```
type
    TCP_Buffer : array [1..522] of byte;
end_type

var_output
    Init_Ok:        BOOL;           (* Flag indicating that initialization is finished. Data
    can be sent and received. *)
    Received_Data:  BOOL;           (* Flag indicating that data from Gateway has been
    received (not necessarily payload data) *)
    Send_Success:   BOOL;           (* Flag indicating that data has been successfully send
    to Gateway *)
end_var

var_in_out
    Buffer:         TCP_Buffer;     (* Byte array for data exchange with Gateway *)
end_var

var_input
    GW_Address:     STRING;         (* e.g. '/ACTIVE /PORT=49000 /IP=192.168.0.100' (/ACTIVE
    for IP connection) *)
    Send_Data:      BOOL;           (* Flag indicating that payload data is to be sent via
    Gateway *)
end_var

var
    ID:             INT;            (* ID of the TCP/IP connection to Gateway *)
    Init_State:     INT;            (* Status of the initialization *)
    Valid:          BOOL;           (* Flag indicating the validity of the TCP/IP connection
    *)
    Sent:           BOOL;           (* Flag indicating that data has been sent *)
    Received:       BOOL;           (* Flag indicating that data has been received *)

    myIP_CONNECT:   IP_CONNECT;     (* Library FB for opening a TCP/IP connection *)
    myIP_USEND:     IP_USEND;       (* Library FB for transmitting data via TCP/IP *)
    myIP_URCV:      IP_URCV;        (* Library FB for receiving data via TCP/IP *)
    myR_TRIG:       R_TRIG;         (* Library FB for detecting rising edges of a signal *)

    Error:          BOOL;           (* Flag storing an error returned by an FB, not
    evaluated *)
    Status:         INT;            (* Integer storing a status returned by an FB, not
    evaluated *)
end_var

(* Open socket *)

IP_CONNECT_1(EN_C:= true, PARTNER:= GW_Address);

ID      :=IP_CONNECT_1.ID;

R_TRIG_1 (CLK := IP_CONNECT_1.VALID);
Valid := R_TRIG_1.Q;

if not IP_CONNECT_1.VALID then
   init_State := 0;
end_if;
```

```
if Valid then
    Init_State := 1;
end_if;


(* Send OP_HELLO *)
if Init_State = 1 then
    Send_Data := true;

    Buffer[1]   :=  word_to_byte (WORD#10 and word#255);  (* IP packet length, lower Byte*)
    Buffer[2]   :=  word_to_byte (WORD#10 /   word#256);  (* IP packet length, upper Byte*)

    Buffer[3]   :=  BYTE#83;    (* 'S' *)
    Buffer[4]   :=  BYTE#1;     (* OpCode OP_HELLO *)

    Buffer[5]   :=  word_to_byte (WORD#6 and word#255);   (* OpCode packet length, lower
    Byte *)
    Buffer[6]   :=  word_to_byte (WORD#6 /   word#256);   (* OpCode packet length, upper
    Byte *)

    Buffer[7]   :=  BYTE#83;    (* 'S' *)
    Buffer[8]   :=  BYTE#72;    (* 'H' *)
    Buffer[9]   :=  BYTE#84;    (* 'T' *)
    Buffer[10]  :=  BYTE#67;    (* 'C' *)
    Buffer[11]  :=  BYTE#80;    (* 'P' *)
    Buffer[12]  :=  BYTE#1;     (* OpCode protocol version *)

end_if;

(* Send OP_READ_CONTINUOUS *)
if Init_State = 3 then
    Send_Data := true;

    Buffer[1]   :=  word_to_byte (WORD#5 and word#255);    (* IP packet length, lower Byte*)
    Buffer[2]   :=  word_to_byte (WORD#5 /   word#256);  (* IP packet length, upper Byte*)

    Buffer[3]   :=  BYTE#83;                              (* 'S' *)
    Buffer[4]   :=  BYTE#7;                               (* OpCode OP_READ_CONTINUOUS *)

    Buffer[5]   :=  word_to_byte (WORD#1 and word#255);   (* OpCode packet length, lower
    Byte *)
    Buffer[6]   :=  word_to_byte (WORD#1 /   word#256);   (* OpCode packet length, upper
    Byte *)

    Buffer[7]   :=  BYTE#1;     (* Enable  *)

end_if;


(* OP_SET_NODE_ID_REQUEST *)
if Init_State = 5 then
    Send_Data := false;
    Init_State := 7;
(*  Buffer[1]       :=  WORD#6;     (* IP packet length *)
(*  Buffer[2].B0    :=  BYTE#83;    (* 'S' *)
(*  Buffer[2].B1    :=  BYTE#10;    (* OpCode packet identifier *)
(*  Buffer[3]       :=  WORD#2;     (* OpCode packet length *)
```

```
(*  Buffer[4].B1    :=  BYTE#255;   (* NID of Gateway *)
(*  Buffer[4].B1    :=  BYTE#255;   (* NID of Gateway *)

end_if;

if Init_State >= 7 then
    Init_Ok := true;
else
    Init_Ok := false;
end_if;

if Init_State > 0 or Send_Data then

    IP_USEND_1(REQ:= Send_Data and not Sent,
               ID:= ID,
             R_ID:= 'Arcnet',
             SD_1:= Buffer);

    Sent    := IP_USEND_1.DONE;
    Error   := IP_USEND_1.ERROR;
    Status  := IP_USEND_1.STATUS;
    Buffer  := IP_USEND_1.SD_1;

    if Sent then
        if not Init_ok then     (* Still initializing *)
            Init_State    := Init_State + 2;
            Send_Success := false;
        else
            Send_Success := true;
        end_if;
    end_if;
end_if;


(* Receiving from ARCNET via Ethernet. The 1st word in the buffer contains the length.
  It has not been directly transmitted in the ARCNET data package *)

Buffer[1] := byte#0;
Buffer[2] := byte#0;

IP_URCV_1(EN_R:= true,
            ID:= ID,
          R_ID:= 'Arcnet',
          RD_1:= Buffer);

    Received    :=IP_URCV_1.NDR;
    Error       :=IP_URCV_1.ERROR;
    Status      :=IP_URCV_1.STATUS;
    Buffer      :=IP_URCV_1.RD_1;

if Received and
  Buffer[1] >= byte#7 then
    Received_Data := true;
else
    Received_Data := false;
end_if;
```