

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!

- Prologue
- 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. Prologue.
2. close sockets.
3. close address(es).
4. lib stop.

continues in next page

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:
 - Shows The Real Power of Adare_Net and Ada in Live Action
 - And demonstrates the powerful interaction between
 - * Sockets.
 - * Socket_buffers (and his rewind operations).
 - * Ada Streams.
 - * Ada Streams.Stream_IO and his file(s) operations.
 - * And many Ada types and constructs uses, in live.

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

- Prologue.
- Adasockets.
- Anet.
- Gnat-sockets.

A3 *Miscellaneous Tips:*

- Use Alire.
- Use a task pool.
- Use Class Wide types (Tagged Types) and Stream Socket_Buffer.

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_address:

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_address:

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);

    -- 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_address:

    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_address:

    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!

- ***Prologue:***

Send has two main variations:

`send_buffer()`

=> `data_to_send` field:
=> can be `socket_buffer_access` and `socket_buffer` .
=> if `send_buffer()` is successfull in sending all data
in `data_to_send` field, `data_to_send` buffer is emptied.

`send_stream()`

=> `data_to_send` field:
=> can be `stream_element_array_access` and `Stream_Element_Array` .
=> never change `data_to_send` field.

Receive has two main variations:

`receive_buffer()`

=> `data_to_receive` field:
=> can be `socket_buffer_access` and `socket_buffer` .
=> if `receive_buffer()` is successfull in getting all data
from `sock` field, `data_to_receive` buffer is appended with
the received data.

`receive_stream()`

=> `data_to_receive` field:
=> mode 'out'
=> can be `stream_element_array_access` only.
=> if `receive_stream()` is successfull in getting all data
from `sock` field, it create a fresh new data in
`data_to_receive` field, but not change the old values.

From Variations before:

`receive_{buffer,stream}()`

=> `received_address` field:
=> mode 'out'
=> can be `socket_address_access` and `socket_address` .
=> if `receive_{buffer,stream}()` are successfull, Its creates
a fresh new data in `received_address` field, but not change
the old values.

Obs.: I'll only show the `buffer` version for next client and server part,
but the `stream` versions are similar.

continue in next page

- *Send and Receive, Client part:*

```
b_client_send :
declare
  client_data_to_send_backup : socket_buffer_access := null;
  client_data_to_send       : socket_buffer_access := new socket_buffer;
  sended_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, -- block
      data_to_send => client_data_to_send,
      send_count => sended_len,
      milliseconds_start_timeout => 0, -- wait until forever for start sending or error
      milliseconds_next_timeouts => 0) -- wait until 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, -- block
      data_to_send => client_data_to_send,
      send_count => sended_len,
      milliseconds_start_timeout => 4000, -- until maximum of 4 seconds or error
      milliseconds_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 (" send length => " & send_len'image);
Text_IO.Put_Line (" last error => " & string_error);

-- exit or "B-Plan".
end if;
end b_client_send;
b_client_receive :
declare
client_data_to_receive : socket_buffer_access := new socket_buffer;
sender_address         : socket_address_access := null;
-- or sender_address   : socket_address;
received_len : int := 0;
begin

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

if not
receive_buffer (sock => client_socket, -- block
data_to_receive => client_data_to_receive,
received_address => sender_address,
receive_count   => received_len,
milliseconds_start_timeout => 0, -- until maximum of forever or error
milliseconds_next_timeouts => 0) -- until maximum of forever between receiving or error
then

Text_IO.New_Line;
Text_IO.Put_Line (" Error while trying receive from remote host:");
Text_IO.Put_Line (" received length => " & received_len'image);
Text_IO.Put_Line (" last error => " & string_error);

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

-- see client and server src examples to learn how show messages
-- received in client_data_to_receive :-)

-- Some Info :

Text_IO.Put_Line (" All messages received from " & get_address (sender_address) &
" and at port := " & get_address_port (sender_address) &
" and type => " & get_address_type (sender_address) &
" and family type => " & get_family_label (sender_address));

-- restart buffer, just example :-D
-- 'buffer' without restart will just append data received in Itself.

clear (client_data_to_receive);

-- way2
-- choose values for start and next

if not
receive_buffer (sock => client_socket, -- block
data_to_receive => client_data_to_receive,
received_address => sender_address,
receive_count   => received_len,
milliseconds_start_timeout => 7000, -- until maximum of 7 seconds or error
milliseconds_next_timeouts => 2000) -- until maximum of 2 seconds between receives or error
then

Text_IO.New_Line;

