```
//
//
//   Example for a communication interface from ORTD using UDP datagrams to e.g.
//   nodejs.
//   webappUDP.js is the counterpart that provides a web-interface to control
//   a oscillator-system in this example.
//
//

// The name of the program
ProgramName = 'UDPio'; // must be the filename without .sce

// And example-system that is controlled via UDP and one step further with the Web-gui
// Superblock: A more complex oscillator with damping
function [sim, x, v]=damped_oscillator(sim, u)
    // create feedback signals
    [sim,x_feedback] = libdyn_new_feedback(sim);

        [sim,v_feedback] = libdyn_new_feedback(sim);

            // use this as a normal signal
            [sim,a] = ld_add(sim, ev, list(u, x_feedback), [1, -1]);
            [sim,a] = ld_add(sim, ev, list(a, v_feedback), [1, -1]);

            [sim,v] = ld_ztf(sim, ev, a, 1/(z-1) * T_a ); // Integrator approximation

            // feedback gain
            [sim,v_gain] = ld_gain(sim, ev, v, 0.1);

            // close loop v_gain = v_feedback
        [sim] = libdyn_close_loop(sim, v_gain, v_feedback);


        [sim,x] = ld_ztf(sim, ev, v, 1/(z-1) * T_a ); // Integrator approximation

        // feedback gain
        [sim,x_gain] = ld_gain(sim, ev, x, 0.6);

    // close loop x_gain = x_feedback
    [sim] = libdyn_close_loop(sim, x_gain, x_feedback);
endfunction

// Send a signal via UDP, a simple protocoll is defined
function [sim]=SendUDP(sim, Signal, NValues_send)
  [sim,one] = ld_const(sim, 0, 1);

  // Packet counter, so the order of the network packages can be determined
  [sim, Counter] = ld_modcounter(sim, ev, in=one, initial_count=0, mod=100000);
  [sim, Counter_int32] = ld_ceilInt32(sim, ev, Counter);

  // Source ID
  [sim, SourceID] = ld_const(sim, ev, 4);
  [sim, SourceID_int32] = ld_ceilInt32(sim, ev, SourceID);

  // Sender ID
  [sim, SenderID] = ld_const(sim, ev, 1295793); // random number
  [sim, SenderID_int32] = ld_ceilInt32(sim, ev, SenderID);

  // print data
  [sim] = ld_printf(sim, ev, Signal, "Signal to send = ", NValues_send);

  // make a binary structure
  [sim, Data, NBytes] = ld_ConcateData(sim, ev, ...
                         inlist=list(SenderID_int32, Counter_int32, SourceID_int32, Signal ),
                         insizes=[1,1,1,NValues_send], ...
                         intypes=[ ORTD.DATATYPE_INT32, ORTD.DATATYPE_INT32, ORTD.DATATYPE_INT32,
                                   ORTD.DATATYPE_FLOAT ] );

  printf("The size of the UDP-packets will be %d bytes.\n", NBytes);

  // send to the network
  [sim, NBytes__] = ld_constvecInt32(sim, ev, vec=NBytes);
  [sim] = ld_UDPSocket_SendTo(sim, ev, SendSize=NBytes__, ObjectIdentifyer="aSocket", ...
                              hostname="127.0.0.1", UDPPort=10000, in=Data, ...
                              insize=NBytes);
endfunction
```

```
function [sim, outlist, userdata]=UDPReceiverThread(sim, inlist, userdata)
  // This will run in a thread. Each time a UDP-packet is received
  // one simulation step is performed. Herein, the packet is parsed
  // and the contained parameters are stored into a memory.

  // Sync the simulation to incomming UDP-packets
  [sim, Data, SrcAddr] = ld_UDPSocket_Recv(sim, 0, ObjectIdentifyer="aSocket", outsize=4+4+4+Nvalues_recv*8 );

  // disassemble packet's structure
  [sim, DisAsm] = ld_DisassembleData(sim, ev, in=Data, ...
                         outsizes=[1,1,1,Nvalues_recv], ...
                         outtypes=[ ORTD.DATATYPE_INT32, ORTD.DATATYPE_INT32, ORTD.DATATYPE_INT32,
ORTD.DATATYPE_FLOAT ] );

  [sim, DisAsm(1)] = ld_Int32ToFloat(sim, ev, DisAsm(1) );
  [sim, DisAsm(2)] = ld_Int32ToFloat(sim, ev, DisAsm(2) );
  [sim, DisAsm(3)] = ld_Int32ToFloat(sim, ev, DisAsm(3) );

  // print the contents
  [sim] = ld_printf(sim, ev, DisAsm(1), "DisAsm(1) (SenderID)       = ", 1);
  [sim] = ld_printf(sim, ev, DisAsm(2), "DisAsm(2) (Packet Counter) = ", 1);
  [sim] = ld_printf(sim, ev, DisAsm(3), "DisAsm(3) (SourceID)       = ", 1);
  [sim] = ld_printf(sim, ev, DisAsm(4), "DisAsm(4) (Signal)         = ", Nvalues_recv);

  // Store the input data into a shared memory
  [sim, one] = ld_const(sim, ev, 1);
  [sim] = ld_write_global_memory(sim, 0, data=DisAsm(4), index=one, ...
                ident_str="ParameterMemory", datatype=ORTD.DATATYPE_FLOAT, ...
