TRUE

TEMPORAL REASONING UNIVERSAL ELABORATION

True System dynamics software

MANUAL Part 05

True functions

Release 2014/03/03

www.true-world.com

Contents

I - TRUE FUNCTIONS

A 'Procedure' action (or equation) contains code to calculate its return value.
The code written in Wlanguage is dynamically compiled when the action is executed.

A) TRUE specific functions

True specific functions are :

- Aresult()
- Svalue()
- Mvalue()
- Fvalue()
- ValueWrite()
- ValueRead()
- Options()
- ReStartAt()
- ReStartStat()
- SetGlobalProcedure()
- CallGlobalProcedure()
- `WITHLOOKUP`()
- `ClearGlobalVar`()

They allow to:

- display results,
- read the current or passed values of stocks, mirror stocks or flows,
- store values in global pseudo-variables,
- enable procedure end options,
- initialize retro-calculation,
- initialize and call a global procedure,
- clear the global variables

Functions Svalue(), Mvalue() and Fvalue() :

- If these functions are called in one action, there chronology parameter should be set properly.

1. Aresult() function

- Usage: display a string in an 'Aresult(x)' window
- These strings can be displayed in Frames (see manual: **Man 08-Frames**)

❖ Example 1

```
//y is the return value of the procedure
y=100 * (50/35.6)
Aresult(1,'y = ' , y , '(y value in Action 1')
```

❖ Example 2

```
ch is a string
ch='Hello, we are the  '+DateSys()+', it is '+Timesys()
Aresult(1,ch)
//Datesys() returns the date, Timesys() return the time
```

❖ Example 3

```
Aresult(2,'Ok')
```

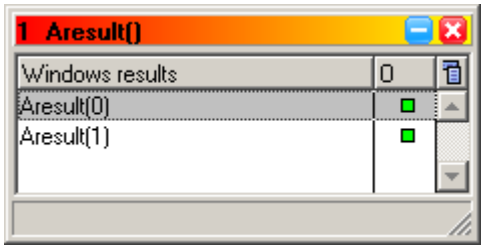
- Syntax
  Aresult (<Number of the window Aresult()>,<Text1>,[<Text2>],[<Text3>],…,…)

- Parameter detail
  - <Number of the window Aresult()> : indicates the number of the Aresult(x) window
    See Man08-Frames : Results chapter
  - <Text1,2,…> : string of characters : the message

Open the Aresult(x) window

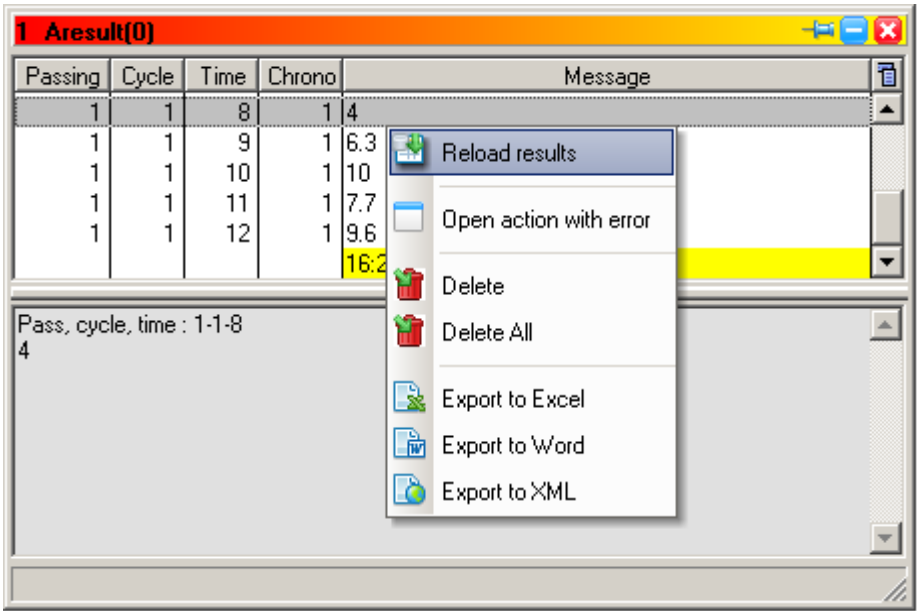
- Select the 'Aresult()' option from the 'Windows' menu

'Aresult()' window contains the list of the 'Aresult(x)' windows



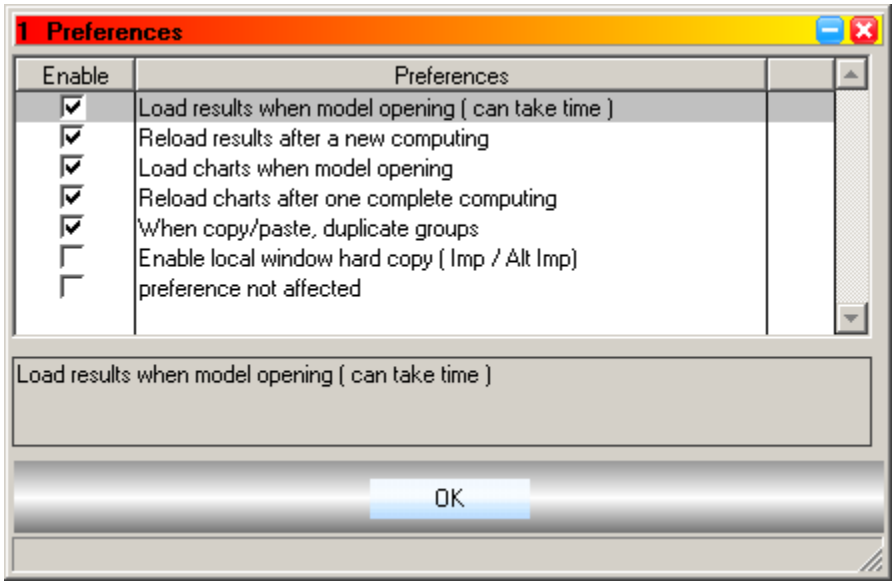
- click on one of the green leds to open an Aresult(x) window.