```

```

Text_IO.Put_Line (" Error while trying receive from remote host:");
Text_IO.Put_Line (" received length => " & received_len'image);
Text_IO.Put_Line (" last error => " & string_error);

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

-- see client and server src examples to learn how show messages
-- received in client_data_to_receive :-)

-- Some Info :

Text_IO.Put_Line (" All messages received from " & get_address (sender_address) &
  " and at port := " & get_address_port (sender_address) &
  " and type => " & get_address_type (sender_address) &
  " and family type => " & get_family_label (sender_address));

end b_client_receive;

```

- *Receive and Send, Server part:*

```

b_server_send :
declare
server_data_to_send_backup : socket_buffer_access := null;
server_data_to_send       : socket_buffer_access := new socket_buffer;
sended_len                : int                := 0;
begin
String'Output (server_data_to_send, "Hi! I'm fine! :-D ");
String'Output (server_data_to_send, "I'm sending to you a unsigned 16bit number, too.");
Unsigned_16'Output (server_data_to_send, Unsigned_16 (19));

server_data_to_send_backup := get_buffer (server_data_to_send);

Text_IO.Put_Line ("Buffer to send size => " &
  Integer_64'(actual_data_size (server_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 => new_socket_accepted, -- block
  data_to_send => server_data_to_send,
  send_count => sended_len,
  miliseconds_start_timeout => 0, -- wait until forever for start sending or error
  miliseconds_next_timeouts => 0) -- wait until 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 (server_data_to_send);

```

```

server_data_to_send := get_buffer (server_data_to_send_backup);

continue in next page

-- way2
-- choose values for start and next

if not
  send_buffer (sock => new_socket_accepted, -- block
    data_to_send => server_data_to_send,
    send_count => send_len,
    milliseconds_start_timeout => 4000, -- until maximum of 4 seconds or error
    milliseconds_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 (" send length => " & send_len'image);
  Text_IO.Put_Line (" last error => " & string_error);

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

b_server_receive :
declare
  server_data_to_receive : socket_buffer_access := new socket_buffer;
  sender_address         : socket_address_access := null;
  -- or sender_address   : socket_address;
  received_len : int := 0;
begin

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

  if not
    receive_buffer (sock => new_socket_accepted, -- block
      data_to_receive => server_data_to_receive,
      received_address => sender_address,
      receive_count   => received_len,
      milliseconds_start_timeout => 0, -- until maximum of forever or error
      milliseconds_next_timeouts => 0) -- until maximum of forever between receiving or error
  then

    Text_IO.New_Line;
    Text_IO.Put_Line (" Error while trying receive from remote host:");
    Text_IO.Put_Line (" received length => " & received_len'image);
    Text_IO.Put_Line (" last error => " & string_error);

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

  -- see client and server src examples to learn how show messages
  -- received in server_data_to_receive :-)

  -- Some Info :

  Text_IO.Put_Line (" All messages received from " & get_address (sender_address) &

```

```

" and at port := " & get_address_port (sender_address) &
" and type => " & get_address_type (sender_address) &
" and family type => " & get_family_label (sender_address));

-- restart buffer, just example :-D
-- 'buffer' without restart will just append data received in Itself.

clear (server_data_to_receive);

-- way2
-- choose values for start and next

if not
  receive_buffer (sock => new_socket_accepted, -- block
    data_to_receive => server_data_to_receive,
    received_address => sender_address,
    receive_count => received_len,
    milliseconds_start_timeout => 7000, -- until maximum of 7 seconds or error
    milliseconds_next_timeouts => 2000) -- until maximum of 2 seconds between receives or error
then

  Text_IO.New_Line;
  Text_IO.Put_Line (" Error while trying receive from remote host:");
  Text_IO.Put_Line (" received length => " & received_len'image);
  Text_IO.Put_Line (" last error => " & string_error);

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

-- see client and server src examples to learn how show messages
-- received in server_data_to_receive :-)

-- Some Info :

Text_IO.Put_Line (" All messages received from " & get_address (sender_address) &
  " and at port := " & get_address_port (sender_address) &
  " and type => " & get_address_type (sender_address) &
  " and family type => " & get_family_label (sender_address));

end b_server_receive;

```

Party End!

