Adare_net Manual

version 0.0.128

Preparing a party

- Create a network address and port
  - many
  - just one
- Create a presence in network (socket)

The server part of the party

- I'm at port (bind)
- I'm listening you! Please connect!
- I accepted you! I waited you forever! thanks for connecting!
- I accepted you! But I'm so Busy! Thanks for connecting or Bye!

The client part of the party

- I'm connecting to you at address and port server!

Party Start!

- receive
- send
- receive_from
- sendto
- plain raw data, vulgo stream_element_array
- buffered data, vulgo socket_buffer
- plain raw data ou buffered data ?

Apendixes

- Full Client and Server TCP/IP example
- Full Client and Server UDP/IP example
- Hints for developers and users of others Network Ada Libraries
  - Anet
  - Gnat-sockets
  - A minimum gnat project to work with.
  - Use a task pool
  - Use Ada Class Wide types (Tagged Types) and Stream Socket_Buffer to see the real power of Adare_Net.

Preparing a party

Create a network address and port

- Many (actually until 10 addresses by each addresses_list)

```
declare
    many_addresses :  addresses_list_access := null;
begin
    init_addresses
      (ip_or_host   =>  "duckduckgo.com",
       port         =>  "25000", -- ignored without bind() or connect().
                                 -- Use "0" to choose automatically.
       ai_socktype  =>  tcp, -- or udp
       ai_family    =>  any, -- or v4 or v6
       addr         =>  many_addresses
      );

    if many_addresses.all'Length < 1 then
      TEXT_IO.Put_Line (" none address discovered ");
      return;
    end if

    utils.show_address_and_port (many_addresses);
end;
```

- Just one

```
declare
    mi_address :  addresses_access := null;
begin
    procedure init_addresses
      (ip_or_host   =>  "duckduckgo.com",
       port         =>  "25000", -- ignored without bind() or connect().
                                 -- Use "0" to choose automatically.
       ai_socktype  =>  tcp, -- or udp
       ai_family    =>  any, -- or v4 or else v6
       addr         =>  mi_address
      );

    if mi_address.all'Length < 1 then
      TEXT_IO.Put_Line (" none address discovered ");
      return;
    end if

    utils.show_address_and_port (many_addresses);
end;
```

Create a presence in network (socket)

```
declare
    mi_presence :  socket_access := null;
begin
    if init_socket (mi_presence, many_addresses) then
      TEXT_IO.Put_Line (" Worked! ");
      return;
    end if
end;
```

- or

```
declare
    mi_presence :  socket_access := null;
begin
    if init_socket (mi_presence, mi_address) then
      TEXT_IO.Put_Line (" Worked! ");
      return;
    end if
end;
```

The server part of the party

I'm at port (bind)

```
begin
    if bind (mi_presence) then -- port already choosed in init_addresses().
      TEXT_IO.Put_Line (" Worked! ");
      return;
    end if
end;
```

I'm listening you! Please connect!

```
declare
    Backlog : constant := 70; -- is up to you the quantitie.
begin
    if listen (mi_presence, Backlog) then -- can be IPV6 too.
      TEXT_IO.Put_Line (" Worked! ");
      return;
    end if
end;
```

I accepted you! I waited you forever! thanks for connecting!

```
declare
    remote_presence : socket_access := null;
begin
    if accept_socket (mi_presence, remote_presence) then

      -- make something util with the remote_presence.
    end if
end;
```

I Want accepted you! But I'm so Busy! Thanks for connecting or Bye!

```
declare
    remote_presence : socket_access := null;
    mi_poll         : aliased polls.poll_type (2);
begin
    polls.add_events (mi_poll'Access, mi_presence, polls.accept_ev);

    if polls.start_events_listen (mi_poll'Access, 15000) < 1 then -- block, 15 seconds timeout

      close (mi_presence); -- to disable 'listen' too.

      Text_IO.Put_Line (" I'm so busy! Have a good day and night, Bye!");

      return;
    end if;

    if accept_socket (mi_presence, remote_presence) then

      -- make something util with the remote_presence.
    end if
end;
```

The client part of the party

I'm connecting to you at address and port server!

```
  declare
        server_address  :  addresses_list_access := null;
        host_sock       :  socket_access := null;
  begin
        init_addresses
        (ip_or_host   =>  "127.0.0.1",
         port         =>  "25000",
         ai_socktype  =>  tcp, -- or udp
         ai_family    =>  v4, -- or any
         addr         =>  server_address
        );

        if server_address.all'Length < 1 then
          TEXT_IO.Put_Line (" none address discovered ");
          return;
        end if

        if not init_socket (host_sock, server_address) then
          TEXT_IO.Put_Line (" cannot init point of presence ");
          TEXT_IO.Put_Line (" error => " & string_error);
          return;
        end if;

        if not connect (host_sock) then
          Text_IO.Put_Line (" Error while trying connect to remote host:");
          Text_IO.Put_Line (" " & string_error);

          return;
        end if;

        -- make something util with the host_sock e.g.: send, receive, poll etc
```

```
end;
```

- or

```
declare
      server_address  :  addresses_list_access := null;
      host_sock       :  socket_access := null;
begin
      init_addresses
      (ip_or_host   =>  "::1",
       port         =>  "25000",
       ai_socktype  =>  tcp, -- or udp
       ai_family    =>  v6, -- or any
       addr         =>  server_address
      );

      if server_address.all'Length < 1 then
        TEXT_IO.Put_Line (" none address discovered ");
        return;
      end if

      if not init_socket (host_sock, server_address) then
        TEXT_IO.Put_Line (" cannot init point of presence ");
        TEXT_IO.Put_Line (" error => " & string_error);
        return;
      end if;

      if not connect (host_sock) then
        Text_IO.Put_Line (" Error while trying connect to remote host:");
        Text_IO.Put_Line (" " & string_error);

        return;
      end if;

      -- make something util with the host_sock e.g.: send, receive, poll etc

end;
```

Party Start!

