Estelle Specification of MIL-STD-188-220B

Datalink Layer Class A (Type 1) Operation

Protocol Engineering Laboratory
Department of Computer and Information Science
University of Delaware

Paul D. Amer
Greg Burch
Mariusz Fecko
Adarsh Sethi
M. Umit Uyar
Dong Zhu

Date: 4/2/96
Revised: 6/11/96
Revised: May 97
Revised: Aug 97
Revised: Dec 97
Update DL.Unitdata/Status.primitives
Updated DL.Unitdata/Status.primitives

Definitions: (for convenience of reference and consistency in the Estelle code.)

request: This is an internal item used by the SAP component only. The "queue" is, as the name implies, a sequence of outstanding requests. The request is a representation of a Protocol Request (e.g., DL_Unitdata_req) which the SAP component knows how to handle. Included in the request is the request_id which is the id assigned to the Protocol Request by the entity that initiated it (e.g., OPERATOR, NL).

message: An internal exchange of information between the SAP component and Station component of the Estelle architecture. These "messages" are used only locally, and consist of a DL_PDU, unique message_id, and P-bit (from SAP to Station only).

transmission: This signifies the bit pattern that the datalink layer sends to physical layer; as such, a transmission consists of a transmission header, and one or more DL_PDU"s (after they have seen bit-stuffing, FEC, TDC, and scrambling).

DL_PDU: datalink layer PDU (with length information) before applying FEC, TDC, data scrambling and 0 bit stuffing.

Other: There are two data structures used implicitly by the Type SAP component. These structures are not declared explicitly, and as such their detailed contents and organizations are not listed here. The first such structure is the "queue." This structure is, as the name suggests, a FIFO list of outstanding requests (Test Requests, Unitdata Requests) waiting to be handled by the type SAP component. The second structure does not have a name and is referenced by procedures such as "add_message." This structure holds information about all outgoing PDU's requiring acknowledgements (including transmission count, which stations have responded, what the DL_Unitdata_id is for the corresponding request which resulted in the construction of that DL_PDU, etc.)

Note: * denotes a recommendation not specified in MIL-STD 188-220B.

specification datalink_layer systemactivity;
default
individual queue;
timescale millisecond;

const
N4 = any integer;  { Type 1 max. retransmission count }
NS = any integer;  { Number of Stations > 0 }
NS_LESS_1 = any integer;  { NS - 1 }
NULL_ID = -1;  { No ID needed for station message }
NEED_COUPLED_ACK = 1;
NOT_NEED_COUPLED_ACK = 0;

{ Estelle-defined transition firing priorities }
LOW = 2;
MEDIUM = 1;
HIGH = 0;

{ Timer values }
BUSY_TIMER_TIMEOUT_PERIOD = any integer;
QUEUE_REQUEST_RETRIEVAL_PERIOD = any integer;  { Fecko, 05/30/97 }

{ Special Link addresses }
NET_ENTRY_ADDRESS = 1;
NETCON_ADDRESS = 2;
IMMEDIATE_RETRANS_ADDRESS = 3;  { Amer, 8/27/97 }
GLOBAL_MULTICAST_ADDRESS = 127;
ALL_ADDRESSEE_LIST_SIZE = any integer;
MAX_NO_OF_DEST_ADDRESS = 16;
STATION_MESSAGE_TABLE_SIZE = any integer;

type
8_bit_field = integer;  { More precisely: array [0..7] of bit }
16_bit_field = integer;  { More precisely: array [8..15] of bit }
FEC_TDC_scrambling = type_8_bit_field;
P_bit = 0..1;

DL_PDU_type = (UI_COMMAND, URR_COMMAND, URR_RESPONSE,
URNR_COMMAND, URNR_RESPONSE,
TEST_COMMAND, TEST_RESPONSE); type
8_bit_field = integer;  { More precisely: array [0..7] of bit }
16_bit_field = integer;  { More precisely: array [8..15] of bit }

message_number: type_8_bit_field;
unique_id: type_32_bit_field;
group_address: type_32_bit_field;
equipment_preamble_time: type_16_bit_field;
message: type_16_bit_field;
message_type: type_DL_PDU_type;
message_length: type_8_bit_field;
message_data: type_data;
message_data_length: type_data_length;
message_parameters: type_DL_PDU_addressee_list;
key_time: type_16_bit_field; 
equipment_postamble_time: type_16_bit_field; 
carrier_dropout_time: type_16_bit_field; 
tolerance_time: type_16_bit_field; 
processing_time: type_8_bit_field; 
NAD_method: type_8_bit_field; 
subscriber_rank: type_8_bit_field; 
number_of_stations: type_8_bit_field; 
urgent_percent: type_8_bit_field; 
priority_percent: type_8_bit_field; 
traffic_load: type_8_bit_field; 
max_transmit_seconds: type_8_bit_field; 
physical_conceation: type_8_bit_field; 
data_link_conceation: type_8_bit_field; 
max_UT_DATA_1_info_bytes: type_16_bit_field; 
type2_ack_timer: type_16_bit_field; 
type4_window: type_8_bit_field; 
quiet_mode: type_8_bit_field; 

end;

type_message_id = integer;
type_multidwell = (SIX, ELEVEN, THIRTEEN);
type_net_activity = (CLEAR, BUSY, BUSY_WITH_DATA, BUSY_WITH_VOICE);
type_unique_id = type_32_bit_field;
type_precedence = (PREC_URGENT, PREC_PRIORITY, PREC_ROUTINE);
type_retransmit_count = 0..N;
type_throughput = (THRU_NORMAL, THRU_HIGH);
type_time = integer;
type_topology_update_id = ...;
type_transmission_status = (TRANSMIT_COMPLETE_OR_IDLE, TRANSMIT_IN_PROCESS, TRANSMIT_ABORTED);
type_DL_PDU: type_data;
length: type_data_length;
end;
type_type1SAP_message = type_DL_PDU;
type_station_message = type_DL_PDU;
type_station_message_table = array [1..STATION_MESSAGE_TABLE_SIZE] of record 
message_id: type_message_id;
message: type_DL_PDU;
precedence: type_precedence;
P_bit: type_P_bit;
end;
type_ack_success_failure = (FAIL, SUCCESS); 
(type_service_type = (TYPE1ACK, TYPE1NOACK, TYPE2P0, TYPE2P1, TYPE4); 
type_test_result = (SUCCESS, FAILURE);