1. Prologue:

Sockets can only be closed by the actual user of It, in particular
if It was copied to use in another section of App, e.g.: to use it in other task.

Address(es) can be cleared/closed all times, but close Its at finishing stage of the
App is really optional; The close of Address(es) at running time is more to free memory
and can be done at the developer's discretion.

p.s.: Enjoy! :-D

2. Close sockets:

```

b_server_close_sockets :
begin
  close (socket_server);
  close (new_socket_accepted);

```

```

end b_server_close_sockets;

b_client_close_sockets :
begin
  close (client_socket);
end b_client_close_sockets;

```

3. *Close address(es):*

```

b_server_and_client_close_addrs :
begin
  close (many_addresses);
  close (one_address);

end b_server_and_client_close_addrs

```

4. *Lib stop:*

```

stop_adare_net; -- need be the last operation in the program, and after the last use of Adare_Net.

```

*Appendices
in next
page.*

Appendices

A1 Examples:

- Full Client and Server TCP/IP.

- Server TCP/IP

```
-- Besides this is a multitask and reasonable complete example with Adare_net, you can do more, as:
--
-- (1) More que one listen sockets,
-- (2) Simultaneous listen event_types,
-- (3) Use of others types beyond String:
-- (3.1) From built-in types and records to
-- (3.2) Wide class(es) and tagged types
-- (3.3) And with a more fine treatment, all records, tagged types included, can be endian proof.
-- (4) 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:
--
-- tcp_server_new is an Adare_net example and work in pair with one or more tcp_client_new clients.
-- the working address can be ipv6 or ipv4. Automatically the first working address will be picked.
-- mostly common choosen address in server part is "0.0.0.0" or ":::" then use localhost or
-- other configured ip address. eg:
-- 127.0.0.1 or ::1 or ? :-) to connect.

with adare_net.base; use adare_net.base;
with adare_net_init; use adare_net_init;
with adare_net_exceptions; use adare_net_exceptions;

with Ada.Text_IO; use Ada;
with Ada.Command_Line;
with Ada.Task_Identification;
with Ada.Strings.Unbounded;
with Interfaces.C; use Interfaces.C;

use Ada.Task_Identification;

procedure tcp_server_new
is
  pragma Unsuppress (All_Checks); -- just to testing, optional in production code.
begin

  start_adare_net;

  b0 :
  declare
    host_socket_addresses : socket_addresses_access;
```



```

tmp_socket_address      : socket_address_access;
host_socket             : socket_access;
begin

if not create_addresses
  (host_or_ip => "",
   network_port_or_service => "25000",
   Addr_family => any,
   Addr_type => tcp,
   response => host_socket_addresses)
then
  Text_IO.Put_Line ("Failed to discover host addresses.");
  Text_IO.New_Line;
  Text_IO.Put_Line ("last error message => " & string_error);

  goto end_app_label1;
end if;

Text_IO.New_Line;

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

while get_address (host_socket_addresses, tmp_socket_address) loop

  Text_IO.Put_Line ("type => " & get_address_type (tmp_socket_address) &
    " family_type => " & get_family_label (tmp_socket_address) &
    " address => " & get_address (tmp_socket_address) &
    " and port => " & get_address_port (tmp_socket_address));

  Text_IO.New_Line;
end loop;

if not create_socket (host_socket_addresses, host_socket, True, True, 35) then

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

  goto end_app_label1;
end if;

get_address (host_socket, tmp_socket_address);

Text_IO.New_Line;

Text_IO.Put_Line (" choosed: host address => " & get_address (tmp_socket_address) &
  " port => " & get_address_port (tmp_socket_address) &
  " type => " & get_address_type (tmp_socket_address) &
  " family_type => " & get_family_label (tmp_socket_address));

b1 :
declare

task type recv_send_task (connected_sock : not null socket_access)
  with Dynamic_Predicate => is_initialized (connected_sock)
  and then is_connected (connected_sock);

task body recv_send_task
is
  task_sock      : constant socket_access      := connected_sock;
  remote_address : constant socket_address_access := get_address (task_sock);

  this_task_id_str : constant String := Image (Current_Task);

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

