Ecosimpro/Proosis modelling language

Functions, arrays, tables, marcro expansions and enumerations

Rogelio Mazaeda, César de Prada

- Functions
- Other types of variables
- Tables
- Arrays and enumerates
- Macro expansions

```
--Returns the angle of the pendulum of length
--when the horizontal displacement is x
FUNCTION REAL findAngle(REAL x, REAL L)

   DECLS
      REAL angle

   BODY
      IF (x != 0) THEN
         angle = asin(x/L)
      ELSE
         angle = 0
      END IF

   RETURN angle

END FUNCTION
```

- Functions introduce the possibility of inserting into the model the traditional sequential programming techniques.
- Functions statements are executed in the order in which they appear
- There are traditional assigments of variables
- Control flow structures can be used: (IF) and loops (WHILE and FOR)

  Syntaxis:

```
WHILE (cond_boolean)
      sentencias_secuenciales
END WHILE
FOR (i IN 1,5)
      k[i]=0.0
END FOR
FOR (i IN 1,5 EXCEPT 2)
      k[i]=0.0
END FOR
INTEGER i
FOR (i=0;i < 5; i=i+1)
      k[i]=0.0
END FOR
```

```
--Returns the angle of the pendulum of length
--when the horizontal displacement is x
FUNCTION REAL findAngle(REAL x, REAL L)

   DECLS
      REAL angle

   BODY
      IF (x != 0) THEN
         angle = asin(x/L)
      ELSE
         angle = 0
      END IF

   RETURN angle

END FUNCTION
```

```
CONST REAL g = 9.81 UNITS "m/s2"

--Pendulum
COMPONENT pendulum
DATA
   REAL  m = 1        UNITS "kg"
   REAL  L = 0.5      UNITS "m"
DECLS
   REAL  x
   REAL  y
   REAL  F
   REAL  angle
CONTINUOUS
   m*x'' = -(x/L)*F
   m*y'' = -(y/L)*F - m*g
   x**2+y**2 = L**2
   angle = findAngle(x,L)
END COMPONENT
```

- Beware that the function **findAngle()** is called, in the example above, at each integration step
- For this reason the loops (WHILE, FOR) should be used with care

```
--A possible long loop
FUNCTION REAL longLoop(INTEGER final)
   DECLS
      REAL     i
      REAL     suma = 0
   BODY
        FOR (i=0;i < final; i=i+1)
           suma = suma + i
        END FOR
   RETURN suma
END FUNCTION
```

```
--Possible very slow integrator
COMPONENT slowIntegrator
DATA
   INTEGER finalTime = 1000
DECLS
   REAL  m
   REAL suma
CONTINUOUS
   suma = longLoop(finalTime)
   m' = 1
END COMPONENT
```

- Beware that the function **findAngle()** is called, in the example above, at each integration step
- For this reason the loops (WHILE, FOR) should be used with care
- REAL data type is the most important one: all continuous variables participating in the computational causality assigment must be REAL
- There are other data types that should be used sparingly: INTEGER (useful for array index, control loop counters), **BOOLEAN** (logical conditions) and **STRING** for e.g. writing messages.

```
--A possible long loop
FUNCTION REAL longLoop(INTEGER final, OUT REAL prod)
   DECLS
      REAL     i
     REAL      suma = 0
   BODY
        prod = 2
        FOR (i=0;i < final; i=i+1)
           suma = suma + i
           prod = prod*2
        END FOR
        WRITE("Termina loop. Producto = %g\n ",prod)
   RETURN suma
END FUNCTION
```

```
--Possible very slow integrator
COMPONENT slowIntegrator
DATA
   INTEGER finalTime = 1000
DECLS
      REAL  m
      REAL  suma
DISCR REAL  producto
CONTINUOUS
   suma=longLoop(finalTime,producto)
   m' = 1
END COMPONENT
```

- Functions can return values via function parameters using OUT modifier
- WRITE statement helps in providing textual information from inside the simulation useful for debugging purposes.
- Use the DISCR modifier in COMPONENT to instruct not to consider some particular variable in the computational causality assignation algorithm

```
--With known function
COMPONENT testFunction
DECLS
   REAL x,y
CONTINUOUS
   y=log(x)
END COMPONENT
```