receive

```
function receive
  (sock     : not null socket_access;
   buffer   : out stream_element_array_access;
   max_len  : Stream_Element_Count := 1500) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => a stream_element_array_access variable. the length is equal to
             returned value or 0. buffer allways return a new buffer in this function,
             but don't touch the old value. buffer can be a null
             stream_element_array_access variable.
  max_len => the _maximum_ length to receive in one go.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size received, 1 or more.
```

```
eg.:

  declare
    mi_buff : stream_element_array_access := null;
    count_receive : ssize_t;
  begin
    count_receive := receive (host_sock, mi_buff);
    -- verify count_receive =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just use buffer.
  end;
```

- or

```
function receive
  (sock     : not null socket_access;
   buffer   : not null socket_buffer_access;
   max_len  : Stream_Element_Count := 1500) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => an initialized socket_buffer. the received data will be
             automatically appended to It.
  max_len => the _maximum_ length to receive in one go.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size received, 1 or more.

eg.:

  declare
    mi_buff : socket_buffer_access := new socket_buffer;
    count_receive : ssize_t;
  begin
    clean (mi_buff); -- optional. will wipe all data.
    count_receive := receive (host_sock, mi_buff);
    -- verify count_receive =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just use buffer.
  end;
```

send

```
function send
  (sock     : not null socket_access;
   buffer   : not null stream_element_array_access) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => an not null stream_stream_element_array_access.
             send(), by Itself, will try send _all_ data in buffer.
             buffer data remain untouched.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size sended => buffer.all'length
```

```
eg.:

declare
  mi_buff : stream_element_array_access := new stream_element_array'(1 .. 4 => 0);
  count_sended  : ssize_t;
begin
  count_sended := send (host_sock, mi_buff);
  -- verify count_sended =>   equal 0? or else equal socket_error?
  -- yes ? show  string_error function
  -- no? just do more work.
end;
```

- or

```
function send
  (sock     : not null socket_access;
   buffer   : not null socket_buffer_access) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => an initialized socket_buffer.
             send(), by Itself, will try send _all_ data in buffer.
              if all data was sended, buffer becomes empty,
              otherwise buffer data remain untouched.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size sended, old actual_data_size (buffer).

eg.:

  declare
    mi_buff : socket_buffer_access := new socket_buffer;
    count_receive : ssize_t;
  begin
    clean (mi_buff); -- optional. will wipe all data.
    String'Output (mi_buff, "Dani & Cia"); -- automatic conversion
    Integer'Output (mi_buff, 738); -- automatic conversion

    count_sended := send (host_sock, mi_buff);
    -- verify count_sended =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just do more work.
  end;
```

receive_from

```
function receive_from
  (sock     : not null socket_access;
   buffer   : out stream_element_array_access;
   from     : out addresses_access;
   max_len  : Stream_Element_Count := 1500) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => a stream_element_array_access variable. the length is equal to
             returned value or 0. buffer allways return a new buffer in this function,
```

```
            but don't touch the old value. buffer can be a null
            stream_element_array_access variable.
  from    => return a new socket_access value. It don´t touch the old value.
  max_len => the _maximum_ length to receive in one go.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size received, 1 or more.
```

eg.:

```
  declare
    mi_buff   : stream_element_array_access := null;
    from_sock : socket_access := null; -- or from someone else.
    count_receive : ssize_t;
  begin
    count_receive := receive_from (host_sock, mi_buff, from_sock);
    -- verify count_receive =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just use buffer.
  end;
```

- or

```
function receive_from
  (sock     : not null socket_access;
   buffer   : not null socket_buffer_access;
   from     : out addresses_access;
   max_len  : Stream_Element_Count := 1500) return ssize_t
   with  pre => initialized (sock);

  sock    => an initialized socket.
  buffer  => an initialized socket_buffer. the received data will be
             automatically appended to It.
  from    => return a new socket_access value. receive_from() don´t touch the old value.
  max_len => the _maximum_ length to receive in one go.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size received, 1 or more.
```

eg.:

```
  declare
    mi_buff   : socket_buffer_access  := new socket_buffer; -- or from someone else
    from_sock : socket_access         := null; -- or from someone else.
    count_receive : ssize_t;
  begin
    count_receive := receive_from (host_sock, mi_buff, from_sock);
    -- verify count_receive =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just use buffer.
  end;
```

sendto

```
function sendto
    (sock     : not null socket_access;
     send_to  : not null addresses_access;
     buffer   : not null stream_element_array_access) return ssize_t
     with  pre => initialized (sock) and then initialized (send_to);

    sock    => an initialized socket.
    send_to => an initialized addresses.
    buffer  => an not null stream_stream_element_array_access.
               sendto(), by Itself, will try send _all_ data in buffer.
               buffer data remain untouched.

    return value =>
      'socket_error' when error
      '0' when remote node closed the remote socket
       if ok return size sended => buffer.all'length
```

eg.:

```
  declare
    mi_buff : stream_element_array_access := new stream_element_array'(1 .. 4 => 0);
    to_addr : addresses_access := null;
    count_sended  : ssize_t;
  begin
    init_addresses
            (ip_or_host   =>  "127.0.0.1",
             port         =>  "25000",
             ai_socktype  =>  tcp, -- or udp
             ai_family    =>  v4, -- or any
             addr         =>  to_addr
            );
    count_sended := sendto (host_sock, to_addr, mi_buff);
    -- verify count_sended =>   equal 0? or else equal socket_error?
    -- yes ? show  string_error function
    -- no? just do more work.
  end;
```

- or

```
function sendto
    (sock     : not null socket_access;
     send_to  : not null addresses_access;
     buffer   : not null stream_element_array_access) return ssize_t
     with  pre => initialized (sock) and then initialized (send_to);

    sock    => an initialized socket.
    send_to => an initialized addresses.
    buffer  => an not null stream_stream_element_array_access.
               sendto(), by Itself, will try send _all_ data in buffer.
               buffer data remain untouched.

    return value =>
      'socket_error' when error
      '0' when remote node closed the remote socket
       if ok return size sended => buffer.all'length
```

eg.:

```
declare
  mi_buff : stream_element_array_access := new stream_element_array'(1 .. 4 => 0);
  to_addr : addresses_access := null;
  count_sended  : ssize_t;
begin
  init_addresses
          (ip_or_host   =>  "::1",
           port         =>  "25000",
           ai_socktype  =>  tcp, -- or udp
           ai_family    =>  v6, -- or any
           addr         =>  to_addr
          );
  count_sended := sendto (host_sock, to_addr, mi_buff);
  -- verify count_sended =>   equal 0? or else equal socket_error?
  -- yes ? show  string_error function
  -- no? just do more work.
end;
```

