	HANAR	18	6174.0	11-07-2008 15:2
7	ECU	2	17800.0	11-06-2008 15:4

In this case, taking year as parameter, we need to retrieve the sales data of this year and last year to prepare the object table. In the object table, list the number of sales person, sales of this year, sales of last year, growth rate, number of clients, big client (number of clients to whom the sales exceed 50 thousand *yuan*), and the proportion of the big client.

Because you need to compute the data for each sales person, after retrieving the sales data of the recent two years, you need to group data by the sequence number of sales person first, collect statistics on the annual sales data, and then collect statistics on the sales data of each client after regrouping by the client.



Program parameter

☐ Set arguments before run

No.	Name	Value	Remark
1	year	2010	

OK  
Cancel  
Add  
Delete  
Up  
Down

	A	B
1	=demo.query("select * from SALES where year(ORDERDATE) in (?)", [year,year-1])	
2	=A1.group(SELLERID)	
3	=create(SellerId,{string(year)+"Sales"},{string(year-1)+"Sales"}, CustCount,BigCustCount, BigCustRate, GrowthRate)	
4	for A2	=A4(1).SELLERID
5		=round(A4.select(year(ORDERDATE)==year).sum(AMOUNT),2)
6		=round(A4.select(year(ORDERDATE)==year-1).sum(AMOUNT),2)
7		=A4.group(CLIENT).(round(~.sum(AMOUNT),2))
8		=B7.count()
9		=B7.count(~>=50000)
10		=round(B9/B8,2)
11		=round((B5-B6)/B6,2)
12		>A3.insert(0,B4,B5,B6,B8,B9,B10,B11)

In A1, retrieve the sales data of this year and last year, taking year as parameter. In A2, group the retrieved data by the sequence number of the sales person. In A3, establish the result TSeq. The code after the 4th row will loop the sales data of each sales person in the A2 to collect the statistics. The sequence numbers of sales person are obtained in B4. The total sale of 2010 is computed in B5. The total sale of 2009 is computed in B6. In B7, the sales data will be re-grouped by CLIENT, and the total value of the orders for each client is computed. According to the computation in B7, the total number of clients of this sales person is computed in B8; and the number of big clients of this sales person is computed in B9. The proportion of big clients is





computed in B10; the growth rate of sales of this sales person in 2010 is computed in B11. After computation, in B12, fill the statistics of this sales person to the resulting TSeq in A3.

After computation, you can check the result of data preparation in A3:

SellerId	2010Sales	2009Sales	CustCount	BigCustCount	BigCustRate	GrowthRate
1	265934.0	313228.0	22	1	0.05	-0.15
2	301502.0	298006.0	22	3	0.14	0.01
3	279986.0	142054.0	17	2	0.12	0.97
4	228070.0	237614.0	19	3	0.16	-0.04
5	175478.0	252294.0	19	2	0.11	-0.3
6	198150.0	246882.0	19	2	0.11	-0.2
7	198100.0	204208.0	16	2	0.12	0.08

### 7.5.3 Payroll Computation

The EMPLOYEE table in the database stores the employee information, such as the name, state, gender, birthday, and basic salary:

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-20	2005-03-11	R&D	7000
2	Ashley	Wilson	F	New York	1980-07-19	2008-03-16	Finan	11000
3	Rachel	Johnson	F	New Mexico	1970-12-17	2010-12-01	Sales	9000
4	Emily	Smith	F	Texas	1985-03-07	2006-08-15	HR	7000
5	Ashley	Smith	F	Texas	1975-05-13	2004-07-30	R&D	16000

Days absent from work this month of each employee are stored in the ATTENDANCE table:

EMPLOYEEID	ABSENCE
1	1.00
3	2.00
6	1.50
9	3.00

The employee evaluation and bonus in this month is stored in the PERFORMANCE table:

EMPLOYEEID	EVALUATION	BONUS
1	0.75	6000
2	0.90	10000
3	1.10	8000
4	0.80	800
5	1.40	3000
6	1.18	4000

– **salaryPaid = SALARY\*(1-ABSENCE/(250/12))+SALARY\*(EVALUATION-1)+BONUS**

The employees not in the ATTENDANCE table are considered full attendance, and the employees not in the PERFORMANCE table are considered to deserve a 1 evaluation and 0 bonus. Compute the name and salary to be paid of top 5 employees with the greatest difference between standard salary and salary to be paid.

Because not all employee records are contained in the PERFORMANCE table and the ATTENDANCE table, you cannot perform computation directly with the sequence numbers. You need to conduct alignment grouping or equal join on the records in the ATTENDANCE table and PERFORMANCE table according to the employee number, and then collect the statistics.



### Style 1: Without join

	A	B	C
1	=demo.query("select * from EMPLOYEE")		
2	=demo.query("select EMPLOYEEID, ABSENCE from ATTENDANCE")		
3	=demo.query("select EMPLOYEEID, EVALUATION,BONUS from PERFORMANCE")		
4	=A2.align(A1:EID,EMPLOYEEID)		
5	=A3.align(A1:EID,EMPLOYEEID)		
6			
7	for A1	=A7.SALARY	=B7
8		if A4(#A7)!=null	>C7=B7*(1-A4(#A7).ABSENCE/(250/12))
9		if A5(#A7)!=null	>C7=C7+B7*(A5(#A7).EVALUATION-1)+A5(#A7).BONUS
10		>A6=A6 round(C7-B7,2)	
11	=A6.psort(abs(-)-1)(to(1,5))		
12	=A1(A11).new(NAME+" "+SURNAME:FullName,A6(A11(#))+SALARY:Salary)		

In A1, A2, and A3, retrieve employee data, attendance data, and performance data from database. In A4 and A5, perform the alignment grouping on attendance and performance data respectively by employee ID. In A6, prepare an empty sequence to store the difference between the actual salary and standard salary of each employee.

In the code block of A7, compute the difference between the actual salary and the standard salary of each employee by loop: in B7, get the standard salary; in C7, store the actual salary; in B8, if this employee has the absence record, then compute the salary according to the days absent from work; In B9, if this employee has any record about performances, then compute the salary according to his performance; in B10, compute the difference between the actual salary and the standard salary and store the result into the sequence in A6.

In A11, compute the sequence numbers of top 5 employees with the greatest salary difference. These sequence numbers are the same as those employee numbers in A1's employee table. Finally, in A12, compute the resulting TSeq on the basis of the data of these 5 employees.

The resulting TSeq in A12 is as follows:

FullName	Salary
Ashley Smith	25400.0
Ashley Wilson	19900.0
Rachel Johnson	17036.0
Matthew Johnson	16188.0
Rebecca Moore	10914.0

### Style 2: With join:



	A	B	C
1	=demo.query("select * from EMPLOYEE")		
2	=demo.query("select EMPLOYEEID, ABSENCE from ATTENDANCE")		
3	=demo.query("select EMPLOYEEID, EVALUATION,BONUS from PERFORMANCE")		
4	=join@1(A1:Info,EID;A2:Attendance, EMPLOYEEID;A3:Performance, EMPLOYEEID)		
5	>A4.derive(Salary)		
6	for A4	=A4.Info.	=B6
7		if A6.Attendance !=null	>C6=B6*(1-A6.Attendance.ABSENCE/(250/12))
8		if A6.Performance !=null	>C6=C6+B6*(A6.Performance.EVALUATION-1)+A6.Performance.BONUS
9		>A6.Salary=round(C6,2)	
10	=A4.sort(abs(Salary-Info.SALARY):-1)(to(1,5))		
11	=A10.new(Info.NAME+" "+Info.SURNAME,FullName,Salary:Salary)		

In A4, through the equal join regarding the employee ID on the EMPLOYEE table, ATTENDANCE table, and PERFORMANCE table, all information of an employee will be consolidated to form a TSeq. @1 option is needed to perform left join during the operation. In A5, the actual salary field will be added to the TSeq of A4. The following computation is similar to the style 1. At last, in A11, return the information of the top five employees with the greatest salary difference, and the result is the same:

FullName	Salary
Ashley Smith	25400.0
Ashley Wilson	19900.0
Rachel Johnson	17036.0
Matthew Johnson	16188.0
Rebecca Moore	10914.0

With the **join()**, the data from different tables can be consolidated to a TSeq clearly. The train of thought is much clearer when collecting statistics.

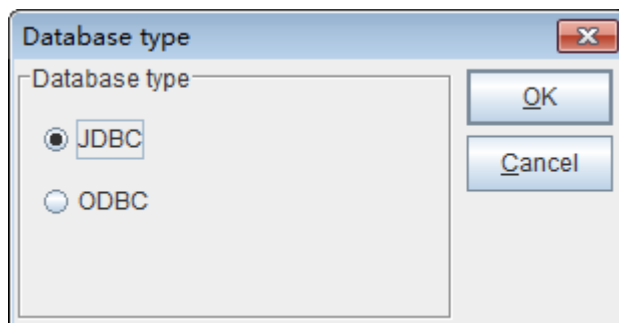
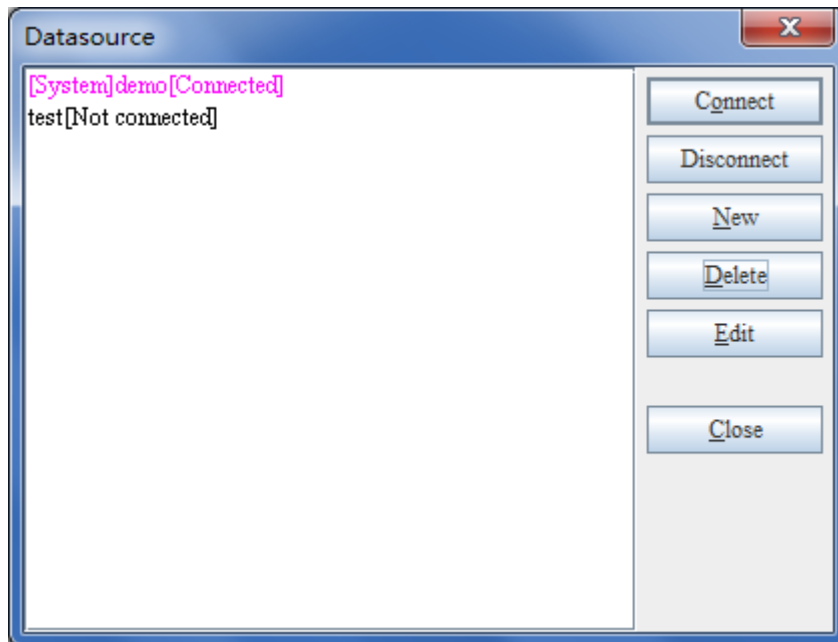
## 8. Database