'Results(0)' window



- Yellow line : indicates a new set of results and displays the time.
- 'Passing','Cycle','Time' columns : indicates the number of passage, cycle, and time of creating the message .
- 'Chrono' column : indicates the chronology of the action when creating the message.
- 'Message' column : string passed as parameter to the function Aresult().

To load the results when model opening, check the 'Load results when the model opening' option, from the 'Preferences' window, in main menu, 'Options'.
'Preferences' window



2. Svalue() function

- Usage : returns a stock value

- Example 1

```
//y is the action return value
y = Svalue('b')
//y = value of the stock b for the current cycle c and current time t
```

- Example 2

```
//y is the action return value
//c is the current cycle
//t is current time unit
c_, t_ are int
c_ = 4; t_ = 10
IF c_ = c AND t_ = t THEN y = Svalue('b',c_,t_)
//y = value of the stock b for cycle c_=4 and time t_ = 10
```

- Example 3

```
//y is the return value of the procedure
//nbt is the current number of time, for all cycles
nbt_ is int
nbt_ = nbt-10
IF nbt_ > 0 THEN y = Svalue('b',False,nbt_)
//y = value of the stock b for time nbt - 10
```

- First syntax

Svalue (<Name of the stock>,[<Cycle>,<Time>])

- Parameter details
  - <Name of the stock> : Character string, the name of the stock
  - <Cycle> : Optional integer, cycle for which the value must be obtained
  - <Time> : Optional integer, time unit for which the value must be obtained
  - If the parameters <Cycle> and <Time> do not correspond to a valid value, the function will return 0

- Second syntax

Svalue (<Name of the stock>,[<False,<Number of time>])

- Parameter details
  - <Name of the stock> : Character string, the name of the stock
  - <False> : Optional boolean, replaces <Cycle> parameter, and indicates that the parameter <Number of time> must correspond to the variable nbt, which represents the number of time for all cycles
  - If parameter <Number of time> does not correspond to a valid value, the function will return 0

- **Important note for Svalue()**:

  When an action calls the Svalue() function to retrieve a stock value, the '**Chrono**' parameter of this action must be **higher** than the last action that modifies the stock, else the retrieved value will not be the last stock value, but a **delayed** value.

  When an action of type **Rate** (Vensim) modifies a stock, the last value of the linked mirror stock **can not** be retrieved by an action.
  Indeed, the value returned by an action of type **Rate** affects the stocks at the **end** of the current time unit, after **all the other actions have been executed**.

  If the Svalue() function is called with parameters initialized to retrieve a delayed stock value, the value of the '**Chrono**' parameter of the action will not affect the stock value returned by Svalue()

3. Mvalue() function

- Usage : returns a mirror stock value
  - The principle is the same as the function `Svalue()` principle

- Example 1

```
//y is the return value of the procedure
y = Mvalue('m')
//y = value of the mirror stock for current cycle c and time t
```

- Example 2

```
//c is the current cycle
//t is the current time unit
c_, t_ are int
c_ = 4; t_ = 10
IF c_ = c AND t_ = t THEN y = Mvalue('b', c_, t_)
//y = value of the stock b for cycle c_=4 and time t= 10
```

- Example 3

```
//nbt is the current number of time, cumulated for all cycles
nbt_ is int
nbt_ = nbt-10
IF nbt_ > 0 THEN y = Mvalue('b', False, nbt_)
//y = value of the stock b for time t =  nbt-10
```

- First syntax

Mvalue (<Name of the mirror stock>,[<Cycle>,<Time>])

- Parameter details
  - <Name of the mirror stock> : Character string, indicates the name of the mirror object.
  - <Cycle> : Optional integer, indicates the cycle for which the value must be obtained.
  - <Time> : Optional integer, indicates the unit of time for which the value must be obtained.
  - If the parameters <Cycle> and <Time> do not correspond to a valid value, the function will return 0

- Second syntax

Mvalue (<Name of the mirror stock>,[<False>,<Number of time>])

- Parameter details
  - <Name of the mirror stock>: Character string, is the mirror's object name.
  - <False>: Optional boolean, replaces the <Cycle> parameter, and indicates that the parameter <Number of time> must correspond to the variable nbt.
  - <Number of time>: Optional integer, which is equal to the number of time for all cycles.
  - If the parameter <Number of time> doesn't correspond to a valid unit of time, the function will return 0.

- **Important note for Mvalue()**:

  When an action calls the Mvalue() function to retrieve a mirror stock value, the '**Chrono**' parameter of this action must be higher than the last action that modifies the mirror stock, else the retrieved value will not be the last value of the mirror stock, but a **delayed** value.