- or

```
function sendto
  (sock     : not null socket_access;
   send_to  : not null addresses_access;
   buffer   : not null socket_buffer_access) return ssize_t
   with  pre => initialized (sock) and then initialized (send_to);

  sock    => an initialized socket.
  send_to => an initialized addresses.
  buffer  => an initialized socket_buffer.
  buffer  => sendto(), by Itself, will try send _all_ data in buffer.
              if all data was sended, buffer becomes empty,
              otherwise buffer data remain untouched.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size sended => buffer.all'length
```

eg.:

```
declare
  mi_buff : socket_buffer_access := new socket_buffer;
  to_addr : addresses_access := null;
  count_sended  : ssize_t;
begin
  init_addresses
          (ip_or_host   =>  "127.0.0.1",
           port         =>  "25000",
           ai_socktype  =>  tcp, -- or udp
           ai_family    =>  v4, -- or any
           addr         =>  to_addr
          );
  clean (mi_buff); -- optional. will wipe all data.
  String'Output (mi_buff, "Dani & Cia"); -- automatic conversion
  Integer'Output (mi_buff, 738); -- automatic conversion

  count_sended := sendto (host_sock, to_addr, mi_buff);
  -- verify count_sended =>   equal 0? or else equal socket_error?
```

```
  -- yes ? show  string_error function
  -- no? just do more work.
end;
```

- or

```
function sendto
  (sock     : not null socket_access;
   send_to  : not null addresses_access;
   buffer   : not null socket_buffer_access) return ssize_t
   with  pre => initialized (sock) and then initialized (send_to);

  sock    => an initialized socket.
  send_to => an initialized addresses.
  buffer  => an initialized socket_buffer.
  buffer  => sendto(), by Itself, will try send _all_ data in buffer.
              if all data was sended, buffer becomes empty,
              otherwise buffer data remain untouched.

  return value =>
    'socket_error' when error
    '0' when remote node closed the remote socket
     if ok return size sended => buffer.all'length

eg.:

declare
  mi_buff : socket_buffer_access := new socket_buffer;
  to_addr : addresses_access := null;
  count_sended  : ssize_t;
begin
  init_addresses
          (ip_or_host   =>  "::1",
           port         =>  "25000",
           ai_socktype  =>  tcp, -- or udp
           ai_family    =>  v6, -- or any
           addr         =>  to_addr
          );
  clean (mi_buff); -- optional. will wipe all data.
  String'Output (mi_buff, "Dani & Cia"); -- automatic conversion
  Integer'Output (mi_buff, 738); -- automatic conversion

  count_sended := sendto (host_sock, to_addr, mi_buff);
  -- verify count_sended =>   equal 0? or else equal socket_error?
  -- yes ? show  string_error function
  -- no? just do more work.
end;
```

Hints about plain raw data, vulgo stream_element_array and about buffered data, vulgo socket_buffer

`You can read more length chunks, eg.: a file and send it 'as is' to a node,`

without need to read (and write) stream by stream from filesystem storage.

`The data received or sended can be used as buffer, and if are missing data, you can`

'rewind' last read, before get/write more data in buffer. the raw data can be setted in buffer with add_raw() and getted with get_raw() from buffer.

To Fill buffer with data, use Ada Streams 'write and 'output. To get data from buffer, use Ada Streams 'read and 'input.

plain raw data ou buffered data ?

Both are OK. It depends more on your project and your needs. Other than that

it's more a matter of which way you like it best.

Appendix

Full Client and Server TCP/IP example

```
-- This is an over simplified example of tcp client with Adare_net, :-)
-- but is yet up to you create a real world champion software with Adare_net and you can do it!! ^^

-- Info about this software:
-- Tcp client with Adare_net example. It work in pair with tcp server

with Ada.Command_Line;
with Ada.Text_IO;
use Ada, Ada.Command_Line;

with adare_net.sockets.polls;
with adare_net.sockets.utils;
use adare_net.sockets;

with adare_net_exceptions;
use adare_net_exceptions;

with adare_net_init;
use  adare_net_init;

with socket_types;
use socket_types;

with Interfaces.C;
use Interfaces.C;

procedure client
is
begin

  start_adare_net;

  if Argument_Count < 4 then

    Text_IO.New_Line;

    Text_IO.Put_Line (" Usage: " & Command_Name & " host port ""message1"" ""message2"" ""message_$n"" ");
    Text_IO.New_Line;
    Text_IO.Put_Line (" Minimum of 2 messages ");
    Text_IO.New_Line (2);
    Text_IO.Put_Line (" It will also show that 'buffer' can be readed and written offline ");

    Text_IO.New_Line;

    Set_Exit_Status (Failure);

    stop_adare_net;

    return;
  end if;
```

```
Text_IO.New_Line;

b0 :
declare
  buffer  : socket_buffer_access := new socket_buffer;
begin
  clean (buffer);

  for qtd in 3 .. Argument_Count loop
    String'Output (buffer, Argument (qtd)); -- automatic conversion
  end loop;

  b1 :
  declare
    remote_addr  : addresses_list_access;

    ok  :  Boolean := False;

    host_sock             : socket_access     := null;
    choosed_remote_addr   : addresses_access  := null;

    bytes_tmp : ssize_t :=  0;

    host_poll : aliased polls.poll_type (2);

    result_from_poll : int := 0;
  begin
    init_addresses
      (ip_or_host    =>  Argument (1),
       port         =>  Argument (2),
       ai_socktype  =>  tcp,
       ai_family    =>  any,
       addr         =>  remote_addr);

    if remote_addr.all'Length < 1 then

      Text_IO.New_Line;
      Text_IO.Put_Line (" Failed to discover remote host addresses.");
      Text_IO.Put_Line (" Quitting.");
      Text_IO.New_Line;

      goto end_app_label1;
    end if;

    Text_IO.Put_Line (" Remote host addresses discovered:");

    utils.show_address_and_port (remote_addr);

    if not init_socket (host_sock, remote_addr) then

      Text_IO.New_Line;
      Text_IO.Put_Line (" Error while trying initialize socket:");
      Text_IO.Put_Line (" " & string_error);
      Text_IO.Put_Line (" Quiting.");

      goto end_app_label1;
    end if;

    if not connect (host_sock) then

      Text_IO.New_Line;
```

```
  Text_IO.Put_Line (" Error while trying connect to remote host:");
  Text_IO.Put_Line (" " & string_error);
  Text_IO.Put_Line (" Quiting.");

  goto end_app_label1;
end if;

choosed_remote_addr  :=  get_addresses (host_sock);

Text_IO.New_Line;

Text_IO.Put_Line (" Connected at address :=  "  & get_addresses (choosed_remote_addr) &
  " and at port := " & get_port (choosed_remote_addr));

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting to send messages. ");

polls.add_events (host_poll'Access, host_sock, polls.send_ev);

result_from_poll := polls.start_events_listen (host_poll'Access, 2500); -- block, 2.5 seconds timeout.

if result_from_poll > 0 then

  bytes_tmp := send (host_sock, buffer);

  if bytes_tmp = socket_error then

    Text_IO.Put_Line (" An error occurred during sending data.");
    Text_IO.Put_Line (" Finishing task.");

    goto end_app_label1;
  end if;

  if bytes_tmp > 0 then

    Text_IO.Put_Line (" Successfull sended " & bytes_tmp'Image & " bytes.");

  else

    Text_IO.Put_Line (" Failed in send messages to " & get_address_and_port (choosed_remote_addr));

    if bytes_tmp = 0 then

      Text_IO.Put_Line ("remote closed the socket");

    else

      Text_IO.Put_Line ("With Error => " & string_error);

    end if;
  end if;

else

  Text_IO.Put_Line (" Failed to send to remote host " & get_address_and_port (choosed_remote_addr));

  if result_from_poll = 0 then

    Text_IO.Put_Line (" But it is just only a normal 2.5 seconds time_out");
```

```
  else

    if polls.hang_up_error (host_poll'Access, host_sock) then

      Text_IO.Put_Line (" Remote Host " & get_address_and_port (choosed_remote_addr) & " closed the conn

      Text_IO.Put_Line (" Nothing more to do. Quitting.");

      goto end_app_label1;

    end if;
  end if;
end if;

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting to receive message(s). ");

polls.update (host_poll'Access, host_sock, polls.receive_ev);

result_from_poll := polls.start_events_listen (host_poll'Access, 2500); -- block, 2.5 seconds timout

if result_from_poll > 0 then

  bytes_tmp  := receive (host_sock, buffer); -- block

  if bytes_tmp = socket_error then

    Text_IO.Put_Line (" An error occurred during receiving data.");
    Text_IO.Put_Line (" Finishing task.");

    goto end_app_label1;
  end if;

  if bytes_tmp > 0 then

    Text_IO.Put_Line (" Received message(s) from " & get_address_and_port (choosed_remote_addr));

    Text_IO.Put_Line (" Messages length " & bytes_tmp'Image & " bytes.");

    Text_IO.New_Line;

    Text_IO.Put_Line (" Messages:");

    b2 :
    begin

      loop3 :
      loop

        Text_IO.Put_Line (" |" & String'Input (buffer) & "|");

      end loop loop3;

    exception
      when buffer_insufficient_space_error =>

        Text_IO.New_Line;
        Text_IO.Put_Line (" All messages received from " & get_address_and_port (choosed_remote_addr) &
    end b2;
```

```
        ok := True;
      else
        Text_IO.Put_Line (" Failed in receive messages from " & get_address_and_port (choosed_remote_addr));
        if bytes_tmp = 0 then
          Text_IO.Put_Line ("remote host closed the socket");
        else
          Text_IO.Put_Line ("With Error => " & string_error);
        end if;
      end if;
    else
      Text_IO.Put_Line (" Failed in receive messages from " & get_address_and_port (choosed_remote_addr));
      if result_from_poll = 0 then
        Text_IO.Put_Line (" But it is just only a normal 2.5 seconds " & " time_out");
        ok := True;
      else
        if polls.hang_up_error (host_poll'Access, host_sock) then
          Text_IO.Put_Line (" Remote Host " & get_address_and_port (choosed_remote_addr) & " closed the conn
          Text_IO.Put_Line (" Besides reconnect, nothing to do in this case." & " quitting.");
        end if;
      end if;
    end if;

    <<end_app_label1>>

    if initialized (host_sock) then
      close (host_sock);
    end if;

    Text_IO.New_Line;

    Text_IO.Put (" " & Command_Line.Command_Name);

    if ok then
      Text_IO.Put (" successfull ");
    else
      Text_IO.Put (" unsuccess ");
    end if;

    Text_IO.Put_Line ("finalized.");
    Text_IO.New_Line;
  end b1;
end b0;
```

```
  stop_adare_net;

end client;



-- Besides this is a multitask and reasonable complete example with Adare_net, you can do more, as:
--
-- (1) More que one listen socket and polls,
-- (2) More length polls,
-- (3) Simultaneous listen event_types,
-- (4) Use of others types beyond String:
-- (4.1) From built-in types and records to
-- (4.2) Wide class(es) and tagged types
-- (4.3) And with a more fine treatment, all records, tagged types included, can be endian proof.
-- (5) Etc. ^^
-- But is yet up to you create a yet better real world champion software with Adare_net and you can do it!! ^^

-- Info about this software:
-- Tcp1 server with Adare_net example. It work in pair with tcp1 client