### 8.1 Connect databases

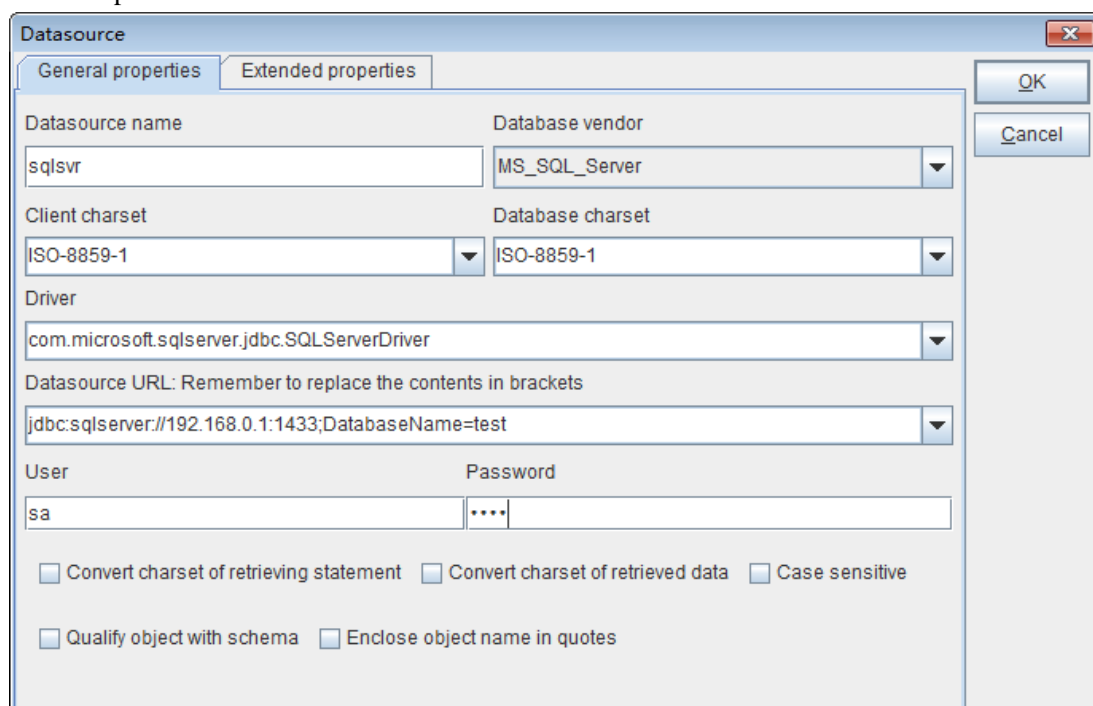
#### 8.1.1 Datasource Manager

To connect the relational database:

- Click the **Tool -> Datasource connection** menu item, open the Datasource Manager. Create a new datasource, and select the type.

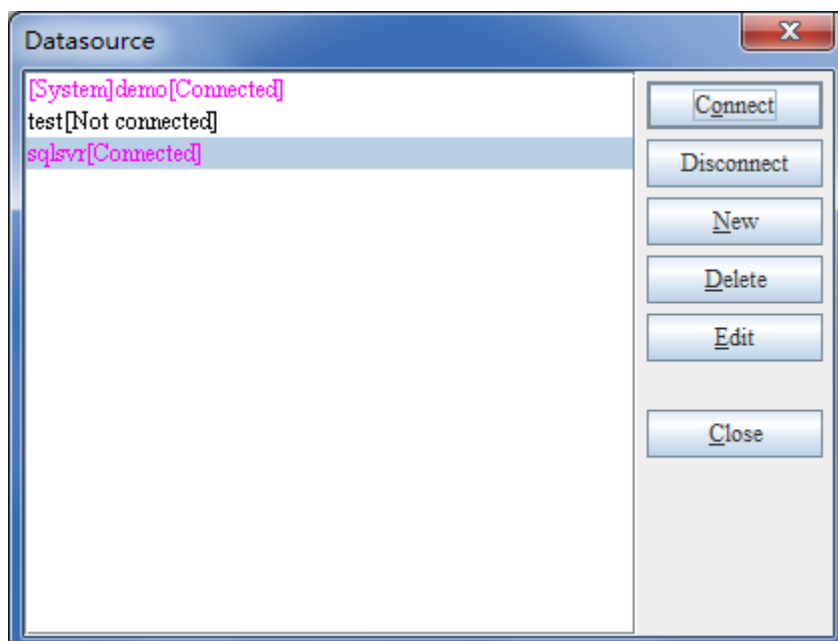


Select database type, charset and other information; set the datasource connection parameters and name the datasource.



Connect, and the Datasource Manager will display if the Datasource is connected

successfully. In the Datasource Manager, you can connect to multiple databases at the same time.



### 8.1.2 Database Connection Function

Besides the connection and disconnection from datasource in the Datasource Manager, the database connection functions are also available in esProc.

#### **connect(*dbn*)**

Connect to the datasource *dbn*, name of the datasource and connection parameters are **configured in the Datasource Manager**. Return the database connection object *db*. This object can be invoked directly with the cell name.

- ❖ **@e** Error message will be returned and handled by the code if errors are caused. Otherwise the execution will be interrupted.

When connect function is used with the **@e** option, you can check the error information with codes:

#### **db.error()**

Return the error code of the previous command regarding the database operations, and 0 indicates no error.

- ❖ **@m** Return the error message string of previous database-related command.

#### **db.close()**

Close the database connection object *db*.

	A	
1	=connect("demo")	Connect to datasource demo
2	=A1.query("select * from FOODS")	
3	>A1.close()	Close the datasource connection in A1

### 8.1.3 Commit transaction and Rollback transaction of the Database

In esProc, non-automatic commit is allowed when you execute certain database functions. Then, you can use functions to control the commit and the rollback of database.

### **db.commit()**

Commit the operation on database to the database.

### **db.rollback()**

Rollback the database, and cancel the operation pending commit.

## **8.2 Data Retrieval from Database**

### **8.2.1 Retrieve TSeq with SQL**

#### **db.query(sql...)**

Execute the *sql* statement in the datasource *db*, and return the resulting TSeq. *sql* statements can have tailing parameters, and you may directly add the parameters after statements.

- **db.query("select \* from employee")**
- **db.query("select \* from employee where dept=?", 3)**

	A	B	C
1	=connect("demo")		
2	=A1.query("select * from ADVENTURE")	=A1.query("select * from ADVENTURE where NAME>?", "M")	=A1.query("select * from ADVENTURE where NAME>? and AID<?", "M", 5)
3	>A1.close()		

AID	NAME	COUNTRY
1	Grand Canyo	US
2	Kailua-Kona	US
3	Yosemite Na	US
4	Moab	US
5	Sequoia and	US

AID	NAME	COUNTRY
3	Yosemite Na	US
4	Moab	US
5	Sequoia and	US
8	Olympic Natio	US

AID	NAME	COUNTRY
3	Yosemite Na	US
4	Moab	US

Annotations: Red arrows point from the SQL queries in the table to the corresponding result sets below. The middle and right result sets are annotated with red boxes: "Scenic spots whose names are alphabetically behind 'M'" and "Scenic spots whose names are alphabetically behind 'M' and serial numbers are below 5".

- ❖ **@1** Only return data of the first row. The returned value is single-value or the sequence instead of the TSeq.

	A
1	=demo.query@1("select * from ADVENTURE")
2	=demo.query@1("select Name from ADVENTURE")

Return a sequence since the data of 1<sup>st</sup> row is from multiple columns

Return single-value since the data of 1<sup>st</sup> row is from one column

Member
1
Grand Canyon National Park
US

Value
Grand Canyon National Park

#### **db.query(A,sql...)**

For the sequence *A*, execute the *sql* statement on the data source *db*, and return a TSeq incorporating the result sets. The *sql* statement can have parameters and you can just add them after it.

A		
Num	=demo.query([2,5,7],"select * from ADVENTURE where AID=?",~)	A is a sequence
[2,4,6]	=create(Num).insert(0,[2,4,6]).insert(0,[5,6]).insert(0,7)	
[5,6]		
7	=demo.query(A2,"select * from ADVENTURE where AID in (?)",Num)	A is a TSeq

AID	NAME	COUNTRY
2	Kailua-Ko	US
5	Sequoia	US
7	Glacier N	US

AID	NAME	COUNTRY
2	Kailua-Ko	US
4	Moab	US
6	Denali Na	US
5	Sequoia	US
6	Denali Na	US
7	Glacier N	US

After retrieving data from a TSeq, you may need to perform the **switch** operation on the TSeq, and the short form is as follows:

**db.query(...;...)**

Execute the **query()** function first, then perform the **switch()** operation on the resulting TSeq.

A		
1	=demo.query("select ABBR, NAME, STATEID from STATES")	
2	=demo.query("select * from CITIES",STATEID,A1.STATEID)	

ABBR	NAME	STATEID
AL	Alabama	1
AK	Alaska	2
AZ	Arizona	3
AR	Arkansas	4
CA	California	5

CID	NAME	POPULATION	STATEID
1	New York	8084316	NY
2	Los Angeles	3798981	CA
3	Chicago	2886251	IL
4	Houston	2009834	TX
5	Philadelphia	1492231	PA

ABBR	NAME	STATEID
CA	California	5

## 8.2.2 \*Execute the Stored Procedure in the Database

**db.proc(sql,a:t:m:v,...)**

Call the stored procedure to compute. For any possible result returned, it will be carried out with the *v* in the parameter.