{*******************************************************************}
{ Functions/Procedures Used In Module m_station_component }{*******************************************************************}
function get_DL_PDU_type(data: type_DL_PDU): type_DL_PDU_type; primitive;

{ Get P/F bit of a given DL_PDU }
function get_PF_bit(message: type_DL_PDU): type_P_bit; primitive;

{ Get source address of a given DL_PDU }
function get_src_address(message: type_DL_PDU): type_address;

{ Get Unique ID of a station who send the DL_PDU }
function get_unique_id(message: type_DL_PDU): type_unique_id;

{*******************************************************************}
{ Functions/Procedures Used In Module m_type1SAP_component }{*******************************************************************}
procedure add_message(message_id: integer;
message: type_type1SAP_message;
NL_id: type_id); primitive;

function addr_index(address: type_address): integer;

function addr_list_size(addr_list: type_addr_list): integer;

function all_acked(message_id: integer): boolean;

function choose_message_id: integer;

function create_busy_address_list(addr_list: type_addr_list): type_addr_list;

function create_non_busy_address_list(addr_list: type_addr_list): type_addr_list;

function create_queued_request(addr_list: type_addr_list;
src_addr: type_address;
precedence: type_precedence;
acknoack: type_ack_required;
immediate_retrans_req: type_immediate_retrans_req;
request_type: type_request_type;
request_id: type_id); type_queued_request;

function create_type1SAP_message(userdata: type_data;
addr_list: type_addr_list;
src_addr: type_address;
precedence: type_precedence;
acknoack: type_ack_required;
immediate_retrans_req: type_immediate_retrans_req;
data: type_data;
data_length: type_data_length;
req_type: type_request_type;
request_id: type_id); type_type1SAP_message;

function create_queue_message(type_message_id: integer;
message: type_type1SAP_message;
NL_id: type_id); primitive;

{ Enter a type1SAP message into some unspecified structure used only by the type1SAP component. Set the list of received acks for that message’s corresponding DL_PDU to nil. Set the transmission count for that message’s corresponding DL_PDU to 1. The NL_id is the ID given back to the NL if there is an eventual ack_failure sent up. }

function get_DL_PDU_type(data: type_DL_PDU): type_DL_PDU_type;

function get_PF_bit(message: type_DL_PDU): type_P_bit;

function get_src_address(message: type_DL_PDU): type_address;

function get_unique_id(message: type_DL_PDU): type_unique_id;

function all_acked(message_id: integer): boolean;

function choose_message_id: integer;

function create_busy_address_list(addr_list: type_addr_list): type_addr_list;

function create_non_busy_address_list(addr_list: type_addr_list): type_addr_list;

function create_queued_request(addr_list: type_addr_list;
src_addr: type_address;
precedence: type_precedence;
acknoack: type_ack_required;
immediate_retrans_req: type_immediate_retrans_req;
data: type_data;
data_length: type_data_length;
req_type: type_request_type;
request_id: type_id); type_queued_request;

function create_type1SAP_message(userdata: type_data;
addr_list: type_addr_list;
data_length: type_data_length);
function get_DL_Unitdata_id_from_id(message_id : integer) : type_DL_Unitdata_id; primitive;
{ Take an already existing message id and create a new typeDL_Unitdata message,
changing the address list to include only those addresses which
have not yet acked (using the ack info for that id as well). }

function create_updated_type1SAP_message(message_id : integer) : type_type1SAP_message;
{ For abstraction purposes. Make a message for the station component out of the given args. }

if an immediate retransmission is requested, include address 3 in the
addr_list in its appropriate place. See restrictions in 188-220
Section 3.4.2.2.2.5 } (Amer 9/4/97)

function get_request_DL_Unitdata_id(request : type_queued_request) :
function get_request_OP_test_req_id(request : type_queued_request) :
{ For abstraction purposes. The request argument MUST have type DL_REQUEST }

function get_nonresponding_addresses(message_id : integer) : type_addr_list; primitive;
{ For abstraction purposes. The request argument MUST have type TEST_REQUEST }

function get_request_data(request: type_queued_request) : type_data; primitive;
{ For abstraction purposes. }

function get_request_data_length(request : type_queued_request) :
function get_DL_Unitdata_id_from_id(message_id : integer) :
{ For abstraction purposes. }

function get_addr_from_list((var) addr_list: type_addr_list) : type_address; primitive;
{ Remove the first address from the list addr_list, and return it. Side effect of changing addr_list IS permanent and intended. }

function get_data(message : type_station_message) : type_data; primitive;
{ For abstraction purposes. }

function get_dest_addr_list(message : type_station_message) : type_addr_list; primitive;
{ For abstraction purposes. }

function get_message_id_being_acked(message : type_station_message): integer; primitive;
{ Get the message_id being responded to in the current URR or URNR
sent from the station component in argument "message". }

function get_message_transmission_count(message_id : integer) : integer; primitive;
{ Return the transmission count for the DL_PDU assoc. with id message_id }

function get_msg_data(message_id : integer) : type_data; primitive;
{ For abstraction purposes. }

function get_msg_data_length(message_id : integer) : type_data_length; primitive;
{ Return the message data length assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_neighbor_detected(message_id : integer ) : type_neighbor_detected;
{ Return the message neighbor detected assoc. with id message_id } (Amer 8/27/97)

function get_msg_immediate_retrans(message_id : integer) : type_immediate_retrans_req;
{ Return message immediate retrans value assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_precedence(message_id : integer) : type_precedence; primitive;
{ Return the message precedence assoc. with id message_id } (Amer 9/4/97)

function get_msg_data(message_id : integer) : type_data; primitive;
{ Return the message data assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_data_length(message_id : integer) : type_data_length; primitive;
{ Return the message data length assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_request_type(message_id : integer) : type_request_type; primitive;
{ Return the message request type assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_src_addr(message_id : integer) : type_address; primitive;
{ Return the message source address assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_DL_Unitdata_or_OP_test_req_id(message_id : integer) : type_id;
primitive;
{ Return the message DL_Unitdata_id or OP_test_req_id assoc. with id message_id }