  When an action of type **Rate** (Vensim) modifies a stock, the last value of the linked mirror stock, can not be retrieved by an action.
  Indeed, the value returned by an action of type **Rate** affects stocks at the end of the current time unit, after **all the other actions have been executed**.

  If the Mvalue() function is called with parameters initialized to retrieve a delayed mirror stock value, the value of the '**Chrono**' parameter of the action will not affect the mirror stock value returned by Mvalue()

4. Fvalue() function

- Usage : returns the value of transfering a flow, which represents the sum of the values transferred by all the actions of this flow.

- Example 1

```
//y is the return value of the procedure
y = Fvalue('ab')
//y = transfer value of flow 'ab' for current cycle c and current time
```

- Example 2

```
//y is the return value of the procedure
//c is the current cycle
//t is the current time unit
c_, t_ are int
c_ = 4; t_ = 10
IF c_ <= c AND t_ <= t THEN y = Fvalue('ab', c_, t_)
//y = transfer value of the flow 'ab' for cycle c_=4 and time t_= 10
```

- Example 3

```
//y is the return value of the procedure
//nbt is the current number of time, cumulated for all cycles
nbt_ is int
nbt_ = nbt-10
IF nbt_ > 0 THEN y = Fvalue('ab', False, nbt_)
//y = transfer value of flow 'ab' for number of time nbt - 10
```

- First syntax

  Fvalue (<Name of the flow>,[<Cycle>,<Time>])

- Parameter details
  - <Name of the flow> : Character string, indicates the name of the flow
  - <Cycle> : Optional integer, indicates the cycle for which the transfer value must be obtained
  - <Time> : Optional integer, indicates the unit of time for which the transfer value must be obtained
  - If the parameters <Cycle> and <Time> do not correspond to a valid value, the function will return 0

- Second syntax

  Fvalue (<Name of the flow>,[<False>,<Number of time>])

- Parameters details
  - <Name of the flow> : Character string, indicates the name of the flow
  - <False> : Optional boolean, replaces the parameter <Cycle> and indicates that the parameter <Number of time> must correspond to the variable nbt
  - If the parameter <Number of time> does not correspond to a valid value, the function will return 0

- **Important note for Fvalue()**:

  When an action calls the **Fvalue()** function to retrieve a flow value, the '**Chrono**' parameter of this action must be higher than the last action that modifies the flow, else the retrieved value will not be the last flow value, but a **delayed** value.

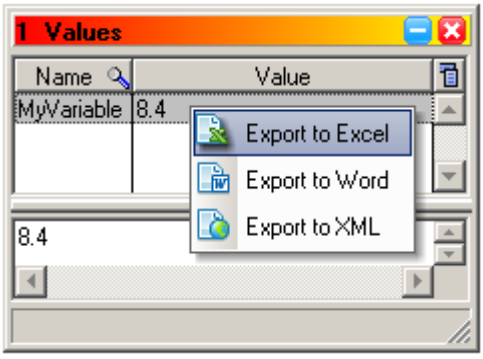
  If the **Fvalue()** function is called with parameters initialized to retrieve a delayed flow value, the value of the '**Chrono**' parameter of the action will not affect the flow value returned by **Fvalue()**

5. ValueWrite() function

- Usage : memorises in a file from the data base, a value corresponding to a name
  This function allows declaration and initialisation of pseudo-variables to be simulated.
  The values can be read with the function ValueRead()

The values of the pseudo-variables are displayed in the 'Values' window, after computing.

'Values' window, opened by selecting the 'ValueWrite' option from the 'Windows' menu



➢ Note : when locally computing the procedure (with the 'F3' key in a 'Procedure' window), the value memorised by global computing are not affected

❖ Examples

```
//y is the return value of the procedure
ValueWrite('ab', y)
// y value is memorised in the pseudo-variable named 'ab'

ValueWrite('product number', 300)
//the value 300 is memorised in the pseudo-variable named 'product number'

ValueWrite('name', 'SMITH')
//the string 'SMITH' is memorised  in the pseudo-variable 'name'

ValueWrite('dim')
//value of the pseudo-variable named 'dim' is deleted
```

❑ First syntax

ValueWrite(<Name of the pseudo-variable>,[<Value of the pseudo-variable>])

❑ Parameter details

- <Name of the pseudo-variable>: integer or character string, indicates the name of the pseudo-variable, limited to 40 characters
- <Value of the pseudo-variable> optional, indicates the value of the pseudo-variable, limited to 100 characters

6. ValueRead() function

- Usage : reads the value of a pseudo-variable value, memorised in the data base by the function ValueWrite().

  The pseudo-variable values are displayed in the 'Values' window after computing.

- Note : when locally computing the procedure (with the 'F3' key in 'Procedure' window), the value memorised by global computing are not affected

- Example 1

```
//y is the return value of the procedure
y = ValueRead('ab')
//y is initialized with the value of the pseudo-variable named 'ab'
```

- Example 2

```
Amount is currency
Amount = ValueRead('amount')
//Amount is initialized with the value of the pseudo-variable named 'amount'
```

- Example 3

```
Account is a currency
Account = ValueRead('amount')
//variable Account is initialized with value of the pseudo-variable 'amount'
```

- Syntax

  <Result> = ValueRead (<Name of the pseudo-variable>)

- Parameter details

  - <Result>: character string containing the value memorised under the name <Name of the pseudo-variable>, limited to 100 characters

  - <Name of the pseudo-variable>: integer or character string specifying the name of the pseudo-variable, limited to 40 characters

7. Options() function

- Usage: initializes one or more functionalities executed at the end of the procedure

- Example 1
  //y is the return value of the procedure
  IF y=50 THEN Options(3)

- Example 2
  //y is the return value of the procedure
  IF y=50 THEN Options(3,5,6,7)

- First syntax
  Options (<Number of the option>,[<Number of the option>,<Number of the option>,…..])