The stored procedure is different to the normal SQL. In addition to the input parameter, there are still the output parameters. The return value of stored procedure will be taken out with the output parameter *v*.

## 8.3 Write to Database

### 8.3.1 Execute the SQL Command in the Database

**db.execute(sql,...)**

Run *sql* statement in datasource *db*, “...” is a parameter. Unlike **query** function, **execute** function will not return the result set. It is usually used to override the database data.

- **db.execute("delete from EMPLOYEE where DEPT=?", 3 )**

Similar to **query()** function, the **execute** function can also work with an RSeq or a Sequence:

### ***db.execute(A,sql,...)***

Run *sql* on every member of Sequence A. If A is an RSeq, you can use the fields of A in the same way as using the loop function in the parameter expressions.

- ***db.execute(A,"update EMPLOYEE set NAME=? where EID=?",Name,Id)***

A		ID	Name	Gender	Age
1	=file("D:/files/txt/students.txt")	1	Emily	F	17
2	=A1.import@t()	2	Elizabeth	F	17
3	>demo.execute(A2,"update STUDENTS1 set NAME=?, GENDER=?,AGE=? where ID=?",Name,Gender,Age,ID)	6	Zachary	M	19
		8	Megan	F	16

Update table students 1 in the database with the table

**execute()** function will usually auto-commit to the database after execution. If the manual control is required, then you can use these options.

- ❖ **@k** Execute the SQL statement with no transaction committed.
- ❖ **@s** Run the static SQL statement directly without pre-parsing, and the parameters cannot be used.

When using the **@k** option, the transaction will not commit automatically. You will need to use function **db.commit()** to commit or use function **db.rollback()** to roll back and cancel the modification.

A		B
1	=file("D:/files/txt/students.txt")	
2	=A1.import@t()	
3	=connect@e("demo")	Establish database connection
4	>A3.execute@k(A2,"update STUDENTS1 set NAME=?, GENDER=?,AGE=? where ID=?",Name,Gender,Age,ID)	
5	=A3.error()	Check the error code after executing A4
6	if A5==0	>A3.commit() Commit if no error
7	else	>A3.rollback() Roll back if any error
8	>A3.close()	

### 8.3.2 Update the Whole TSeq

You can also use **update** function to write the whole TSeq to the database.

### ***db.update(A,tbl,F;xi,...;P,...)***

Update field  $F_i$  in *tbl* table according to A and assign value  $x_i$  to it; use the field value of the same name if  $x_i$  is omitted. The  $P,...$  in the expression is the primary key for updating. If not specified, then the primary key will be retrieved from table *tbl*. If the retrieval fails, then the primary key of A will become the substitute.

- ***db.update(A1,EMPLOYEE, NAME:FullName,EID)***

A is an RSeq usually. For every record in A, if the primary key value exists in *tbl*, then an **UPDATE** statement will be generated. Otherwise, **INSERT** statement will be generated.



	A		ID	Name	Gender	Age
1	=file("D:/files/txt/students.txt")		1	Emily	F	17
2	=A1.import@t()		2	Elizabeth	F	17
3	>demo.update(A2,STUDENTS1,ID,NAME:Name, GENDER:Gender,AGE:Age,ID)		6	Zachary	M	19
			8	Megan	F	16

Update the retrieved TSeq in the file to the STUDENTS1 table in the database

When **update()** function is working, several options below are often used to perform various operations:

- ❖ **@u** Only generate **UPDATE** statement.
- ❖ **@i** Only generate **INSERT** statement.
- ❖ **@a** Empty the target table before updating.
- ❖ **@k** After updating, do not commit transaction. The transaction will be committed automatically by default.
- ❖ **@1** The first field is the independent increment field. During **updating**, no value will be assigned to this field.

## 8.4 Use SQL Directly

**\$(db)sql;...**

Execute the SQL statement directly in the cell in the data connection **db**. If the **(db)** is omitted, then use the **db** just worked in the previous statement. The SQL statement can be with parameters which simply follow the semicolon. When using the SQL statement, you are not required to add the equal mark before it or mark it with quotes. The beginning words are limited to **select/insert/update/delete** only. Using the **select** statement will return the result set.

- **\$(db)select \* from EMPLOYEE**
- **\$select \* from EMPLOYEE where DEPT=?;3**
- **\$update EMPLOYEE set NAME=? where EID=?;"Harry",1**

	A		AID	NAME	COUNTRY
1	=connect("demo")		1	Grand Canyon	US
2	\$(A1)select * from ADVENTURE	Specify the datasource to execute the select statement with (A1), and return the result set.	2	Kailua-Kona	US
3	\$(A1)select * from ADVENTURE where NAME=? and AID<?;"M",5		3	Yosemite National	US
4	\$update STUDENTS1 set NAME=? WHERE ID=?;"Nancy",1	Omit the (db) and thus use the db connection in the previous sql statement	4	Moab	US
5	=A1.close()	Execute update statement and return the updated number of the records	5	Sequoia National	US

AID	NAME	COUNTRY
3	Yosemite National	US
4	Moab	US

Value
1

## 9. High-level Code and Its Application

### 9.1 Long Statement

If any long statements cannot be displayed fully when written in one cell, then you can write them in multiple cells according to **the sequential cell rule**.

If a computational cell or an execution cell ends with ",", ";", or "(", it will concatenate with the contents of the next cell till the cell doesn't ends with these characters or reaches the end of this code block.

	A	B	C	D
1	68			
2	==if(A1>=80:"Heavyweight", A1>=68:"Middleweight", A1>=58:"Lightweight", "Flyweight")			

Value
Middleweight

	A	B
1	Deputy Division Chief	
2	==case(A1,	
3		"Division Chief":500,
4		"Deputy Division Chief":300,
		"Section Chief":500)

Value
300

The first cell of a long statement generally begins with sign "==" or ">>". This writing style tells esProc to process the entire code block, which take this cell as its master cell, as a single statement. Without these signs, other cells in the code block will be mistakenly executed by esProc and errors will thus occur.

### 9.2 Computational Substatement

Following the JAVA convention, the assignment statement is also regarded as an expression, which will return the value itself and support the parenthesis operator.

- **a=4** Assign 4 to *a* and return 4.
- **(a=3,a\*a)** Assign 3 to *a* and return 9, the computed result of the expression.

	A
1	=a=4
2	=a
3	=(b=3,b*b)
4	=b

Value
4
Value
4
Value
9
Value
3

Some long statements are very complex, in which the temporary variables above may be used for the computation, for example:

- **demo.query("select \* from EMPLOYEE").select((a=demo.query("select \* from FAMILY where**

EID=?", EID).select(RELATION: "child").sort(Age: -1),if(a.len())<2, false, a(1).AGE-a(2).AGE>= 2)))

The above code will find out the employees whose age differences of their first two children are not less than 2.

This long statement is not allowed to be written into multiple cells by practicing the above-mentioned mechanism of long statement writing style because the temporary variables are used in it. In this case, you will need to follow the rule of computational statement to write the long statement into multiple cells.

	A		C
1	<code>==demo.query("select * from EMPLOYEE").select(??)</code>	A1: The employees whose age differences of their first two children are not less than 2 are	
2		<code>=demo.query("select * from FAMILY where EID=?",EID)</code>	
3		<code>=B2.select(RELATION:"child")</code>	
4		<code>if B3.len()&gt;=2</code>	Sort it in descending age
5			<code>=B3.sort(AGE:-1)</code>
6		The age difference of the first two children is not less than 2	<code>=(C5(1).AGE-C5(2).AGE&gt;=2)</code>
7		else	<code>=false</code>

In cell A1, you may find a "??" operator.

??

The operator ?? equals a function. It will compute in order the cells of the code block except the current cell, which is the master cell, and return the cell value of the last computational cell.

The whole statement becomes clearer with this writing style.

The computational substatement is a long statement that is quite special and also begins with double equal sign "=", without which esProc will execute again the other cells in the code block, and thus errors may occur.

## 9.3 Assignment Block, Execution Block and Comment Block

### 9.3.1 Assignment Block and Computation Block

In the above example of long statement and the computational substatement, cell strings of the master cells all begin with == instead of =.

This writing style is to make esProc realize that the whole code block with the current cell as its master cell is actually a single statement, which thus must be computed as a whole and cannot be split. In this case, esProc will ignore other cells in this code block after executing this statement according to the rule. Otherwise, without the operator, esProc will compute other cells in the code block by default, and this may cause errors.

The executable cell works in the same way. When it begins with >>, it indicates that the whole code block with the current cell as its master cell is a single statement.

	A	B
1	=connect@e("demo")	=A1.query("select * from EMPLOYEE")
2	>A1.execute("drop table EMPLOYEE1")	>A1.execute("create table EMPLOYEE1 (EID int,FULLNAME varchar(30), GENDER varchar(10))")
3	>>A1.execute(  	

These code blocks, according to the types of statements being executed, are respectively called assignment block and **execution block**. Accordingly, the above statement block beginning with == is called **computation block**.

### 9.3.2 Comment Block

Similarly, when the cell string of a cell begins with //, it indicates that the whole code block with this cell as its master cell is the comment. This code block is thus called comment block. When esProc runs into this kind of cell strings, it will skip the whole code block.

	A	B
1	//comment:	
2		1.note1...
3		2.note2...
4		3.note3...
5	=1+3	

All the comment block is the comment

The ending line of a comment block starts

## 9.4 Macro

To increase the flexibility of the code, you need to generate the statement string dynamically, and a macro may be used here.

The macro is bracketed with sign **\${}** , and esProc will compute the expression in **\${...}** and take the result as the macro value to replace **\${}** . If **\${}**  is quoted, then it will not be replaced. After the replacement, esProc will resolve the statement. In this way, you can generate the expression dynamically.