function get_nonresponding_addresses(message_id : integer) : type_addr_list; primitive;
{ For abstraction purposes. }

function get_request_DL_Unitdata_id(request : type_queued_request) :
function get_NON_test_req_id : type_id; primitive;
{ For abstraction purposes. }

function get_request_OP_test_req_id(request : type_queued_request) : type_id; primitive;
{ For abstraction purposes. }

function get_request_data(request: type_queued_request) : type_data; primitive;
{ For abstraction purposes. }

function get_request_data_length(request : type_queued_request) :
function get_DL_Unitdata_id_from_id(message_id : integer) :
{ For abstraction purposes. }

function get_addr_from_list((var) addr_list: type_addr_list) : type_address; primitive;
{ Remove the first address from the list addr_list, and return it. Side effect of changing addr_list IS permanent and intended. }

function get_data(message : type_station_message) : type_data; primitive;
{ For abstraction purposes. }

function get_dest_addr_list(message : type_station_message) : type_addr_list; primitive;
{ For abstraction purposes. }

function get_message_id_being_acked(message : type_station_message): integer; primitive;
{ Get the message_id being responded to in the current URR or URNR
sent from the station component in argument "message". }

function get_message_transmission_count(message_id : integer) : integer; primitive;
{ Return the transmission count for the DL_PDU assoc. with id message_id }

function get_msg_data(message_id : integer) : type_data; primitive;
{ For abstraction purposes. }

function get_msg_data_length(message_id : integer) : type_data_length; primitive;
{ Return the message data length assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_neighbor_detected(message_id : integer ) : type_neighbor_detected;
{ Return the message neighbor detected assoc. with id message_id } (Amer 8/27/97)

function get_msg_immediate_retrans(message_id : integer) : type_immediate_retrans_req;
{ Return message immediate retrans value assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_precedence(message_id : integer) : type_precedence; primitive;
{ Return the message precedence assoc. with id message_id } (Amer 9/4/97)

function get_msg_data(message_id : integer) : type_data; primitive;
{ Return the message data assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_data_length(message_id : integer) : type_data_length; primitive;
{ Return the message data length assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_request_type(message_id : integer) : type_request_type; primitive;
{ Return the message request type assoc. with id message_id } (Fecko, 05/30/97)

function get_msg_src_addr(message_id : integer) : type_address; primitive;
{ Return the message source address assoc. with id message_id } (Fecko, 05/30/97)
Functions/Procedures Used In Module m_ack_timer

function compute_delay(number_acks: integer) : type_time; primitive;
{ Compute the time required for coupled responses to be issued,
in succession, by a number of stations equal to number_acks }

Functions/Procedures Used In Module m_station_component

Test if the DL_PDU is valid.
A DL_PDU is invalid if it has one or more of the following:
a. not bounded by a beginning and ending flag
b. too short
c. too long
d. has an invalid address or control field
e. has an FCS error
function DL_PDU_valid(DL_PDU: type_DL_PDU): boolean; primitive;

Get number of expected acks (DL_PDU: type_DL_PDU) : integer; primitive;

Buffer message from type1SAP in station_message_table
procedure buffer_message_in_station_message_table(
  var station_message_table: type_station_message_table;
  message_id: type_message_id;
  message: type_DL_PDU;
  precedence: type_precedence); primitive;

Get message(s), applying FEC, TDC, data scrambling (if appropriate) and 0 bit stuffing, concatenate them together if required (data_link_concatenation = on), desired and possible
procedure construct_and_concatenate_DL_PDU_of_highest_priority(
  var transmission_to_be_sent: type_data;
  var transmission_length: type_data_length;
  var need_coupled_ack: boolean; { TRUE if coupled ack is expected }
  var message_id: type_message_id;
  var no_of_expected_acks: integer;
  var station_message_table: type_station_message_table;
  FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  multidwell: type_multidwell); primitive;

Construct a coupled ack for the DL_PDU_received
procedure construct_coupled_ack(
  var coupled_ack: type_DL_PDU;
  src_address: type_address;
  DL_PDU_received: type_DL_PDU); primitive;

Construct a transmission by applying FEC, TDC and data scrambling (if required) and 0 bit stuffing to message in station_message_table. Used for coupled ack
procedure construct_transmission(
  var transmission_to_be_sent: type_data;
  var transmission_length: type_data_length;
  DL_PDU: type_DL_PDU;
  FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  multidwell: type_multidwell); primitive;

Construct a coupled ack for the DL_PDU_received
procedure construct_coupled_ack(
  var coupled_ack: type_DL_PDU;
  src_address: type_address;
  DL_PDU_received: type_DL_PDU); primitive;

Construct a transmission by applying FEC, TDC and data scrambling (if required) and 0 bit stuffing to message in station_message_table. Used for coupled ack
procedure construct_transmission(
  var transmission_to_be_sent: type_data;
  var transmission_length: type_data_length;
  DL_PDU: type_DL_PDU;
  FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  multidwell: type_multidwell); primitive;

Fill in DL_PDU_addressee_list with all addresses who will appear in the next DL_PDU
procedure decide_DL_PDU_addressee(
  var all_addressee_list: type_all_addressee_list;
  var all_address: type_all_address_list); primitive;

Fill in all_addressee_list with all addresses this station want to send message to
procedure decide_all_addressee(
  var all_addressee_list: type_all_addressee_list); primitive;

Check to see whether desired addressee exist(s).
There are 2 types of addressee: P=0 or P=1
function exist_address(
  all_address: type_all_address_list;
  P_bit: type_P_bit); primitive;

Test if DL_PDU_addressee_list is empty
function DL_PDU_addressee_list_empty(DL_PDU_addressee_list: type_DL_PDU_addressee_list): boolean; primitive;

Get n_th DL_PDU in a transmission
function get_DL_PDU_from_transmission(
  transmission: type_data;
  transmission_length: type_data_length;
  n_th: integer; { n_th DL-PDU in transmission }
  FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  multidwell: type_multidwell): type_DL_PDU; primitive;