- Parameter details
  - <Number of the option>: integer, indicates the number of the option to be executed. Seven options can't be simultaneously defined.
  - The functions are described in the chapter 'Actions', 'Procedures', 'Options for ending a procedure'.

8. ReStartAt() function

- Usage: this function allows the computation to restart at a given time unit.

  An action which wants to make retro-computing, must open a gate towards the past with the function ReStartAt(). It returns the number of the opened gate.

  The number of the gate is associated to the current action and to the current 'number of time' (nbt)

  There cannot be more than one gate for the same action and the same nbt

  It is associated a counter of passing by the gate into the past.

  This counter is incremented when passing by the gate into the past.

  The status of the gates of the model can be retrieved by the function ReStartStat()

- Example 1

```
//The action 'Act' calls the function ReStartAt() for the nbt 50
//The future number of the gate number returned by this function is 1.
//The gate number 1 will be created after exiting the action,
//and the first passage by this action will be set to 1.

//Retro-computing will be done for the temporal parameters c and t
//passed to the function, ReStartAt(c,t).

//When the action 'Act' comes back to nbt 50 and recalls the
//ReStartAt() function, the return toward the past will be done by the
//previous created gate, the counter of passage will be incremented and
//will be set to 2.

//If the action 'Act', at nbt 50, doesn't recall the function ReStartAt()
//the gate number 1 will be deleted, with the passing counter.

//c is the current cycle, t is the current unit time
gate is int
c_, t_ are int

IF c = 5 AND t = 10 //nbt = 50
      IF y <> <Waited Result>
            c_ = 2; t_ = 5
            gate = ReStartAt(c_,t_)
//computing will be restarted at time unit 5 of the cycle 2, after exiting the current action
      END
END
```

❖ Example 2

```
//Opening a gate towards the past when a gate is already open:

//The action 'Act' has opened the gate number 1 and has done 3 passages by this gate.

//The action is coming back towards gate number1, but before reach this last one, its opens a new gate.

//The number of the new gate will be 2, and the associated passing counter will be 1.

//It is the same way when opening new gates by other actions.

//y is the return value of the procedure
//nbt is the current number of unit time, for all cycles

nbt_ is int
gate is int

IF nbt = 30
      IF y <> <Waited Result>
            nbt_ = 20
            gate = ReStartAt(False,nbt_)
      END
END
```

- First syntax

  <Result>=ReStartAt(<Cycle>,<Time>,[<max_passing>],[<max_action_gate>],
  [<max_flow_gate>],<max_global_gate>])

- Parameter details

  - <Result> : number of the created gate, or already created gate, by the action, for the current 'number of time' (nbt)
    If the gate already exists, passing counter is incremented

  - <Cycle> : integer which indicates for which cycle you want to start a retro-calculation

  - <Time> : integer which indicates for which unit time you want to start a retro-calculation
    If parameters <Cycle> and <Time> do not correspond to a valid unit time, function returns 0
    This parameters must correspond to a current or past unit of time

  - <max_passing> : optional integer which limits the number of passages by this gate

  - <max_action_gate> : optional integer which limits the numbers of gate which can be opened by the current action
    Indeed, other gates can be opened for other 'numbers of time ' (nbt), by the current action

  - <max_flow_gate> : optional integer which limits the number of gates which can be opened by the current flow
    Indeed, other gates can be opened for other 'numbers of time ' (nbt), by the flow containing the current action

  - <max_global_gate> : optional integer which limits the number of gates which can be opened by all the actions of the model

➢ Note: if one of the limits is reached, the gate is not created
If the gate already exists, it is deleted, and the passage towards the past is not done

- Second syntax

  <Resultat>=ReStartAt(<False>,<Number of time>,[<max_passing>],[<max_action_gate>],[<max_flow_gate>],[<max_global_gate>])