	A	B	C
1	[1,2,3,4]		
2	func	=A1.\${A2}()	return B2
3	func	=A1.\${left(A3,3)}()	return B3
4	=func(A2,"sum")	Return A1.sum()	
5	=func(A3,"avgTest")	Return A1.avg()	

Value
10

Value
2.5

## 9.5 String Constant

**\$[...]** indicates that the string constant consisting of "..." is equal to "..." in the esProc expressions.

But there are a lot of differences between **\$[...]** and "..." in the esProc edit interface. The cell names in **\$[...]**, along with other standard expressions, will change according to the editing operation; or the expression will be modified in executing copy and paste. While the contents of "..." will not change according to the editing operation.

Thus, in some special circumstances that strings need to be adjust-paste along with the editing, the "..." may be substituted with **\$[...]**, for example, the arguments of **eval** function, and SQL statements in **query** function, etc. These arguments are all presented as strings, but they essentially are expressions or SQL substatements to be parsed.

	A	B	C	D
1	=eval(\$[C1+D1])	=eval("C1+D1")	1	4
2	=eval(\$[C2+D2])	=eval("C1+D1")	2	5
3	=eval(\$[C3+D3])	=eval("C1+D1")	3	6

String constant. The cell name in the string can be adjusted when pasting with Ctrl+Alt+V.

String in the "...". Contents in the string cannot be adjusted when pasting with Ctrl+Alt+V

	A	B
1	= "EID=4"	= "EID=14"
2	=demo.query(\$[select * from FAMILY where \${A1}])	=demo.query(\$[select * from FAMILY where \${B1}])

If macro exists in the string constant, then the cell name used in the macro can be adjusted along with inserting and deleting columns, or pasting with Ctrl+Alt+V

In the Section **2.2.2 String Constant Cell**, we can define the string constant by starting with the **'**, and the data in the string constant cell will be resolved as string directly, not requiring any additional escape character. In the string constant cell, you can define the character string starting with the equal mark or containing special characters.

	A
1	'=, >, <'
2	'C:\file.txt'

Value
=, >, <

Value
C:\file.txt

## 9.6 Special Identifiers



Some identifiers may include non-alphabetic or non-numeric characters such as %, () or **spaces**, so they may be misinterpreted as a percent sign or parentheses by the engine. To avoid this, a pair of single quotation marks may be used to enclose the identifier so as to differentiate the identifiers from the special signs.

	A
1	=demo.query("select * from STATES")
2	>A1.alter(ABBR,POPULATION,Area (Sq.mi.):AREA)
3	>A1.derive(round(POPULATION/Area (Sq.mi.),'2):Population/Area)

Parentheses, spaces, decimal point, and other special characters may exist in the column name. In order to avoid confusion, you may need to indicate them with single quote.

ABBR	POPULATION	Area (Sq.mi.)	Population/Area
AL	4779736	52419	91.18
AK	710231	663267	1.07
AZ	6392017	113998	56.07
AR	2915918	52897	55.12
CA	37253956	163700	227.57

## 9.7 Cross-Cellset Calling

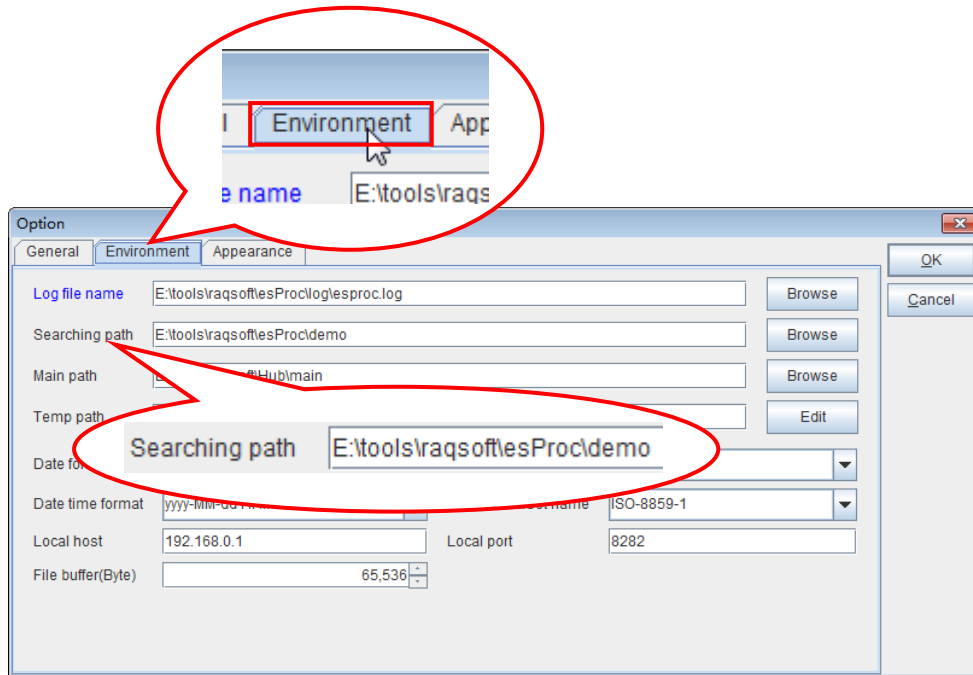
### 9.7.1 File Path

- ❖ **file@*s(fn)*** Define the file object at the path *fn*. It is allowed that *fn* does not contain any file path. For those file names not containing any path, esProc will search the class path first, and then the list of searching path.

	A
1	=file("D:/files/txt/students.txt")
2	=file@s("students.txt")

If it is not the full file path that in the file name, you can use the @s option to indicate that the file is in the application directory

Click the **Options** button on the **Tool** menu option of the menu bar to display the Option window. In the Option window, set the list of searching path on the **Environment** tab. If multiple paths are required to be set, use the semicolon to separate them:



### 9.7.2 Return Value and End of Cellset

result  $x_i, \dots$

In esProc, to run the whole cellset, you can take it as a subroutine. In the cellset, use **result** to return the result of expression  $x_i, \dots$ . Return and terminate the program to release resources.

### 9.7.3 Cross-Cellset Calling

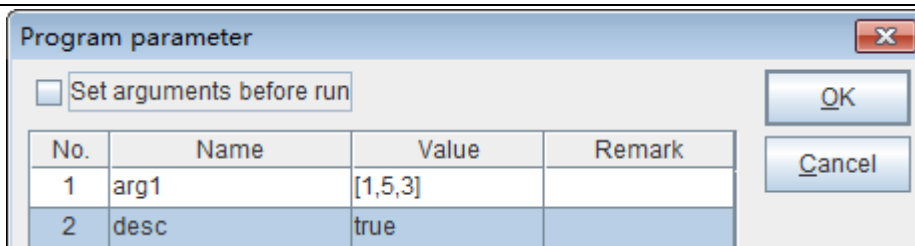
In esProc, you can execute cross-cellset calling. In other words, you can run the program from other cellsets or use the computation in other cellsets.

**call(df $x$ ,...)**

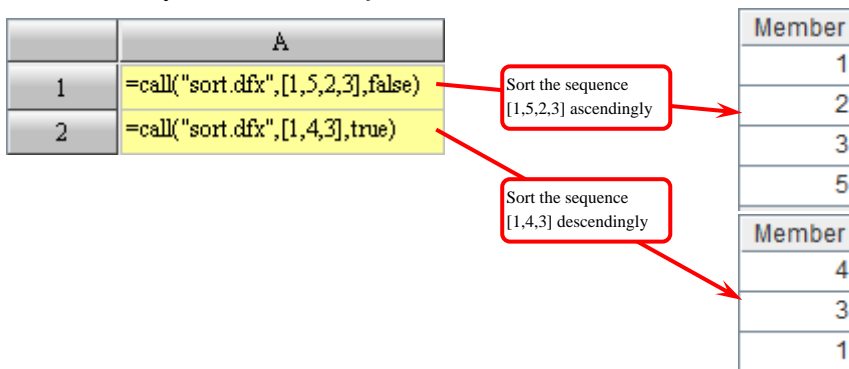
Call the cellset  $dfx$ , and take "... " as the parameter to input in correct order. When specifying  $dfx$ , you can use the absolute path name of the cellset directly. If the absolute path isn't used, @s will find the file automatically. If there is any cellset parameter in the  $dfx$ , then use these parameters "... " when calling the **call()**. These parameter values will be given to each parameter of  $dfx$ , which are irrelevant to the parameter names in the  $dfx$  parameter list.  $dfx$  can also be a file object. In that case, it will be loaded and executed directly.

For example, the cellset file *sort.dfx* is to sort the sequence in the parameter *arg1* according to the settings in the parameter *desc* (true for descending and false for ascending):

	A	B
1	if desc	result arg1.sort(~:-1)
2	else	result arg1.sort()



In other files, you can call *sort.dfx* across cellsets:

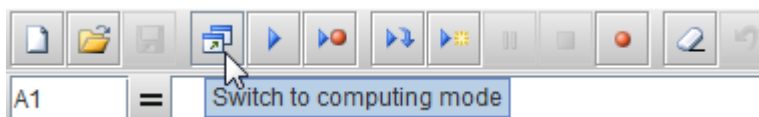


When multiple results are to be returned from the cellset being called using **result**, you will get a sequence as the final result.

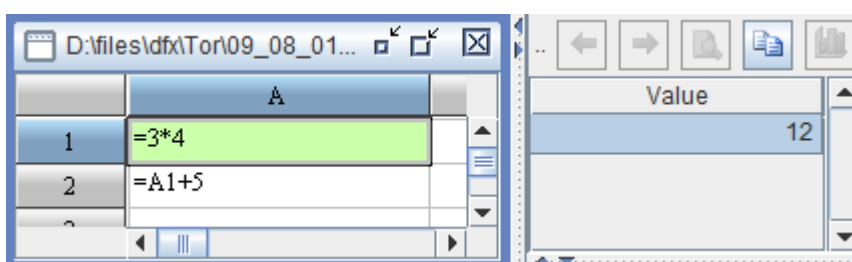
## 9.8 Instant Computation Mode

esProc can also run in the instant computation mode.