Get topology_update_id from a transmission
function get_topology_update_id_from_transmission(transmission: type_data)
  topology_update_id: type_topology_update_id; primitive;

Init FEC_TDC_scrambling variable
procedure init_FEC_TDC_scrambling(
  var FEC_TDC_scrambling: type_FEC_TDC_scrambling); primitive;

procedure init_link_parameter_table(
  var link_parameter_table: type_link_parameter_table); primitive;

Init parameters
procedure init_parameters(
  var link_parameter_table: type_link_parameter_table;
  var FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  var multidwell: type_multidwell;
  var my_address: type_address;
  var quiet_mode: type_quiet_mode); primitive;

procedure init_station_message_table(
  var station_message_table: type_station_message_table); primitive;

Assign an unique ID to this station
procedure init_unique_id(var unique_id: type_unique_id); primitive;

Test if the received DL_PDU is coupled ack or not
function is_coupled_ack(DL_PDU_received: type_DL_PDU): boolean; primitive;

Test if the message is addressed to this station
function is_for_me(DL_PDU_received: type_DL_PDU; my_address: type_address): boolean; primitive;

Having retrieved no_retrieved DL_PDU from transmission, test if there are more
function more_DL_PDU_in_transmission(
  transmission: type_data;
  transmission_length: type_data_length;
  no_retrieved: integer); boolean; primitive;

When a response is received, the source address of the response is removed
procedure remove_addressee_from_DL_PDU_addressee_list(
  addressee: type_address;
  var DL_PDU_addressee_list: type_DL_PDU_addressee_list); primitive;
function require_coupled_ack(DL_PDU_received: type_DL_PDU): boolean; primitive;

function station_message_table_empty(station_message_table: type_station_message_table): boolean; primitive;

function service_type_to_be_used(delay_req: type_delay; throughput: type_throughput; reliability: type_reliability) : type_service_type; primitive;

function is_immediate_retrans_requested(delay_req: type_delay; throughput: type_throughput; reliability: type_reliability; precedence: type_precedence) : type_immediate_retrans_req; primitive;

channel NL_DL (network, datalink);
by network:
  DL_Unitdata_Req (DL_Unitdata_id: type_DL_Unitdata_id; dest_addr_list: type_addr_list; src_addr: type_address; topology_update_id: type_topology_update_id; precedence: type_precedence; throughput: type_throughput; delay_req: type_delay; reliability: type_reliability; data: type_data; data_length: type_data_length);
by datalink:
  DL_Unitdata_Ind (dest_addr_list: type_addr_list; src_addr: type_address; topology_update_id: type_topology_update_id; data: type_data; data_length: type_data_length);
  DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id; ack_success_failure: type_ack_success_failure; topology_update_id: type_topology_update_id; data: type_data; data_length: type_data_length);

channel DL_PL (datalink, physical);
by datalink:
  PL_Unitdata_Req (data: type_data; data_length: type_data_length; FEC_TDC_scrambling: type_FEC_TDC_scrambling; multidwell: type_multidwell; coupled: type_coupled (*));
by physical:
  PL_Unitdata_Ind (data: type_data; data_length: type_data_length; FEC_TDC_scrambling: type_FEC_TDC_scrambling; multidwell: type_multidwell; topology_update_id: type_topology_update_id);
preference: type_precedence;
acknoack: type_ack_required;
immmediate_retrans_req: type_immediate_retrans_req; {Amer 8/27/97}
data: type_data;
data_length: type_data_length);

by type1SAP:
DL_Unitdata_Ind (dest_addr_list: type_addr_list;
src_addr: type_address;
topology_update_id: type_topology_update_id;
data: type_data;
data_length: type_data_length);

by type1SAP:
DL_Status_Ind (DL_Unitdata_id: type_DL_Unitdata_id;
ack_success_failure: type_ack_success_failure;
address_list: type_addr_list;
type2_connection_status: type_type2_connection_status;
neighbor_detected: type_neighbor_detected);  {Amer 8/27/97}

channel station_ack_timer (station, ack_timer);
by station:
start_ack_timer (message_id: type_message_id; no_expected_acks: integer);
start_timer (time: type_time);
stop_ack_timer; { Fecko, 05/30/97 }

by ack_timer:
ack_timer_timeout;
ack_timer_stopped; { Fecko, 05/30/97 }

channel station_TP_timer (station, TP_timer); { Fecko, 05/30/97 }
by station:
start_TP_timer (no_expected_acks: integer);
stop_TP_timer;

by TP_timer:
TP_timer_timeout;
current_no_queue_entries := 0;
end;
enend;

trans

{ Unitdata Request (NOACK) received from upper layer }
from ACTIVE_STATE to same
when ip_station_upper_mux.DL_Unitdata_Req provided (acknoack = NOACK)
name TYPE1SAP_2:
begin
non_busy_addresses := create_non_busy_address_list(dest_addr_list);
busy_addresses := create_busy_address_list(dest_addr_list);

if (addr_list_size (non_busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
v_message := create_type1SAP_message(
data, non_busy_addresses, data_length, 0,
immediate_retrans_req); (Amer 9/4/97)
id_num := 0;
output ip_station_lower_mux.type1SAP_message(
v_message, precedence, id_num, topology_update_id)
end;

if (addr_list_size (busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
request_type := DL_REQUEST;
request := create_queued_request(busy_addresses, src_addr, precedence,
acknoack, immediate_retrans_req, (Amer 9/4/97)
data, data_length, request_type,
DL_Unitdata_id);
queue_request(request)
end
end;