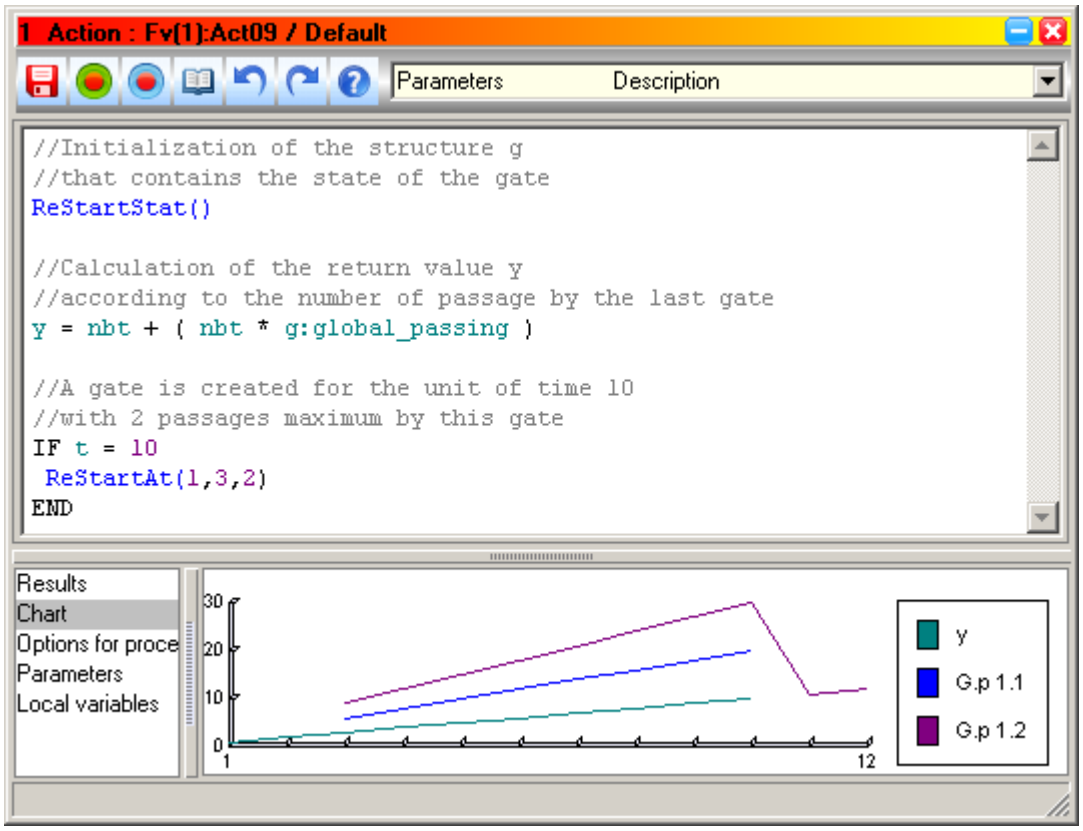
- Parameter details

  - <False> : boolean, replaces the parameter <Cycle>, and indicates that the parameter <Number of time> must correspond to the variable nbt, which represents the number of time for all cycles

  - <Number of time> : integer, indicates for which 'number of time ' you want to start retro-calculation
    If the parameter <Number of time> doesn't correspond to a valid value, the function will return 0

  - See the previous syntax for the other parameters

'Action' window with a 'ReStartAt()' function



➢ Note : the caption of the graph 'G.p 1.2' means 'Gate 1, passage 2'

9. ReStartStat() function

- Usage : the function ReStartStat() returns the status of the gates created by the function ReStartAt()

  It initializes the global structure g which it is accessible in the procedure of the current action

  She can also return the current number of passage for a specified gate.

❖ Example 1

```
// To know the status of the gates
passing is int

ReStartStat()

y= g:global_passing // get the current number of passage for the last opened gate in the model

IF y<>0
    <Action>// if a passing exists, you can start an action

//get the number of gates for the current action
    y = Dimension(g:t_action_gate)

// get the number of passages for gate number 3
    IF y<=3
      passing=g:t_action_gate[3]
    END

END
```

❖ Example 2

```
// To know the number of passages of one gate
gate,passing is int
gate = 3

// get the number of passages for the gate number 3
passing = ReStartStat(gate)

// start actions according to the current number of passage
SWITCH passing
    CASE  1 : <Action 1 >
    CASE  2 : <Action 2 >
END
```

- First syntax

  ReStartStat()

  The global structure g is initialised by the function ReStartStat()
  The members of this structure are :

  - g:action_gate = number of the gate opened by the current action and the current nbt, if exists
  - g:action_passing = number of passing for the gate opened by the current action and the current nbt, if exists
  - g:nb_action_gate = number of gate opened in the current action
  - g:nb_flow_gate = number of gate opened in the current flow
  - nb_global_gate = number of gate opened in the model
  - g:global_passing = last number of passing for the last gate created in the model
  - g:f_action_gate = boolean, true if a gate was opened in the current action for the current nbt
  - g:f_flow_gate = boolean, true if a gate was opened in the current flow for the current nbt
  - g:f_global_gate = boolean, true if a gate was opened in the model for the current nbt
  - g:t_action_gate = array of the numbers of the gates opened in the current action
  - g:t_flow_gate = array of the numbers of the gates opened in the current flow

➢ Note : the dimension of the array g:t_action_gate and g:t_flow_gate can be returned by the function Dimension() of the WLanguage, see example 2 above

- Second syntax

  <Result>=ReStartStat(<Gate number>)

- Parameter details

  - <Result> : integer containing the number of passages for the gate <Gate number>
    If the gate doesn't exist, the function return 0
  - <Gate nupmber> : integer containing the gate for which you want to obtain the number of passages

10. SetGlobalProcedure() function

- Usage : declares a new global procedure.
  - The code of the procedure containing this function may be called by another procedure with the function CallGlobalProcedure()

- Example

```
SetGlobalProcedure('my global procedure')
 //... following the code of the procedure code...
 //y = sin(t)...

 //The code of this procedure may be called by another procedure with the function
 CallGlobalProcedure('my global procedure')
```

- Syntax

  SetGlobalProcedure (<Name of the global procedure>)

- Parameter detail

  - <Name of the global procedure> : string of characters, indicates the name of the global procedure.