```

```

tmp_tmp_socket_address : socket_address_access := null;

size_tmp : int := 0;

use Ada.Strings.Unbounded;

message : Unbounded_String := To_Unbounded_String ("");
begin

clear (recv_send_buffer);  -- optional, reset all data in buffer
clear (recv_send_buffer2); -- optional, reset all data in buffer

Text_IO.New_Line (2);

Text_IO.Put_Line (" " & this_task_id_str & " remote host connected from [" &
  get_address (remote_address) & "]: " & get_address_port (remote_address) &
  " type => " & get_address_type (tmp_socket_address) &
  " family_type => " & get_family_label (tmp_socket_address));

Text_IO.Put_Line (" " & this_task_id_str &
  " will wait until 2 seconds to start receive data.");
Text_IO.Put_Line (" " & this_task_id_str &
  " will wait until 0.5 seconds between continuous receive.");

if not receive_buffer (sock => task_sock,
  data_to_receive => recv_send_buffer,
  received_address => tmp_tmp_socket_address,
  receive_count => size_tmp,
  milliseconds_start_timeout => 2000,
  milliseconds_next_timeouts => 500) or else size_tmp < 1
then
  Text_IO.Put_Line (" " & this_task_id_str & " An error occurred " &
    "while receiving or the length of message received is zero.");
  Text_IO.Put_Line (" " & this_task_id_str & " Nothing to do.");
  Text_IO.Put_Line (" " & this_task_id_str & " Last error message => " & string_error);
  Text_IO.Put_Line (" " & this_task_id_str & " Finishing...");

  goto finish1_task_label;
end if;

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

Text_IO.Put_Line (" " & this_task_id_str & " message length " &
  size_tmp'Image & " bytes.");

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;

    Text_IO.Put_Line (" " & this_task_id_str & " waiting until 2 " &
        "seconds to start send data to remote host");
    Text_IO.Put_Line (" " & this_task_id_str & " will wait until 0.5 " &
        "seconds between continuous send.");

    if not send_buffer (sock => task_sock,
        data_to_send => recv_send_buffer2,
        send_count => size_tmp,
        milliseconds_start_timeout => 2000,
        milliseconds_next_timeouts => 500) or else size_tmp < 1
    then
        Text_IO.Put_Line (" " & this_task_id_str & " An error occurred while " &
            "sending data to remote host.");
        Text_IO.Put_Line (" " & this_task_id_str & " Nothing to do.");
        Text_IO.Put_Line (" " & this_task_id_str & " Last error message => " & string_error);
        Text_IO.Put_Line (" " & this_task_id_str & " Finishing...");

        goto finish1_task_label;
    end if;

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

    <<finish1_task_label>>

    if is_initialized (task_sock) then

        close (task_sock);
    end if;

end recv_send_task;

type recv_send_access is access all recv_send_task;

working_task : recv_send_access
    with Unreferenced;

msg_seaa : stream_element_array_access := null;

tmp_received_socket_access : socket_access := null;

begin

    Text_IO.New_Line;

    Text_IO.Put_Line (" Start Accepting connect in Main Server.");
    Text_IO.Put_Line (" 20 seconds max timeout between clients.");
    Text_IO.New_Line (2);

    loop2 :
    loop
        if not wait_connection (sock => host_socket,
            response => tmp_received_socket_access,
            data_received => msg_seaa,
            milliseconds_start_timeout => 20000)
        then
            close (host_socket); -- to disable 'listen' too.

            Text_IO.New_Line (2);

            Text_IO.Put_Line (" Main event 20 seconds Time_out.");

```

```

    Text_IO.Put_Line (" Waiting 5 seconds to allow enough time " &
        "for working tasks finish.");

    Text_IO.New_Line (2);

    delay 5.0;

    Text_IO.Put_Line (" Have a nice day and night. Bye!");
    Text_IO.New_Line (2);

    exit loop2;
end if;

-- 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 (tmp_received_socket_access);

Text_IO.New_Line (2);

Text_IO.Put_Line (" restarting 20 seconds timeout.");

    end loop loop2;
end b1;

<<end_app_label1>>

if is_initialized (host_socket) then

    close (host_socket);
end if;

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

Text_IO.New_Line;

end b0;

stop_adare_net;

end tcp_server_new;

```

– Client TCP/IP

```

-- This is an over simplified, but complete enough, 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.base; use adare_net.base;
with adare_net_init; use adare_net_init;
with adare_net_exceptions; use adare_net_exceptions;

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

procedure tcp_client_new