{ Unitdata Request (YESACK) received from upper layer }
from ACTIVE_STATE to same
when ip_station_upper_mux.DL_Unitdata_Req provided (acknoack = YESACK)
name TYPE1SAP_3:
begin
non_busy_addresses := create_non_busy_address_list(dest_addr_list);
busy_addresses := create_busy_address_list(dest_addr_list);

if (addr_list_size (non_busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
v_message := create_type1SAP_message(
data, non_busy_addresses, data_length, 1,
immediate_retrans_req); (Amer 9/4/97)
id_num := choose_message_id;
output ip_station_lower_mux.type1SAP_message(
v_message, precedence, id_num, get_topology_update_id)
end;

if (addr_list_size (busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
request_type := TEST_REQUEST;
request := create_queued_request(busy_addresses, src_addr, precedence,
acknoack, NO_IMMED_RETRANS, (Test never use immed retrans; Amer 9/4/97)
data, data_length, request_type,
OP_test_req_id);
queue_request(request);
add_message(id_num, v_message, OP_test_req_id)
end
end;

{ Test Request (YESACK) received from Operator }
from ACTIVE_STATE to same
when ip_OPERATOR.OP_test_req provided (acknoack = YESACK)
name TYPE1SAP_4:
begin
non_busy_addresses := create_non_busy_address_list(destinations);
busy_addresses := create_busy_address_list(destinations);

if (addr_list_size (non_busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
v_message := create_type1SAP_message(
data, non_busy_addresses, data_length, 1,
NO_IMMED_RETRANS); (Test never use immed retrans; Amer 9/4/97)
id_num := 0;
output ip_station_lower_mux.type1SAP_message(
v_message, precedence, id_num, get_topology_update_id)
end;

if (addr_list_size (busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
request_type := TEST_REQUEST;
request := create_queued_request(busy_addresses, src_addr, precedence,
acknoack, NO_IMMED_RETRANS, (Test never use immed retrans; Amer 9/4/97)
data, data_length, request_type,
OP_test_req_id);
queue_request(request);
add_message(id_num, v_message, OP_test_req_id)
end
end;

{ Test Request (NOACK) received from Operator }
from ACTIVE_STATE to same
when ip_OPERATOR.OP_test_req provided (acknoack = NOACK)
name TYPE1SAP_5:
begin
non_busy_addresses := create_non_busy_address_list(destinations);
busy_addresses := create_busy_address_list(destinations);

if (addr_list_size (non_busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
v_message := create_type1SAP_message(
data, non_busy_addresses, data_length, 0,
NO_IMMED_RETRANS); (Test never uses immediate retrans; Amer 9/4/97)
id_num := 0;
output ip_station_lower_mux.type1SAP_message(
v_message, precedence, id_num, get_topology_update_id)
end;

if (addr_list_size (busy_addresses) > 0) then { Fecko, 05/30/97 }
begin
request_type := TEST_REQUEST;
request := create_queued_request(busy_addresses, src_addr, precedence,
acknoack, NO_IMMED_RETRANS, (Test never uses immed retrans; Amer 9/4/97)
data, data_length, request_type,
OP_test_req_id);
queue_request(request)
end
end;

{ Message received from Station Component }
from ACTIVE_STATE to same
when ip_station_lower_mux.station_message provided
name TYPE1SAP_6to14;
begin
msg_type := get_message_type(message);
case msg_type of
  UIC: begin { TYPE1SAP_6 }
    Station Component received a UI Command packet
    output ip_station_upper_mux.DL_Unitdata_Ind(
      get_dest_addr_list(message),
      get_src_addr(message),
      topology_update_id,
      get_data(message),
      get_data_length(message));
  end;
  TESTR: begin { TYPE1SAP_9 }
    Station Component received a TEST Response packet
    output ip_OPERATOR.OP_test_ind(SUCCESS,
      get_dest_addr(message),
      get_src_addr(message),
      get_data(message),
      get_data_length(message));
  end;
  TESTC: begin { TYPE1SAP_10 }
    Station Component received a TEST Command packet
    v_message := create_type1SAP_message(get_data(message),
      make_addr_list(get_src_addr(message)),
      get_data_length(message),
      0, NO_IMMED_RETRANS);
    output ip_station_lower_mux.type1SAP_message(
      v_message, PREC_ROUTINE, 0, topology_update_id);
  end;
  URRC: begin { TYPE1SAP_11 }
    Station Component received a URR Command packet
    index := addr_index(get_src_addr(message));
    type1_busy_status[index] := STATION_FREE;
    output ip_busy_timer[index].stop_busy_timer;
  end;
  URRR: begin { TYPE1SAP_12 }
    Station Component received a URR Response packet
    index := addr_index(get_src_addr(message));
    type1_busy_status[index] := STATION_BUSY;
    output ip_busy_timer[index].start_busy_timer;
  end;
end; {case}
end; {transition}

{ Ack timer for current outgoing packet (at lowest level) times out,
  but this is not the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout

{ Ack timer for current outgoing packet (at lowest level) times out,
  and this is the N4th transmission of that packet }
from ACTIVE_STATE to same
when ip_ack_timer.ack_timer_timeout

begin
  { set ack_success_failure_ind and connection_status first, then...}
  ack_success_failure_ind := SUCCEED; {Amer 2/7/97}
  output ip_station_upper_mux.DL_Status_Ind(
    get_DL_Unitdata_id_from_id(message_id),
    ack_success_failure_ind,
    get_nonresponding_addresses(message_id),
    connection_status,
    test_neighbor_detected(message_id)); {Amer 8/27/97}
end;

{ Any station which was BUSY has its busy_timer timeout }
from ACTIVE_STATE to same

when ip_busy_timer[i].busy_timer_timeout

begin
  if (addr_list_size(non_buses_addresses) > 0) then { Fecko, 05/30/97 }

  index := addr_index(get_src_addr(message));
  type1_busy_status[index] := STATION_BUSY;

  begin
    { Notify UL of successful ack. App’d 19Dec97 Comment Resolution }
    ack_success_failure_ind := SUCCEED; {Amer 9/4/97 }
    output ip_station_upper_mux.DL_Status_Ind(
      get_DL_Unitdata_id_from_id(message_id),
      ack_success_failure_ind,
      get_nonresponding_addresses(message_id),
      connection_status,
      test_neighbor_detected(message_id)); {Amer 8/27/97}
  end;
end;
name TYPE1SAP_17:
begin
    type1_busy_status[i] := STATION_FREE;
end;