11. CallGlobalProcedure() function

- Usage : call a global procedure.
  - The code of the global procedure declared by the function SetGlobalProcedure() may be called by the function CallGlobalProcedure()

- Example

```
CallGlobalProcedure('my global procedure')

//The code of the procedure containing the function SetGlobalProcedure('my global procedure')
is called and executed in the procedure containing the function
CallGlobalProcedure('my global procedure')
```

- Syntax

```
CallGlobalProcedure (<Name of the global procedure>)
```

- Note : when the function CallGlobalProcedure() is present in the code of a procedure, all the codes from this procedure, before computing, are replaced by the code of the procedure that contains the function SetGlobalProcedure()

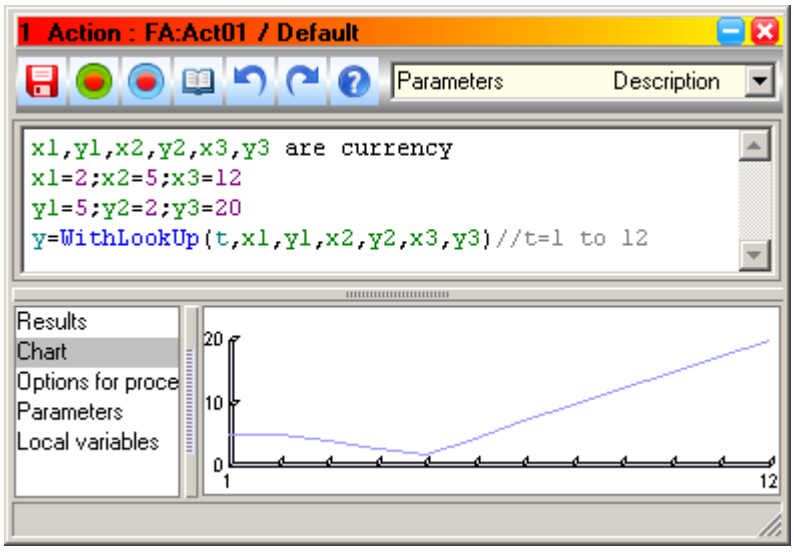
- Parameter details

  - <Name of the global procedure> : string of characters, indicates the name of the global procedure.

12. WITHLOOKUP() function

- Usage : specifies a nonlinear relationship between the input x and the output by passing the input through a series of x,y pairs specified as numbers

- Example



- Syntax
  <Result> = WithLookUp (<InputVar>,<x1>,<y1>,<x2>,<y2>,<xn>,<yn>,...,...)

- Parameter details

  - <Result>: currency

  - <x,y>: pairs of currency

B) Functions added to the Wlanguage

List of added functions :

- Rand()
- Srand()
- NormalMS()
- NormalMSMM()
- NormalMM()
- MeanSigma()
- SawtoothN()
- SawtoothP()
- Triangle()
- TriangleCos()
- TriangleS()
- SinusPulse()
- Sign()

1. Rand() function

- Usage
  - The function Rand() returns an random number between 0 and 1/32767

- Example

```
result is real
result=Rand()
//result will be between 0 and 1/32767
```

- Syntax
  <Result>=Rand()

- Parameter detail
  <Result> : real, return value between 0 and 1/32767

- Notes
  Le 'pool' in which the random number is 'chosen' can be initialized by the function Srand()

  If the function Srand() is not called, the random values generated by the function Rand() will be the same every time the program will run

  The function Rand() is used in the functions NormalMS(), NormalMSMM(), NormalMM()

2. Srand() function

- Usage
  - Initializes the random generator for the Rand() function

- Example

```
n is int
n = 3
Srand(n)
 //Initializes the generator of random number fur the Rand() function
```

- Syntax
  Srand([<n>])

- Parameter detail
  <n> : optional integer, base number of the generator of random number for the Rand() function

  If this parameter is1 the generator is reset
  If this parameter is different than 1 or if it is not specified, the generator uses the system time to generate a random number

- Notes
  Random number

  If the Srand() function is not called, the random values generated by the function Rand() will the same every time the program will run

  The Rand() function is used in the NormalMS(), NormalMSMM(), NormalMM() functions

3. NormalMS() function

- Usage
  - The NorMalMS() function returns a pseudo-random number according to the normal distribution

- Example

```
mean, sigma, normal are real
mean= 1000 ( mean of the gauss curve)
sigma= 250 ( standard deviation, inflexion point of the gauss curve)
normal=NormalMS(mean, sigma)
 //get a pseudo-random value according the normal distribution for a value around 1000 and a
 standard deviation = 250
```

- Syntax
  <Result>=NormalMS(<mean>,<sigma>)

- Parameter details
  <Result >: real, pseudo-random number according to the normal distribution

  <mean>: real, mean of the gauss curve

  <sigma>: real, standard deviation, inflexion point of the gauss curve
  If this parameter is <=0 the function will return 0

- Notes
  Random number
  The Rand() function is used in the NormalMS(), NormalMSMM(), NormalMM() functions
  The Srand() function determines the random values returned by the Rand() functions

4. NormalMSMM() function

- Usage
  The NorMalMSMM() function returns a pseudo-random number according to the normal distribution, between a minimum and a maximum value

- Example

```
mean, sigma, minimum, maximum, normal are real
mean= 1000 ( mean of the gauss curve)
sigma= 250 ( standard deviation)
minimum= 500 ( minimum value)
maximum= 1500 (  maximum value )
normal=NormalMS(mean, sigma, minimum, maximum)
 //get a pseudo-random value according to the normal distribution for a value around 1000 and a
 standard deviation = 250, between 500 and 1500
```

- Syntax
  <Result>=NormalMSMM(<mean>,<sigma>,<min>,<max>[<reject>])

