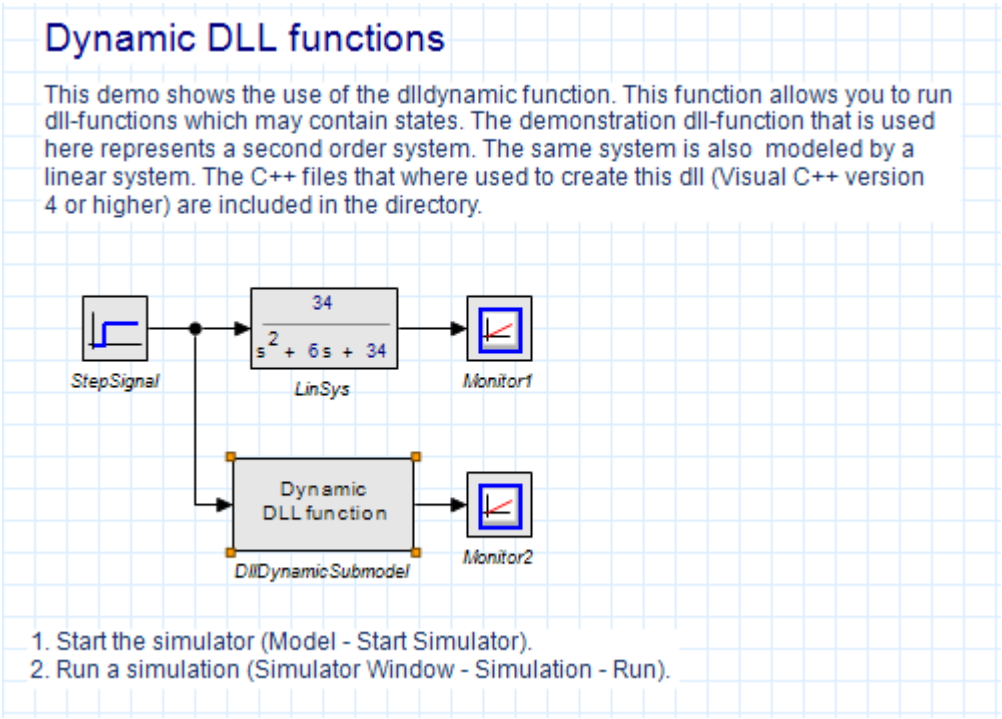
Example of 20-sim using a dll function

20-sim program



Call of the dll function in 20-sim

```
parameters
   string dllName = 'demoDynamicDll.dll';
   string functionName = 'SFunctionCalculate';
equations
   output = dlldynamic (dllName, functionName, input);
```

C code

```
#include "stdafx.h"
#include "SimulatorSFunctionStruct.h"

/*****************************************************************************
 * in this source file we are gonna describe a linear system which is defined by
 * the following transfer function description:

              34
       Y = ---------------- * U
            s^2 + 6s + 34

       or A, B, C, D system:

       A = [ 0, -3.4;
               10, -6];
       B = [ -3.4;
               0];
       C = [0, -1];
       D = 0
       which has two poles on (-3 + 5i) and (-3 -5i)

  steady state = 1

  *****************************************************************************/

#define DllExport   __declspec( dllexport )

extern "C"
{

// called at begin of the simulation run
DllExport int Initialize()
{
       // you can perform your own initialization here.

       // success
       return TRUE;
}

// called at end of the simulation run
DllExport int Terminate()
{
       // do some cleaning here

       // success
       return TRUE;
}

DllExport int SFunctionInit (SimulatorSFunctionStruct *s)
{
       // tell our caller what kind of dll we are
       s->nrIndepStates = 2;
       s->nrDepStates = 0;
       s->nrAlgLoops = 0;

       // dubious information, since 20-sim itself does not check and need this info
       s->nrInputs = 1;
       s->nrOutputs = 1;
```

```
	// return 1, which means TRUE
	return 1;
}

DllExport int SFunctionGetInitialStates (double *x0, double *xd0, double *xa0,
SimulatorSFunctionStruct *s)
{
	// fill in the x0 array here. Since we specified no Dependent states, and No
algebraic loop variables
	// the xd0 and xa0 may not be used.

	// initial value is zero.
	x0[0] = 0;
	x0[1] = 0;

	// return 1, which means TRUE
	return 1;
}

DllExport int SFunctionCalculate(double *u, double *x, double *y, double *dx,
SimulatorSFunctionStruct *s)
{
	// we could check the SimulatorSFunctionStruct here if we are in an
initialization state and/or we are
	// in a major integration step.
#if 0
	if( s->initialialOutputCalculation )
		; // do something

	// possibly do some explicit action when we are in a major step.
	if( s->major == TRUE )
		; // do something
#endif
	dx[0] = -3.4 * x[1] -3.4 * u[0];
	dx[1] = 10 * x[0] -6 * x[1];


	y[0] = -x[1];

	// return 1, which means TRUE
	return 1;
}

}// extern "C"
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                       )
{
    return TRUE;
}
```