- PROOSIS have a collection of internal, native functions: **sin()**, **cos()**, **log()**, **exp()**, **sqrt()** …
- These functions can be mathematically manipulated in the partition generation phase.
- In the example: if y is declared as the boundary condition, PROOSIS “knows” how to invert the function (when possible) to use, in this case, the exponential.

```
--Custom exponential
FUNCTION REAL customExp(REAL x)

   DECLS
   BODY
   RETURN exp(x)

END FUNCTION

--With unknown user defined function
COMPONENT testCustomFunc
DECLS
   REAL x,y
CONTINUOUS
   y=customExp(x)
END COMPONENT
```

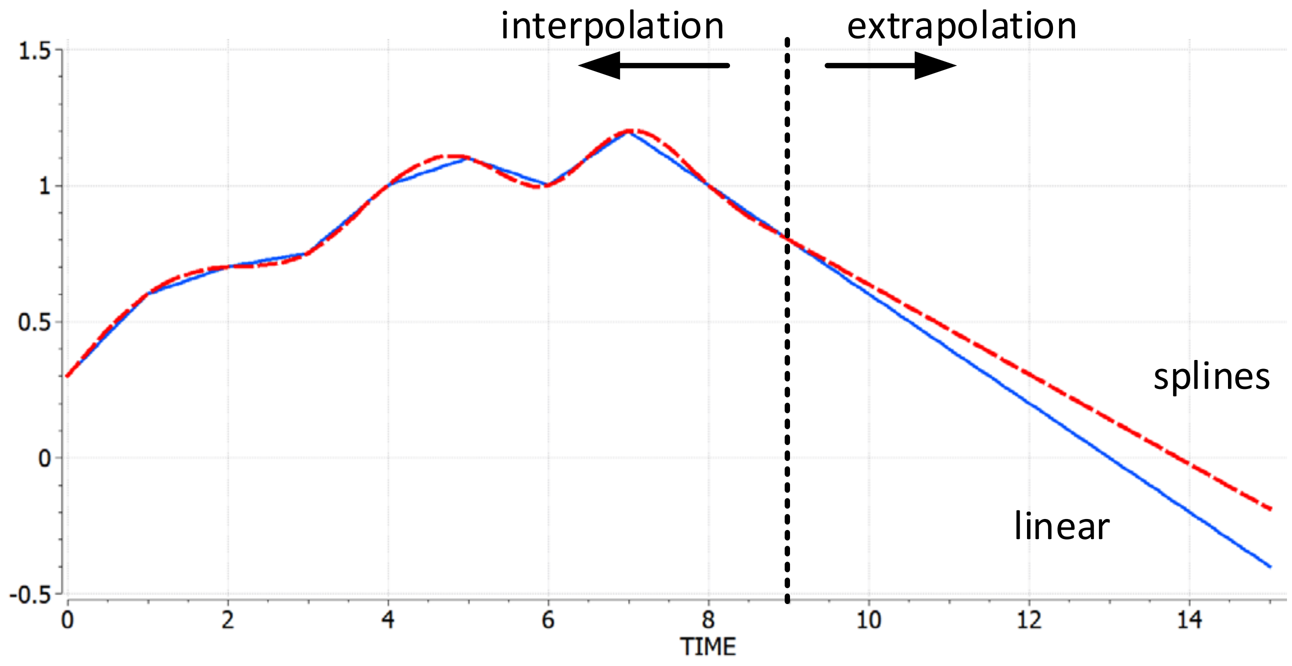
- With custom functions this is ability is lost. PROOSIS does not know how to invert a user-defined function
- Thus, in the cases in which the boundary is the y variable, the generated partition must have a algebraic loop

```
--Custom natural log.
FUNCTION REAL customLog(REAL x)
   DECLS
   BODY
   RETURN log(x)
END FUNCTION
--Custom exponential
FUNCTION REAL customExp(REAL x)

   DECLS
   BODY
   RETURN exp(x)

END FUNCTION
--With  user defined function
--but with alternative definition
COMPONENT testFunctionBothWays
DECLS
  REAL x,y
CONTINUOUS
              y=customExp(x)
  INVERSE (x) x = customLog(y)
END COMPONENT
```

- An advanced modelling technique is to provide to PROOSIS alternative functions to “invert” user defined function: teach PROOSIS how to invert our functions (in the cases this should be possible)
- The **INVERSE** modifier instructs the partition generation algorithm to use an alternative function if the need arises.

```
--Trying out tables and interpolation
COMPONENT inter
DATA
   TABLE_1D tabT= { {0., 1, 2, 3, 4, 5, 6, 7, 8, 9}, -- input
                    {0.3,0.6,0.7,0.75,1, 1.1, 1, 1.2,1, 0.8 } } --Output
DECLS
   REAL Tinterp,Tspline
CONTINUOUS
   Tinterp = linearInterp1D(tabT, TIME)
   Tspline = splineInterp1D(tabT, TIME)
END COMPONENT
```

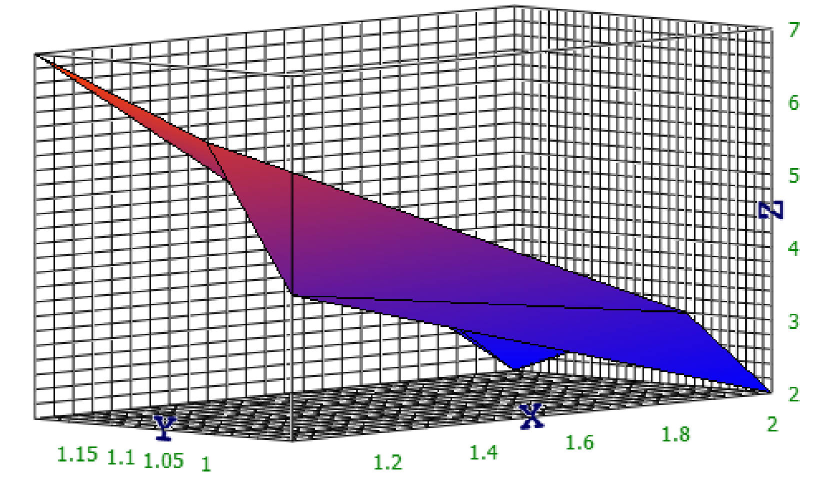
- The language offers the possibility of defining values by means of tables (describing mathematical functions as interpolation/extrapolation on 1D, 2D and 3D tables)
- It is possible to customize the interpolation and extrapolation methods.
- There are interpolation with history variants to speed up search in big tables.



2D table

```
TABLE_2D t = { { 1, 2},         -- X values
{ 0.9 ,1.0, 1.2 },              -- Y values
{ {4,6,7} , { 2, 3, 2 } } }-- output
```

	X →	1	2	3
Y ↓		0.9	1.0	1.2
1	1	4	6	7
2	2	2	3	2
3				



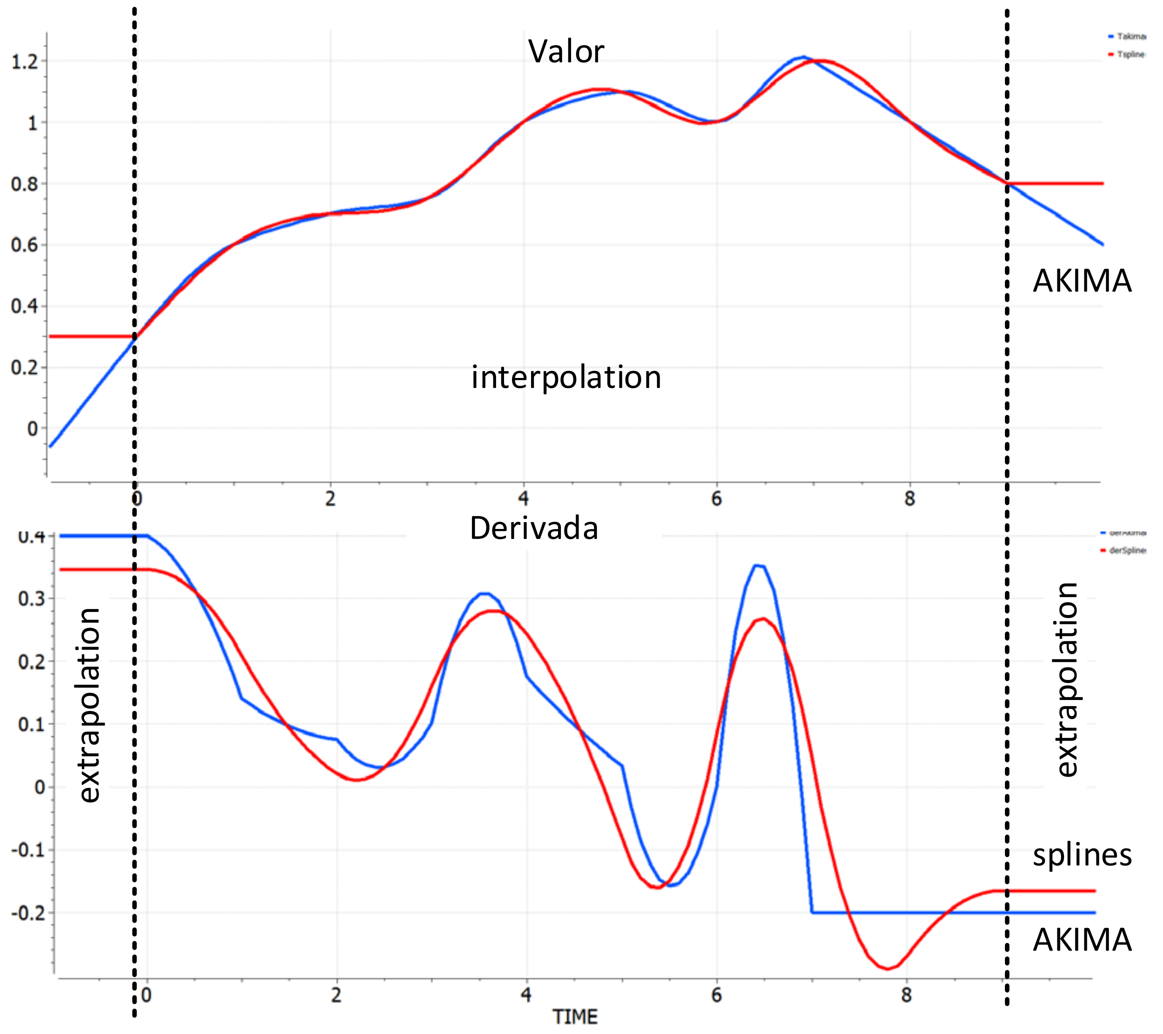
- General interpolation (extrapolation) function for tables

```
FUNCTION REAL interp1D(TABLE_1D tbl,    -- 1D table
IN ENUM t_interp tin,                   -- interpolation method
IN ENUM t_interp tex,                   -- extrapolation method
IN REAL x,                              -- input value x
OUT REAL dx=DUMMY_REAL)
```

DIMENSION	METHOD	INTERPOLATION	EXTRAPOLATION	DERIVATIVES
1D	FORBIDDEN	NO	YES	NO
1D	LINEAR	YES	YES	YES
1D	CONSTANT	YES	YES	YES
1D	SPLINE	YES	YES	YES
1D	AKIMA	YES	YES	YES
1D	CUBIC	NO	NO	NO
2D	FORBIDDEN	NO	YES	NO
2D	LINEAR	YES	YES	YES
2D	CONSTANT	YES	YES	YES
2D	SPLINE	YES	YES	YES
2D	AKIMA	YES	YES	YES
2D	CUBIC	YES	YES	YES
2DM(*)	Same as 1D	YES	YES	NO
3D	FORBIDDEN	NO	YES	NO
3D	LINEAR	YES	YES	YES
3D	CONSTANT	YES	YES	YES
3D	SPLINE	YES	YES	YES
3D	AKIMA	NO	NO	NO
3D	CUBIC	NO	NO	NO

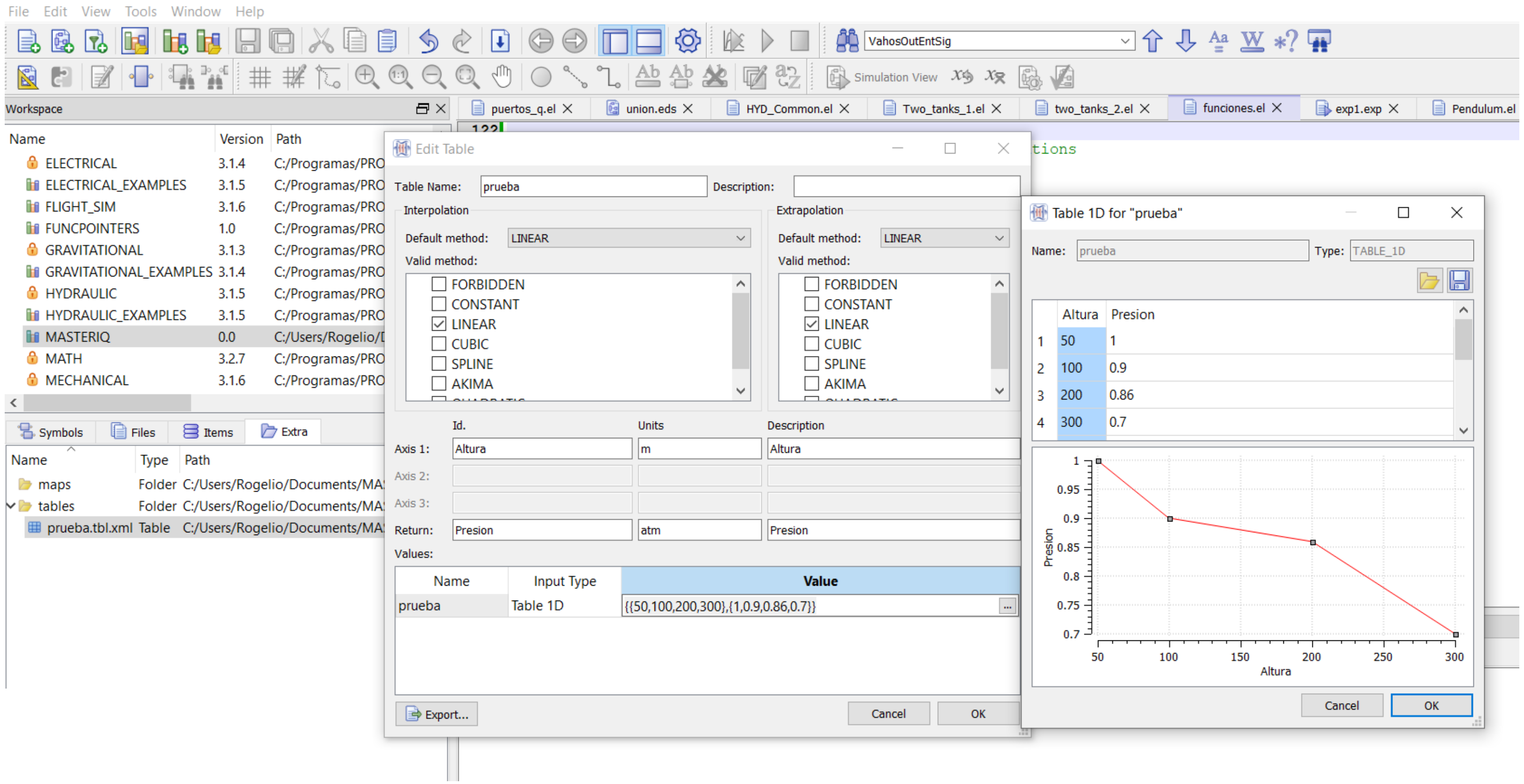
- General interpolation (extrapolation) function for tables

```
--Trying out 1D tables with with general interpolation funtions
COMPONENT interGeneral
DATA
   TABLE_1D tabT= { {0., 1, 2, 3, 4, 5, 6, 7, 8, 9}, -- input
                  {0.3,0.6,0.7,0.75,1, 1.1, 1, 1.2,1, 0.8 } } --Output
DECLS
   REAL Takima,Tspline
   REAL derAkima, derSpline
CONTINUOUS
   Takima = interp1D(tabT, AKIMA, AKIMA, TIME, derAkima)
   Tspline = interp1D(tabT, SPLINE, CONSTANT, TIME, derSpline)
END COMPONENT
```



Visual editing and reading of external defined tables.

- Proosis /EcosimPro has a tool for visual editing tables using the extra tag in the library panel
- The defined table can be saved externally as an XML file
- The resulting XML file can be used by any COMPONENT (the xml file should be placed in the specified folder)




```
--Reads table from file (XML)
COMPONENT testReadTable
    DATA
        FILEPATH tableFileXml = "tables/prueba.tbl.xml"
    DECLS
        TABLE_1D  txml
        REAL      y
    INIT
        readTable1D(tableFileXml,txml,2)  -- read from 2:XML
    CONTINUOUS
        y = linearInterp1D(txml, TIME)
END COMPONENT
```

```
--Fille array of size Num with values starting at initialValue
FUNCTION NO_TYPE fillArray(INTEGER Num, OUT REAL Arr[], REAL initialValue, REAL inc)
DECLS
   INTEGER  i
   REAL     value
BODY
   value = initialValue
   FOR (i=1; i <= Num; i = i + 1)
      Arr[i] = value
      value = value + inc
   END FOR
END FUNCTION
```

```
--Arrays of different dimensions
COMPONENT testArray(INTEGER N = 100)
    DATA
       REAL A[2,3] = {{1,2,3},{4,5,6}}
    DECLS
       REAL B[10,10,10]

       REAL C[N]     --One dimensional array (size is given by
                     --construction parameter)
    INIT
      FOR (i IN 1,10)
         FOR (j IN 1,10)
            FOR (k IN 1,10)
               B[i,j,k] = j**2 + i**2 +k**2
            END FOR
         END FOR
      END FOR

    CONTINUOUS
    fillArray(N,C,TIME,100)
END COMPONENT
```

- It is possible to work with arrays of one or more dimensions.
- Arrays can be passed as parameters to functions
- Notice the use of construction parameter (N) in the definition of the component

- In components, arrays can be to used in creating flexible components.
- EXPAND statements offers the language MACRO expansion facilities at component compile time

```
USE MATH
--Función para sumar el volumen de una serie de depósitos
FUNCTION REAL sumaVolumen(INTEGER N, REAL h[], REAL A)

   DECLS
      REAL     volumen = 0
   BODY

         FOR (i IN 1,N)
            volumen = volumen + A*h[i]
         END FOR

   RETURN volumen

END FUNCTION
--Depósitos en serie
COMPONENT seriesIntegrators(INTEGER N = 10)
DATA
   REAL A          = 10        "Area  de todos los depósitos m^2)"
   REAL hmax       = 2         "Altura max. de los depósitos m"
   REAL flujoIn    = 1         "Flujo de entrada al primer depósito m^3/s"
   REAL alpha      = 10        "Coeffiecient for Francis formula"
DECLS
      REAL  h[N]
      REAL  flujoOut
      REAL  Vtotal
CONTINUOUS

   EXPAND(i IN 1,N EXCEPT 1)  A*h[i]' = alpha*max(0,h[i-1]-hmax)**3.0/2.0-alpha*max(0,h[i]-hmax)**3.0/2.0

   A*h[1]'=flujoIn - alpha*max(0,h[1]-hmax)**3.0/2.0
   flujoOut = alpha*max(0,h[N]-hmax)**3.0/2.0
   Vtotal = sumaVolumen(N, h, A)

END COMPONENT
```

Enumerative types

- Enumerations allows for the assigment of intuitive names to a finite collection of discrete values
- Then arrays (with one or more dimensions) can be use, as indexes, enumeration types

```
ENUM Elements = {H, He, Li, Be, B, C, N, O}
ENUM Property = {AN, AM}

CONST STRING name[Elements] = {"Hidrogen","Helium", "Lithium", "Berilium", "Boron", "Carbon", "Nitrogen", "Oxigen"}
CONST REAL someProp[Property, Elements] = {{1,2,3,4,5,6,7,8},{1.008, 4.003, 6.941, 9.012, 10.811, 12.011, 14.007,15.999}}

COMPONENT testElements
DECLS
INIT
   WRITE("Atomic Mass of %s is %g and its Atomic Number %g\n", name[He], someProp[AM,He],someProp[AN,He])
CONTINUOUS
END COMPONENT
```

- Enumerative types: defines a set of possible discrete values.
- SET_OF types chooses subsets of a given enumeration.
- Allows for general description of components. Especially useful in chemical eng. modelling
- Used frequently as construction parameters

```
--Example of enumeration and SET_OF
ENUM Chemicals = {H2O, solute1, solute2, O2, H2SO4, CO2, N2 }
SET_OF(Chemicals) Air = {O2, N2, H2O, CO2}
SET_OF(Chemicals) subs = {H2O, solute1, solute2}
--A simple port
PORT prod (SET_OF(Chemicals) Mix = subs)
   EQUAL        REAL    P
   SUM        REAL    W
   SUM  IN    REAL    Wi[Mix]
   EQUAL OUT  REAL    C[Mix]
CONTINUOUS
   EXPAND (i IN subs) Wi[i] = C[i]*W
END PORT
```

```
--A simple tank
COMPONENT Tank(SET_OF(Chemicals) Mix = subs)
PORTS
   IN  prod(Mix) input
   OUT prod(Mix) output
DATA

DECLS

   REAL m[Mix]
   REAL mt
INIT
   FOR (comp IN Mix)
      m[comp] = 0
   END FOR
CONTINUOUS

      EXPAND_BLOCK (i IN Mix)
         m[i]' = input.Wi[i] - output.W*output.C[i]
         output.C[i] = input.C[i]
      END EXPAND_BLOCK

      mt = SUM (i IN Mix ; m[i])
END COMPONENT
```