- Parameter details
  <Result >: real, pseudo-random number according to the normal distribution

  <mean>: real, mean of the gauss curve

  <sigma>: real, standard deviation, inflexion point of the gauss curve
  If this parameter is <=0 the function will return 0

  <min>: real, minimum of the return value

  <max>: real, maximum of the return value
  If <max> < <min> then <min> will be the maximum and <max> the minimum

  <reject>: optional integer, number of values out of the minimum and maximum

- Notes
  Random number
  The Rand() function is used in the NormalMS(), NormalMSMM(), NormalMM() functions
  The Srand() function determines the random values returned by the Rand() function

5. NormalMM() function

- Usage
  - The NorMalMM() function returns a pseudo-random number according to the normal distribution, between a minimum and a maximum value, for a mean and standard deviation calculated according the minimum and the maximum values

- Example

```
minimum, maximum, normal are real
minimum= 500 ( minimum value)
maximum= 1500 ( maximum value)
normal=NormalMM(minimum, maximum)
 //get a pseudo-random value according to the normal distribution for a value between 500 and
 1500
 for a mean= 1000 and a standard deviation = 204,1241452319
```

- Syntax
  <Result>=NormalMM(<min>,<max>,[<reject>])

- Parameter details
  <Result >: real, pseudo-random number according to the normal distribution

  <min>: real, minimum of the return value

  <max>: real, maximum of the return value
  If <max> < <min> then <min> will be the maximum and <max> the minimum

  <reject>: optional integer, number of values out of the minimum and maximum

- Notes
  Random number
  The Rand() function is used in the NormalMS(), NormalMSMM(), NormalMM() functions
  The Srand() function determines the random values returned by the Rand() function

6. MeanSigma() function

- Usage
  - The MeanSigma() function calculates the mean and the standard deviation according to the minimum and maximum values of one pseudo-random number to be generated by the NormalMS() function according to the normal distribution

- Example

```
minimum, maximum, mean, sigma, normal are real
minimum= 500 ( minimum value )
maximum= 1500 ( maximum value )

MeanSigma(minimum, maximum, mean, sigma)
mean = 1000 ( value calculated  by the function )
sigma = 204,1241452319  ( value calculated  by the function  )

normal=NormalMS(mean, sigma)
 //get a pseudo-random value according to the normal distribution for a value around 1000 and a
 standard deviation =  204,1241452319
```

- Syntax
  MeanSigma(<min>,<max>,<mean>,<sigma>)

- Parameter details

  <min>: real, minimum of the return value

  <max>: real, maximum of the return value
  If <max> < <min> then <min> will be the maximum and <max> the minimum

  <mean>: real, mean of the gauss curve

  <sigma>: real, standard deviation, inflexion point of the gauss curve

- Notes
  Random number
  The Rand() function is used in the NormalMS(), NormalMSMM(), NormalMM() functions
  The Srand() function determines the random values returned by the Rand() function

7. SawtoothP(), SawtoothP(), Triangle(), TriangleCos() functions

- Usage
  Theses functions returns a value = f(t) between -1 and 1 or -0,5 and 0,5