( Periodically process the request at the head of the request queue. Transition is made periodic to prevent processing requests in an infinite loop. HIGH priority allows to avoid non-determinism, should TYPE1SAP_18 and TYPE1SAP_15 become fireable at the same time. )
( Fecko, 05/30/97 )
from ACTIVE_STATE to same
provided (current_no_queue_entries > 0)
delay (QUEUED_REQUEST RETRIEVAL_PERIOD)  ( Fecko, 05/30/97 )
priority HIGH
name TYPE1SAP_18:
begin
    process_queue_entry;
end;
end;  { b_type1SAP_component }

{*******************************}
{    URR/URNR Busy Timer        }{*******************************}
module m_URR_URNR_busy_timer activity;
    ip ip_type1SAP: type1SAP_busy_timer (busy_timer);
end;body b_URR_URNR_busy_timer for m_URR_URNR_busy_timer;
state
    TIMER_OFF, TIMER_RUNNING, TIMER_RESTART;
initialize
to TIMER_OFF
begin
end;
trans
    { Start the busy timer }from TIMER_OFF to TIMER_RUNNING
when ip_type1SAP.start_busy_timer
name BT_1:
begin
end;
trans
    { Request to stop a non_running busy_timer }from TIMER_OFF to same
when ip_type1SAP.stop_busy_timer
name BT_2:
begin
end;
trans
    { Request to start an already_running busy timer }from TIMER_RUNNING to TIMER_RESTART
when ip_type1SAP.start_busy_timer
name BT_3:
begin
    { Change state to restart the timer }end;
trans
    { Stop a running busy_timer }from TIMER_RUNNING to TIMER_OFF
when ip_type1SAP.stop_busy_timer
name BT_4:
begin
end;

{***************************}
{    Type 1 Ack Timer        }{***************************}
module m_ack_timer activity;
    ip ip_type1SAP: type1SAP_ack_timer (ack_timer);
    ip_station: station_ack_timer (ack_timer);
end;body b_ack_timer for m_ack_timer;
state
    TIMER_OFF, TIMER_RUNNING;
var
    delay_value          : type_time;
    message_idnum        : type_message_id;
initialize
to TIMER_OFF
begin
delay_value := 0;
end;
trans
    { Start ack timer }from TIMER_OFF to TIMER_RUNNING
when ip_station.start_ack_timer
name ACK_TIMER_1:
begin
    message_idnum := message_id;
    delay_value := compute_delay(no_expected_acks);
end;
trans
    { Stop ack timer which is not running }from TIMER_OFF to same
when ip_type1SAP.stop_ack_timer
name ACK_TIMER_2:
begin
    { This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. } ( Fecko, 05/30/97 )
end;
trans
    { Stop ack timer which is running }from TIMER_RUNNING to TIMER_OFF
when ip_type1SAP.stop_ack_timer
name ACK_TIMER_3:
begin
end;
output ip_station.ack_timer_stopped; {Fecko}
end;

{ Start ack timer which is already running }
from TIMER_RUNNING to same
when ip_station.start_ack_timer
name ACK_TIMER_4:
begin
 { Ignore - allows timer to continue; do not restart }
end;

{ Timer timeout }
from TIMER_RUNNING to TIMER_OFF
delay(delay_value)
name ACK_TIMER_5:
begin
output ip_type1SAP.ack_timer_timeout(message_idnum); {Fecko}
{ This allows station component to move out of WAIT_FOR_TP state and resume transmitting frames. } { Fecko, 05/30/97 }
output ip_station.ack_timer_stopped; {Fecko}
end;

{ Stop ack timer which is not running } { Fecko, 05/30/97 }
from TIMER_OFF to same
when ip_station.stop_ack_timer
name ACK_TIMER_6:
begin
{ Ignore }end;

{ Stop ack timer which is running. Station component, while awaiting a coupled ack, gets another typel frame with P/F=1 and needs to abort TP procedures. } { Fecko, 05/30/97 }
from TIMER_RUNNING to TIMER_OFF
when ip_station.stop_ack_timer
name ACK_TIMER_7:
begin
{ Send "fake" timeout to TYPE1SAP to make it update retransmission counter immediately. }
output ip_type1SAP.ack_timer_timeout(message_idnum);
end;
end; { b_ack_timer }

/*******************************************************************************/
{ Type 1 TP Timer } { Fecko, 05/30/97 }
*******************************************************************************/
module m_TP_timer activity;
  ip ip_station: station_TP_timer (TP_timer);
end;
body b_TP_timer for m_TP_timer;

state
  TIMER_OFF, TIMER_RUNNING;
var
  delay_value : type_time;
initialize
to TIMER_OFF
begin
  delay_value := 0;
end;

trans
  { Start TP timer }
  from TIMER_OFF to TIMER_RUNNING
  when ip_station.start_TP_timer
  name TP_TIMER_1:
  begin
    delay_value := compute_delay(no_expected_acks);
  end;

  { Stop TP timer which is not running }
  from TIMER_OFF to same
  when ip_station.stop_TP_timer
  name TP_TIMER_2:
  begin
    { Ignore }
  end;

  { Stop TP timer which is running }
  from TIMER_RUNNING to TIMER_OFF
  when ip_station.stop_TP_timer
  name TP_TIMER_3:
  begin
  end;

  { Start TP timer which is already running }
  from TIMER_RUNNING to same
  when ip_station.start_TP_timer
  name TP_TIMER_4:
  begin
    { Ignore - allows timer to continue; do not restart }
  end;

  { Timer timeout }
  from TIMER_RUNNING to TIMER_OFF
  delay(delay_value)
  when ip_station.stop_TP_timer
  name TP_TIMER_5:
  begin
    output ip_station.TP_timer_timeout;
  end;
end; { b_TP_timer }