Click "**Switch to computing mode**" button on the Tool bar, and enter the instant computation mode.



In the instant computation mode, press the Enter key on the cell to compute the cell value of this cell instantly:



In the instant computation mode, you must pay attention to the order of computation because you can only compute the selected cell value each time. If the uncomputed cells are called in the expression, errors may occur.

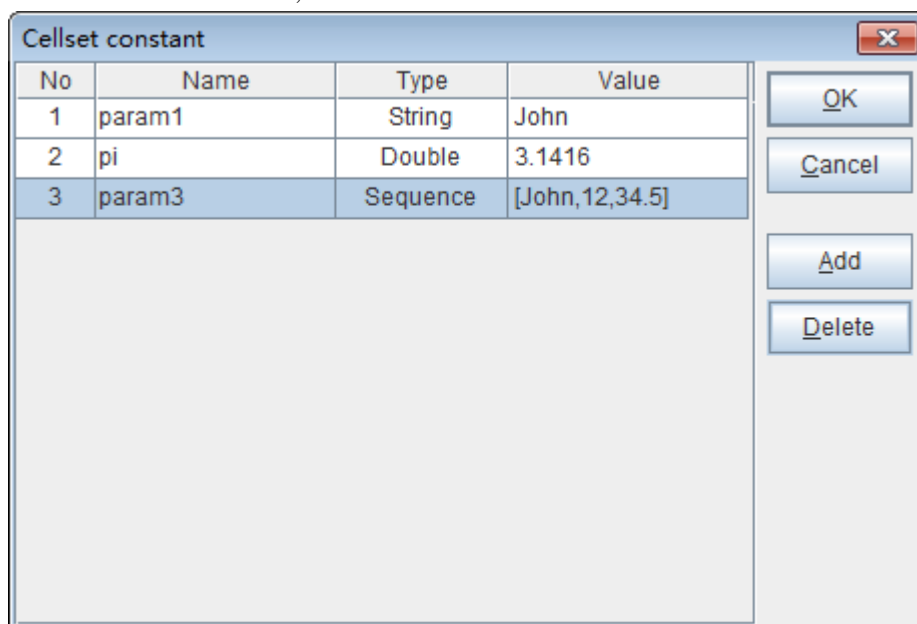
## 9.9 Use Constants in the Cellset

In the cellset computation, parameters are used frequently. For some repetitively used parameters, you can set them as constants. The usage of constants is the same as that of the cellset parameters, and you can call them with the constant names. However, constants and parameters in the cellset are a bit different. The assignment is allowed for parameters in the cell while it is not

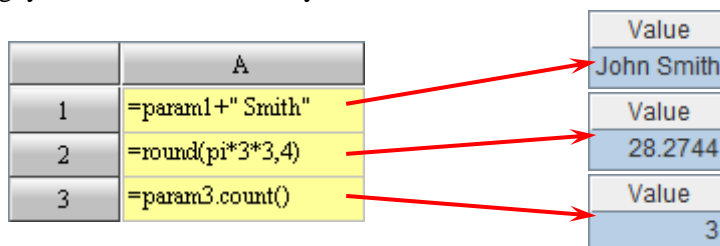


allowed for the constants.

To view **Cellset constant**, click the **Cellset constant** button on the **Tool** menu:



The constants can be various data types, such as string, integer, date, and sequence. When using, you can call them directly with their names:



## 9.10 Execute Cellset Files with Command Line

Under esProc installation directory esProc\bin, you can find the esprocx.exe file, and you can compute the cellset files with command line in the DOS environment.

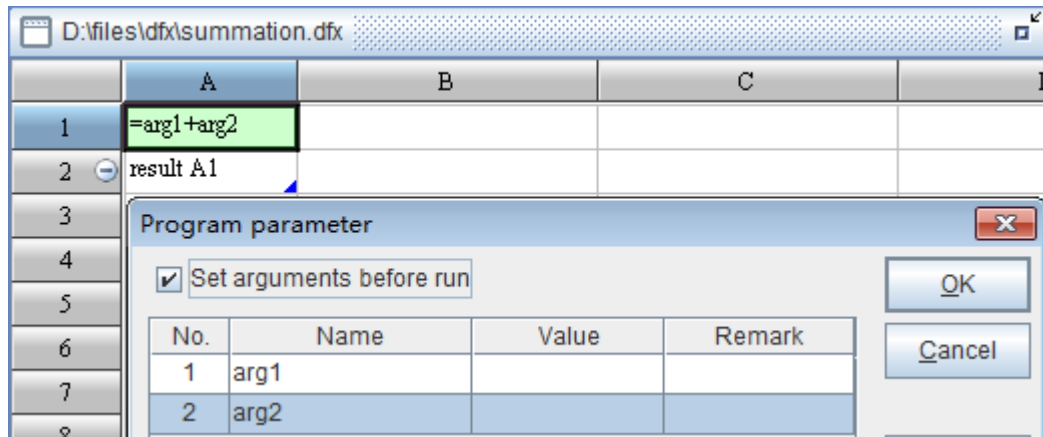
**esprocx <options> <dfxFile> [arg0] [arg1]...**

esProc can be launched with a command line in DOS. The **option** in the command line provides options for setting the launch. The **dfxFile** is the cellset for esProc computation, and **[arg0] [arg1]...** is the cellset parameters for computation. In computing the cellset file, the program will automatically search the config.xml to retrieve the environment configuration data. The config.xml file contains the basic configuration data of esProc, such as registration code, searching path, home directory, and data source configuration.

- ❖ **-R** Export result. TSeq/RSeq is exported in the form of text
- **esprocx C:\c.dfx** Execute the computation of cellset file *c.dfx*.

When the cellset file is being executed with the command line, the messages of startup and end will be printed out on the screen.

- No export



Execute the command line to pass-in the arguments 100 and 5, and the result is as shown below:

```
D:\Program Files64\esProc\bin>esprocx F:\summation.dfx 100 5
[2014-01-23 16:01:21] : [DEBUG] - Before calculating:Thu Jan 23 16:01:21 CST 2014
[2014-01-23 16:01:21] : [DEBUG] - After calculated:Thu Jan 23 16:01:21 CST 2014
D:\Program Files64\esProc\bin>
```

Use **-R** option to print out the result of calculation.

	A	B
1	=connect("demo")	
2	=A1.query("select DEPT,NAME,SALARY from EMPLOYEE where EID < 10")	
3	>A1.close()	
4	result A2	

After the command is executed, the result is as shown below:

```
D:\Program Files64\esProc\bin>esprocx -R F:\queryEmployee.dfx
[2014-01-23 16:27:23] : [DEBUG] - Before calculating:Thu Jan 23 16:27:23 CST 2014

Result in A4:
R&D      Rebecca  7000
Finance  Ashley   11000
Sales    Rachel    9000
HR       Emily     7000
R&D      Ashley   16000
Sales    Matthew  11000
Sales    Alexis    9000
Marketing Megan     11000
HR       Victoria  3000
[2014-01-23 16:27:23] : [DEBUG] - After calculated:Thu Jan 23 16:27:23 CST 2014
```

## 10. File System and Cursor

During data computing, analysis and processing, the file access is the common and important demand. The file read-write and file data computing, and etc. all belong to the file access.

### 10.1 File System



In the **2.8 Read and Write data with File Object** , we explained the basic uses of file objects in esProc. In this section, we will further explain the related functions of file system.

### **filename(*fn*)**

Get the file name with extension, by splitting it out from the full path *fn*.

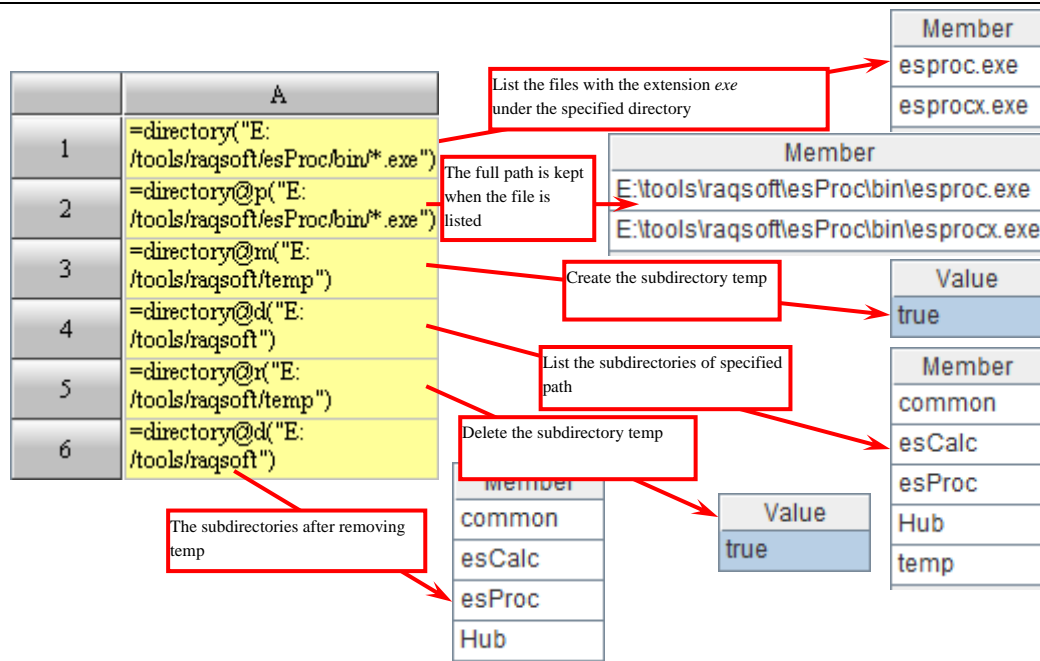
- ❖ **@e** Only split the extension name out
- ❖ **@n** Split the file name out with no extension kept
- ❖ **@d** Split the directory out
- ❖ **@t** It is for local use only. Under the same directory of *fn*, the temporary file of the same extension name is generated and the file name is returned. If *fn* is omitted, under the system temporary directory, a file name equivalent to the home directory will be generated.

