Get a Party! The Joy of Ada Language + Adare_Net Network Programming!

Adare_Net Version 2.17.5-dev.

Init Adare_Net!

- lib start.

Continue Preparing Party!

- Server part:
  1. Create a network address and port.
  2. Create a presence in network (socket).
     1. bind option.
     2. listen option.
        - backlog Option.
  3. I'm waiting you... connect to my socket!
     - I want you! I waited you forever! Thanks for connecting!
     - I want you! But I'm so Busy! Thanks for connecting or Bye!

- Client part:
  1. Create a network address and port.
  2. Create a presence in network (socket).
     1. bind option.
        - just ignore.
     2. listen option.
        - backlog Option.
          * just ignore both.
  3. I'm connecting to you, please accept me server!
     - I'm successfull connected to you! Thank's!
     - I'm not successfull connected:
       * timeout...
       * connection refused...

Party Start!

- send and receive:
  - client part:
    1. send to server.
    2. receive from server.
  - server part:
    1. receive from client.
    2. send to client.

Party End!

1. close sockets.
2. close addresses.
3. lib stop.

Appendices:

A1 *Examples:*

- Full Client and Server TCP/IP.
- Full Client and Server UDP/IP.
- How to Discover Network Addresses and Their Characteristics.

- A working Micro-Version of Embedded and Distributed Database.

  It shows the powerfull interaction of: sockets + socket_buffers (and his rewind operations) +
  Ada Streams + Ada Streams.Stream_IO (and his file(s) operations) +
  'normal' and 'class wide' types.

  It serve as a example of the real power of Adare_Net and Ada. Enjoy!! :-D

A2 ***Hints for Users of Others Network Ada Libs:***

- Adasockets.
- Anet.
- Gnat-sockets.

A3 ***Miscellaneous Tips:***

- Use Alire.
- Use a task pool.
- Use Ada Class Wide types (Tagged Types) and Stream Socket_Buffer to see the real power of Adare_Net.

Init Adare_Net!

- lib start:
  - `start_adare_net;  -- need be the first operation in the program, and before first use of Adare_Net.`

Continue Preparing Party!

- ***Server part:***
  1. Create a network address and port:
     - *many* => max 'quantity' choosed by user, between 1 and 65535, defaults to 9 addresses:

```
b_address_many :
declare
  --
  -- 'socket_addresses' and 'socket_addresses_access' types work as circular types and
  -- rewind is automatic after last address. For user convenience, exist rewind() procedures, too.
  --

  many_addresses :  socket_addresses_access := null;
  -- or many_addresses :  socket_addresses;

begin

  if not
    create_addresses (host_or_ip => "", -- Empty String "" implies choosing the ips of the
                                        -- current host or "::" or "0.0.0.0" .

      network_port_or_service => "25000", -- Ignored without 'bind' or connect(),
                                          -- Use "0" to choose one free random port automatically.
      Addr_family => any, -- ipv4 and ipv6.
      Addr_type => tcp,
      response => many_addresses,
      quantity => 9) -- quantity has a default value of 9 .
  then
    Text_IO.Put_Line ("Failed to discover host addresses.");
    Text_IO.New_Line;
    Text_IO.Put_Line ("last error message => " & string_error);

    -- exit or "B-Plan".
  end if;

end b_address_many;
```

     - *one* => get one address: from addresses (showed here, in three different ways) or from socket (to be showed):

```
b_address_one :
declare

  one_address :  socket_address_access := null;
  -- or one_address :  socket_address;

  ok  : Boolean :=  False;

begin
  -- remember, when ok is False, it flag or real error or last address getted.


  -- way1: get one or more addresses, one address at a time:

  ok := get_address (many_addresses, one_address);
  -- make some thing with 'one_address' var.
```

```
    -- ok := get_address (many_addresses, one_address);
    -- make some thing...with 'one_address' var.

    -- ok := get_address (many_addresses, one_address);
    -- make some thing with 'one_address' var.


    -- way2: loop it with get_adddress:

    rewind (many_addresses); -- go to first address, optional, just to start at begining address.

    loop2 :
    loop
      if get_address (many_addresses, one_address) then
        -- make some thing with 'one_address' var.


        goto end_loop2_label; -- 'continue' :-D
      end if;

      exit loop2;

      <<end_loop2_label>>
    end loop loop2;


    -- way3: loop it with get_adddress:

    rewind (many_addresses); -- go to first address, optional, just to start at begining address.

    loop3 :
    while get_address (many_addresses, one_address) loop

      -- make some thing with 'one_address' var.

    end loop loop3;

  end b_address_one;
```