```

```

is
  pragma Unsuppress (All_Checks); -- just to testing, optional in production code.
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 read and written offline ");

    Text_IO.New_Line;

    Set_Exit_Status (Failure);

    stop_adare_net;

    return;
  end if;

  Text_IO.New_Line;

b0 :
declare
  buffer : constant socket_buffer_access := new socket_buffer;
  ok      : Boolean := False;
begin
  clear (buffer); -- optional

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

b1 :
declare
  remote_addr : socket_addresses_access;
  choosed_addr : socket_address_access;
  rcv_addr : socket_address_access;
  host_sock : socket_access;

  bytes_tmp : int := 0;

begin
  if not create_addresses
    (host_or_ip => Argument (1), network_port_or_service => Argument (2),
    Addr_family => any, Addr_type => tcp, response => remote_addr)
  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:");

```

```

while get_address (remote_addr, choosed_addr) loop
    Text_IO.Put_Line ("type => " & get_address_type (choosed_addr) &
        " , family_type => " & get_family_label (choosed_addr) &
        " , address => " & get_address (choosed_addr) &
        " , and port => " & get_address_port (choosed_addr));

    Text_IO.New_Line;
end loop;

if not create_socket (remote_addr, host_sock) 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;

get_address (host_sock, choosed_addr);

Text_IO.Put_Line ("type => " & get_address_type (choosed_addr) &
    " , family_type => " & get_family_label (choosed_addr) &
    " Connected at address := " & get_address (choosed_addr) &
    " and at port := " & get_address_port (choosed_addr));

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting until 2 seconds to start sending messages. ");
Text_IO.Put_Line (" with until 0,5 seconds between sending remaining messages. ");

Text_IO.Put_Line (" buffer size " & Integer_64'(actual_data_size (buffer))'Image);

if not send_buffer (sock => host_sock,
    data_to_send => buffer,
    send_count => bytes_tmp,
    miliseconds_start_timeout => 2000,
    miliseconds_next_timeouts => 500) or else bytes_tmp < 1
then
    Text_IO.Put_Line (" An error occurred while sending data to remote server.");
    Text_IO.Put_Line (" Nothing to do.");
    Text_IO.Put_Line (" Last error message => " & string_error);
    Text_IO.Put_Line (" Finishing.");

    goto end_app_label1;
end if;

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

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting until 5 seconds to receive message(s). ");
Text_IO.Put_Line (" with until 0,5 seconds between receive remaining messages. ");

```

```

if not receive_buffer (sock => host_sock,
    data_to_receive => buffer,
    received_address => rcv_addr,
    receive_count => bytes_tmp,
    miliseconds_start_timeout => 5000,
    miliseconds_next_timeouts => 500) or else bytes_tmp < 1
then
    Text_IO.Put_Line (" An error occurred while receiving or the length of " &
        "message received is zero.");
    Text_IO.Put_Line (" Nothing to do.");
    Text_IO.Put_Line (" Last error message => " & string_error);
    Text_IO.Put_Line (" Finishing.");

    goto end_app_label1;
end if;

Text_IO.Put_Line (" Received message(s) from " & get_address (choosed_addr) &
    " and at port := " & get_address_port (choosed_addr) &
    " , type => " & get_address_type (choosed_addr) &
    " , family type => " & get_family_label (choosed_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 (choosed_addr) &
            " and at port := " & get_address_port (choosed_addr) &
            " and type => " & get_address_type (choosed_addr) &
            " and family type => " & get_family_label (choosed_addr) & " showed.");
end b2;

ok := True;

<<end_app_label1>>

if is_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 tcp_client_new;

```

- Full Client and Server UDP/IP.

- *Server UDP/IP*

```

-- Besides this is a multitask and reasonable complete example with Adare_net, you can do more, as:
--
-- (1) More que one listen sockets,
-- (2) Simultaneous listen event_types,
-- (3) Use of others types beyond String:
-- (3.1) From built-in types and records to
-- (3.2) Wide class(es) and tagged types
-- (3.3) And with a more fine treatment, all records, tagged types included, can be endian proof.
-- (4) 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_new is an Adare_net example and work in pair with one or more udp_client_new clients.
-- the working address can be ipv6 or ipv4. Automatically the first working address will be picked.
-- mostly common choosen address in server part is "0.0.0.0" or ":::" then use localhost or
-- other configured ip address. e.g.: 127.0.0.1 or ::1 or ? :-) to connect.

with adare_net.base; use adare_net.base;
with adare_net_init; use adare_net_init;
with adare_net_exceptions; use adare_net_exceptions;

with Ada.Text_IO; use Ada;
with Ada.Command_Line;
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Strings.Unbounded;
with Ada.Streams; use Ada.Streams;
with Interfaces.C; use Interfaces.C;

procedure udp_server_new
is
    pragma Unsuppress (All_Checks); -- just to testing, optional in production code.
begin

    start_adare_net;

    b0 :
    declare
        host_socket_addresses : socket_addresses_access;
        tmp_socket_address    : socket_address_access;
        host_socket           : socket_access;
    begin

        if not create_addresses
            (host_or_ip => "",
             network_port_or_service => "25000",

```



```

    Addr_family => any,
    Addr_type => udp,
    response => host_socket_addresses)
then
    Text_IO.Put_Line ("Failed to discover host addresses.");
    Text_IO.New_Line;
    Text_IO.Put_Line ("Last error message => " & string_error);

    goto end_app_label1;
end if;

Text_IO.New_Line;

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

while get_address (host_socket_addresses, tmp_socket_address) loop

    Text_IO.Put_Line ("type => " & get_address_type (tmp_socket_address) &
        " family type => " & get_family_label (tmp_socket_address) &
        " address => " & get_address (tmp_socket_address) &
        " port => " & get_address_port (tmp_socket_address));

    Text_IO.New_Line;
end loop;

if not create_socket (host_socket_addresses, host_socket, True, True, 35) then
    Text_IO.Put_Line (" Failed to initialize socket: " & string_error);

    goto end_app_label1;
end if;

get_address (host_socket, tmp_socket_address);

Text_IO.New_Line;

Text_IO.Put_Line (" Chooosed: type => " & get_address_type (tmp_socket_address) &
    " family type => " & get_family_label (tmp_socket_address) &
    " address => " & get_address (tmp_socket_address) &
    " and port => " & get_address_port (tmp_socket_address));

b1 :
declare

    task type recv_send_task (connected_sock : not null socket_access;
        pre_message : stream_element_array_access)
        with Dynamic_Predicate => is_initialized (connected_sock)
        and then is_connected (connected_sock);

    task body recv_send_task
    is

        task_socket      : constant socket_access := connected_sock;
        remote_address   : constant socket_address := get_address (task_socket);

        this_task_id_str : constant String := Image (Current_Task);

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

        tmp_tmp_socket_address : socket_address_access := null;

        size_tmp : int := 0;

        use Ada.Strings.Unbounded;

```

```

message : Unbounded_String := To_Unbounded_String ("");
begin
  clear (recv_send_buffer); -- optional, reset all data in buffer
  clear (recv_send_buffer2); -- optional, reset all data in buffer

  if pre_message /= null then
    Stream_Element_Array'Write (recv_send_buffer, pre_message.all);
  end if;

  Text_IO.New_Line (2);

  Text_IO.Put_Line (" " & this_task_id_str &
    " remote host connected from [" & get_address (remote_address) & "]" &
    get_address_port (remote_address) &
    " and type => " & get_address_type (remote_address) &
    " , and family type => " & get_family_label (tmp_socket_address));

  Text_IO.Put_Line (" " & this_task_id_str &
    " will wait until 2 seconds to start receive data.");
  Text_IO.Put_Line (" " & this_task_id_str &
    " will wait until 0.5 seconds between continuous receive.");

  if not receive_buffer (sock => task_socket,
    data_to_receive => recv_send_buffer,
    received_address => tmp_tmp_socket_address,
    receive_count => size_tmp,
    milliseconds_start_timeout => 2000,
    milliseconds_next_timeouts => 500) or else size_tmp < 1
  then
    if pre_message = null or else pre_message.all'Length < 1 then

      Text_IO.Put_Line (" " & this_task_id_str &
        " An error occurred while receiving or the length of message received is zero.");
      Text_IO.Put_Line (" " & this_task_id_str & " Nothing to do.");
      Text_IO.Put_Line (" " & this_task_id_str & " Last error message => " & string_error);
      Text_IO.Put_Line (" " & this_task_id_str & " Finishing...");

      goto finish1_task_label;
    end if;
  end if;

  size_tmp := size_tmp + (if pre_message = null then 0 else pre_message.all'Length);

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

  Text_IO.Put_Line (" " & this_task_id_str & " message length " & size_tmp'Image & " bytes.");

  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;

Text_IO.Put_Line (" " & this_task_id_str &
    " waiting until 2 seconds to start send data to remote host");
Text_IO.Put_Line (" " & this_task_id_str &
    " will wait until 0.5 seconds between continuous send.");

if not send_buffer (sock => task_socket,
    data_to_send => recv_send_buffer2,
    send_count => size_tmp,
    milliseconds_start_timeout => 2000,
    milliseconds_next_timeouts => 500) or else size_tmp < 1
then

    Text_IO.Put_Line (" " & this_task_id_str &
        " An error occurred while sending data to remote host.");
    Text_IO.Put_Line (" " & this_task_id_str & " Nothing to do.");
    Text_IO.Put_Line (" " & this_task_id_str & " Last error message => " & string_error);
    Text_IO.Put_Line (" " & this_task_id_str & " Finishing...");

    goto finish1_task_label;
end if;

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

<<finish1_task_label>>

if is_initialized (task_socket) then

    close (task_socket);
end if;

end recv_send_task;

type recv_send_access is access all recv_send_task;

working_task : recv_send_access
    with Unreferenced;

msg_seaa : stream_element_array_access := null;

tmp_received_socket : socket_access := null;

begin

    Text_IO.New_Line;

    Text_IO.Put_Line (" Start Accepting connect in Main Server.");
    Text_IO.Put_Line (" Until 20 seconds max timeout between clients.");
    Text_IO.New_Line (2);

loop2 :
loop
    if not wait_connection (sock => host_socket, response => tmp_received_socket,
        data_received => msg_seaa, milliseconds_start_timeout => 20000)
    then
        close (host_socket); -- to disable 'listen' too.

        Text_IO.New_Line (2);

```

```

Text_IO.Put_Line (" Main event 20 seconds Time_out.");
Text_IO.Put_Line (" Waiting 5 seconds to allow enough time for working tasks finish.");

Text_IO.New_Line (2);

delay 5.0;

Text_IO.Put_Line (" Have a nice day and night. Bye!");
Text_IO.New_Line (2);

    exit loop2;
end if;

-- 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 (tmp_received_socket, msg_seaa);

Text_IO.New_Line (2);

Text_IO.Put_Line (" restarting 20 seconds timeout.");

    end loop loop2;
end b1;

<<end_app_label1>>

if is_initialized (host_socket) then

    close (host_socket);
end if;

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

Text_IO.New_Line;

end b0;

stop_adare_net;

end udp_server_new;

```

– Client UDP/IP

```

-- 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.base; use adare_net.base;
with adare_net_init; use adare_net_init;
with adare_net_exceptions; use adare_net_exceptions;

with Interfaces.C; use Interfaces.C;

```

```

procedure udp_client_new
is
  pragma Unsuppress (All_Checks); -- just to testing, optional in production code.
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 read and written offline ");

    Text_IO.New_Line;

    Set_Exit_Status (Failure);

    stop_adare_net;

    return;
  end if;

  Text_IO.New_Line;

  b0 :
  declare
    buffer : constant socket_buffer_access := new socket_buffer;
    ok      : Boolean := False;
  begin
    clear (buffer); -- optional

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

    b1 :
    declare
      remote_addr   : socket_addresses_access := null;
      choosed_addr  : socket_address_access  := null;
      rcv_addr      : socket_address_access  := null;
      host_sock     : socket_access          := null;

      bytes_tmp     : int := 0;
    begin
      if not create_addresses
        (host_or_ip   => Argument (1), network_port_or_service => Argument (2),
         Addr_family  => any, Addr_type => udp, response => remote_addr)
      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:");
    end
  end

```

```

while get_address (remote_addr, choosed_addr) loop

    Text_IO.Put_Line ("type => " & get_address_type (choosed_addr) &
        " family type => " & get_family_label (choosed_addr) &
        " address => " & get_address (choosed_addr) &
        " and port => " & get_address_port (choosed_addr));

    Text_IO.New_Line;
end loop;

if not create_socket (remote_addr, host_sock) then

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

    goto end_app_label1;
end if;

-- connect() is optional when the connection is in udp

get_address (host_sock, choosed_addr);

Text_IO.Put_Line (" Connected at: address := " & get_address (choosed_addr) &
    " port := " & get_address_port (choosed_addr) &
    " type := " & get_address_type (choosed_addr) &
    " family type => " & get_family_label (choosed_addr));

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting until 2 seconds to send messages");

if not send_buffer (sock => host_sock,
    data_to_send => buffer,
    send_count => bytes_tmp,
    miliseconds_start_timeout => 2000,
    miliseconds_next_timeouts => 500) or else bytes_tmp < 1
then

    Text_IO.Put_Line (" An error occurred while sending data to remote server.");
    Text_IO.Put_Line (" Nothing to do.");
    Text_IO.Put_Line (" Last error message => " & string_error);
    Text_IO.Put_Line (" Finishing.");

    goto end_app_label1;
end if;

Text_IO.Put_Line (" Successfull sended " & bytes_tmp'Image & " bytes.");
Text_IO.Put_Line (" But remember, this client app_is_a_udp_client ^^ ");

Text_IO.New_Line;

Text_IO.Put_Line (" Waiting until 5 seconds to receive message(s). ");
Text_IO.Put_Line (" with until 0,5 seconds between receiving remaining messages. ");

if not receive_buffer (sock => host_sock,
    data_to_receive => buffer,
    received_address => rcv_addr,
    receive_count => bytes_tmp,
    miliseconds_start_timeout => 5000,

```

```

        milliseconds_next_timeouts => 500) or else bytes_tmp < 1
then
    Text_IO.Put_Line (" An error occurred while receiving or " &
        "the length of message received is zero.");
    Text_IO.Put_Line (" Nothing to do.");
    Text_IO.Put_Line (" Last error message => " & string_error);
    Text_IO.Put_Line (" Finishing.");

    goto end_app_label1;
end if;

Text_IO.Put_Line (" Received message(s) from: type => " & get_address_type (rcv_addr) &
    " family type => " & get_family_label (rcv_addr) &
    " address => " & get_address (rcv_addr) &
    " and port => " & get_address_port (rcv_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: type => " & get_address_type (rcv_addr) &
            " family type => " & get_family_label (rcv_addr) &
            " address => " & get_address (rcv_addr) &
            " and port => " & get_address_port (rcv_addr) & " showed.");

end b2;

ok := True;

<<end_app_label1>>

if is_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 udp_client_new;

```

- How to Discover Network Addresses and Their Characteristics.

```

--
with adare_net_init; use adare_net_init;
with adare_net.base; use adare_net.base;

with Ada.Text_IO; use Ada;

procedure test_get_multi_addresses is
begin

    start_adare_net;

    b0 :
    declare
        mi_addresses      : socket_addresses;
        -- or mi_addresses : socket_addresses_access := null;
        mi_address         : socket_address;
        -- or mi_address   : socket_address_access  := null;
    begin

        Text_IO.New_Line;

        Text_IO.Put_Line (" addresses from google.com ");

        if create_addresses
            (host_or_ip => "google.com",
             network_port_or_service => "0",
             Addr_family => any,
             Addr_type => tcp,
             response => mi_addresses)
        then

            Text_IO.New_Line;

            while get_address (mi_addresses, mi_address) loop

                Text_IO.Put_Line ("type => " & get_address_type (mi_address) &
                                   " , family_type => " & get_family_label (mi_address) &
                                   " , address => " & get_address (mi_address) &
                                   " , and port => " & get_address_port (mi_address));

                Text_IO.New_Line;
            end loop;

        end if;

        Text_IO.Put_Line ("Completed.");
    end b0;

    stop_adare_net;

end test_get_multi_addresses;

```

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

TBD

Continues Next Page

A2 Hints for Users of Others Network Ada Libs:

- *Prologue*

AdaSockets:

I'm GrateFull to Samuel Tardieu and AdaSockets, by be the first one
free network lib, yet in the init of Ada95, long before ipv6 existed,
with mswindows support and long before gnat-sockets existed.

He was also always helpful when I contacted him to fix bugs and add
new features to AdaSockets.

As the ages passed I needed IPv6 support and discovered that it was not possible.
An extreme and strenuous rework is necessary to use IPv6 in adasockets,
but this would lead to creating a 100% new Adasockets.
As the first network lib available in Ada, Adasockets could, who knows?, already
running in nuclear plants, even military embedded systems
and (mybe?) cannot be updated.

And the AdaSockets Author don't need ipv6... :-D Bye! ^^

Annet and gnat-sockets prologue TBD

- *Adasockets.*

TBD

- *Anet.*

TBD

- *Gnat-sockets.*

TBD