*******************************************************************************/
{ Station Component body definition: Active }*******************************************************************************/
state
  ACTIVE,
  WAIT_FOR_TP,
  WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION;
stateset
  ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION = [
    ACTIVE,
    WAIT_FOR_TP ];
  ALL_BUT_WAIT_FOR_TP = [
    ACTIVE,
    WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION ];
  ALL_STATES = [
    ACTIVE,
    WAIT_FOR_TP,
WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION ];

var
{ Main station component parameters and data structures }
  v_FEC_TDC_scrambling: type_FEC_TDC_scrambling;
  v_link_parameter_table: type_link_parameter_table;
  v_multidwell: type_multidwell;
  v_my_address: type_address;
  v_net_activity: type_net_activity;
  v_quiet_mode: type_quiet_mode;
  v_rexmis_count: type_rexmis_count;
  v_station_message_table: type_station_message_table;
  v_topology_update_id: type_topology_update_id;
  v_unique_id: type_unique_id;

{ Other station component variables and data structures }
  v_DL_PDU_received: type_DL_PDU;
  v_DL_PDU_address_list: type_DL_PDU_address_list;
  v_coupled_ack: type_DL_PDU;
  v_message_id: type_message_id;
  v_need_coupled_ack: boolean;
  v_n_th: integer;
  v_no_of_expected_acks: integer;
  v_service: type_service_type;
  v_transmission_to_be_sent: type_data;
  v_transmission_length: type_data_length;

modvar
  URR_URNR_busy_timer: array [0..NS_LESS_1] of m_URR_URNR_busy_timer;
type1SAP_component: m_type1SAP_component;
  ack_timer: m_ack_timer;
  TP_timer: m_TP_timer;  {Fecko 10/20/97}
initialize to ACTIVE
var
  i: integer;

begin
  init type1SAP_component with b_type1SAP_component;
  init ack_timer with b_ack_timer;
  for i:= 0 to NS_LESS_1 do
    init URR_URNR_busy_timer[i] with b_URR_URNR_busy_timer;
  end;
  connect ip_transmission to ip_DL_PDU;
  for i:= 0 to NS_LESS_1 do
    connect type1SAP_component.ip_busy_timer[i] to URR_URNR_busy_timer[i].ip_type1SAP;
    connect type1SAP_component.ip_ack_timer to ack_timer.ip_type1SAP;
    attach ip_O_type1SAP_comp to type1SAP_component.ip_OPERATOR;
    connect type1SAP_component.ip_lower_mux to type1SAP_component.ip_station_lower_mux;
    connect ip_ack_timer to TP_timer.ip_station;
  end;
  connect TP_timer with b_TP_timer;  {Fecko/Amer 10/20/97}
  connect TP_timer to TP_timer.ip_station;  {Fecko/Amer 10/20/97}
  v_multidwell := SIX;  { 6 segments per packet * }
  v_my_address := NET_ENTRY_ADDRESS;
  v_net_activity := BUSY;
  v_quiet_mode := OFF;
  init_unique_id(v_unique_id);
  init_link_parameter_table(v_link_parameter_table);
  init_station_message_table(v_station_message_table);

  init_FEC_TDC_scrambling(v_FEC_TDC_scrambling);  { * }
  end;

trans
{**************************}
{ Active Phase Transitions }
{**************************}
{ TYPE1SAP Component provide a new message }
{ from ACTIVE to same }
when ip_type1SAP_lower_mux.type1SAP_message
  priority MEDIUM
name A_1:
begin
  buffer_message_in_station_message_table(
    v_station_message_table, message_id, message, precedence);
end;

{ PL_Unitdata_Ind arrives from Physical Layer by way of internal queue }

{ Fecko, 05/30/97 }
from ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to WAIT_FOR_TP
when ip_DL_PDU.DL_PDU_form.PL_Unitdata_Ind
  provided require_coupled_ack(DL_PDU)
  priority MEDIUM
name A_3_1:
begin
  v_DL_PDU_received := DL_PDU;
  if is_for_me(v_DL_PDU_received, v_my_address) and (v_quiet_mode = OFF) and (v_net_activity = CLEAR) then
    begin
      construct_coupled_ack(v_coupled_ack, v_my_address, v_DL_PDU_received);
      { could be TEST, URR or URNR response. In the case of URR/URNR
        response, station_busy variable is checked so that correct response
        could be constructed }
      construct_transmission(v_transmission_to_be_sent, v_transmission_length,
        v_coupled_ack, v_FEC_TDC_scrambling, v_multidwell);
      { transmit in the appropriate time slot }
      output ip_PSAP.PL_Unitdata_Req(v_transmission_to_be_sent, v_transmission_length,
        v_FEC_TDC_scrambling, v_multidwell, COUPLED);
    end;
  end;

{ PL_Unitdata_Ind arrives from Physical Layer by way of internal queue }

{ Fecko, 05/30/97 }
from ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same
when ip_DL_PDU.DL_PDU_form.PL_Unitdata_Ind
  provided not require_coupled_ack(DL_PDU)
  priority MEDIUM
name A_3_2:
begin
  v_DL_PDU_received := DL_PDU;
  if is_for_me(v_DL_PDU_received, v_my_address) then
    output ip_type1SAP_lower_mux.station_message(
      v_message_id, v_DL_PDU_received, topology_update_id);
  end;