2. Create a presence in network (socket):

```
b_server_socket :
declare
  server_socket : socket_access;
  -- or server_socket : socket;
begin

  -- way1: pick the first working address:

  if not
    create_socket (sock_address => many_addresses,
      response      => server_socket,
      bind_socket   => True,
      listen_socket => True,
      backlog       => 323); -- a true mini monster server queue.
  then
    Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

    -- exit or "B-Plan".
  end if;


  -- way2: pick the only address:
```

```
    if not
      create_socket (sock_address => one_address,
        response      => server_socket,
        bind_socket   => True,
        listen_socket => True,
        backlog       => 323); -- a true mini monster server queue.
    then
      Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

      -- exit or "B-Plan".
    end if;

  end b_server_socket;
```

3. I'm waiting you... connect to my socket!

   – I want you! I waited you forever! thanks for connecting!

```
    b_server_accept :
    declare
      msg  : stream_element_array_access := null; -- can be ignored when 'tcp'

      new_socket_accepted : socket_access := null;
      -- or new_socket_accepted : socket;
    begin

      if not
        wait_connection (sock =>  server_socket, -- block
          response  => new_socket_accepted,
          data_received =>  msg,
          miliseconds_start_timeout => 0) -- until forever
      then

        Text_IO.Put_Line (" Accept failed. Error => " & string_error);
        Text_IO.New_Line (2);

        -- exit or "B-Plan".
      end if;

      -- make some thing with 'new_socket_accepted' var

    end b_server_accept;
```

   – I want you! But I'm so Busy! Thanks for connecting or Bye!

```
    b_server_accept :
    declare
      msg  : stream_element_array_access := null; -- can be ignored when 'tcp'

      new_socket_accepted : socket_access := null;
      -- or new_socket_accepted : socket;
    begin

      if not
        wait_connection (sock =>  server_socket, -- block
          response  => new_socket_accepted,
          data_received =>  msg,
          miliseconds_start_timeout => 20000) -- until around 20 seconds.
      then
```

```
      Text_IO.Put_Line (" I waited for you for around 20 seconds. Bye.");
      Text_IO.New_Line (2);

      Text_IO.Put_Line (" last error message => " & string_error);
      Text_IO.New_Line (2);

      -- exit or "B-Plan".
    end if;

    -- make some thing with 'new_socket_accepted' var.

  end b_server_accept;
```

- ***Client part:***

1. Create a network address and port

   – *many* => max 'quantity' choosed by user, between 1 and 65535, defaults to 9 addresses:

```
b_address_many :
declare
  --
  -- 'socket_addresses' and 'socket_addresses_access' types work as circular types and
  -- rewind is automatic after last address. For user convenience, exist rewind() procedures, too.
  --

  many_addresses :  socket_addresses_access := null;
  -- or many_addresses :  socket_addresses;

begin

  if not
    create_addresses (host_or_ip => "::1", -- just example.

      network_port_or_service => "25000", -- Ignored without 'bind' or connect() .
                                          -- Use "0" to choose one free random port automatically.
      Addr_family => any, -- ipv4 and ipv6
      Addr_type => tcp,
      response => many_addresses,
      quantity => 3) -- quantity has a default value of 9
  then
    Text_IO.Put_Line ("Failed to discover host addresses.");
    Text_IO.New_Line;
    Text_IO.Put_Line ("last error message => " & string_error);
  end if;

end b_address_many;
```

   – *one* => get one address: from addresses (showed here, in three different ways) or from socket (to be showed):

```
b_address_one :
declare

  one_address :  socket_address_access := null;
  -- or one_address :  socket_address;

  ok  : Boolean :=  False;

begin
  -- remember, when ok is False, it flag or real error or last address getted.


  -- way1: get one or more addresses, one address at a time:
```

```
      ok := get_address (many_addresses, one_address);
      -- make some thing with 'one_address' var.

      -- ok := get_address (many_addresses, one_address);
      -- make some thing...with 'one_address' var.

      -- ok := get_address (many_addresses, one_address);
      -- make some thing with 'one_address' var.


      -- way2: loop it with get_adddress:

      rewind (many_addresses); -- go to first address, optional, just to start at begining address.

      loop2 :
      loop
        if get_address (many_addresses, one_address) then
          -- make some thing with 'one_address' var.


          goto end_loop2_label; -- 'continue' :-D
        end if;

        exit loop2;

        <<end_loop2_label>>
      end loop loop2;


      -- way3: loop it with get_adddress:

      rewind (many_addresses); -- go to first address, optional, just to start at begining address.

      loop3 :
      while get_address (many_addresses, one_address) loop

        -- make some thing with 'one_address' var

      end loop loop3;

    end b_address_one;
```

