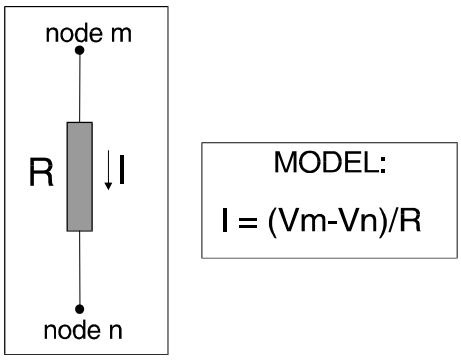
Appendix 6
Analogue simulation examples

Since most of the users prefer learning from examples, in this Appendix some simple electronic analogue and mechanical examples are given. Usage of model cards, `clone` command, and other advanced modelling techniques are not described here.

A6.1. Electronic models

Some of the models described here are already built-in components of Alecsis, so you do not need to describe such models. They are here given as introduction, as it is always easier to start from simple examples.

A6.1.1. Resistor



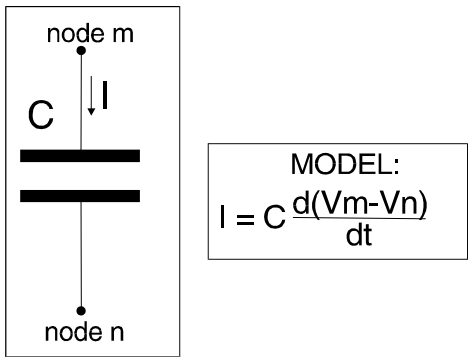
Model stamp:

	Vm	Vn	rhs
m	1/R	-1/R	
n	-1/R	1/R	

Model code:

```
module MyResistor (node m, n)
{
   action (double value=0.0) {
      // avoid division by zero
      process initial {
         if (!value)
            warning ("zero resistance", 1);
      }

      process initial {
         double G;
         // conductance for equation matrix
         G = 1./value;
         // equation (stamp):
         eqn {m,n}.i = G*{m,n}.v;
      }
   }
}
```

A6.1.2. Capacitor



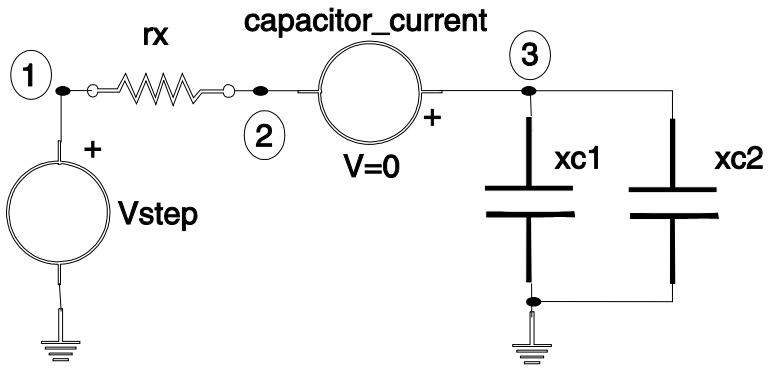
Model stamp:

	Vm	Vn	rhs
m	C/h	-C/h	(C/h)*(Vmp-Vnp)
n	-C/h	C/h	(C/h)*(Vmp-Vnp)

Model code:

```
module MyCapacitance (node m, n) {
   // capacitance C=0.0pF -> default value
   action (double C=0.0) {
      process per_moment {
         //equation (stamp) matrix!
         eqn {m,n}.i = C*ddt{m,n}.v;
      }
   }
}
```

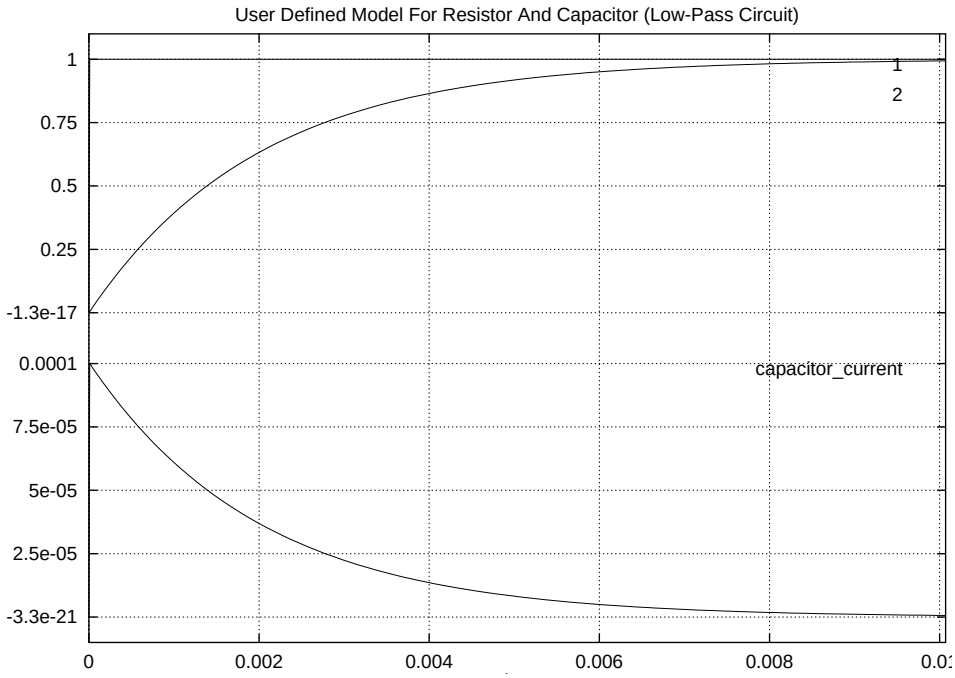
A6.1.3. Example 1



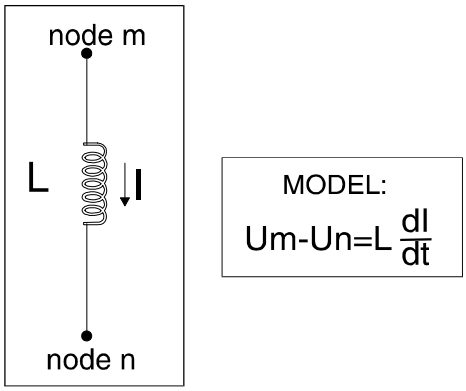
System description:

```
#include <alec.h>
#define Period 10 ms
root module eq ()
{
   // declarations
   MyResistor r;
   MyCapacitance xc1, xc2;
   vgen capacitor_current; // built-in voltage source
   vpwl vin;               // built-in piecewise linear volt. gen.

   // connections
   rx(1,2) 10k;
   capacitor_current (2,3) 0V; // used to measure current
   xc1(3,0) 100nF;
   xc2(3,0) 100nF;
   vin (1, 0) { 0,0; 1ns, 1; Period, 1; }

   // simulation control
   timing { tstop = Period;   a_step=Period/1000; }
   plot { caption
        "User Defined Model For Resistor
         And Capacitor (Low-Pass Circuit)";
         node 1, 2;    // these two nodes use the same scaling
         current capacitor_current; }
}
```

Simulation results:


User Defined Model For Resistor And Capacitor (Low-Pass Circuit)
1
0.75
0.5
0.25
-1.3e-17
0.0001
7.5e-05
5e-05
2.5e-05
-3.3e-21
0
0.002
0.004
0.006
0.008
1
2
capacitor_current

A6.1.4. Inductor



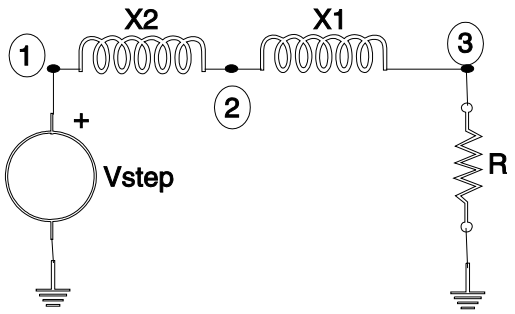
Model stamp. Current must be independent variable in the system of equations:

	Vm	Vn	I	rhs
m			+1	
n			-1	
new	1	-1	L/h	(L/h)*Ip

Model description. Branch current is returned using name of the module:

```
module current MyInductance (node m, n) {
   // inductance L=0.0uH -> default value
   action (double L=0.0) {
      // zero inductance check
      process initial {
         if (!L)
             warning ("zero inductance", 1);
      }

      process per_moment {
         // equation (stamp)
         eqn MyInductance, {m.n}.v = L*ddt{MyInductance};
      }
   }
}
```

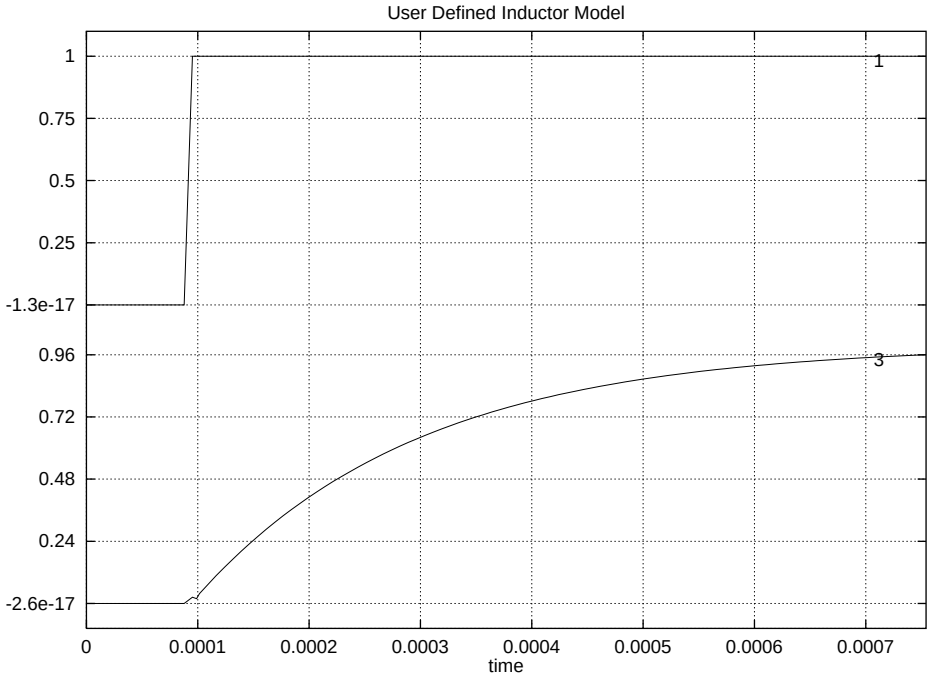
A6.1.5. Example 2



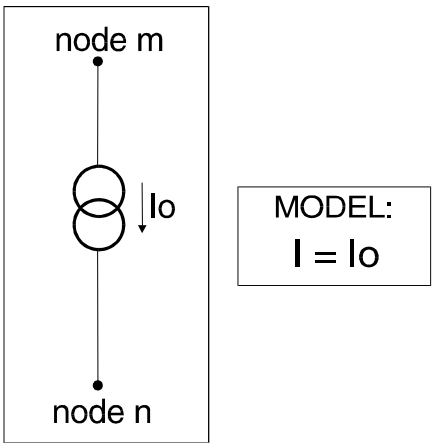
Circuit description:

```
#include <alec.h>
#define Period 750 us

root eq ()
{
   MyResistor r;
   MyInductance x1,x2;
   vpwl vin;      // built-in

   r(3,0) 1.;
   x1(2,3) L=100uH;
   x2(1,2) L=100uH;

   vin (1, 0) { 0,0; Period/8.,0;
                Period/8.,1;  Period, 1; }
   timing { tstop = Period;
            a_step=Period/1000; }
   plot { caption
            "User Defined Inductor Model";
          node 1; node 3;     // two separate waveforms
        }
}
```