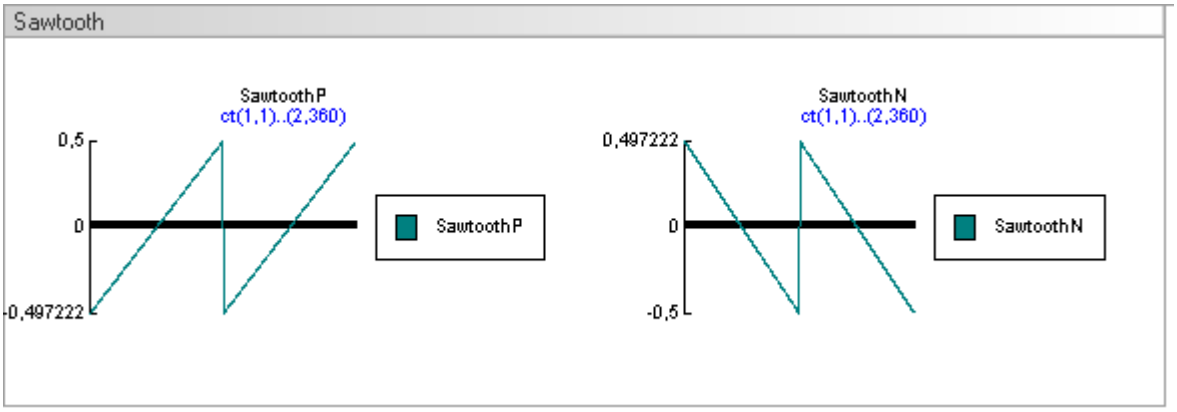
- Example

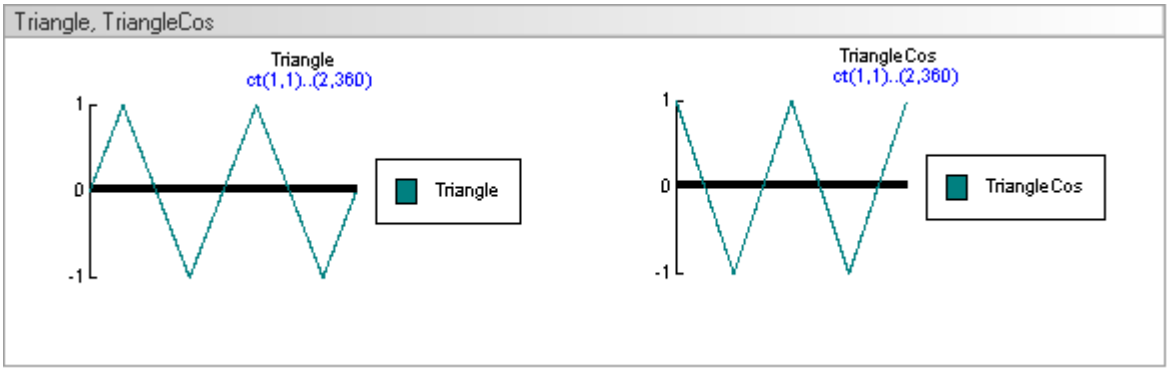
```
result is currency
result=SawtoothP(t) //t between 0 and 360
//result between -0,5 and 0,5

result=Triangle(t)  //t between 0 and 360
//result between -1 and 1
```

- Syntax
  <Result>=SawtoothP()

- Parameter detail
  <Result> : currency, return value

- Examples for t=0 to 360





8. TriangleS() function

- Usage
  Theses functions returns a value = f(t,centre) between -0,5 and 0,5

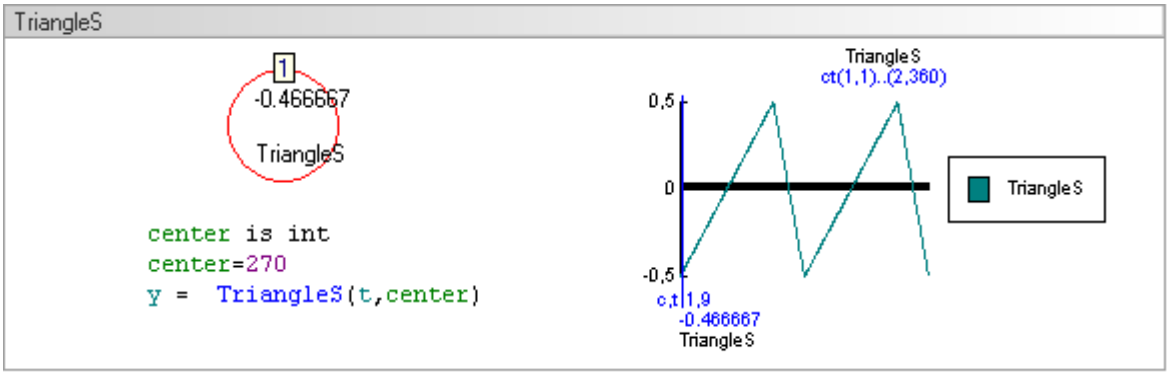
- Example

```
angle, centre is int
y=TriangleS(angle, centre) //angle and centre between 0 and 360
//result between -0,5 and 0,5
```

- Syntax
  <Result>=TriangleS(<angle>,<centre>)

- Parameter detail
  <Result> : currency, return value
  <angle> : int, angle value : 0 to 360
  <centre> : int, centre of signal : 0 to 360

- Chart



9. SinusPulse() function

- Usage
  - This function returns a value = f(t,centre,width) between 0 and 1

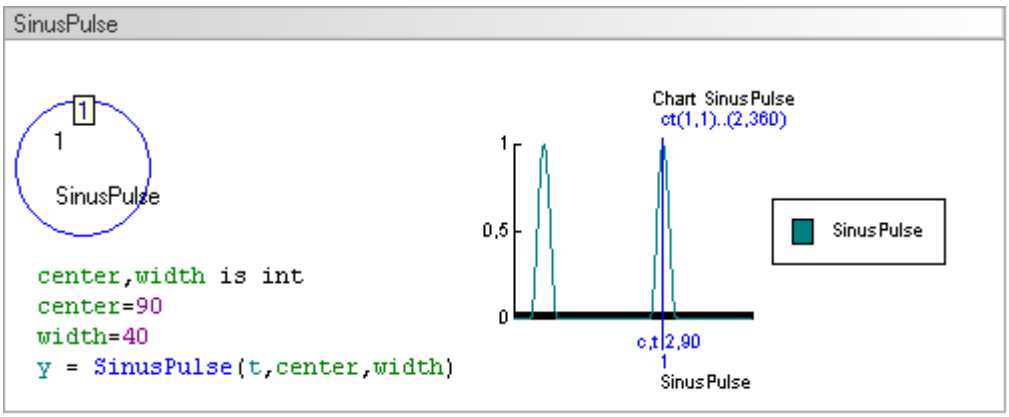
- Example

```
angle, centre, width is int
y=SinusPulse(angle, centre,width) //angle and centre between 0 and 360
//result between 0 and 1
```

- Syntax
  <Result>=SinusPulse(<angle>,<centre>,<width>)

- Parameter detail
  <Result> : currency, return value
  <angle> : int, angle value : 0 to 360
  <centre> : int, centre of signal : 0 to 360
  <width> : int, width of the pulse

- Chart



10. Sign() function

- Usage
  - This function returns the sign of one value, -1,0 or 1

- Example

```
v is currency = -50
y=Sign(v)
//y = -1
```

- Syntax
  <Result>=Sign(<v>)

- Parameter detail
  <Result> : int, indicates the return value (-1,0,1)
  <v> : currency or int....