2. Create a presence in network (socket).

```
b_client_socket :
declare
  client_socket : socket_access;
  -- or client_socket : socket;
begin

  -- way1: pick the first working address:

  if not
    create_socket (sock_address => many_addresses,
      response      => client_socket,
      bind_socket   => False,
      listen_socket => False,
      backlog       => 1); -- ignored. the choosed '1' value is just to fill with something.
  then

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

```
    -- exit or "B-Plan".
  end if;


  -- way2: pick the only address:

  if not
    create_socket (sock_address => one_address,
      response      => client_socket,
      bind_socket   => False,
      listen_socket => False,
      backlog       => 1); -- ignored. the choosed '1' value is just to fill with something.
  then
    Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

    -- exit or "B-Plan".
  end if;

end b_client_socket;
```

3. I'm connecting to you server!
   - Please accept me!

```
b_client_connect :
begin

  if not connect (client_socket) 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.");

    --  obs.:
    --    timeout... => mostly time: there are a ip and configured port in choosed socket
    --      address server, but the server may either:
    --       (1) be very busy or (2) undergoing maintenance. Try again later.
    --
    --    connection refused... => mostly time: (1) app server not fully started or
    --      (2) app server fully finished or (3) firewall rules in client or server or both.

    -- exit or "B-Plan".
  end if;

  --  I'm successfull connected to you server! Thank's!

  -- make some use of client_socket

end b_client_connect;
```

Party Start!

- send and receive, client part:

```
b_client_send_and_receive :
declare
  client_data_to_send_backup  : socket_buffer_access := null;
  client_data_to_send         : socket_buffer_access := new socket_buffer;
  client_data_to_receive      : socket_buffer_access := new socket_buffer;
  sended_len    : int   := 0;
  received_len  : int   := 0;
```

```
  begin
    String'Output (client_data_to_send, "Hi! Server! how are you? :-D ");
    String'Output (client_data_to_send, "I'm sending to you a unsigned 16bit number ");
    Unsigned_16'Output (client_data_to_send, Unsigned_16 (9));

    client_data_to_send_backup := get_buffer (client_data_to_send);

    Text_IO.Put_Line ("Buffer to send size => " &
      Integer_64'(actual_data_size (client_data_to_send))'image);

    -- way1
    -- start        => wait forever or error
    -- after start  => wait forever or a low value or error

    if not
      send_buffer (sock   => client_socket,
        data_to_send  => client_data_to_send,
        send_count  => sended_len,
        miliseconds_start_timeout =>  0, -- wait forever for start sending or error
        miliseconds_next_timeouts =>  0) -- wait forever between sends or error
    then

      Text_IO.New_Line;
      Text_IO.Put_Line (" Error while trying send to remote host:");
      Text_IO.Put_Line (" sended length => " & sended_len'image);
      Text_IO.Put_Line (" last error => " & string_error);

      -- exit or "B-Plan".
    end if;


    -- restart buffer, just example :-D

    clear (client_data_to_send);

    client_data_to_send := get_buffer (client_data_to_send_backup);


    -- way2
    -- choose values for start and next

    if not
      send_buffer (sock   => client_socket,
        data_to_send  => client_data_to_send,
        send_count  => sended_len,
        miliseconds_start_timeout =>  4000, -- until maximum of 4 seconds or error
        miliseconds_next_timeouts =>  2000) -- until maximum of 2 seconds between sends or error
    then

      Text_IO.New_Line;
      Text_IO.Put_Line (" Error while trying send to remote host:");
      Text_IO.Put_Line (" sended length => " & sended_len'image);
      Text_IO.Put_Line (" last error => " & string_error);

      -- exit or "B-Plan".
    end if;

  end b_client_send_and_receive;
```

- receive and send, server part: