Adare_net Manual

version 0.0.109

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
- send_to
- 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)

```
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);
    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 don´t are modified.

  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).
```

receive_from

- send_to
- plain raw data, vulgo stream_element_array
- buffered data, vulgo socket_buffer
- plain raw data ou buffered data ?