                ElementsToWrite=Nvalues_recv);

  // output of schematic
  outlist = list();
endfunction

// The main real-time thread
function [sim, outlist, userdata]=Thread_MainRT(sim, inlist, userdata)
  // This will run in a thread
  [sim, Tpause] = ld_const(sim, ev, 1/20);  // The sampling time that is constant at 20 Hz in this example
  [sim, out] = ld_ClockSync(sim, ev, in=Tpause); // synchronise this simulation

  //
  // Add you own control system here
  //

  // Open an UDP-Port
  [sim] = ld_UDPSocket_shObj(sim, ev, ObjectIdentifyer="aSocket", Visibility=0, hostname="127.0.0.1",
UDPPort=10001);

  // Number of parameters
  Nvalues_recv = 2;

  // initialise a global memory for storing the input data for the computation
  [sim] = ld_global_memory(sim, ev, ident_str="ParameterMemory", ...
              datatype=ORTD.DATATYPE_FLOAT, len=Nvalues_recv, ...
              initial_data=[zeros(Nvalues_recv,1)], ...
              visibility='global', useMutex=1);

  // Create thread for the receiver
  ThreadPrioStruct.prio1=ORTD.ORTD_RT_NORMALTASK, ThreadPrioStruct.prio2=0, ThreadPrioStruct.cpu = -1;
  [sim, startcalc] = ld_const(sim, 0, 1); // triggers your computation during each time step
  [sim, outlist, computation_finished] = ld_async_simulation(sim, 0, ...
            inlist=list(), ...
            insizes=[], outsizes=[], ...
            intypes=[], outtypes=[], ...
            nested_fn = UDPReceiverThread, ...
            TriggerSignal=startcalc, name="Thread1", ...
            ThreadPrioStruct, userdata=list() );

  // Read the parameters
  [sim, readI] = ld_const(sim, ev, 1); // start at index 1
  [sim, Parameter1] = ld_read_global_memory(sim, ev, index=readI, ident_str="ParameterMemory", ...
                          datatype=ORTD.DATATYPE_FLOAT, 1);

  [sim, readI] = ld_const(sim, ev, 2); // start at index 2
  [sim, Parameter2] = ld_read_global_memory(sim, ev, index=readI, ident_str="ParameterMemory", ...
                          datatype=ORTD.DATATYPE_FLOAT, 1);

  [sim] = ld_printf(sim, ev, Parameter1, "Parameter1 ", 1);
  [sim] = ld_printf(sim, ev, Parameter2, "Parameter2 ", 1);

  // The system to control
  T_a = 0.1; [sim, x,v] = damped_oscillator(sim, Parameter1);

  // send
  [sim, Signal] = ld_mux(sim, 0, 2, list(x,v));
  [sim]=SendUDP(sim, Signal, NValues_send=2);

  outlist = list();
endfunction
```

```
// This is the main top level schematic
function [sim, outlist]=schematic_fn(sim, inlist)

//
// Create a thread that runs the control system
//

        ThreadPrioStruct.prio1=ORTD.ORTD_RT_NORMALTASK; // or  ORTD.ORTD_RT_NORMALTASK
        ThreadPrioStruct.prio2=0; // for ORTD.ORTD_RT_REALTIMETASK: 1-99 as described in   man sched_setscheduler
                                  // for ORTD.ORTD_RT_NORMALTASK this is the nice-value (higher value means less
                                  // priority)
        ThreadPrioStruct.cpu = -1; // The CPU on which the thread will run; -1 dynamically assigns to a CPU,
                                   // counting of the CPUs starts at 0

        [sim, StartThread] = ld_initimpuls(sim, ev); // triggers your computation only once
        [sim, outlist, computation_finished] = ld_async_simulation(sim, ev, ...
                              inlist=list(), ...
                              insizes=[], outsizes=[], ...
                              intypes=[], outtypes=[], ...
                              nested_fn = Thread_MainRT, ...
                              TriggerSignal=StartThread, name="MainRealtimeThread", ...
                              ThreadPrioStruct, userdata=list() );

   // output of schematic (empty)
   outlist = list();
endfunction



//
// Set-up (no detailed understanding necessary)
//

thispath = get_absolute_file_path(ProgramName+'.sce');
cd(thispath);
z = poly(0,'z');

// defile ev
ev = [0]; // main event

// set-up schematic by calling the user defined function "schematic_fn"
insizes = []; outsizes=[];
[sim_container_irpar, sim]=libdyn_setup_schematic(schematic_fn, insizes, outsizes);

// pack the simulation into a irpar container
parlist = new_irparam_set();
parlist = new_irparam_container(parlist, sim_container_irpar, 901); // pack simulations into irpar container with
id = 901
par = combine_irparam(parlist); // complete irparam set
save_irparam(par, ProgramName+'.ipar', ProgramName+'.rpar'); // Save the schematic to disk

// clear
par.ipar = []; par.rpar = [];
```