-- Automatically, the working address can be ipv6 or ipv4. The first working one will be picked.

with Ada.Text_IO;
use Ada;

with Ada.Task_Identification;

with Ada.Command_Line;
with Ada.Strings.Unbounded;

with adare_net.sockets.utils;
with adare_net.sockets.polls;
use adare_net.sockets;

with adare_net_exceptions;
use adare_net_exceptions;

with socket_types;
use  socket_types;

with adare_net_init;
use  adare_net_init;

with Interfaces.C;
use Interfaces, Interfaces.C;

procedure server
is
begin

  start_adare_net;

  b0 :
  declare
    host_addr           : addresses_list_access := null;
    choosed_remote_addr : addresses_access := null;
    host_sock           : socket_access := null;
    ok                  : Boolean := False;
  begin
    init_addresses (ip_or_host =>  "", -- host addresses
                      port        =>  "25000",
```

```
                  ai_socktype =>  tcp,
                  ai_family   =>  any, -- choose ipv4 and ipv6
                  addr        =>  host_addr);

if host_addr.all'Length < 1 then

  Text_IO.Put_Line (" Failed to discover addresses in this host. Quitting.");

  goto end_app_label1;
end if;

Text_IO.Put_Line (" Addresses Discovered in this host:");

utils.show_address_and_port (host_addr);

if not init_socket (host_sock, host_addr) then

  Text_IO.New_Line;
  Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

  goto end_app_label1;
end if;

reuse_address (host_sock);

if not bind (host_sock) then

  Text_IO.New_Line;
  Text_IO.Put_Line (" Bind error: " & string_error);

  goto end_app_label1;
end if;

if not listen (host_sock, 9) then

  Text_IO.New_Line;
  Text_IO.Put_Line (" Listen error: " & string_error);

  goto end_app_label1;
end if;

choosed_remote_addr  :=  get_addresses (host_sock);

Text_IO.New_Line;

Text_IO.Put_Line (" Binded and Listening at address "  &
  get_addresses (choosed_remote_addr) &
  " and at port " & get_port (choosed_remote_addr));

b1 :
declare

  incomming_socket  : socket_access;
  mi_poll           : aliased polls.poll_type (1);

  task type recv_send_task (connected_sock  : socket_access);

  task body recv_send_task
  is
    remote_address  : addresses_access := get_addresses (connected_sock);
  begin
```

```
if not (initialized (connected_sock) or else connected (connected_sock)) then
  Text_IO.Put_Line (" Incomming socket not initialized or connected.");
  Text_IO.Put_Line (" Quitting this working task.");

  goto finish2_task_label;
end if;

bt0 :
declare

  use Task_Identification;

  this_task_id_str  : constant String := Image (Current_Task);

  task_poll         : aliased polls.poll_type (1);

  recv_send_buffer  : socket_buffer_access := new socket_buffer;
  recv_send_buffer2 : socket_buffer_access := new socket_buffer;

  size_tmp          : ssize_t  := 1;
  result_from_poll  : int := 0;

  use  Ada.Strings.Unbounded;

  message : Unbounded_String := To_Unbounded_String ("");
begin
  clean (recv_send_buffer);
  clean (recv_send_buffer2);

  Text_IO.Put_Line (" " & this_task_id_str & " remote host " &
    get_address_and_port (remote_address));

  polls.add_events (task_poll'Access, connected_sock, polls.receive_ev); -- all *_ev events can be or'

  Text_IO.Put_Line (" " & this_task_id_str & " waiting to receive data.");

  result_from_poll  := polls.start_events_listen (task_poll'Access, 3000); -- block, 3 seconds time_ou

  if result_from_poll > 0 then

    size_tmp  := receive (connected_sock, recv_send_buffer); -- block and initialize recv_send_buffer

    if size_tmp = socket_error then

      Text_IO.Put_Line (" " & this_task_id_str & " An error occurred during reception.");
      Text_IO.Put_Line (" " & this_task_id_str & " finishing task.");

      goto finish1_task_label;
    end if;

    Text_IO.Put_Line (" " & this_task_id_str & " received message:");
    Text_IO.Put_Line (" " & this_task_id_str & " message len " & size_tmp'Image & " bytes.");

    if size_tmp > 0 then

      bt1 :
      begin
        String'Output (recv_send_buffer2, "Thank you for send ");

        loop1 :
```

```
      loop
        message := To_Unbounded_String (String'Input (recv_send_buffer));

        String'Output (recv_send_buffer2, To_String (message));

        Text_IO.Put_Line (" " & this_task_id_str & " message |" & To_String (message) & "|");
      end loop loop1;

    exception

      when buffer_insufficient_space_error =>

        Text_IO.Put_Line (" " & this_task_id_str & " all messages showed.");

    end bt1;

  end if;
else

  Text_IO.Put_Line (" " & this_task_id_str & " failed in receive data.");

  if result_from_poll = 0 then

    Text_IO.Put_Line (" " & this_task_id_str & " but it is a normal time_out.");

  else
    if polls.hang_up_error (task_poll'Access, connected_sock) then

      Text_IO.Put_Line (" " & this_task_id_str & " remote host closed the connection. Quitting.");

      goto finish1_task_label;
    end if;
  end if;
end if;

polls.clear_all_event_responses (task_poll'Access);

polls.update (task_poll'Access, connected_sock, polls.send_ev);

Text_IO.Put_Line (" " & this_task_id_str & " waiting to send data to remote host");

result_from_poll  := polls.start_events_listen (task_poll'Access, 3000); -- block, 3 seconds timeout

if result_from_poll > 0 then

  size_tmp  := send (connected_sock, recv_send_buffer2); -- block

  Text_IO.Put_Line (" " & this_task_id_str & " sended messages !");

else

  Text_IO.Put_Line (" " & this_task_id_str & " failed in send data.");

  if result_from_poll = 0 then

    Text_IO.Put_Line (" " & this_task_id_str & " but it is a normal time_out");

  else

    if polls.hang_up_error (task_poll'Access, connected_sock) then
```

```
            Text_IO.Put_Line (" " & this_task_id_str & " remote host closed the connection. Quitting");

          end if;
        end if;
      end if;

      <<finish1_task_label>>

      if initialized (connected_sock) then

        close (connected_sock);

      end if;
    end bt0;

    <<finish2_task_label>>
  end recv_send_task;

  type recv_send_access is access all recv_send_task;

  working_task  : recv_send_access
    with Unreferenced;

begin

  Text_IO.New_Line;

  polls.add_events (mi_poll'Access, host_sock, polls.accept_ev);

  ok := True;

  loop2 :
  loop

    if polls.start_events_listen (mi_poll'Access, 15000) < 1 then -- block, 15 seconds timeout

      close (host_sock); -- to disable 'listen' too.

      Text_IO.Put_Line (" Main event 15 seconds Time_out.");
      Text_IO.Put_Line (" Waiting 5 seconds to allow enough time for working tasks finish.");

      Text_IO.New_Line;

      delay 5.0;

      Text_IO.Put_Line (" Have a nice day and night. Bye!");
      Text_IO.New_Line;

      exit loop2;
    end if;

    if accept_socket (host_sock, incomming_socket) then -- block. accepted socket incomming_socket is allw
      -- For the curious: We believe the task(s) will not leak.
      -- Reason: ARM-2012 7.6 (9.2/2) :-)
      working_task  :=  new recv_send_task (incomming_socket);
    end if;

    polls.clear_all_event_responses (mi_poll'Access);

  end loop loop2;
end b1;
```

```
    <<end_app_label1>>

    if initialized (host_sock) then

      close (host_sock);

    end if;

    Text_IO.Put (" " & Command_Line.Command_Name);

    if ok then
      Text_IO.Put_Line (" successfully finalized.");
    else
      Text_IO.Put_Line (" failed.");
    end if;

    Text_IO.New_Line;

  end b0;

  stop_adare_net;

end server;
```

Full Client and Server UDP/IP example

```
-- This is an over simplified example of udp client with Adare_net, :-)
-- but is yet up to you create a real world champion software with Adare_net and you can do it!! ^^

-- Info about this software:
-- Udp client with Adare_net example. It work in pair with udp server


with Ada.Command_Line;
with Ada.Text_IO;
use Ada, Ada.Command_Line;

with adare_net.sockets.polls;
with adare_net.sockets.utils;
use adare_net.sockets;

with adare_net_exceptions;
use adare_net_exceptions;

with adare_net_init;
use  adare_net_init;

with socket_types;
use socket_types;

with Interfaces.C;
use Interfaces.C;

procedure client
is
begin
```

```
start_adare_net;

if Argument_Count < 4 then

  Text_IO.New_Line;

  Text_IO.Put_Line (" Usage: " & Command_Name & " host port ""message1"" ""message2"" ""message_$n"" ");
  Text_IO.New_Line;
  Text_IO.Put_Line (" Minimum of 2 messages ");
  Text_IO.New_Line (2);
  Text_IO.Put_Line (" It will also show that 'buffer' can be readed and writed offline ");

  Text_IO.New_Line;

  Set_Exit_Status (Failure);

  stop_adare_net;

  return;
end if;

Text_IO.New_Line;

b0 :
declare

  buffer  : socket_buffer_access  := new socket_buffer;

begin

  clean (buffer);

  for qtd in 3 .. Argument_Count loop
    String'Output (buffer, Argument (qtd)); -- automatic conversion
  end loop;

  b1 :
  declare
    local_addr    : addresses_list_access := null;
    local_addr2   : addresses_access := null;
    remote_addr   : addresses_access  := null;
    remote_addr2  : addresses_access  := null;
    local_sock    : socket_access := null;

    bytes_tmp     : ssize_t :=  0;

    local_poll    : aliased polls.poll_type (2);

    poll_result   : int := 0;

    ok  : Boolean := False;
  begin

    init_addresses
      (ip_or_host   =>  Argument (1),
       port         =>  Argument (2),
       ai_socktype  =>  udp,
       ai_family    =>  any,
       addr         => remote_addr);

    init_addresses
```

```
  (ip_or_host   =>  "",
   port         =>  "0", -- ignored without 'bind'
   ai_socktype  =>  udp,
   ai_family    =>  get_address_family (remote_addr),
   addr         =>  local_addr);

if is_null (remote_addr) then

  Text_IO.New_Line;

  Text_IO.Put_Line (" Failed to discover remote host addresses. Quitting.");

  Text_IO.New_Line;

  goto end_app_label1;
end if;

if local_addr.all'Length < 1 then

  Text_IO.New_Line;

  Text_IO.Put_Line (" Failed to discover local host addresses. Quitting.");

  Text_IO.New_Line;

  goto end_app_label1;
end if;

Text_IO.Put_Line (" Remote host addresses discovered:");

utils.show_address_and_port (remote_addr);

Text_IO.New_Line;

Text_IO.Put_Line (" Host addresses discovered:");

utils.show_address_and_port (local_addr);

Text_IO.New_Line;

if not init_socket (local_sock, local_addr) then

  Text_IO.New_Line;

  Text_IO.Put_Line (" Error while trying initialize socket because: " & string_error & ".");
  Text_IO.Put_Line (" Quiting.");

  goto end_app_label1;
end if;

local_addr2 := get_addresses (local_sock);

Text_IO.New_Line;

Text_IO.Put_Line (" Host address choosed :=  "  & get_addresses (local_addr2) &
  " and at port := " & get_port (local_addr2));

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting to send messages. ");
```

```
polls.reset_all (local_poll'Access);

polls.add_events (local_poll'Access, local_sock, polls.send_ev);

poll_result :=  polls.start_events_listen (local_poll'Access, 2500); -- block, 2.5 seconds timeout.

if poll_result > 0 then

  bytes_tmp := sendto (local_sock, remote_addr, buffer);

  if bytes_tmp = socket_error then

    Text_IO.Put_Line (" An error occurred during sending data.");
    Text_IO.Put_Line (" Finishing task.");

    goto end_app_label1;
  end if;

  Text_IO.Put_Line (" Successfull sended " & bytes_tmp'Image & " bytes.");

else

  Text_IO.Put_Line (" Failed to send to remote host " & get_address_and_port (remote_addr));

  if poll_result = 0 then

    Text_IO.Put_Line (" But it is just only a normal 2.5 seconds time_out");

  else

    if polls.hang_up_error (local_poll'Access, local_sock) then

      Text_IO.Put_Line (" Remote Host " & get_address_and_port (remote_addr) & " closed the connection.

      Text_IO.Put_Line (" Nothing more to do. Quitting.");

      goto end_app_label1;
    end if;
  end if;
end if;

Text_IO.New_Line;
Text_IO.Put_Line (" Waiting to receive message(s). ");

polls.clear_all_event_responses (local_poll'Access);

polls.update (local_poll'Access, local_sock, polls.receive_ev);

poll_result := polls.start_events_listen (local_poll'Access, 2500); -- block, 2.5 seconds timeout.

if poll_result > 0 then

  clean (buffer);

  bytes_tmp  := receive_from (local_sock, buffer, remote_addr2); -- block

  if bytes_tmp = socket_error then

    Text_IO.Put_Line (" An error occurred during receiving data.");
    Text_IO.Put_Line (" Finishing task.");
```

```
    goto end_app_label1;
  end if;

  if bytes_tmp > 0 then

    Text_IO.Put_Line (" Received message(s) from " & get_address_and_port (remote_addr2));

    Text_IO.Put_Line (" Messages length " & bytes_tmp'Image & " bytes.");

    Text_IO.New_Line;

    Text_IO.Put_Line (" Messages:");

    b2 :
    begin
      loop1 :
      loop

        Text_IO.Put_Line (" |" & String'Input (buffer) & "|");

      end loop loop1;

    exception
      when buffer_insufficient_space_error =>

        Text_IO.New_Line;

        Text_IO.Put_Line (" All messages received from " & get_address_and_port (remote_addr2) & " showe

    end b2;

    ok := True;

  else

    Text_IO.Put_Line (" Failed in receive messages from " & get_address_and_port (remote_addr));

    if bytes_tmp = 0 then

      Text_IO.Put_Line ("remote closed the socket");

    elsif bytes_tmp < 0 then

      Text_IO.Put_Line ("With Error => " & string_error);

    end if;
  end if;

  Text_IO.New_Line;

else

  Text_IO.Put_Line (" Failed in receive messages from " & get_address_and_port (remote_addr));

  if poll_result = 0 then

    Text_IO.Put_Line (" But it is just only a normal 2.5 seconds " & " time_out");

    ok := True;

  else
```

```
          if polls.hang_up_error (local_poll'Access, local_sock) then

            Text_IO.Put_Line (" Remote Host " & get_address_and_port (remote_addr) & " closed the connection.

            Text_IO.Put_Line (" Besides reconnect, nothing to do in this case." & " quitting.");

          end if;
        end if;
      end if;

      <<end_app_label1>>

      if initialized (local_sock) then

        close (local_sock);

      end if;

      Text_IO.New_Line;

      Text_IO.Put (" " & Command_Line.Command_Name);

      if ok then
        Text_IO.Put (" successfull ");
      else
        Text_IO.Put (" unsuccess ");
      end if;

      Text_IO.Put_Line ("finalized.");

      Text_IO.New_Line;
    end b1;
  end b0;

  stop_adare_net;

end client;



-- Besides this is a multitask and reasonable complete example with Adare_net, you can do more, as:
--
-- (1) More que one listen socket and polls,
-- (2) More length polls,
-- (3) Simultaneous listen event_types,
-- (4) Use of others types beyond String:
-- (4.1) From built-in types and records to
-- (4.2) Wide class and tagged types
-- (4.3) And with a more fine treatment, all records, tagged types included, can be endian proof.
-- (5) Etc. ^^
-- But is yet up to you create a yet better real world champion software with Adare_net and you can do it!! ^^

-- Info about this software:
-- Udp server with Adare_net example. It work in pair with udp client
-- The working address can be ipv6 or ipv4, automatically. The first working one will be picked.

with Ada.Text_IO;
use Ada;

with Ada.Task_Identification;
```

```
with Ada.Command_Line;
with Ada.Strings.Unbounded;

with adare_net.sockets.utils;
with adare_net.sockets.polls;
use adare_net.sockets;

with adare_net_exceptions;
use adare_net_exceptions;

with socket_types;
use  socket_types;

with adare_net_init;
use  adare_net_init;

with Interfaces.C;
use Interfaces, Interfaces.C;

procedure server
is
begin
  start_adare_net;

  b0 :
  declare
    host_address  : addresses_list_access := null;
    host_socket   : socket_access := null;
    ok            : Boolean :=  False;

  begin

    init_addresses
      (ip_or_host =>  "", -- host addresses
      port        =>  "25000",
      ai_socktype =>  udp,
      ai_family   =>  any, -- choose ipv4 and ipv6
      addr        => host_address);

    if host_address.all'Length < 1 then

      Text_IO.Put_Line (" Failed to discover addresses in this host");
      Text_IO.Put_Line (" Quitting.");

      goto end_app_label1;
    end if;

    Text_IO.Put_Line (" Addresses Discovered in this host:");

    utils.show_address_and_port (host_address);

    if not init_socket (host_socket, host_address) then

      Text_IO.New_Line;

      Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

      goto end_app_label1;
    end if;
```

```
reuse_address (host_socket);

if not bind (host_socket) then

  Text_IO.New_Line;

  Text_IO.Put_Line (" Bind error: " & string_error);

  goto end_app_label1;
end if;

Text_IO.New_Line;

Text_IO.Put_Line (" Binded  at address " &
  get_addresses (get_addresses (host_socket)) &
  " and at port " & get_port (get_addresses (host_socket)));

b2 :
declare

  task type recv_send_task (
    host_address_family : Address_family;
    remote_actual_addr  : not null addresses_access;
    host_old_buff       : not null socket_buffer_access
  );

  task body recv_send_task -- See ARM-2012 7.6 (9.2/2)
  is
    use Task_Identification;

    this_task_id_str    : constant String   := Image (Current_Task);

    use  Ada.Strings.Unbounded;

    message : Unbounded_String := To_Unbounded_String ("");

  begin

    if is_null (remote_actual_addr) then

      Text_IO.Put_Line (this_task_id_str & " remote address not configured. quitting.");

      goto finish2_task_label;
    end if;

    if is_empty (host_old_buff) or else actual_data_size (host_old_buff) < 1 then

      Text_IO.Put_Line (this_task_id_str & " buffer is empty. quitting.");

      goto finish2_task_label;
    end if;

    Text_IO.Put_Line (" " & this_task_id_str & " remote host " &
      get_address_and_port (remote_actual_addr));

    Text_IO.Put_Line (" " & this_task_id_str & " received messages:");

    bt0 :
    declare
      remote_buff : socket_buffer_access := new socket_buffer;
```

```
begin

  clean (remote_buff);

  bt1 :
  begin

    String'Output (remote_buff, "Thank you for send ");

    loop1 :
    loop

      message := To_Unbounded_String (String'Input (host_old_buff));

      String'Output (remote_buff, To_String (message));

      Text_IO.Put_Line (" " & this_task_id_str & " message |" & To_String (message) & "|");

    end loop loop1;

  exception

    when buffer_insufficient_space_error =>
      Text_IO.Put_Line (" " & this_task_id_str & " all messages showed.");

  end bt1;

  bt2 :
  declare

    host_local_address  : addresses_list_access;

    socket_send   : socket_access;
    poll_send     : aliased polls.poll_type (1);

    poll_result   : int := 0;
    bytes_send    : ssize_t := 0;

  begin
    init_addresses
      (ip_or_host   => "", -- host addresses.
       port         =>  "0", -- ignored without 'bind'
       ai_socktype  =>  udp,
       ai_family    =>  host_address_family,
       addr         =>  host_local_address);

    if host_local_address.all'Length < 1 then

      Text_IO.Put_Line (" " & this_task_id_str & " Failed to get server host address.");

      Text_IO.Put_Line (" " & this_task_id_str & " Quitting this working task.");

      goto finish2_task_label;
    end if;

    if not init_socket (socket_send, host_local_address) then

      Text_IO.Put_Line (" " & this_task_id_str & " Failed to create temporary socket.");

      Text_IO.Put_Line (" " & this_task_id_str & " Quitting this working task.");
```

```
          goto finish2_task_label;
        end if;

        polls.add_events (poll_send'Access, socket_send, polls.send_ev);

        Text_IO.Put_Line (" " & this_task_id_str & " waiting to send data to remote host");

        poll_result  := polls.start_events_listen (poll_send'Access, 2500); -- block, 2.5 seconds timeout.

        if poll_result > 0 then

          bytes_send := sendto (socket_send, remote_actual_addr, remote_buff); -- block

          Text_IO.Put_Line (" " & this_task_id_str & " sended messages !");

        else

          Text_IO.Put_Line (" " & this_task_id_str & " failed in send data");

          Text_IO.Put_Line (" " & this_task_id_str & " to " &
            get_address_and_port (remote_actual_addr) & " host.");

          if poll_result = 0 then

            Text_IO.Put_Line (" " & this_task_id_str & " but it is a normal time_out");

          end if;

        end if;

        if initialized (socket_send) then

          close (socket_send);

        end if;
      end bt2;
    end bt0;

    <<finish2_task_label>>
  end recv_send_task;

  working_task  : access recv_send_task
    with Unreferenced;

  host_socket_family  : constant Address_family := get_address_family (get_addresses (host_socket));
  remote_address      : addresses_access        := null;
  host_buffer         : socket_buffer_access    :=  new socket_buffer;
  host_poll           : aliased polls.poll_type (1);

begin

  clean (host_buffer);

  Text_IO.New_Line;

  polls.add_events (host_poll'Access, host_socket, polls.receive_ev);

  ok := True;

  loop2 :
  loop
```

```
      if polls.start_events_listen (host_poll'Access, 15000) < 1 then -- block, 15 seconds timeout.

        if initialized (host_socket) then

          close (host_socket);

        end if;

        Text_IO.Put_Line (" Main event 15 seconds Time_out.");
        Text_IO.Put_Line (" Waiting 5 seconds to allow enough time for working tasks finish.");

        Text_IO.New_Line;

        delay 5.0;

        Text_IO.Put_Line (" Have a nice day and night. Bye!");
        Text_IO.New_Line;

        exit loop2;
      end if;


      if receive_from (host_socket, host_buffer, remote_address) > 0 then -- block

        -- For the curious: We believe the task(s) will not leak.
        -- Reason: ARM-2012 7.6 (9.2/2) :-)

        working_task  := new recv_send_task (host_socket_family, remote_address,
          get_buffer_init (host_buffer));

      end if;

      polls.clear_all_event_responses (host_poll'Access);

      clean (host_buffer);

    end loop loop2;
  end b2;

  <<end_app_label1>>

  if initialized (host_socket) then

    close (host_socket);

  end if;

  Text_IO.Put (" " & Command_Line.Command_Name);

  if ok then
    Text_IO.Put_Line (" successfully finalized.");
  else
    Text_IO.Put_Line (" failed.");
  end if;

  Text_IO.New_Line;

end b0;

stop_adare_net;
```

```
end server;
```

Hints for developers and users of others Network Ada Libraries

Anet

```
1.  There are just one universal addresses and universal adresses_list type(s).
    1.  Configure with:
        1. ip or host domain in string form.
        2. net port in string form, can be a service too. eg.: "ftp" etc
        3. udp or tcp
        4. any or v6 or v4

2.  There are just one universal socket type.
    1. initialize it with the addresses above.
3.  bind has just one parameter
4.  use raw data in fuctions send/sendto receive/receive
7.  obs.: you can listen() in IPv6 too.
5.  That's It.
```

Gnat-sockets

```
"equal Anet"
```

A minimum gnat project to work with.

```
Just include:  'with adare_net' in your gnat project.
```

Use a task pool

```
The examples showed a new task per connection but you can use a 'task pool'.
You can implement It yourself, but before that,
see a really good implementation in https://github.com/jrcarter/PragmARC
Thanks Jr.Carter !
```

Obs.: Use Ada Class Wide types (Tagged Types) and Stream Socket_Buffer,

with dispatching calls and automatic discover about data to see the real power of Adare_Net.