Code of the structure called

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

// Insert your headers here
#define WIN32_LEAN_AND_MEAN		// Exclude rarely-used stuff from Windows headers

#include <windows.h>



#ifndef __SIMULATION_SFUNCTION_STRUCT
#define __SIMULATION_SFUNCTION_STRUCT

#ifndef BOOL
typedef int   BOOL;
#endif

struct SimulatorSFunctionStruct
{
	double versionNumber;
	int nrInputs;
	int nrOutputs;
	int nrIndepStates;
	int nrDepStates;
	int nrAlgLoops;
	double simulationStartTime;
	double simulationFinishTime;
	double simulationCurrentTime;
	BOOL major;
	BOOL initialOutputCalculation;
};


/*
   This structure is both used for initialization of the DLL
  SFunction as for the calling of the SFunction funtion itself

  the versionNumber in the structure is to be able to detect changes in the structure
  definition for future use. New options and fields will be added to the end of the
  structure, so that older dll's will remain working.

1.Initialization:
  In de DLL the function "int SFunctionInit(SimulatorSFunctionStruct *simStruct)"
  is called.
  Return value is 0 means error. every other value succes
  Argument is a pointer to the simstructure.
  On initialization the following fields should be filed in:
  nrIndepStates
  nrDepStates
  nrAlgLoop

  The following fields already have valid values:
  simulationStartTime            : giving the start time of the simulation
  simulationFinishTime           : giving the finish time of the simulation
  simulationCurrentTime          : giving the current time (actually the start time
at the moment of initialization)


2.Initial values for the states:
  Return value is 0 means error. every other value succes
```

```
  int SFunctionGetInitialStates(double *initialIndepStates,
                                        double *initialDepRates,
                                        double *initialAlgloopIn,
                                        SimulatorSFunctionStruct *simStruct);

  The initial value for the independent states, dependent rates and algebraic
  loop variables can be specified by the dll in this function. It is just called
  before the initial output calculation function in step 3. If all the initial values
  are zero, nothing has to be specified.

3.Initial Output Calculation
  It is possible that the DLL-function can give an initial output.
  A separate function is called so that the DLL can
  calculate it's initial output values.
  The boolean initialOutputCalculation in the simulatorSFunction structure
  is used. Just the sFunction is called. as in point 4

4.SFunction calling:
  Here all the fields of the SimulatorSFunctionStruct are
  input for the function. The inputArray, stateArray, outputArray and rateArray
  are always given as arguments of the function. Dependent on the number of dependent
  states and algebraic loop variables more arguments can be given as shown
  in the functions below (sFunctionName is the name defined by the parameter name
specified
  by the user):
  Return value is 0 means error. every other value succes

  case: no dependent states, no algebraic loop variables
  int sFunctionName(double *inputArray,
                                double *stateArray,
                                double *outputArray,
                                double *rateArray,
                                SimulatorSFunctionStruct *simStruct);

  case: dependent states, no algebraic loop variables
  int sFunctionName(double *inputArray,
                                double *stateArray,
                                double *dependentRateArray,
                                double *outputArray,
                                double *rateArray,
                                double *dependentStateArray,
                                SimulatorSFunctionStruct *simStruct);

  case: no dependent states, algebraic loop variables
  int sFunctionName(double *inputArray,
                                double *stateArray,
                                double *algLoopInArray,
                                double *outputArray,
                                double *rateArray,
                                double *algLoopOutrray,
                                SimulatorSFunctionStruct *simStruct);

  case: dependent states, algebraic loop variables
  int sFunctionName(double *inputArray,
                                double *stateArray,
                                double *dependentRateArray,
                                double *algLoopInArray,
                                double *outputArray,
                                double *rateArray,
                                double *dependentStateArray,
                                double *algLoopOutrray,
```

```
                                    SimulatorSFunctionStruct *simStruct);

  the boolean major in the SimulatorSFunctionStruct determines whether the evaluation
  of the model is done at the time output is generated (major == TRUE ) or that the
  model is evaluated because of determining model characteristics. For example Runge-
Kutta4
  integration method uses three minor steps before taking a major step where output is
generated.
  Higher order methods can have different number of minor steps before a major step is
taken.

*/



#endif // __SIMULATION_SFUNCTION_STRUCT
```