Simulation results:


User Defined Inductor Model
1
0.75
0.5
0.25
-1.3e-17
0.96
0.72
0.48
0.24
-2.6e-17
0
0.0001
0.0002
0.0003
0.0004
0.0005
0.0006
0.0007
time
1
3

A6.1.6. Current source



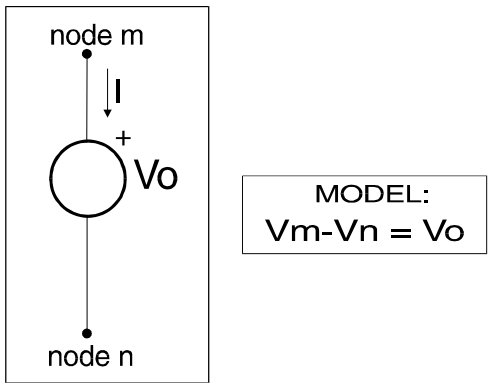
Model stamp:

node	Vm	Vn	rhs
m			-Io
n			+Io

Model description:

```
module Current_source (node m, n) {
   // default current Io=0.0
   action double Io=0.0) {
      virtual process initial {
         eqn {m,n}.i = Io;
      }
   }
}
```

A6.1.7. Voltage source



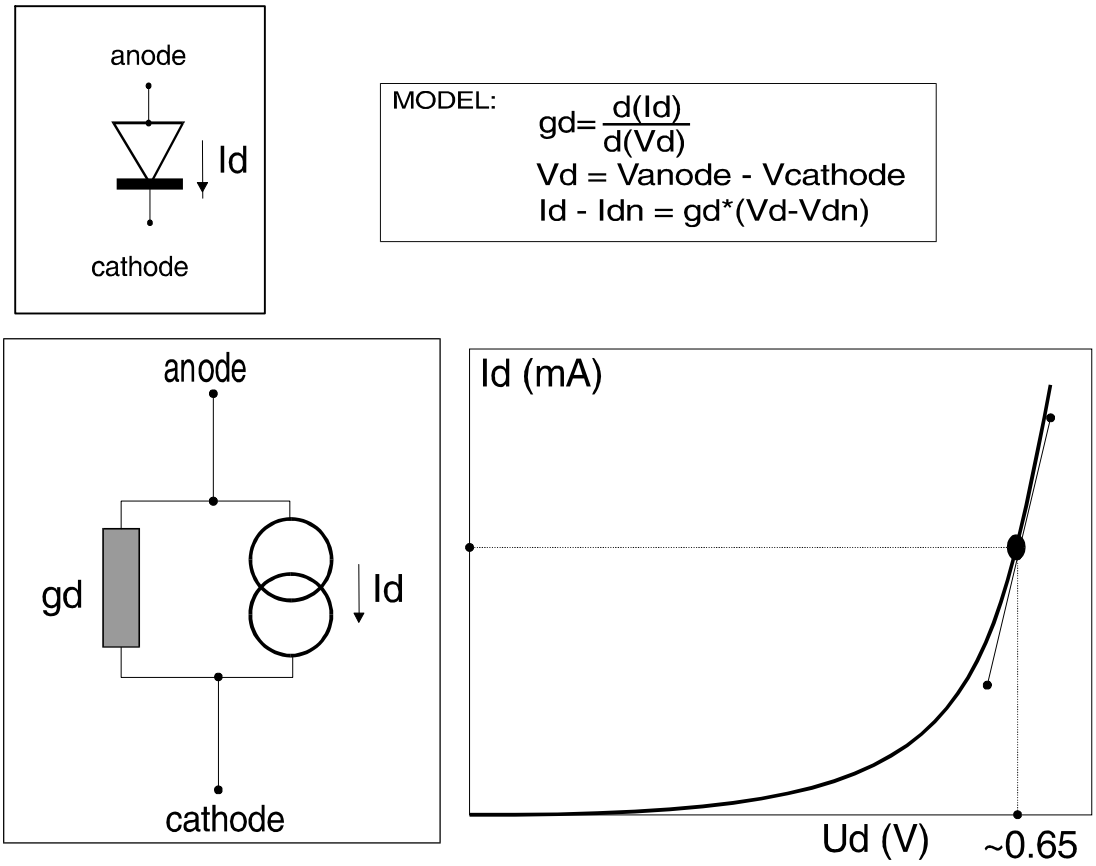
Model stamp. Branch current has to be independent quantity in the system of equations:

node	Vm	Vn	I	rhs
m			+1	
n			-1	
I	1	-1		Vo

Model description:

```
module current Voltage_source (node m, n){
   // default voltage Vo=0.0
   action (double Vo = 0.0) {
      virtual process initial {
         eqn Voltage_source, {m,n}.v = Vo;
      }
   }
}
```

A6.1.8. Diode



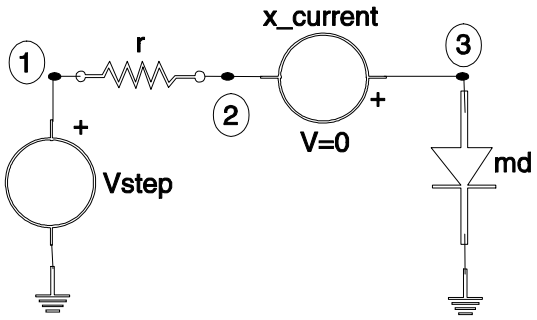
Model with user defined linearized structure. Real diode model should have more parameters and model card, this is simplified description:

```
module MyDiode1 (node m, n) {
   action (double Is=1e-14) {
      process per_iteration {
         double gd,Vd_p,Id_p;
         double Vt=25.8mV;

         // voltage from the last iteration
         Vd_p = m-n;
         // current from the last iteration
         Id_p = Is*exp((m-n)/Vt);
         // slope of the linearized equation - d(Id)/d(Vd)
         gd = Id_p/Vt;

         // linearized equation definition
         eqn {m,n}.i = gd*{m,n}.v - gd*Vd_p + Id_p;

         /* alternative description of the same equation:
            eqn m:  gd*{m}-gd*{n} = -Id_p+gd*Vd_p;
            eqn n: -gd*{m}+gd*{n} = +Id_p+gd*Vd_p;
         */
      }
   }
}
```

Automated description of the linearized model using `nlcgen` - nonlinear current generator:

```
module MyDiode2 (node m, n){
   nlcgen Id;
   Id (m,n,m,n);//connected between m and n, controlled by m and n

   action (double Is=1e-14) {
      process per_iteration {
         double id, gd;
         double Vt=25.8mV;

         id = Is*(exp((m-n)/Vt)-1);
         gd = (id+Is)/Vt;

         // linearized equation with partial derivatives
         nlcgen Id = id { @m=gd; @n=-gd; }

         /* alternative description - automated linearization,
            gd is not necessary:
            nlcgen Id=id;
         */
      }
   }
}
```

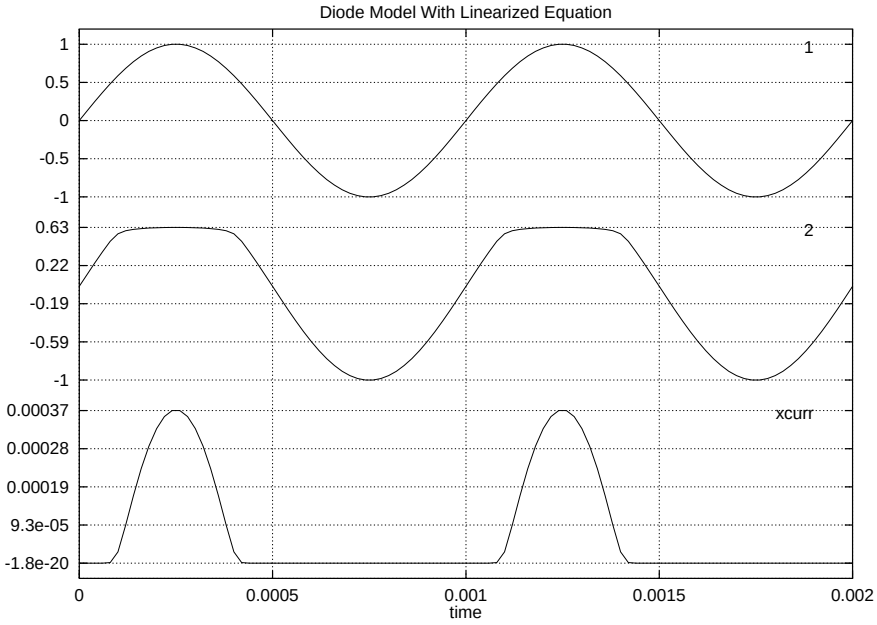
A6.1.9. Example 3



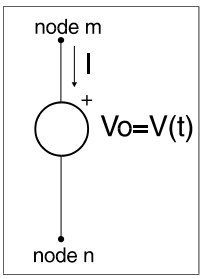
Circuit description:

```
#include <alec.h>
#define Period 2 ms

root module eq ()
{
   MyResistor r;
   MyDiode1 md;
   vsin vin;
   Voltage_source xcurr;

   md(3,0);
   xcurr (2,3) 0V;
   r(2,1) 1e3;
   vin (1, 0) { amp=1V; freq=1kHz; }

   timing { tstop = Period;           a_step=Period/100; }
   plot { caption "Diode Model With Non-Linear Current Generator";
         node 1; node 2;
         current xcurr;
       }
}
```

Simulation results:


Diode Model With Linearized Equation
1
0.5
0
-0.5
-1
0.63
0.22
-0.19
-0.59
-1
0.00037
0.00028
0.00019
9.3e-05
-1.8e-20
1
2
xcurr
0
0.0005
0.001
0.0015
0.002
time

A6.1.10. Sinusoidal voltage generator



MODEL:
V(t) = offset+
amp*sin(twopi*freq*now+phase)

Model description:

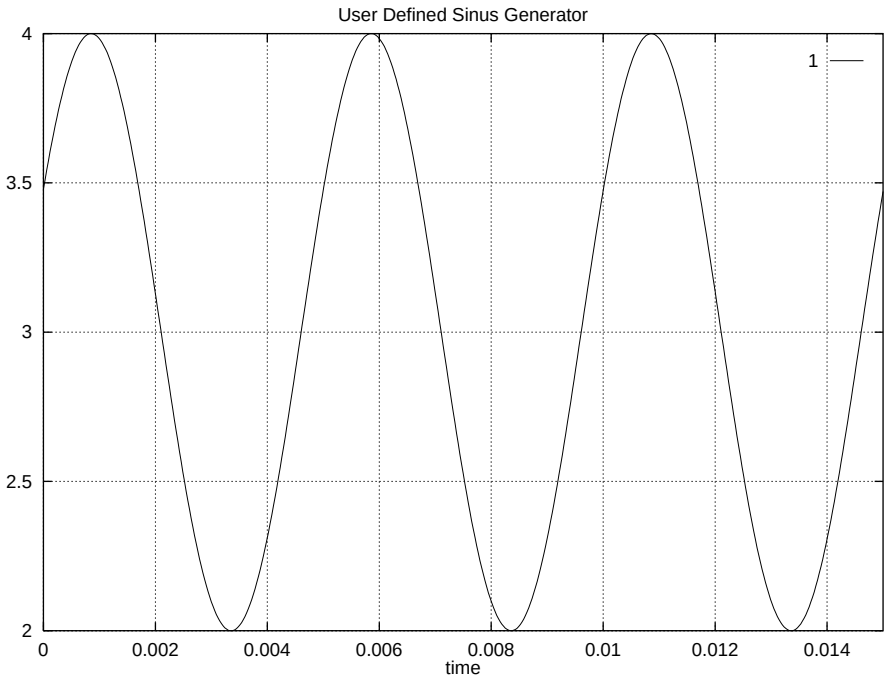
```
#define twopi 6.282

module current Sinus(node m, n) {
   vgen Sinus; // built-in const. voltage src. is used as a basis

   return Sinus(m,n); // connection - branch current of vgen
                      // is returned using name of the module

   // default parameter definition
   action per_moment (double amp=1.0v,
                      double freq=1kHz,
                      double phase=0.0rad,
                      double offset = 0V){
      Sinus->value = offset + amp*sin(twopi*freq*now+phase);
   }
}
```

A6.1.11. Example 4

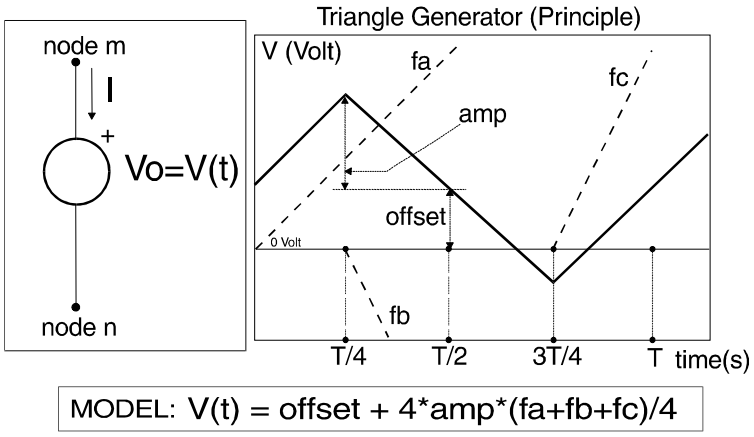
Circuit description:

```
#include <alec.h>
#define Period 15 ms
root eq ()
{
  MyResistor r;
  Sinus singen;

  r(1,0)1k;
  singen (1,0) {amp=1V; freq=200Hz; phase=0.5rad; offset=3V;}

  timing { tstop = Period; a_step=Period/500; }
  plot { caption "User Defined Sinus
                  Generator";
         node 1; }
}
```

Simulation results:



A6.1.12. Triangular voltage generator



Model description:

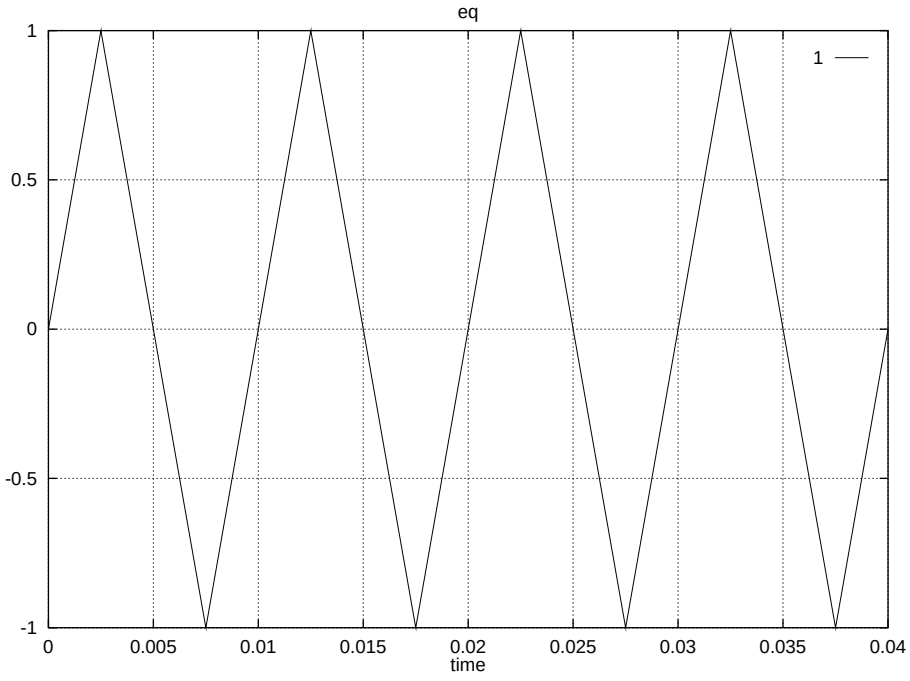
```
module current Triangle (node m, n) {
  vgen Triangle;
  return Triangle(m, n);   // connection definition

  action per_moment ( double amp=1.0V,
                      double T=1s,
                      double offset=0V) {
    // control parameter
    double X;
    double fa=0, fb=0, fc=0;

    X = now;
    while(X > T) X -= T;

    fa = X;

    if (X>T/4) fb=-2*(X-T/4);
     else fb=0;

    if (X> 3*T/4) fc=2*(X-(3*T/4));
    else fc=0;

    Triangle->value = offset + 4*amp*(fa+fb+fc)/T;
  }
}
```

A6.1.13. Example 5

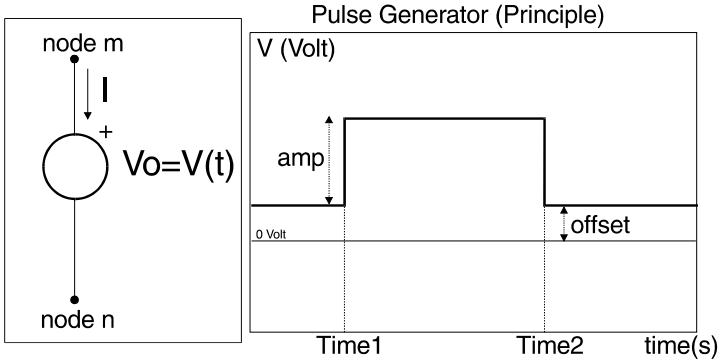
Circuit description:

```
#include <alec.h>
#define Period 40 ms

root eq ()
{
   MyResistor r;
   Triangle t;


  r(1,0)1k;
  t(1,0) {amp=1V; T=10ms; offset=0V;}

  timing { tstop = Period; a_step=Period/10000; }
  plot { node 1; }
}
```

Simulation results:



A6.1.14. Pulse voltage generator



MODEL: V(t) = offset + amp, if Time1<t<Time2
V(t) = offset, if t<Time OR t>Time2

Model description:

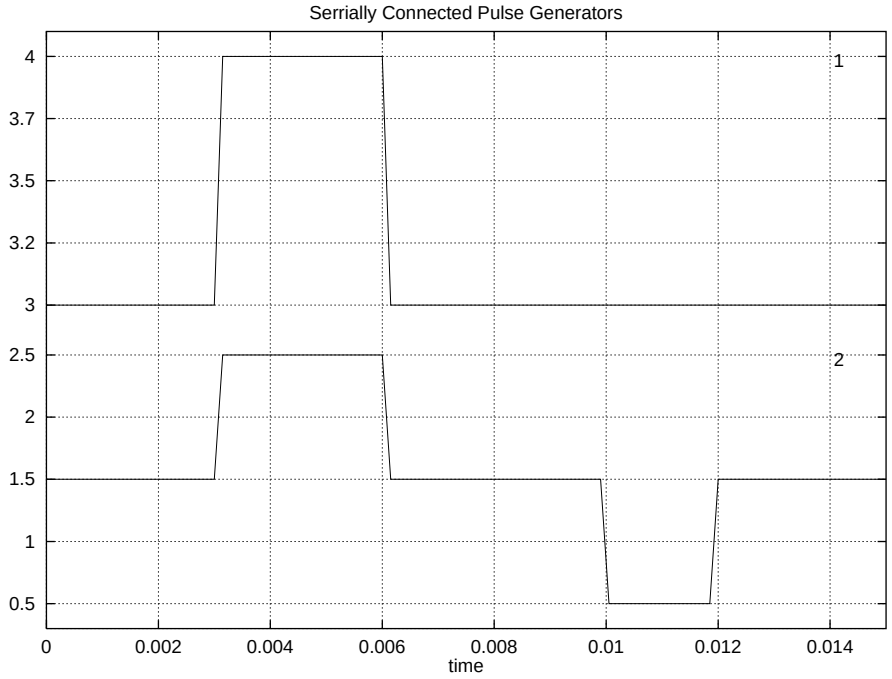
```
module current Pulse (node m, n) {
   vgen Pulse;
   return Pulse(m, n);
   // default parameters definition
   action per_moment (double amp=1.0V,
                      double time1=1ms,
                      double time2=2ms,
                      double offset=0V) {
      if ((now>=time1) && (now<time2))
      Pulse->value = amp+offset;

      if ((now<time1) || (now>=time2))
      Pulse->value = offset;
   }
}
```

A6.1.15. Example 6

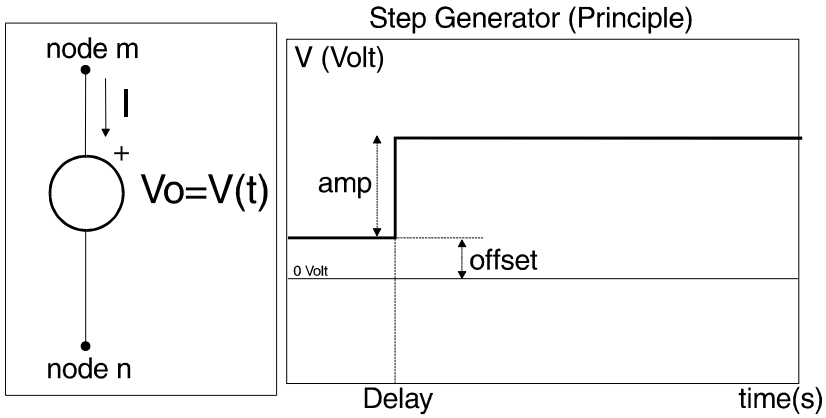
Circuit description:

```
#include <alec.h>
#define Period 15 ms

root eq ()
{
   MyResistor r;
   Pulse p1, p2;

   r(2,0)1k;
   // two generators produce independant,
   // but superimposed pulses
   p1 (1,0) {amp=1V; time1=3ms; time2=6ms; offset=3V;};
   p2 (1,2) {amp=1V; time1=10ms; time2=12ms; offset=1.5V;};

   timing { tstop = Period; t_step=Period/100; }
   plot { caption  "Serrially Connected Pulse Generators";
       node 1;  node 2; }
}
```

Simulation results:



A6.1.16. Step voltage generator



MODEL:	V(t) = offset, if t<Delay
	V(t) = offset + amp, if t>Delay

Model description:

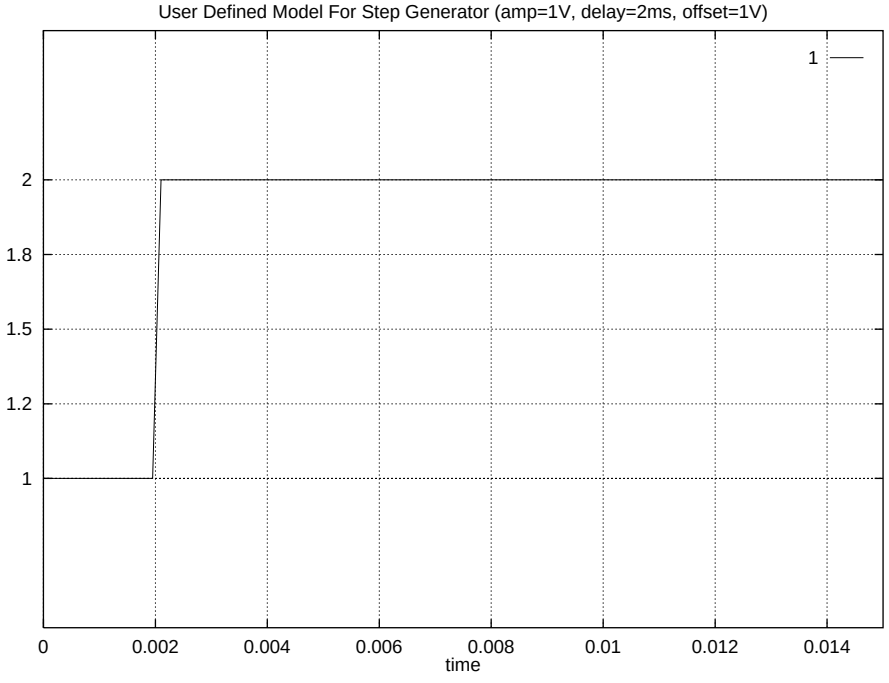
```
module current Step(node m, n) {
 vgen Step;
 return Step (m,n);    // node definition

 action per_moment (double amp=1.0V,
                    double delay=1ms,
                    double offset = 0V) {
   if (now < delay)
      Step->value = offset;
   else
      Step->value = offset + amp;
   }
}
```

A6.1.17. Example 7

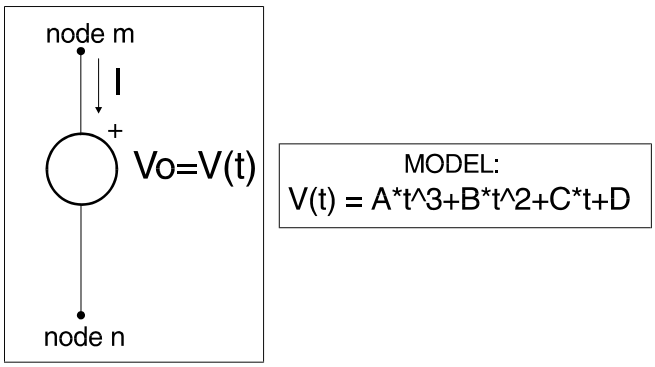
Circuit description:

```
#include <alec.h>
#define Period 15 ms

root module eq () {
  MyResistor r;
  Step sgen;

  r (1,0)1k;
  sgen (1,0) {amp=1V; delay=2ms; offset=1V;};

  timing { tstop = Period; a_step=Period/100; }
  plot { caption "User Defined Model For \
                  Step Generator (amp=1V, delay=2ms, offset=1V)";
         node 1; }
}
```

Simulation results:



A6.1.18. Polynomial voltage generator



Model description:

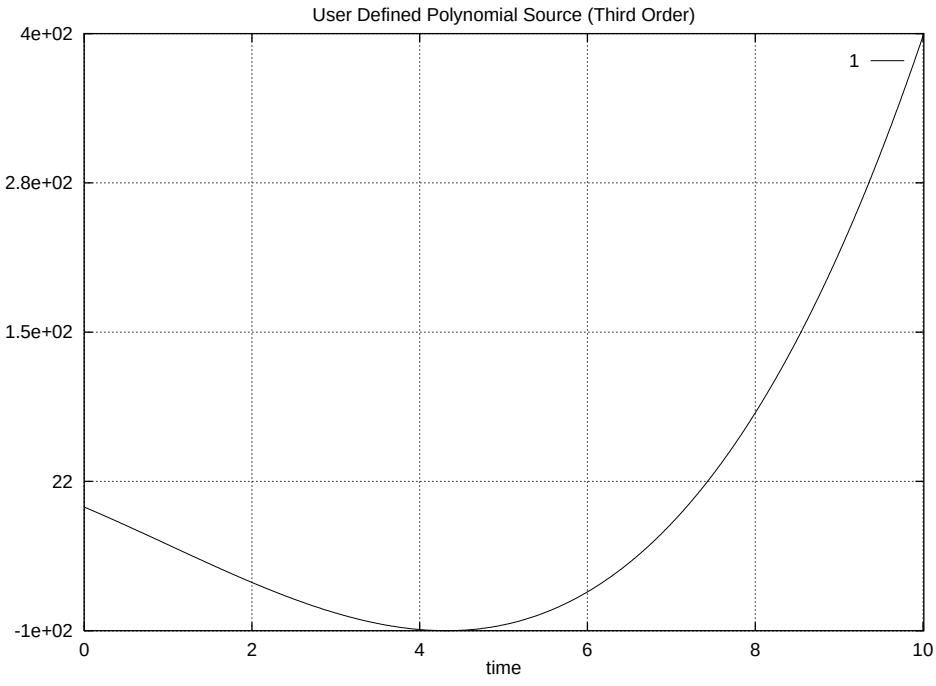
```
module current Poly (node m, n) {
   vgen Poly;

   return Poly (m, n);

   action per_moment (double A=0.0,
                      double B=0.0,
                      double C=0.0,
                      double D=0.0) {
      Poly->value = A*now*now*now+B*now*now+C*now+D;
   }
}
```