	A	
1	=filename("D:/files/txt/abc.txt")	Value abc.txt
2	=filename@e("D:/files/txt/abc.txt")	Value txt
3	=filename@n("D:/files/txt/abc.txt")	Value abc
4	=filename@d("D:/files/txt/abc.txt")	Value D:/files/txt
5	=filename@t("D:/files/txt/abc.txt")	Value D:/files/txt/tmpdata1780127675025597244.txt

### **directory(*path*)**

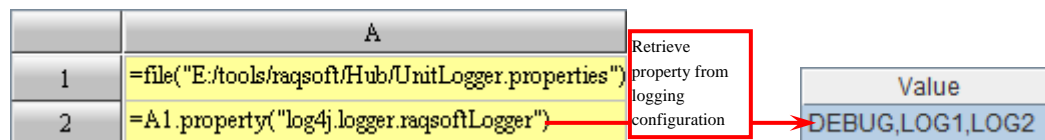
It is for local use only. List the file names matching the wildcard path *path*, excluding the path names.

- ❖ **@d** List the subdirectory
- ❖ **@p** Keep the full path name
- ❖ **@m** Create directory
- ❖ **@r** Delete empty directory



### ***f.property(p)***

From the property file object *f*, get the value of property *p* by name. If *p* is omitted, return a TSeq composed of all properties.



## **10.2 Database Cursor**

You can retrieve massive data from database in batches by taking advantage of database cursor.

### ***db.cursor(sql,...)***

Open a database cursor based on *sql* and return it.

### ***cs.fetch(n:x)***

With the cursor *cs*, retrieve *n* records or keep on retrieving records till the expression *x* changes. The retrieved records will be organized and returned in the form of TSeq. Null will be returned if the retrieval reaches its end. When the remaining records in the database are less than *n*, retrieve all of them. The cursor will be closed once all records are retrieved.

### ***cs.skip(n:x)***

*n* records are skipped or you keep on skipping the records one by one until expression *x* changes, and then return the actual number of skipped records.

### ***cs.close()***

Close the cursor *cs*.

	A	B	C
1	=demo.cursor("select * from EMPLOYEE")	Open a cursor	
2	for		Retrieve data from the cursor, 150 records each time.
3		=A1.fetch(150)	Discontinue the loop once finish retrieving the data
4		if B3==null	break
5		else	Operate on the data retrieved.
			...

Once the last record is fetched or skipped in a cursor, then this cursor will be closed automatically.

**for  $cs,n:x$**

Loop through the cursor, retrieve  $n$  records each time or retrieve the records till  $x$  changes, then return a TSeq. Once the retrieval ends, close the cursor.

	A	B
1	=demo.cursor("select * from EMPLOYEE")	Loop through the cursor, retrieve 100 records each time and return, close the cursor when the loop is
2	for A1,100	>A3=A3+A2.len()
3	0	

## 10.3 File Object Cursor and Memory RSeq Cursor

In the section **10.2 Database Cursor**, we introduced the basic method to retrieve the database data with the cursor. In addition, you can also retrieve file data with cursor or build a cursor by means of memory RSeq.

### 10.3.1 File Object Cursor

**$f.cursor(F_i:type_i,...;s,b:e)$**

Based on file  $f$ , create the cursor and return it. Creating a cursor is similar to retrieving a TSeq from files. You can also define the beginning and ending position of data and the fields to be retrieved. Similarly, you can also use various options like the **@t**, **@b**, **@z** and **@s**. For the related contents, please refer to the section **5.3.1 Retrieve and Write TSeq from File**.

❖ **@x** Delete the file automatically when the file object is closed.

The file object cursor is used in the same way as the database cursor:

	A	B	C
1	=file("D:/files/txt/employee.txt")		
2	=A1.cursor@t()		
3	for	=A2.fetch(150)	
4		if B3==null	break
5		=B3.select(STATE=="California")	
6		>A7=A7 B5	
7			

In this example, the file object cursor is used to retrieve the data of 150 employees each time, of which the Californian employees are selected to store in the A7. The result is shown below:



EID	NAME	SURNA...	GENDER	STATE	BIRTHD...	HIRED...	DEPT	SALARY
1	Rebecca	Moore	F	California	1974-11-	2005-03-	R&D	7000
6	Matthew	Johnson	M	California	1984-07-	2005-07-	Sales	11000
8	Megan	Wilson	F	California	1979-04-	1984-04-	Marketing	11000
23	Joseph	Turner	M	California	1983-08-	2003-08-	Finance	6000
27	Alexis	Jones	F	California	1983-12-	2003-12-	Marketing	10000
29	Olivia	Harris	F	California	1979-08-	2009-08-	Sales	8000

### ***F.cursor()***

Based on the binary file object - sequence *F*, create a cursor and return it. Each file will take up a field.

### **10.3.2 Memory RSeq Cursor**

### ***P.cursor()***

Convert an RSeq to a cursor with the aim to use the RSeq data in the cursor function, for example, **merge@x** and **join@x**.

	A	B	C
1	Administration	4	40000
2	Finance	24	177500
3	HR	19	138000
4	Marketing	99	733500
5	Production	91	663000
6	R&D	29	239000
7	Sales	187	1362500
8	Technology	47	344000
9	=create(DEPT,Count,TotalSalary).record([A1:C8])		
10	=A9.cursor()		

The memory RSeq cursor in A10 is generated from the RSeq in A9.

You can also create a cursor from a *dfx* file.

### ***pcursor(dfx,...)***

Call the cellset *dfx*, and take "..." as the parameter and input it in correct order, and return the result as a cursor.

## **10.4 Usage of Cursor**

### **10.4.1 Cursor for Data Retrieval**

In addition to the basic methods introduced in the section **10.2 Database Cursor** for cursor retrieval and skip, esProc additionally provides many other cursor-related functions. We will continue to explore these functions in this section.

### ***cs.select(x)***

Filter the records in the cursor *cs* on condition *x*, and return a cursor. It is equivalent to performing **select(x)** on records from *cs*.

- ❖ **@c** Only return the first continuous band of records that makes *x* true.



	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.select(STATE=="California")
4	=A3.fetch()
5	=A2.fetch()

Filtering data with a cursor directly will lead to the same result as filtering after data retrieving. The result from A4 is the same as that in the section **10.3.1 File Object Cursor**. One thing to note is that cursor in A3 is generated from cursor in A2. So, the cursor in A3 is to retrieve the data with the cursor in A2. Once data is retrieved with the cursor in A3, the cursor in A2 will also complete its data retrieval. Thus the result from A5 is null.

➤ ***f.export(cs,x:F,...;s)***

Write the data from cursor *cs* to the file object *f*. It supports **@t**, **@b**, **@p**, **@z**, and **@s** option of *export* function. Similar to ***f.export(A,x:F,...;s)*** in the section **5.3.1 Retrieve and Write TSeq from File**, you can also specify parameters *x*, *F* and *s*.

***F.export(cs,x<sub>i</sub>:F<sub>i</sub>,...;s)***

Write the data from cursor *cs* to each file object of the file object sequence *F*. Each field will be written to a file.

❖ **@a** Append

***cs.new(x...)***

Create a new TSeq according to the definition in *x*. In data retrieving, get the new records by computing the expression based on the records from cursor *cs*, and return a cursor. This is similar to performing ***new(x...)*** on the records from *cs*.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.new(NAME+" "+SURNAME:FULLNAME, interval@y(BIRTHDAY,now()):AGE)
4	=A3.fetch()
5	=A2.fetch()

With the file data, compute and return the full name and age of the employees. The ***cs.new()*** works in a similar way as creating a new TSeq based on an existing TSeq. The new cursor in A3 needs the cursor in A2 to retrieve the data. Once data is retrieved in the A3, the data retrieval in A2 is also completed.

The computed result of A4 is as follows:



FULLNAME	AGE
Rebecca Moore	39
Ashley Wilson	33
Rachel Johnson	43
Emily Smith	28
Ashley Smith	38
Matthew Johnson	29

### **cs.derive(...)**

Add computational fields to cursor *cs*, and return it as a new cursor. It is similar to run **derive(...)** against the result set returned from *cs*.

### **cs.run(x)**

Compute the expression *x* with the records from the cursor *cs* to generate the new records, and return them as a cursor, which is similar to performing **run(x)** on the records from *cs*.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME, SURNAME, STATE)
3	=A2.run(NAME=NAME+" "+SURNAME)
4	=A3.fetch()

NAME	SURNAME	STATE
Rebecca Moore	Moore	California
Ashley Wilson	Wilson	New York
Rachel Johnson	Johnson	New Mexico
Emily Smith	Smith	Texas
Ashley Smith	Smith	Texas

### **cs.switch ( $F_i, A_i;x;...$ )**

Switch the records in the cursor *cs* to get the new records and return them as a cursor. This is similar to executing **switch(...)** on the records in *cs*.

@i If the corresponding value of  $F_i$  cannot be find after a record is computed based on  $A_i$ , then delete the whole record.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME, SURNAME, STATE)
3	=demo.query("select NAME, ABBR, CAPITAL from STATES")
4	=A2.switch(STATE,A3:NAME)
5	=A4.fetch()

NAME	SURNAME	STATE
Rebecca	Moore	California
Ashley	Wilson	New York
Rachel	Johnson	New Mexico
Emily	Smith	Texas
Ashley	Smith	Texas

NAME	ABBR	CAPITAL
New Mexico	NM	Santa Fe

You can see from these cursor functions that when a new cursor is generated from a cursor function, the old cursor will become invalid and, in principle, shouldn't be used any more. Otherwise the new cursor and the old one will have mutual effect during data retrieval on each other, which will produce unexpected situations or make the data retrieval failed.

### **cs.news(...)**

Based on the expression..., perform the computation by using the records in the cursor *cs*. Each record will be split into a Sequence or an RSeq. The members or records in the results will be merged to return a new cursor. This is similar to the process that the **news(...)** is performed on the records of *cs*.





	A	
1	=file("D:/files/txt/employee.txt")	
2	=A1.cursor@t()	
3	=A2.news(create(NAME,Item,Date).record([NAME,"Birthday",BIRTHDAY,NAME,"HireDate",HIREDATE]))	
4	=A3.fetch(5)	
5	>A3.close()	

NAME	Item	Date
Rebecca	Birthday	1974-11-20
Rebecca	HireDate	2005-03-11
Ashley	Birthday	1980-07-19
Ashley	HireDate	2008-03-16
Rachel	Birthday	1970-12-17

It can be seen that the number of records will be computed based on the splitting results when you use **fetch** function to retrieve data.

➤ **cs regex(rs,F<sub>i</sub>,...)**

Execute regex(rs) on each member of string cursor cs and combine the matching results into a TSeq whose field name is F<sub>i</sub>,...

- ❖ @i Not case-sensitive.
- ❖ @u Use *unicode* to match.

The cursor's *regex* function works in a way similar to how a TSeq works. Their difference is that the former begins execution as the **fetch** function is fetching data.

#### 10.4.2 Merge and Unconditionally Join Cursor Sequences

In some cases, the data is from multiple cursors. Considering this, esProc provides some functions to merge or join the data from multiple cursors.

**CS.merge@x(x<sub>i</sub>,...)**

Merge the records from cursor sequence CS, and return the cursor. During merging, the records of each cursor must be ordered for expression x<sub>i</sub>,.... and in the same order as that of it. Similar to **A.merge()**, this function supports @u, @i and @d and thus can be used to perform operations of union, intersection and difference during merging.

Take the following case for example. The employee data for each department is stored respectively in the *txt* files which are named after the department names. For example, the data of Sales department is stored in *Sales.txt*. The employee data of each department is sorted by employee number EID in ascending order. In this case, the data of each department will be retrieved with the proper file object cursor and the employee data from all departments will be listed by the employee number. So, we need to merge the records in each cursor:

	A
1	[Finance,Sales,HR,Administration,Marketing,Production,R&D,Technology]
2	=A1.(file("D:/files/txt/"+~+".txt"))
3	=A2.(~.cursor@t())
4	=A3.merge@x(EID)
5	=A4.fetch()

A2 generates the sequence of file object of each department according to the department sequences in A1; A3 uses the file object sequence to generate the sequence of file object cursors. In generating the cursors, the @t option will enable you to read out the first row of text file to build the data structure of each cursor; A4 merges the data of every cursor in A3. In A5, you can



view the following result:

EID	NAME	SURNAME	GENDER	STATE	BIRTHDAY	HIREDATE	DEPT	SALARY
59	David	Texas	M	Pennsylvania	1977-12-24	2007-12-24	Sales	5000
60	Taylor	Smith	F	Texas	1974-12-16	2004-12-16	Production	12000
61	Joseph	Smith	M	California	1974-11-11	2006-04-01	Sales	5000
62	Emily	Jones	F	Texas	1977-05-06	2008-06-01	Sales	6500
63	Samantha	Martin	F	Missouri	1976-11-21	2006-01-01	Sales	7000
64	Nathan	Johnson	M	Michigan	1986-09-29	2003-08-01	Sales	5000
65	Michael	Smith	M	Connecticut	1971-03-03	2004-03-01	Sales	8000
66	Madison	Davis	F	Florida	1982-11-08	2010-01-01	Sales	6500
67	Cameron	Jones	M	New York	1970-03-14	2003-07-01	Sales	5000
68	Ashley	Moore	F	New York	1980-05-09	2007-04-01	Sales	10000
69	Brandon	Wilson	M	Pennsylvania	1981-05-15	2002-11-01	Sales	7000
70	Brandon	Johnson	M	New York	1976-07-26	2008-02-01	Sales	10000

Retrieving the data from the cursors all at once is to describe the structure of the result. In the practical big data computation, the data usually cannot be retrieved into the memory all at once. In this case, if you need to merge and compute the data returned from multiple servers, you can use the cursors to merge.

The cursor sequence in the merge operation can be composed of various cursors, for example, file object cursor, database cursor, or remote cursor.

In some cases, you will have to combine the data from multiple cursors when using cursor to retrieve data. You may choose the unconditional joining and conjunction with cursor.

### CS.pjoin()

It is to join each cursor  $cs_i$  of cursor sequence  $CS$  in alignment and return the result as a new cursor whose minimum length is indicated by the minimum length of  $cs_i$ . The alignment join will add the fields from multiple cursors to the resulting TSeq one by one.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME, SURNAME, STATE)
3	=demo.cursor("select NAME, ABBR, CAPITAL from STATES")
4	=A2, A3.pjoin()
5	=A4.fetch()

The result of data retrieving in A5 is shown below:

NAME	SURNAME	STATE	NAME	ABBR	CAPITAL
Rebecca	Moore	California	Alabama	AL	Montgomery
Ashley	Wilson	New York	Alaska	AK	Juneau
Rachel	Johnson	New Mexico	Arizona	AZ	Phoenix
Emily	Smith	Texas	Arkansas	AR	Little Rock
Ashley	Smith	Texas	California	CA	Sacramento
Matthew	Johnson	California	Colorado	CO	Denver

As can be seen from the result, the simple horizontal join is just to join the fields in various



cursors. Even if there are duplicate fields such as NAME, they will not be processed in this non-conditional joining. Of the data returned from the result cursor, the number of records will be the minimum number of records in each cursor. During the non-conditional joining, it is required generally that **the number of records in each cursor should be the same**.

### CS.conj@x()

Vertically and unconditionally, join each cursor  $cs_i$  of cursor sequence CS, and return the result as a cursor. The data structure will depend on the first cursor  $cs_1$ . The vertical join will add records from multiple cursors to the resulting TSeq successively. All cursors are often required to have **the same number of fields**. Records will be retrieved from  $cs_i$  in proper order. Note: In order to differentiate it with the conjunction function **A.conj()** for normal sequence, an option **@x** must be added to the conjunction function for cursor sequence.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t(NAME, SURNAME, STATE)
3	=demo.cursor("select NAME, ABBR, CAPITAL from STATES")
4	=[A2,A3].conj@x()
5	=A4.fetch()

The result of data-retrieving in A5 is shown below:

NAME	SURNAME	STATE
Nicole	Smith	South Carolina
Joseph	Smith	Pennsylvania
Alabama	AL	Montgomery
Alaska	AK	Juneau
Arizona	AZ	Phoenix
Arkansas	AR	Little Rock

From the result, we can see that the conjunction of cursors is simply to merge the records one by one, and the data structure is determined by  $cs_1$ . The data retrieved from  $cs_i$  will be populated to fields of  $cs_1$  based on its position.

### 10.4.3 Cursor Join

In addition to the unconditional joining, esProc also allows for setting the equivalent conditions, and associate the data in the cursor with the data from other TSeqs or cursors.

#### join@x( $cs_i:F_i, x_j, \dots; \dots$ )

According to the corresponding fields or values of expression  $x_j, \dots$  connect the cursors  $cs_i, \dots$ . Generate and return the cursor with  $F_i, \dots$  as the field. During retrieving, the value of  $F_i$  is the record retrieved from  $cs_i$ . Because the records of each cursor is retrieved individually, the records in each cursor must not only be ordered for expression  $x_i, \dots$  but also have the same order. Similar to **join()**, you can also use **@f** and **@1** to compute full join or left join in joining cursors together.

For example, in the following case, join the record of each employee with its departmental record:



	A	B	C
1	Administration	4	40000
2	Finance	24	177500
3	HR	19	138000
4	Marketing	99	733500
5	Production	91	663000
6	R&D	29	239000
7	Sales	187	1362500
8	Technology	47	344000
9	=create(DEPT,Count,TotalSalary).record([A1:C8])		
10	=[A1:A8] (file("D:/files/txt/"+"~+.txt"))		
11	=A10.(~.cursor@t())		
12	=A11.merge@x(DEPT)		
13	=join@x1(A12:Employee,DEPT,A9.cursor():DEPT,DEPT)		
14	=A13.fetch()		

Note: We must make sure that the records in each cursor in A12 and A9 are sorted for the corresponding departmental records. In A14, we can check the data retrieved from the cursors all at once:

Employee	DEPT
18	Administration
20	Administration
26	Administration
42	Administration
2	Finance
13	Finance
23	Finance
2	Finance

DEPT	Count	TotalSalary
Finance	24	177500

EID	NAME	SURN...	GEND...	STATE	BIRTH...	HIRE...	DEPT	SALA...
23	Joseph	Turner	M	Californ	1983-0	2003-0	Finance	6000

In many cases, we are just required to query data in two or more tables and display it, according to the relation of certain columns in them. This is similar to what **join** statements do in SQL, which aims not to get a TSeq composed of records, like the result in the above example. To deal with this kind of computation, fields can be added to the cursor directly according to the relation.

**cs.join** (*x:... ,A:y:...z:F,...;x:... ,A:y:...z:F,...;*)

Join the cursor *cs* with RSeq *A* on condition that the results of computing expression *x:...*  on the records in *cs* are the same as the results of computing expression *y:...*  on the records in *A*. If the *y* is omitted, the condition for join will be the primary key of *A*. Once joined, the field *F,...*  will be added to the records of *cs*. The value of this field is the result of computing expression *z* in



A. This function also allows for joining multiple RSeqs with semicolon-delimited parameters.

- ❖ **@i** If no corresponding foreign key in *A* is found for a record in *cs*, then remove the record from *cs*.

In addition, use cursor *A* instead of the RSeq in the join function *cs.join()*. In computing, all records will first be retrieved from the cursor *A* to generate a TSeq. In other words, when using the cursor *A* in the function, you must ensure that there will not be too massive/big data in the cursor *A* to be read into the memory.

This join function is often used for processing big data table. When retrieving records from cursor *cs*, you can associate it with other TSeqs or cursors on certain conditions.

	A
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.new(EID,NAME+" "+SURNAME:FullName,STATE,GENDER)
4	=create(Code,Gender).record(["F","Female","M","Male"])
5	=demo.cursor("select * from STATES")
6	=A3.join(GENDER,A4.Code,Gender:FullGender,STATE,A5.NAME,ABBR:StateAbbr)
7	=A6.fetch()

In the above figure, the code in A3 gets the desired fields from the file object cursor in A2; A6 joins the TSeq in A4 with the database cursor in A5 using the cursor returned from A3, in order to get the full name of the genders of employees as well as the abbreviations of the states where they come from. To illustrate the usage of this function, the data is retrieved from the cursor all at once in A7. The result is as follows:

EID	FullName	STATE	GENDER	FullGender	StateAbbr
1	Rebecca Moore	California	F	Female	CA
2	Ashley Wilson	New York	F	Female	NY
3	Rachel Johnson	New Mexico	F	Female	NM
4	Emily Smith	Texas	F	Female	TX
5	Ashley Smith	Texas	F	Female	TX
6	Matthew Johnson	California	M	Male	CA

In practice, the result sets returned by *cs.join()* are generally quite large, the data will be partly fetched from cursor each time.

#### 10.4.4 Group and Aggregated Summarization

##### *cs.groupx(x)*

Group the records in the cursor *cs* with **group@n()** and return a cursor sequence. Because it is impossible to load the data from the big data table into the memory all at one time, it seems that the big data table cannot be grouped in the same way as that for the normal TSeqs. To serve this purpose, esProc especially provides this grouping function specific to cursor. In grouping, the result of expression *x* is expected to indicate the sequence numbers of groups directly. After grouping, multiple temporary files holding the text data will be generated to record the data from each group, and a sequence composed of the cursors of these files will be returned.

	A	Member
1	=file("D:/files/txt/employee.txt")	com.raqsoft.dm.cursor.BFileCursor@14eefbf
2	=A1.cursor@t()	com.raqsoft.dm.cursor.BFileCursor@1fbd4eb
3	=A2.groupx(if(GENDER=="F",1,2))	

Cursor grouping and summarizing is similar to **A.groups(x:F,...;y:G,...)** in **7.2.3 Group summarization**, it uses the method of **aggregation**.

#### **cs.groups (x:F,...;y:G,...)**

Perform aggregated group summarization on the data from cursor *cs* and return the resulting TSeq. The function's parameters are similar to those of **A.groups()**. It can also use **@o** option and **@n** option. The expression *y* for aggregating is the same as that used in **A.groups()**, and only the simple aggregate functions like **sum/count/max/min/topx** can be used.

The aggregated group summarization for cursors can be used in performing big data grouping and summarizing. The usage is similar to that of grouping and summarizing a TSeq. What's more, it is also used to return the data through multiple cursors. For example, the server returns multiple remote cursors in parallel computing.

Modify the *dfx* in section **10.4.2 Merge and Unconditionally Join Cursor Sequences**, first merge the data, then summarize and group the data of returned cursor list to compute the number of male and female employees in each state:

	A
1	[Finance,Sales,HR,Administration,Marketing,Production,R&D,Technology]
2	=A1.(file("D:/files/txt/"+~+".txt"))
3	=A2.(~.cursor@t())
4	=A3.conj@x().groups(STATE,count(GENDER=="M"):Male,count(GENDER=="F"):Female)

After performing aggregated grouping and summarizing on the returned data of remote cursor list, you can see the result in A4:

STATE	Male	Female
Maryland	3	1
Massachusetts	2	6
Michigan	16	12
Minnesota	1	2
Mississippi	4	0
Missouri	2	7

### **10.4.5 Sort Big Data Cursors**

#### **cs.sortx(x...;n)**

Sort big data cursor *cs* and return the result as a cursor. Parameter *n* indicates the size of buffer. During creating the cursor, retrieve *n* records for sorting from *cs* at one time. The sorted result will be cached as the temporary file each time. Perform the operation in cycles. When data is retrieved from the result cursor, the retrieval will be performed on the file cursors of all temporary files



A	
1	=file("D:/files/txt/employee.txt")
2	=A1.cursor@t()
3	=A2.sortx(STATE;50)
4	=A3.fetch()

EID	NAME	SURNAME	GEN...	STATE	BIR...	HIR...	DEPT	SAL...
102	Christian	Smith	M	Alabama	1972	2000	Sales	12000
282	Kayla	Davis	F	Alabama	1970	2006	Sales	6500
419	Sara	Davis	F	Alabama	1984	2004	Sales	5000
477	Lauren	Smith	F	Alabama	1975	2007	Market	6500
100	Daniella	Smith	F	Arizona	1986	2000	Teacher	7000

The usage of **cs.sortx()** is the same as that of normal **T.sort()**, and the same result will be returned. During sorting a table holding massive amount of data, usually the records cannot be retrieved entirely due to the limited size of the memory, or the memory cannot hold the result table. In this case, the massive data sorting function **cs.sortx()** is a good choice.

## 11. Authorization

### 11.1 Registration code

Normally, the edition of esProc is controlled by registration code. Please refer to our **official website** for the differences between editions and how to get a registration code.

After getting the registration code, you can register in the esProc. The operation steps are as follows:

- Open IDE, and click the **Help -> Registration** menu item:

Registration

Registration code:

Product name: esProc 3.1

Edition: Free Edition

Serial No.: 0

Expiration date: For ever

User information

Buttons: Scan, Register, Check points, Close, More Editions

After entering the product registration code, click **Scan** button to see if the registration information is correct. If it is correct, click the **Register** button to start registering. The registration code for free edition is empty.

### 11.2 Cellset Description and Cell Tips

### 11.2.1 Description on Cellset

In esProc, you can set the cellset description in the cellset file.

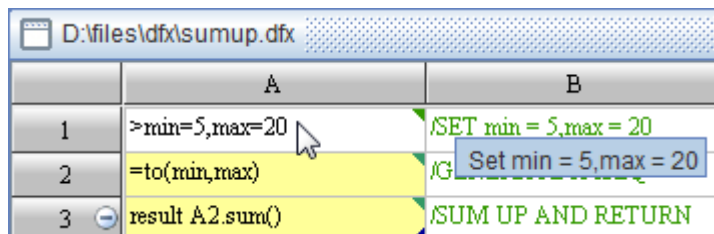
In the **Tool** option on the menu bar, click the **CellSet Description** button. The cellset description window will appear for you to view or set the cellset description.

### 11.2.2 Cell Tips

In the cell, you can add tips and comments to the computation in the cell.

Right click the selected cell and select **Tips** on the right-click menu. The cell tips window will appear for you to set the cell tips in the window.

In esProc, if tips are set in the cell, then a green triangle will appear on the top right corner of the cell. If the cursor hovers on the cell with tips, then the tips that you've set in the cell will appear next to the cell indicated by the mouse.



	A	B
1	>min=5,max=20	/SET min = 5,max = 20
2	=to(min,max)	/G Set min = 5,max = 20
3	result A2.sum()	/SUM UP AND RETURN

In order to illustrate clearly, the cells of column B are filled with comments. Note that the comments are different from the cell tips.

## 11.3 File Encryption

### 11.3.1 Permission

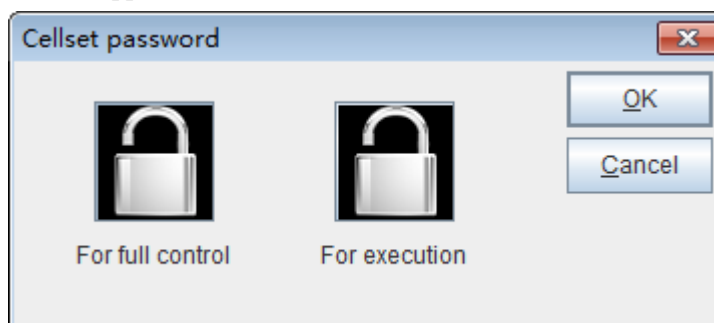
In order to prevent the unauthorized view of or access to the cellset, esProc has developed the cellset encryption with the following two levels of authorizations:

**Execution Right:** You can set the parameters and execute the cellset, check the cellset descriptions and the cell hints, but you are not allowed to change the information. When you are only allowed to execute, you cannot view any string in the cell and cellset variables in the cellset. After execution, you can view the cell value.

**Full Control:** You can perform any operation on the cellset.

### 11.3.2 Password Setup

In the **Tool** option of the menu bar, click the **Cellset password** button. The Cellset password window will appear:



Click **For Full Control** button, you can set the full control password.

Click **For Execution** button, you can set the execution password:

**Note:**



The privilege can be nullified with the corresponding password for such privilege. Once the file is opened under the full control privilege, you can rescind the full control privilege and execution privilege for this file. If the file is opened with the execution privilege, you can only nullify the execution privilege password for it.

### 11.3.3 Control of Authority

If a password is set for the cellset, then **Input cellset password** window will appear to prompt you to enter the password when you open cellset in esProc.

If the password is correct, then you will open the cellset according to the corresponding privilege granted by such password. If the password of the execution privilege is the same as that of the full control privilege, then the cellset will be opened according to the highest level of privilege, that is, full control privilege.

If you've only got the **Execution Privilege**, then you are unable to view the cell string as well as even the comments in the cell or the cellset variables. However, you can view the cell tips:

D:\files\dfx\sumup.dfx		
	A	B
1		
2		Set min = 5,max = 20
3		

Then, you can compute the cellset normally. If the cellset requires the parameter being set before computation, you can set the execution parameter. After running the program, you can view the resulting cell value:

D:\files\dfx\sumup.dfx		
	A	B
1		
2		Set min = 5,max = 20
3		Generate a seq

Having the execution privilege, you can still run the cell and check the result.

Member
5
6
7
8
9

With the full control privilege, you can view all contents in the cellset and take any actions on the cellset:

D:\files\dfx\sumup.dfx		
	A	B
1	>min=5,max=20	/SET min = 5,max = 20
2	=to(min,max)	/G Set min = 5,max = 20
3	result A2.sum()	/SUM UP AND RETURN