{ Station has buffered message to be sent }  
from ACTIVE to WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION  
provided ((not station_message_table_empty(v_station_message_table))  
and (v_quiet_mode = OFF) and (v_net_activity = CLEAR))  
priority MEDIUM  
name A_6:  
begin  
construct_and_concatenate_DL_PDU_of_highest_priority(vv_transmission_to_be_sent,  
v_transmission_length, v_need_coupled_ack, v_message_id,  
v_no_of_expected_acks, v_station_message_table, v_FEC_TDC_scrambling,  
v_multidwell);  
output ip_PSAP.PL_Unitdata_Req(v_transmission_to_be_sent, v_transmission_length,  
v_FEC_TDC_scrambling, v_multidwell, NOT_COUPLED);  
end;  
{ Wait for Physical Layer transmission until transmit is COMPLETE, IDLE OR ABORTED }  
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to ACTIVE  
provided ((v_need_coupled_ack = FALSE) or  
(transaction_status = TRANSMIT_COMPLETE_OR_IDLE) or  
(transaction_status = TRANSMIT_ABORTED))  
priority MEDIUM  
name A_7_1:  
begin  
v_net_activity := net_activity;  
end;  
{ Wait for Physical Layer transmission until transmit is COMPLETE, IDLE OR ABORTED.  
Start an acknowledgement timer. }  
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to WAIT_FOR_TP  
provided ((v_need_coupled_ack = TRUE) and  
(transaction_status = TRANSMIT_COMPLETE_OR_IDLE) or  
(transaction_status = TRANSMIT_ABORTED))  
priority MEDIUM  
name A_7_2:  
begin  
output ip_ack_timer.start_ack_timer(  
v_message_id, v_no_of_expected_acks);  
v_net_activity := net_activity;  
end;  
{ Wait for Physical Layer transmission if TRANSMIT_IN_PROCESS }  
from WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same  
provided (transmission_status = TRANSMIT_IN_PROCESS)  
priority MEDIUM  
name A_8:  
begin  
v_net_activity := net_activity;  
end;  
{ Wait for ack timer to be stopped }  
from WAIT_FOR_TP to ACTIVE  
when ip_ack_timer.ack_timer_stopped  
priority MEDIUM  
name A_9:  
begin  
end;  
{ Wait for TP timer to expire }  
from WAIT_FOR_TP to ACTIVE  
when ip_TP_timer.TP_timer_timeout  
priority MEDIUM  
name A_10:  
begin  
end;  
{ End of Active Phase Transitions }  

***************  
{ Common Transitions }  
***************  
{ Receive PL_Unitdata_Ind from Physical Layer; enqueue valid DL_PDU(s)  
destined for this station }  
from ALL_STATES to same  
when ip_PSAP.PL_Unitdata_Ind  
priority MEDIUM  
name C_1:  
begin  
v_n_th := 1;  
while more_DL_PDU_in_transmission(data, data_length, v_n_th) do  
begin  
v_DL_PDU_retrieved := get_DL_PDU_from_transmission(  
data, data_length, v_n_th, v_FEC_TDC_scrambling, v_multidwell);  
v_topology_update_id_received := get_topology_update_id_from_transmission(data);  
v_n_th := v_n_th + 1;  
if DL_PDU_valid(v_DL_PDU_retrieved) then  
output ip_transmission.DL_PDU_form_PL_Unitdata_Ind(  
v_DL_PDU_retrieved, v_topology_update_id_received);  
end;  
end;  
{ Set v_net_activity according to PL_Status_Ind from Physical Layer if  
not in WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION }  
from ALL_BUT_WAIT_FOR_PHYSICAL_LAYER_TRANSMISSION to same  
when ip_PSAP.PL_Status_Ind  
priority MEDIUM  
name C_4:  
begin  
v_net_activity := net_activity;  
end;  
{ Operator ask to change quiet mode }  
from ALL_STATES to same  
when ip_OSAP.OP_quiet_mode_Req  
priority HIGH  
name C_5:  
begin  
v_quiet_mode := quiet_mode;  
end;  
{ Forward DL_Unitdata_Req from NL to appropriate SAP_component }  
from ALL_STATES to same  
when ip_LSAP.DL_Unitdata_Req  
priority HIGH  
name C_12:  
begin  
{ check QOS information (delay, throughput, reliability) in the DL_Unitdata_Req  
to determine which type service is required. From that, determine which  
type of service is to be used (based on what is available). Forward the  
request to the appropriate SAP component. }  
v_service := service_type_to_be_used(delay_req, throughput, reliability);  
if v_service = TYPE1NOACK then  
output ip_type1SAP_upper_mux.DL_Unitdata_Req(  
...);  
end;  
end;
initialize
var
  i: integer;
begin
for i := 0 to NS_LESS_1 do
begin
  init NL[i] with b_NL;
  init station_component[i] with b_station_component;
  init OPERATOR[i] with b_OPERATOR;
  init PL[i] with b_PL;
  connect NL[i].ip_datalink to station_component[i].ip_LSAP;
  connect station_component[i].ip_PSAP to PL[i].ip_datalink;
  connect OPERATOR[i].ip_type1SAP to station_component[i].ip_O_type1SAP_comp;
end;
end;
end.  { end of entire specification }

******************************************************************************
| Main Specification body |
******************************************************************************
modvar
  NL: array [0..NS_LESS_1] of m_NL;
  station_component: array [0..NS_LESS_1] of m_station_component;
  OPERATOR: array [0..NS_LESS_1] of m_OPERATOR;
  PL: array [0..NS_LESS_1] of m_PL;

initialize
var
  i: integer;
begin
for i := 0 to NS_LESS_1 do
begin
  init NL[i] with b_NL;
  init station_component[i] with b_station_component;
  init OPERATOR[i] with b_OPERATOR;
  init PL[i] with b_PL;
  connect NL[i].ip_datalink to station_component[i].ip_LSAP;
  connect station_component[i].ip_PSAP to PL[i].ip_datalink;
  connect OPERATOR[i].ip_type1SAP to station_component[i].ip_O_type1SAP_comp;
end;
end;
end.  { end of entire specification }

******************************************************************************
| Main Specification body |
******************************************************************************
modvar
  NL: array [0..NS_LESS_1] of m_NL;
  station_component: array [0..NS_LESS_1] of m_station_component;
  OPERATOR: array [0..NS_LESS_1] of m_OPERATOR;
  PL: array [0..NS_LESS_1] of m_PL;

initialize
var
  i: integer;
begin
for i := 0 to NS_LESS_1 do
begin
  init NL[i] with b_NL;
  init station_component[i] with b_station_component;
  init OPERATOR[i] with b_OPERATOR;
  init PL[i] with b_PL;
  connect NL[i].ip_datalink to station_component[i].ip_LSAP;
  connect station_component[i].ip_PSAP to PL[i].ip_datalink;
  connect OPERATOR[i].ip_type1SAP to station_component[i].ip_O_type1SAP_comp;
end;
end;
end.  { end of entire specification }

******************************************************************************
| Main Specification body |
******************************************************************************
modvar
  NL: array [0..NS_LESS_1] of m_NL;
  station_component: array [0..NS_LESS_1] of m_station_component;
  OPERATOR: array [0..NS_LESS_1] of m_OPERATOR;
  PL: array [0..NS_LESS_1] of m_PL;