A6.1.19. Example 8

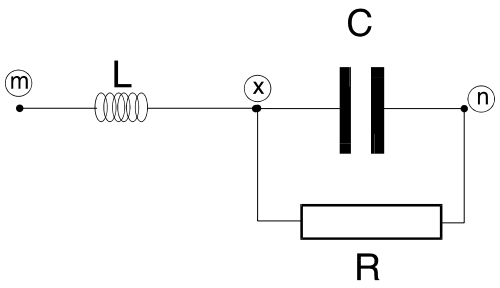
Circuit description:

```
#include <alec.h>
#define Period 10. s

root eq () {
  MyResistor r;
  Poly pgen;

  r (1,0) 1k;
  pgen (1,0) {A=1; B=-3; C=-30; D=0; }

  timing { tstop = Period; a_step=Period/1000; }
  plot { caption "User Defined Polynomial
           Source (Third Order)";
         node 1; }
}
```

Simulation results:



A6.1.20. Submodule (subcircuit) example



Submodule is described as a module without functional part (`action` block):

```
module coilcap (node m, n) {
  node x;

  resistor R;
  capacitor C;
  coil L;

  R(x,n) 1k;
  L(m,x) 1uH;
  C(x,n) 50pF;
}
```

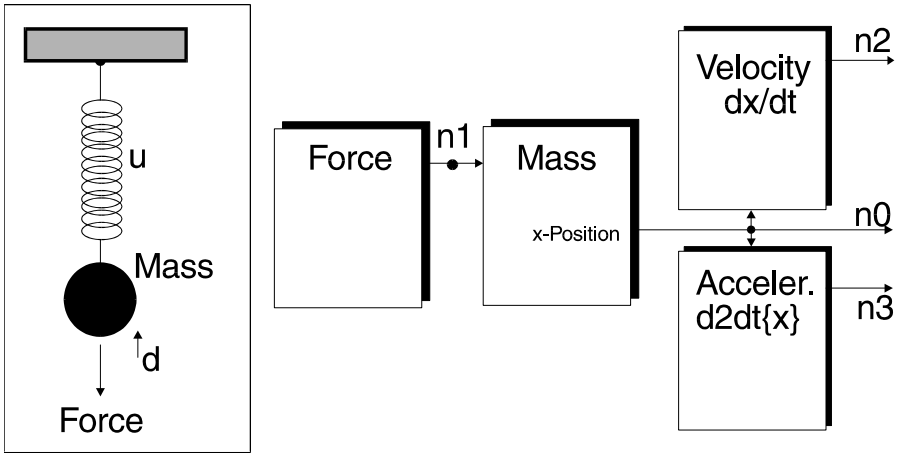
A6.2. Mechanical models

Description of mechanical models does not differ significantly from models of electronic components. One should use analogue links of type `flow` (instead of `node`, `current` and `charge` in electronics), which allows separate control of tolerance parameters for nonelectrical links.

A6.2.1. Oscillating mass

Model:

$$m\frac{d^2x}{dt^2} + d\frac{dx}{dt} + ux = F(t)$$



Model description: Nonelectrical quantites are declared as `flow`s. Modules for calculating velocity and acceleration are not necessary for simulation, they are used just to print out those results:

```
// Swinging Mass
module Swinging_Mass (flow x, force) {
  action (double m, double u, double d) {
    process per_moment {
      // equation of mechanical equilibrium
      eqn x: m*d2dt2{x} +d*ddt{x} + u{x} - {force} = 0;
    }
  }
}

// Force
module Force (flow force) {
  action (double force_value) {
    double force_out;
    process per_moment {
      force_out = force_value*exp(-now);
      eqn force: {force} = force_out;
```

```
    }
  }
}

// Velocity (used for printing out, not necessary for simulation)
module Velocity (flow x, velocity) {
  action {
    process per_moment {
      eqn velocity: {velocity} - ddt{x} = 0;
    }
  }
}

// Acceler. (used for printing out, not necessary for simulation)
module Acceleration (flow x, acceleration) {
  action {
    process per_moment {
      eqn acceleration: {acceleration} - d2dt2{x} = 0;
    }
  }
}
```

A6.1.2. Example 1

System description:

```
#include <alec.h>
#define Period 15. s

root eq () {
  flow n0, n1, n2, n3;
  // simple mechanical system
  Swinging_Mass p;
  Force F;
  Velocity V;
  Acceleration A;

  p (n1,n0){m=1 ; u=1 ; d=0.35;}
  F (n0){force_value=10;}
  V (n1,n2);
  A (n1,n3);

  timing { tstop = Period; a_step=Period/1000; }
  plot { caption "Swinging Mass Model
       (RED=position of the mass,
       GREEN=velocity,
       BLUE=acceleration)
       * EXCITATION: Fo*EXP(-Time)";
       flow n1; flow n2; flow n3;
       }
}
```

Simulation results: