An overview of the emulation environment

This chapter contains information about the lab environment. First an introduction to the 3GEP is given. Then there is an overview of the different protocol layers and their communications procedure as well as a description of the signals involved. In the end of the chapter there is a description of the user interfaces used in the project.

Overview of the 3GEP

The 3GEP (3G Emulation Project) is a student environment, which emulates a UMTS network within a PC. The system consists of programs emulating the UMTS base station (Node B) and network intelligence together called UMTS Terrestrial Radio Access Network (UTRAN) and programs emulating the UMTS mobile phones called User Equipment (UE). These programs communicate via sockets, which represent the radio media.

The UTRAN consists of a protocol stack and a network adapter. The protocol stack emulates the protocols involved in UMTS and handles all communication between the UTRAN and UE. The stack is implemented in SDL, a graphical symbol oriented programming tool, which compiles the code to executable files via C code. The network adapter handles the communication between the stack and the socket connections.

The UE is similar to the UTRAN since it consists of practically the same components although they have a somewhat different content. The GUI is in this case implemented as a mobile phone with a user interface supporting three services: SMS, Audio and FTP. The GUI also gives information of the underlying communication status, displaying important parts the traffic between the UE and the UTRAN. The SDL protocol stack functionality is similar to the UTRAN with some minor differences, while the network adapter is the same.

The UMTS stack of 3GEP

The UE is the mobile phone and the UTRAN is the Node B (base station) and the network intelligence. The core of the UE and the Node B is the UMTS protocol stack. This is a communications software consisting of the first four layers of the OSI model and is standardized by the 3G Partnership Project (3GPP). From top to bottom they are: Radio Resource Control (RRC), Radio Link Control (RLC), Media Access Control (MAC) and the Physical layer (PHY).

The RRC represents the intelligence in the application. It dynamically establishes and releases logical communication channels (Transport entities), which are used by the various services. It controls the available parameters, for example: bit rate, level of retransmission and coding scheme.

The RLC is responsible for retransmissions, segmentation and reassembly. This layer contains the transport entities, which are created and deleted dynamically in pairs as services are established or released. One transport entity handles the incoming traffic and the other handles the outgoing traffic.

The MAC layer handles the timing of the packet releases and the adding of transport entity addresses on the outgoing traffic. The received traffic is sent to the corresponding transport entity via the MAC layer, which reads the address and removes it.

The physical layer takes care of coding, interleaving and adding CRC to the packets. This is the only layer, which is identical in the UE and the Node B.

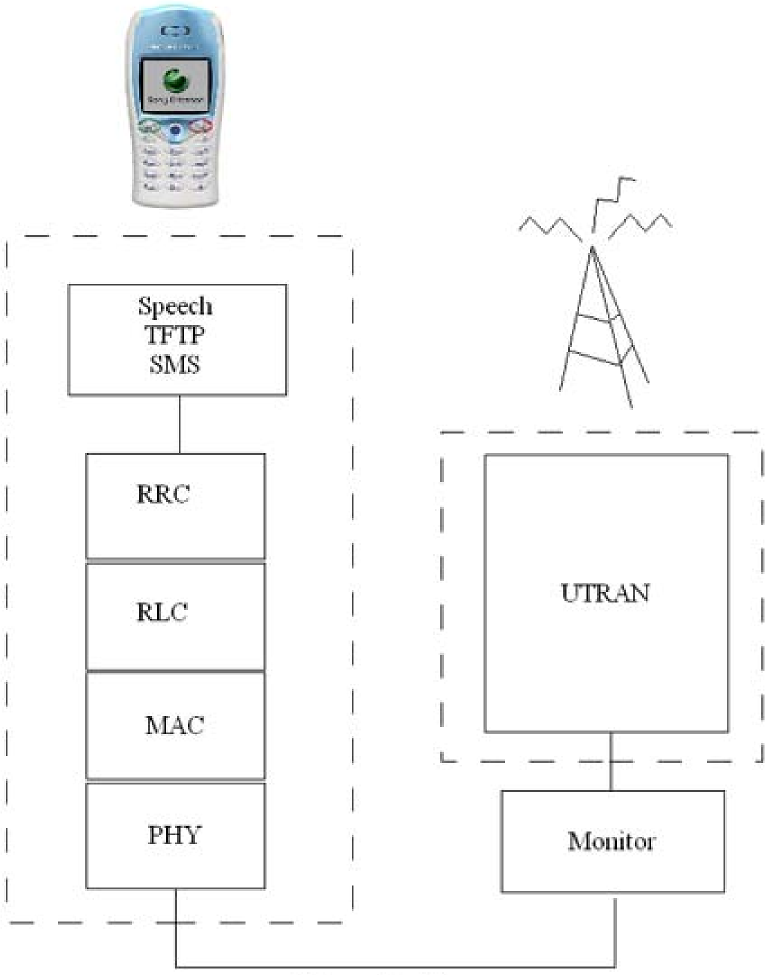


Figure 1: Protocol layers in 3GEP.

A general description of the layers in the 3GEP

This paragraph will describe the most important features of the different layers. As mentioned before the intelligence of the stack lies mostly in the RRC layer. It has direct signal routes to all of the other layers and is the only layer that has contact with the client. For example when the user turns on the mobile via the client, the mobile sends a signal to the RRC Layer, which processes it and initiates the start-up procedure. Other signals that the RRC layer can receive from the client are: power off, request to send SMS, request to initiate a voice call and a request to initiate a TFTP (data transfer) connection.

The RRC layer processes all of these requests and starts the different channels involved with the appropriate signal transactions. When the request is processed the RRC Layer is responsible for releasing the channels.

The RLC layer is composed of a monitor and a number of transport entities (TrE:s). The monitor creates and destroys TrE:s. Whenever a TrE is needed it is requested by the RRC layer which tells the monitor to start a new TrE. A TrE is a logical channel and it can send or receive data and they are therefore created in pairs. All data that is sent or received pass through a TrE if its destination is a layer above or below the RLC layer. The MAC layer and the RRC layer must therefore know which TrE:s that are available and send the data to the appropriate TrE. The TrE adds and removes the length of the packet to the PDU and it is also responsible for the segmentation and the reassembly of the packets if the packets are longer than the specified packet length.

The main responsibility of the MAC layer is to control the timing of the system. Each TrE has its own timing and after a given time interval the MAC layer sends a request to the corresponding TrE to see whether there is any data to send. The data that the MAC layer receives from the physical layer is sent to the corresponding TrE immediately.

When the PDU enter the physical layer (PHY) the CRC is calculated and added last to the PDU. The CRC is 16 bits long and checked by the physical layer on the receiving side.

After the CRC calculation the bits are coded with the coding rate 1/3, which means that the output bit string has three times the bits of the input bit string. The last step is an interleaving, which is made in two steps with two different block sizes and two different permutations.

Communication procedure

The UTRAN is broadcasting a paging message on a regular basis. This paging contains information about the UTRAN and how to contact it. When the UE receives the paging it will respond and an information exchange will start with the purpose of letting the UE enter the network. Figure 2 explains the process.

When a user activates the mobile it will eventually receive a paging and send a paging response and a request to make a connection to the UTRAN. UTRAN will confirm the request by sending an RRC_Connection_Setup signal. These signals will in turn be followed by RRC_Connection_Complete and an RRC_Connection_Established signal. The UE and UTRAN have now established a communication channel and can send information on this new channel.

The original paging channel is only used to broadcast information, therefore the UE and the UTRAN need to establish a private communication channel. Establishment of every new channel consists in the same communication procedure, which is shown in Figure 2.

The concept of channels is vital in the UMTS architecture. Each channel is a unidirectional logical link between the two RRC layers in the UE and the UTRAN. There are different types of channels with different purposes. The paging channel described above is used to broadcast information. After the paging channel a private control channel is established as described above. This channel is used to send all control information between the UE and the UTRAN.

If other services are requested, for example voice, SMS or Internet, separate communication channels are established and released dynamically.

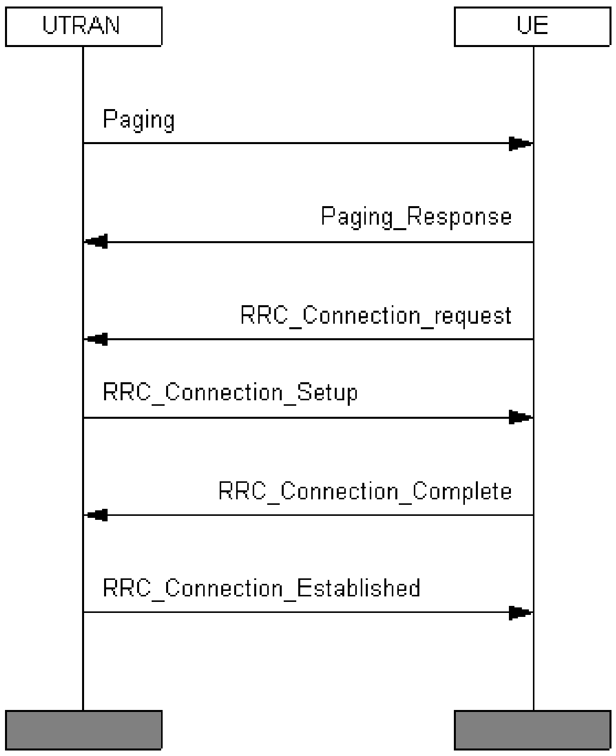


Figure 2: Communication procedure between UE and UTRAN.

Overview of the signals in the RRC layer

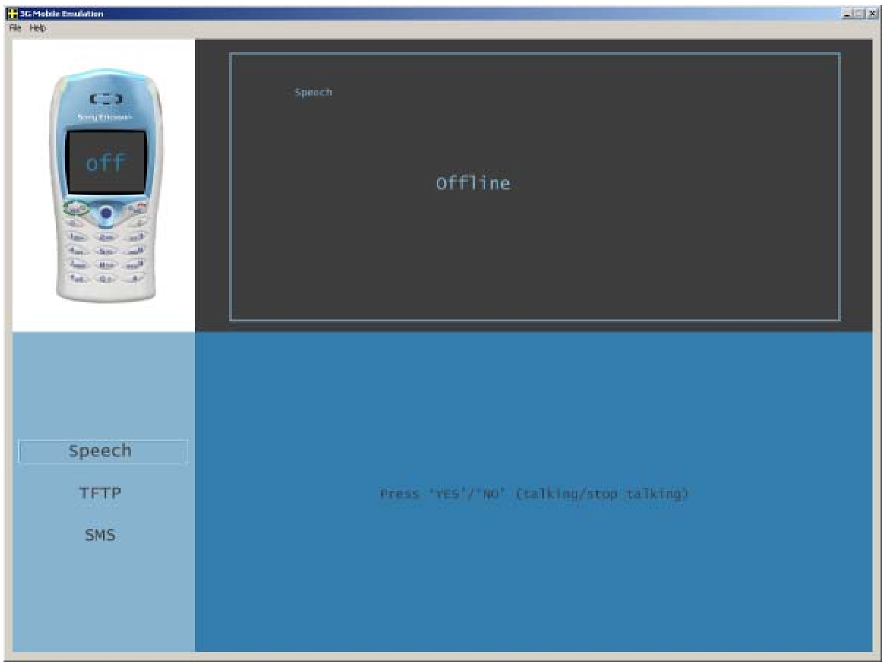
There are a number of signals, which can be sent and received in the RRC layer. Their names and functions are explained below, for a more thorough description of the parameters and a declaration of the constants used in the signals see the system block called "threegstack" in the SDL code.

RR_EST_REQ:	This signal is sent from the client and indicates a command or a service requested by the client. The different commands or services are represented by different variable values in the signal.
CRLC_CONFIG_Req:	When a transport channel is to be established or released, this signal is sent to the RLC layer. Its parameters describe the values of the channel and if it is to be established or released.
RR_EST_CNF:	This is the corresponding signal to the signal RR_EST_REQ signal. The UE uses this signal to send acknowledgements to the client. The value of the parameter lets the client identify the purpose of the acknowledgement.

EPId:	Epid is received from the RLC layer and indicates that a new Transport Entity (TrE) is created. All transport channels have corresponding TrE:s that act as controllers.
SetupConnection:	This signal is sent to the MAC layer and contains information about created TrE:s. The MAC layer needs this information since it must know which TrE to contact when information is sent to higher layers in the stack.
RLC_TM_DATA_IND:	All information that has entered the stack from the physical layer (originating from UTRAN) arrives via this signal if the final destination is the RRC layer.
RR_DATA_REQ:	This signal is similar to RR_EST_REQ with the difference that data can be sent besides the command parameter.
CloseConnection:	When a TrE is released this signal is sent to the MAC layer to indicate that the signal is no longer used.
RLC_TM_DATA_REQ	This signal carries data to UTRAN and travels through the entire stack to the peer RRC layer in UTRAN.

Description of the Graphical User Interface for User Equipment (UE)

The UE-GUI is used for emulating one mobile phone and it contains three applications: Speech, Trivial File Transfer Protocol (TFTP) and SMS. This UE-GUI is supposed to be working together with the UE SDL-stack and sends and receives all its signals to/from the stack.



Press 'YES'/'NO' (talking/stop talking)

Figure 3. Image of the GUI for UE

Starting and shutting down the phone (UE)

When the GUI (Figure 1) is launched, the UE waits in a shut down mode and the only working function is the Power-On function.

To start the phone, press the **Yes**-button on the phone image. The UE sends a Power-On message to the stack. The stack initiates itself, waits for a paging message from UTRAN, connects to the UTRAN and responds back to the UE with a Power-On message. The phone display shows the Sony-Ericsson logo if properly started and initiated. It is now possible to use the three available applications.

Shut down the phone at any time by pressing the **No**-button.

Speech: Using speech is essential for a phone. However, because the SDL executable occupies a major part of the CPU clock cycles and leaves little left for the GUI to perform speech encoding and other restrictions, the sound quality will not be especially good. Normally, when using a phone to talk in, there's someone at the other end. In this case, there's currently only a UTRAN.

TFTP: The Trivial FTP is a light variant of the “real” FTP: it uses the stop-and-wait acknowledge method instead of sliding window. To get into TFTP mode, click the “TFTP”-text on the left pane. To setup a TFTP-session, click the “Choose”-text and select a file to send. When a file is chosen, it shows up in the “File Name” and “File Size” fields. To send it, click the “Send”-text. It is possible to cancel the transmission by clicking the “Cancel”-text.

SMS

To get into the SMS view, click the “SMS”-text on the left pane. There are a few different ways to use the SMS, see Figure 4.

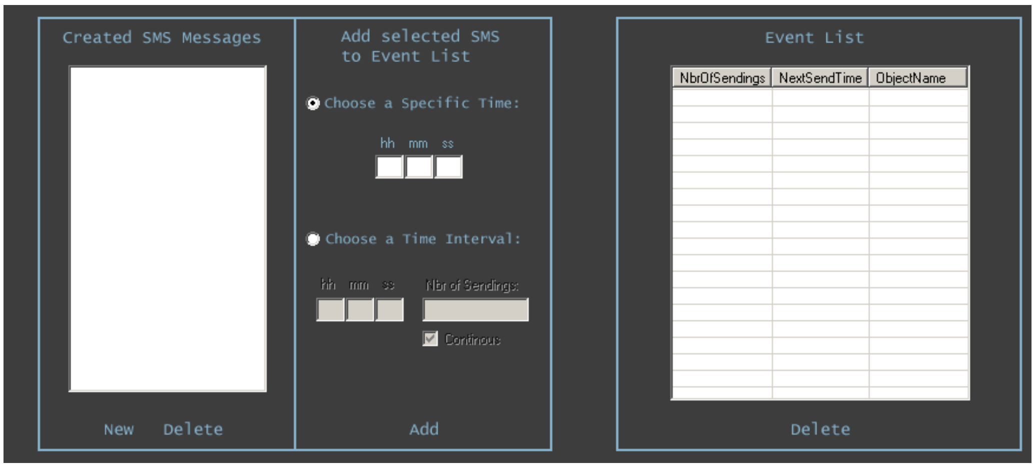


Figure 4. The SMS pane details

Start by clicking the “New”-text. Write your message, name it and save it. Then it shows in the “Created SMS Messages”-pane. Use the “Add selected...” field to send the message at a specific time or to send it in intervals. When you have done your selections, click “Add” and the message is moved to the “Event List”-pane waiting to be sent.

SMS

The section gives an introduction to the SMS service, and some practical information regarding the project development environment. The project template code is listed at the end of this document.

Overview of the SMS service

Short Message Service (SMS) is the ability to send and receive short messages to and from mobile telephones. The messages can be text messages viewed on the phone or they can be data messages (for example a new ring tone, which has been bought and sent to a phone via SMS). An SMS can be point-to-point (between two users) or point-to-omnipoint (a cell broadcast). A point-to-point message can either be mobile originated (MO) or mobile terminated (MT). If the message is mobile originated it is sent from the mobile telephone to the MSC (Mobile Switching Centre) and if the message is mobile terminated it is sent from the MSC to the mobile phone.

The SMS can contain up to 140 octets of data payload. These octets can be 140 bytes of binary data, or 160 characters of text using 7-bit ASCII, or 140 characters using 8-bit ASCII. The format of the text SMS is described below.

The following example shows the PDU for the message HALLO WORLD, sent from the number ++39 347 3820955 at 04^{h}:55^{m}:16^{s} PM of the 13th of January 2002. The mobile used was an Ericsson T10s and the Service Centre number was ++39 349 2000509:

01	80	11	00	0A	81	43	37	28	90	55	00	00	A7	0B	C8	20	93	F9	04	5D	9F	52	26	11
SCA		ID	MR	DA							PID	DCS	VP	UDL	USER DATA									
		SMS-SUBMIT TPDU																						
PDU RETURNED BY THE AT+CMGL COMMAND																								

Each field of the SMS is described in the table below:

SCA	Service Centre Address	Network operator's Service Centre number. Not required by some mobiles. A hex value of 00 or 01-80 means "unknown": the mobile will use the default number stored in mobile's settings.	1 or 2 to 12 bytes
ID	TPDU type identifier	SMS-DELIVER or SMS-SUBMIT identifiers and flags (e.g. request of a status report or presence of VP field) .	1 byte
MR	Message Reference	Progressive number (0 to 255).	1 byte
OA or DA	Originating or Destination Address	Senders or destination phone number. Note that a different number encoding from that of SCA is used.	2 to 12 bytes
PID	Protocol Identifier	Nature of data transported (FAX, voice, etc.), used by the Service Centre for a better routing.	1 byte
DCS	Data Coding Scheme	Format of the data transported (7 or 8 bits, alphabet, etc.) and where to store it (Mobile memory, SIM module, or for immediate display).	1 byte
VP	Validity Period	How long the network operator service centre will hold the message, if undelivered (A7=24 hours).	0, 1, or 7 bytes
UDL	User Data Length	Length of data, prior to encoding (e.g. 11 7-bit characters fit into 10 bytes).	1 byte
UD	User Data	Our message data, "HALLO WORLD"	0-140 bytes

Phone numbers packing

Phone numbers start with the number's length, intended as field length in *bytes* for the service centre (SCA), and *digits* for the remaining numbers (DA, OA). The second byte specifies the numbering plan: 80 = unknown, 81 = national number, 91 = international number. Then follow the digits, swapped in pairs and each occupying a nibble. This is how the number ++39 349 200-059 is encoded:

1st byte *(length)*	2nd byte *(format)*	3rd byte	4th byte	5th byte	6th byte	7th byte	8th byte
0 7	9 1	9 3	4 3	2 9	0 0	5 0	F 9

If the length is odd, the unused nibble (*semi-octet* in ETSI language) is padded with $F.

Some mobiles do not require the SCA, or accept 00 or 01-80 as valid values for the service centre address: in that case the mobile will use its default service centre number.

Message packing

An SMS message, according to ETSI specification, can be up to 140 bytes long (*octets* in ETSI terminology). The usual GSM alphabet requires only 7 bits per character (a *septet*), allowing for the packing of up to 140 * 8 / 7 = 160 characters.

The following is an example of how a 7-bit data is packed between successive bytes.

The 7-bit binary encoding of the string “GSM” is: G = 1000111, S = 1010011, M = 1001101

Let G0 be the bit 0 of letter G, G1 be the bit 1 of letter G, and so on: then the PDU will pack data as:

First Byte								Second Byte								Third Byte							
S0	G6	G5	G4	G3	G2	G1	G0	M1	M0	S6	S5	S4	S3	S2	S1	zp	zp	zp	M6	M5	M4	M3	M2
1	1	0	0	0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	1	0	0	1	1

Note how the last three spare places are padded with zeroes (zp).

The SMS environment

Overview

This chapter gives practical information about the environment you will work in. It lists the signals used by SMS service as well as the template code which is to be used in this assignment.

Practical information about the SMS environment

If you want to find more information about different data types and their operators you can press the blue question mark in the editor or the organizer. In the help window chose the search tab and write “bitstring” and press enter. If you double-click on the first topic in the list below (Using SDL Data Types) you will find useful information about the predefined types and their operators.

If you see a print out on the screen that you wish to find in the code you can do a system wide search for it. To do the search, select the system *threegstack* in the organizer and press the button with the binoculars picture.

Note that if you drag and select something in the UE or UTRAN window the program pauses until the selection is cancelled. Note also that if you drag the windows when it sends or receive a signal there is a probability that the signal is lost. Also, avoid dragging the Client window when the phone is on, because sometimes this causes the program to crash.

For the SMS assignment you have to wait until all four channels are connected until you try to send anything with the client GUI. This process is finished when the text “paging phase completed" is written on both UE and UTRAN. If it does not appear in 30 seconds restart UE, UTRAN and the client.

If the text "signal with length 0" appears in any of the UE or UTRAN windows data has been lost and the systems will most likely restart. In some occasions the systems will be unable to find each other again. Restart the system as described above if it takes more than 30 seconds.

When you are ready to test your SMS solution, choose the **"Generate" menu** in the organizer with the **system threegstack selected** and choose **"Targeting Expert"**. Do not change any settings in the "Targeting Expert" since this can make your output code **permanently corrupt**. To compile your code and create an executable file press the **"Full Make"** button. If you receive any errors during the compilation you can go to the erroneous part of the code by double clicking on the error. Your executable file will be called **component_apc.exe** and will be placed in the subfolder **Target\threegstack._0\Application_CA**.

For further instructions on compilation and testing, please refer to "Getting started with 3GEP project". You should also restart all components after each completed test since they can be in an erroneous state.

If you receive any errors during the execution you can use the printouts on the screen, add your own printout (F:=F // 'My variable:' // MyVariable, F:=putnewline(F)) or you can look in the log file. The log file (log.txt) is a file with all the signals sent in the system and their parameters. It is located in the same directory as the executable file. This is an example of a signal from the log file:

```
RLC_RM_DATA_REQ________RRC 10--RLC__ThisEntityId=510__PDUT=00000001
```

The first entry is the name of the signal, RLC_RM_DATA_REQ . The second entry RRC 10--RLC is composed of the unique id of the signal and the receiving layer. The last two entries are the signal's parameters and their values.

Note that the entries of the log file are added by using RRCLog procedure. If a signal is added to the system it must also be registered (manually) using RRCLog if it is to be visible in the log file. For further information, see the block RRCLog.

RRC-layer code in the UE

Below is the template code of the RRC layer of the UE, where you will implement the SMS service. There should not be any differences between the code in this document and the code on the computer, but if there are its the code on the computer that is correct.

This is a short list of the 8 bits commands that are sent between UE and UTRAN. These commands are valid for the signals: RLC_TM_DATA_REQ and RLC_TM_DATA_IND.

Paging	0000 0000	0
Paging Response	0000 0001	1
RRC connection request	0000 0010	2
RRC connection setup	0000 0011	3
SMS connection request	0000 0100	4
RRC connection established	0000 0101	5
Speech connection request	0000 0110	6
TFTP packet	0000 0111	7
Control connection request	0000 1000	8

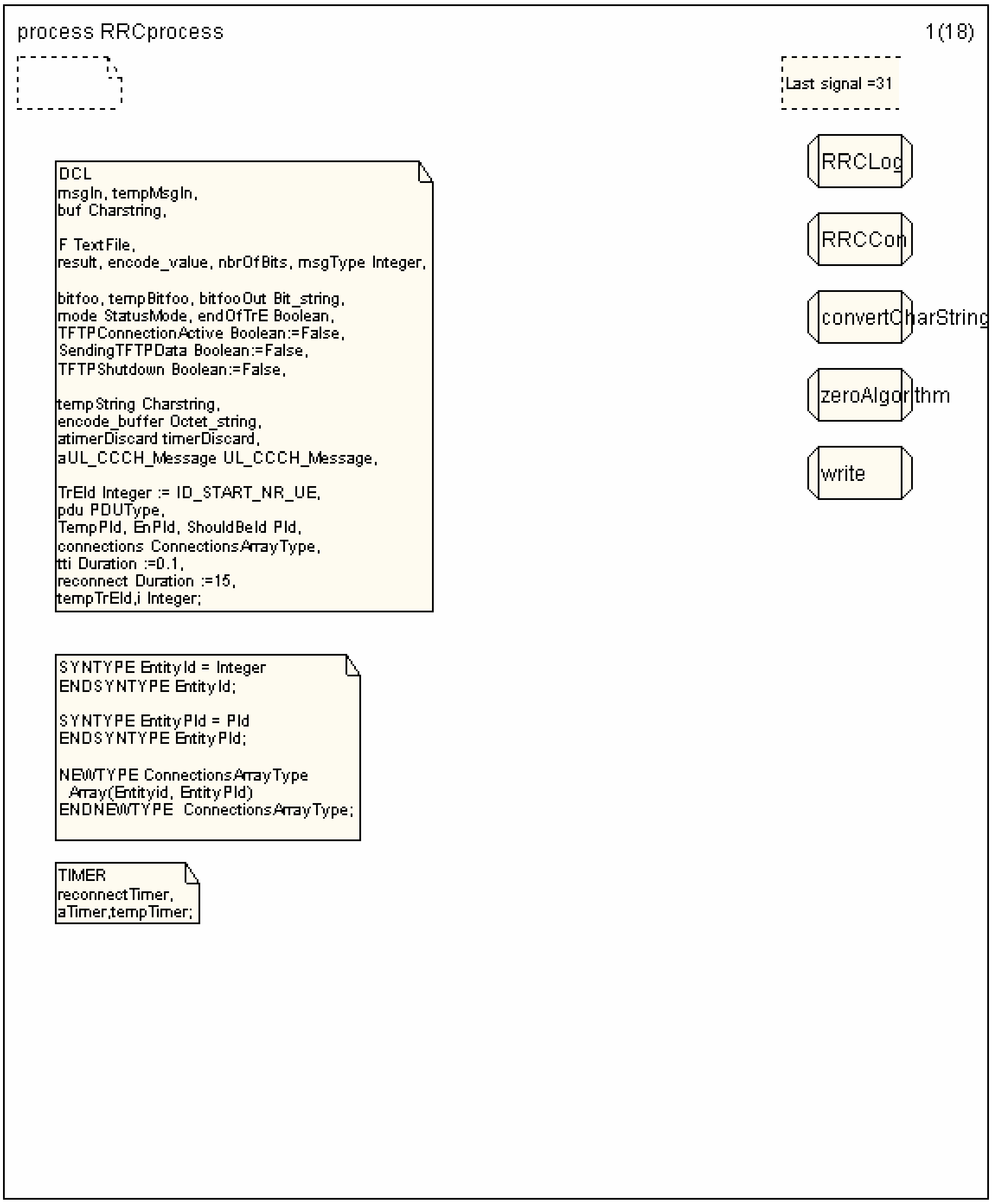
UTRAN connection reset	0000 1001	9
TFTP connection request	0000 1010	10
Not Used	0000 1011	11
RRC connection setup complete	0000 1100	12
SMS transfer complete	0000 1101	13

Some notes on the subroutines in the RRC layer:

RRCLog is a subroutine used to send information to MSCWriter about the signals. It has two charstring IN arguments, the first is the identity of the signal and the second is the name of the signal. The identity is composed of the name of the sending layer (for example RRC) a unique number (for example 17) and the name of the receiving layer (for example RLC). The example identity will then be RRC 17 RLC. RRCLog is similar to a switch statement, which handles the different signals. It adds the relevant data to the name and identity of the signal and sends it to MSCWriter.

RRCCon is used to convert bitstrings or integers to charstrings. The first parameter is an integer and the second is a bitstring. If the second parameter is an empty string (bitstr('')) the integer will be converted otherwise the bitstring will be converted. The third parameter is a Boolean, which decides if the bitstring shall be converted 1:1 or 1:8. If the value is true each bit will become one byte and if it is false 8 bits will become one byte. The result will be stored as a charstring in the fourth parameter.

ZeroAlgorithm converts between bitstrings while replacing bytes with the value zero. If a signal parameter, sent to the environment, contains a byte with the value zero, the parameter will terminate prematurely and the data after the zero will be lost. This subroutine will replace every zero with the string "AB" and every byte containing an 'A' with the string "AA" when converting from bitstrings to charstrings. It will also convert a charstring, with the above-mentioned changes, to the correct bitstring. The subroutine has three parameters. The first is the bitstring, which is to be converted and the second is where the output bitstring will be stored. The third parameter is a Boolean, which if true will bring back the zeros and if false will remove them.

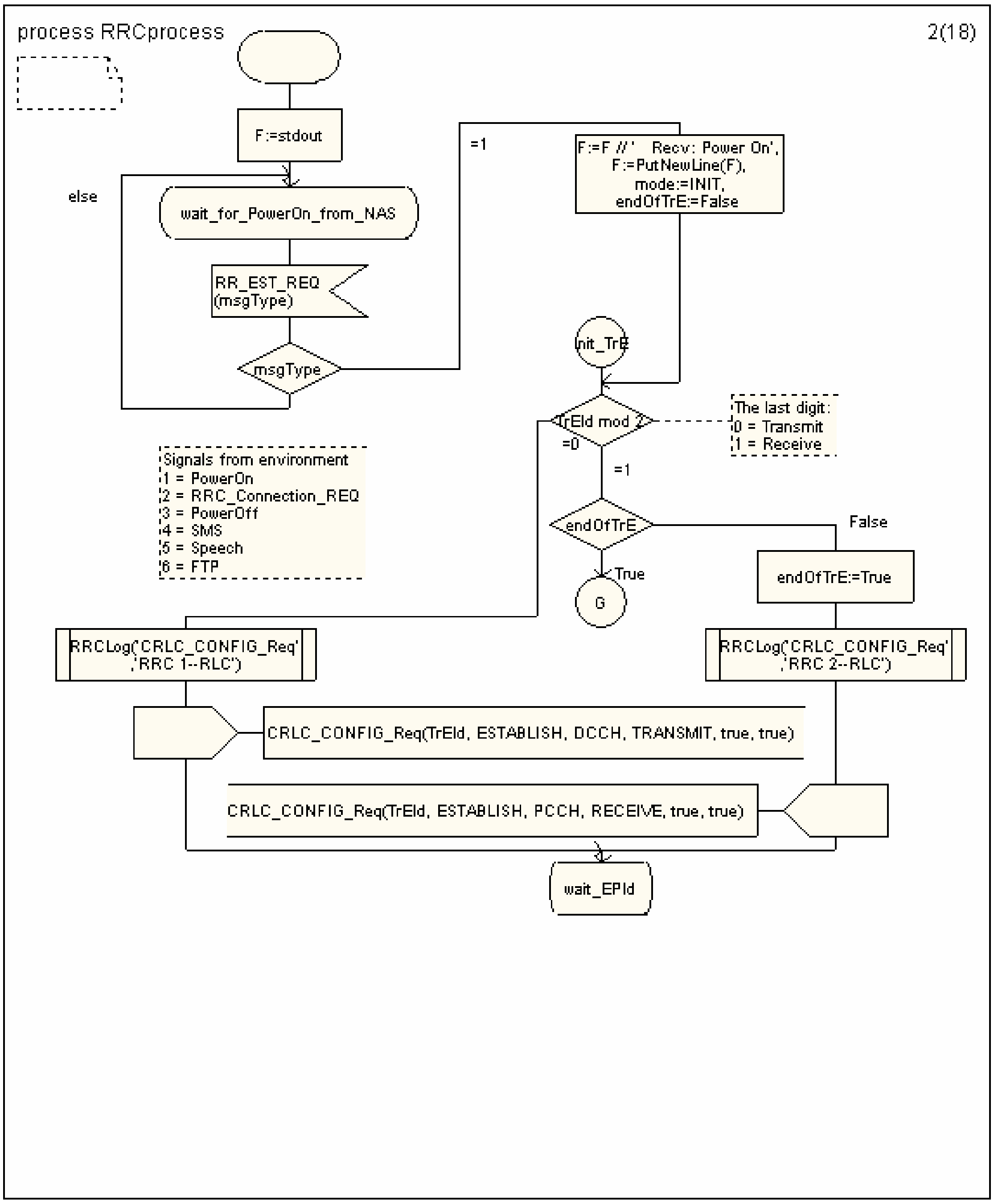


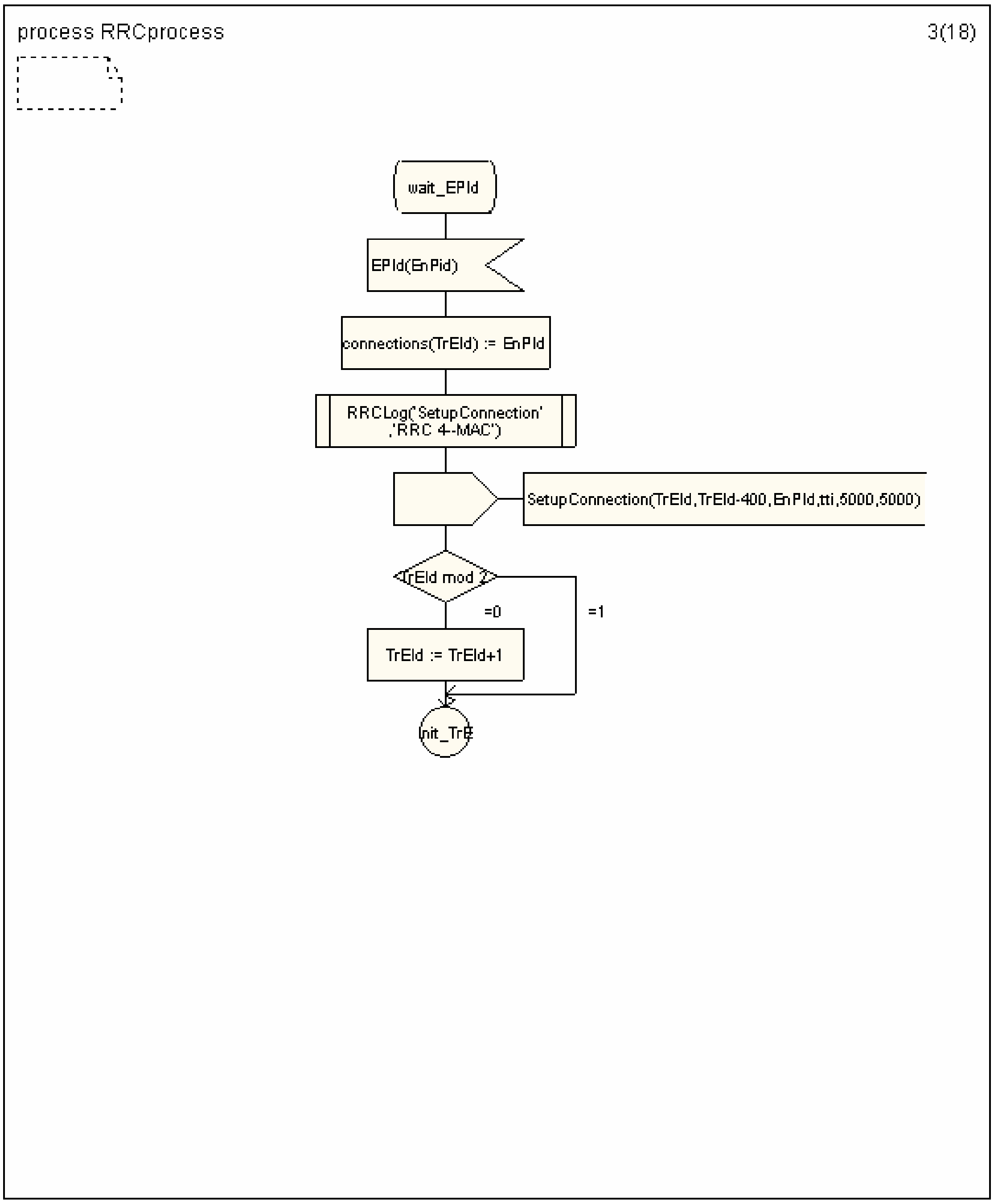
process RRCprocess 1(18)

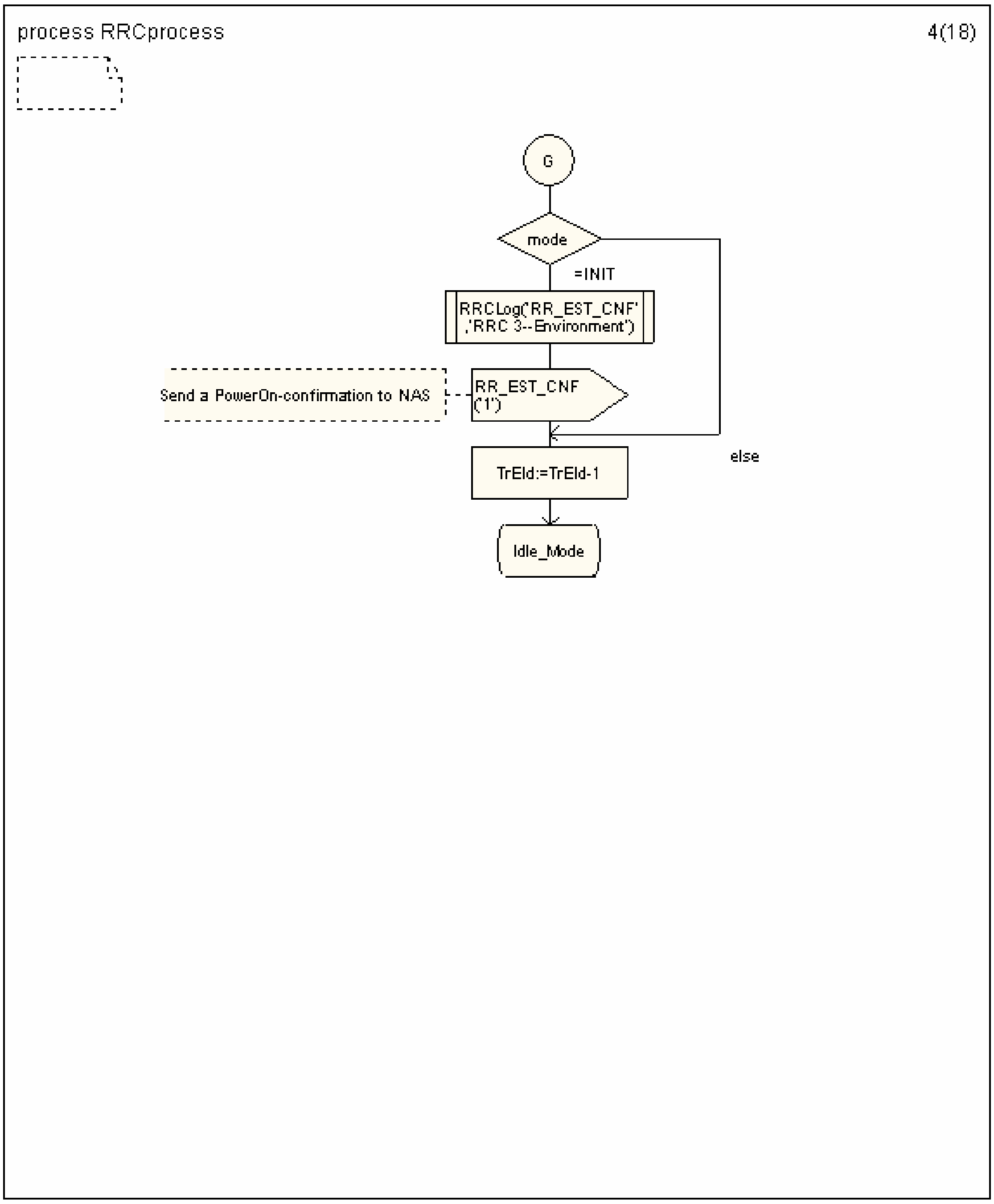
```
DCL
msgIn, tempMsgIn,
buf Charstring,

F TextFile,
result, encode_value, nbrOfBits, msgType Integer,

bitfoo, tempBitfoo, bitfooOut Bit_string,
mode StatusMode, endOfTrE Boolean,
TFTPConnectionActive Boolean:=False,
SendingTFTPData Boolean:=False,
TFTPShutdown Boolean:=False,

tempString Charstring,
encode_buffer Octet_string,
atimerDiscard timerDiscard,
aUL_CCCH_Message UL_CCCH_Message,

TrEId Integer := ID_START_NR_UE,
pdu PDUType,
TempPId, EnPId, ShouldBeId PId,
connections ConnectionsArrayType,
tti Duration :=0.1,
reconnect Duration :=15,
tempTrEId,i Integer;
```

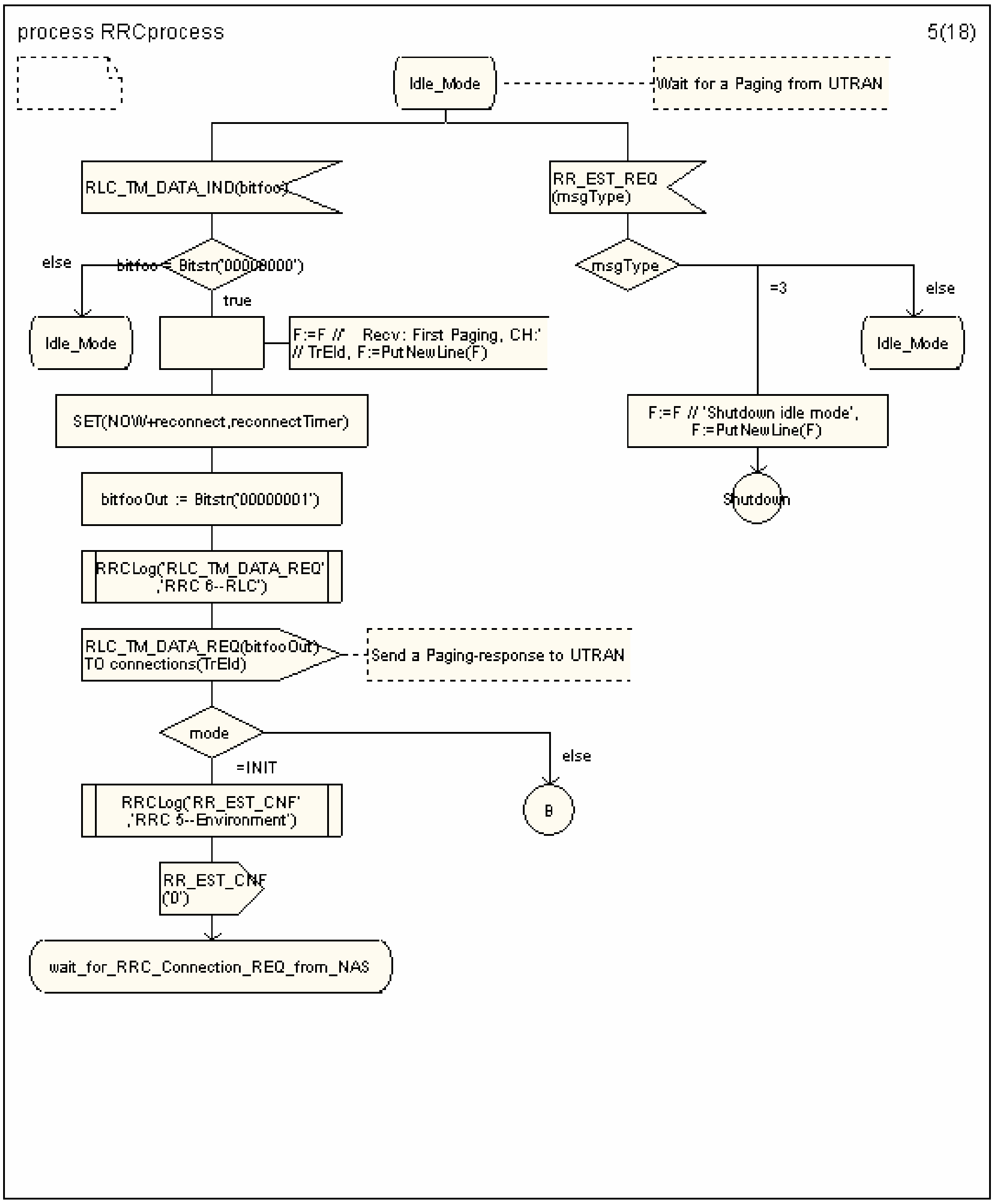
```
SYNTYPE EntityId = Integer
ENDSYNTYPE EntityId;

SYNTYPE EntityPId = PId
ENDSYNTYPE EntityPId;

NEWTYPE ConnectionsArrayType
  Array(Entityid, EntityPId)
ENDNEWTYPE  ConnectionsArrayType;
```

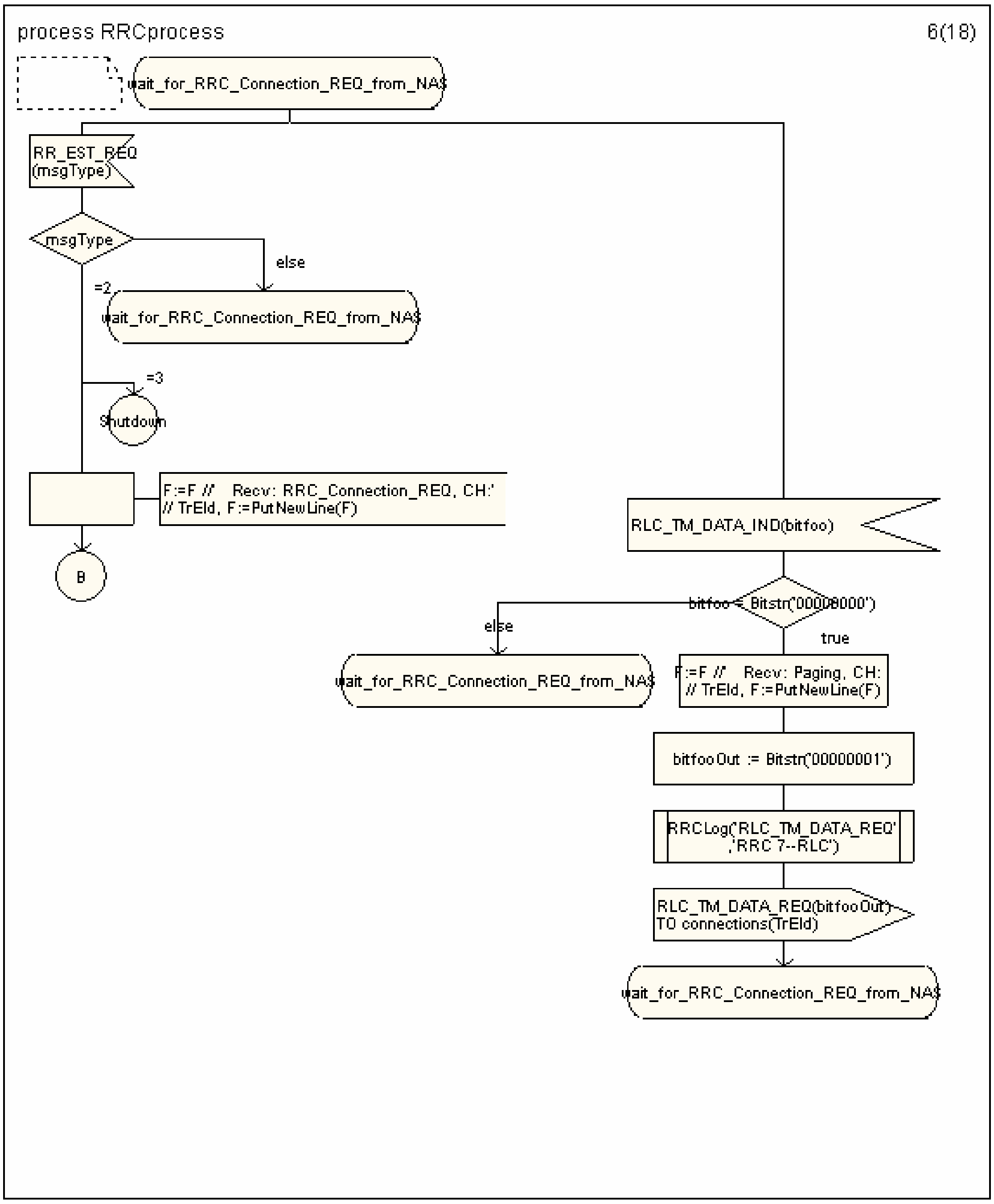
```
TIMER
reconnectTimer,
aTimer,tempTimer;
```

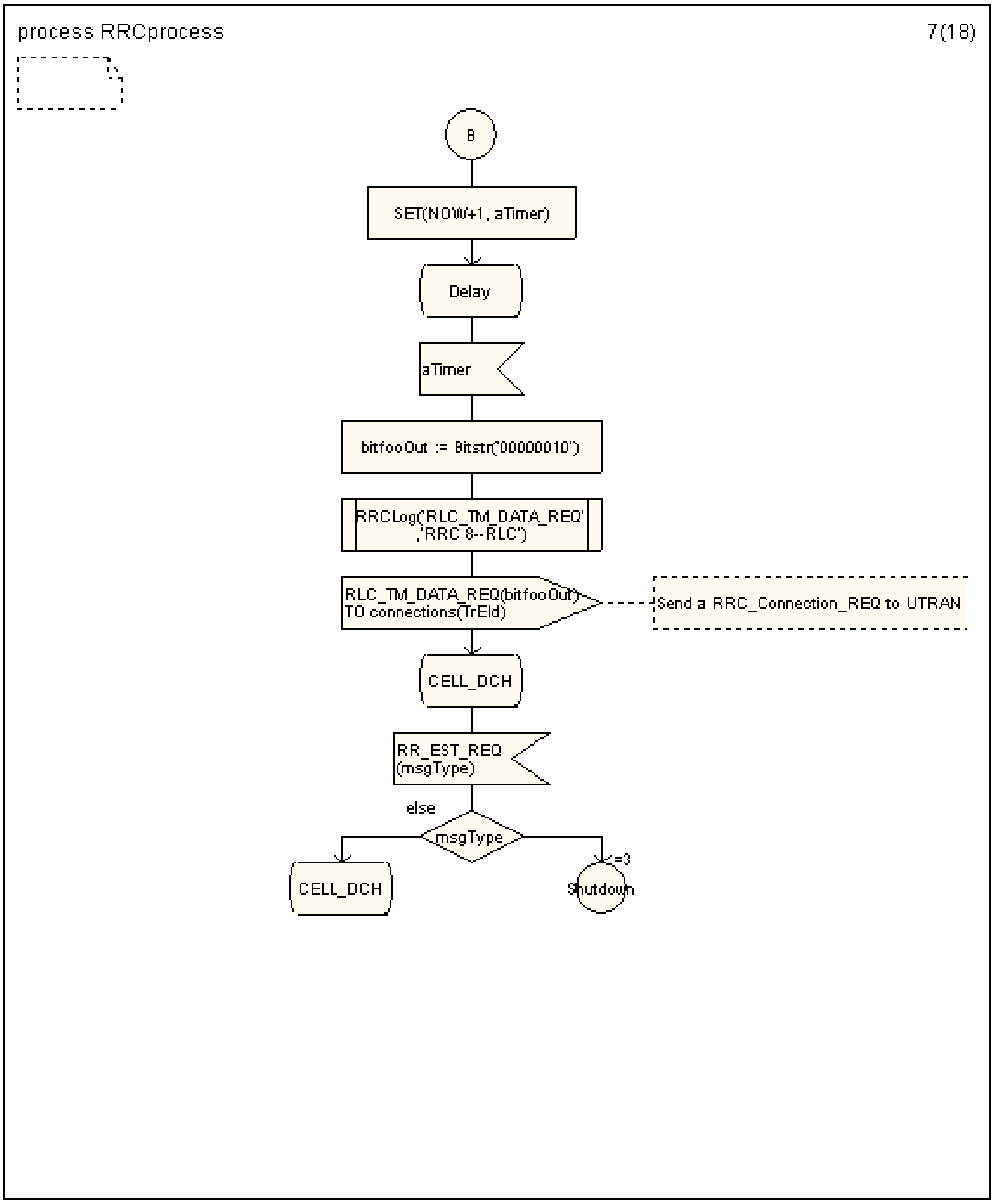

process RRCprocess
2(18)
F:=stdout
else
wait_for_PowerOn_from_NAS
RR_EST_REQ (msgType)
msgType
=1
F:=F // '   Recv: Power On',
F:=PutNewLine(F),
mode:=INIT,
endOfTrE:=False
Init_TrE
TrEId mod 2
The last digit:
0 = Transmit
1 = Receive
=0
=1
Signals from environment
1 = PowerOn
2 = RRC_Connection_REQ
3 = PowerOff
4 = SMS
5 = Speech
6 = FTP
endOfTrE
False
True
G
endOfTrE:=True
RRCLog('CRLC_CONFIG_Req', 'RRC 1--RLC')
RRCLog('CRLC_CONFIG_Req', 'RRC 2--RLC')
CRLC_CONFIG_Req(TrEId, ESTABLISH, DCCH, TRANSMIT, true, true)
CRLC_CONFIG_Req(TrEId, ESTABLISH, PCCH, RECEIVE, true, true)
wait_EPId

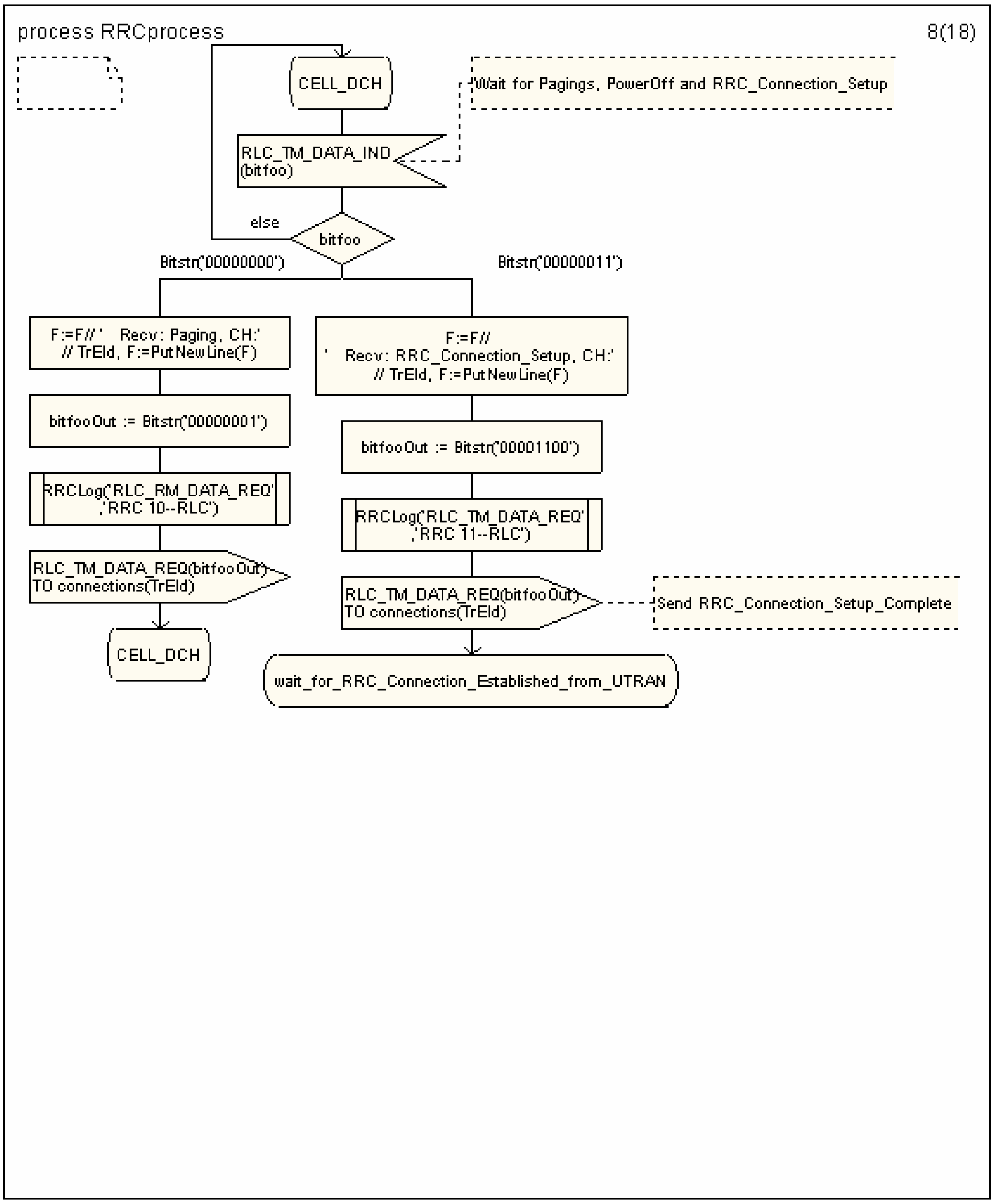

process RRCprocess
3(18)
wait_EPId
EPId(EnPid)
connections(TrEId) := EnPId
RRCLog('SetupConnection' ,'RRC 4-MAC')
SetupConnection(TrEId,TrEId-400,EnPId,tti,5000,5000)
TrEId mod 2
=0
=1
TrEId := TrEId+1
init_TrE

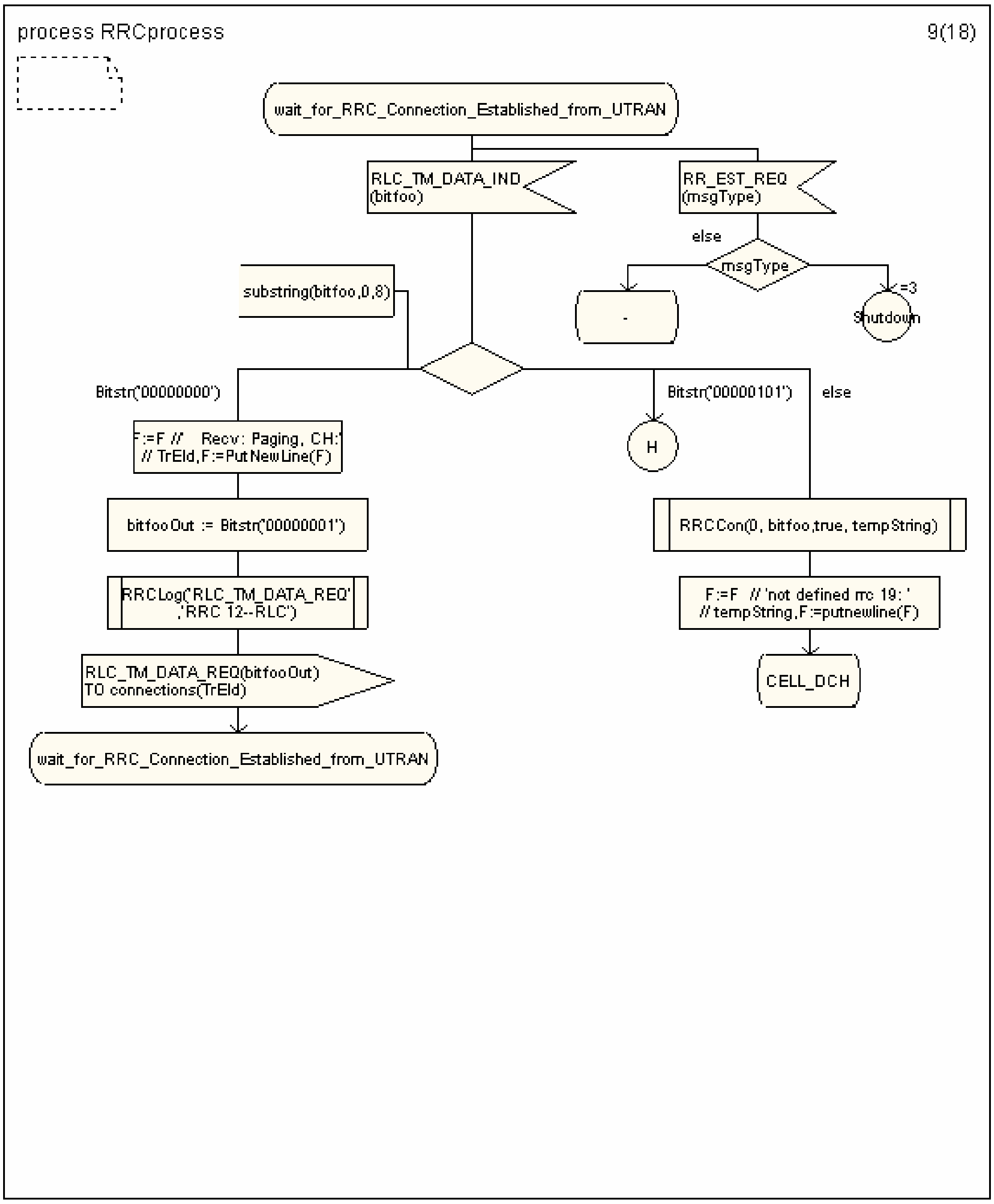

process RRCprocess
4(18)
G
mode
=INIT
RRCLog('RR_EST_CNF' ,'RRC 3--Environment')
Send a PowerOn-confirmation to NAS
RR_EST_CNF ('1')
else
TrEId:=TrEId-1
Idle_Mode

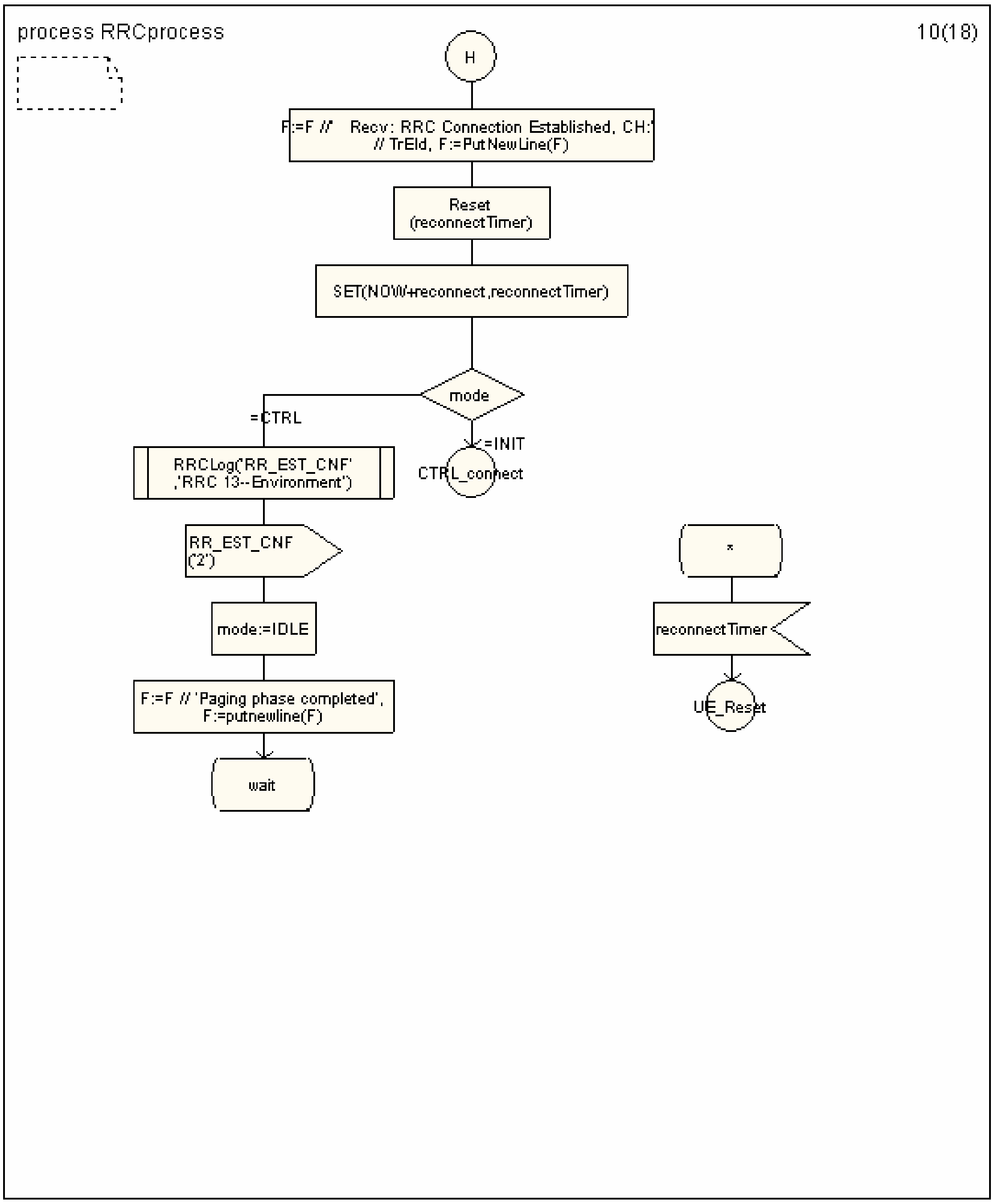

process RRCprocess
5(18)
Idle_Mode
Wait for a Paging from UTRAN
RLC_TM_DATA_IND(bitfoo)
RR_EST_REQ (msgType)
else
bitfoo = Bitstr('00008000')
true
msgType
=3
else
Idle_Mode
F:=F // ' Recv: First Paging, CH:' // TrEId, F:=PutNewLine(F)
Idle_Mode
SET(NOW+reconnect,reconnectTimer)
F:=F // 'Shutdown idle mode', F:=PutNewLine(F)
bitfooOut := Bitstr('00000001')
Shutdown
RRCLog('RLC_TM_DATA_REQ' ,'RRC 6--RLC')
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEId)
Send a Paging-response to UTRAN
mode
=INIT
else
RRCLog('RR_EST_CNF' ,'RRC 5--Environment')
B
RR_EST_CNF ('0')
wait_for_RRC_Connection_REQ_from_NAS

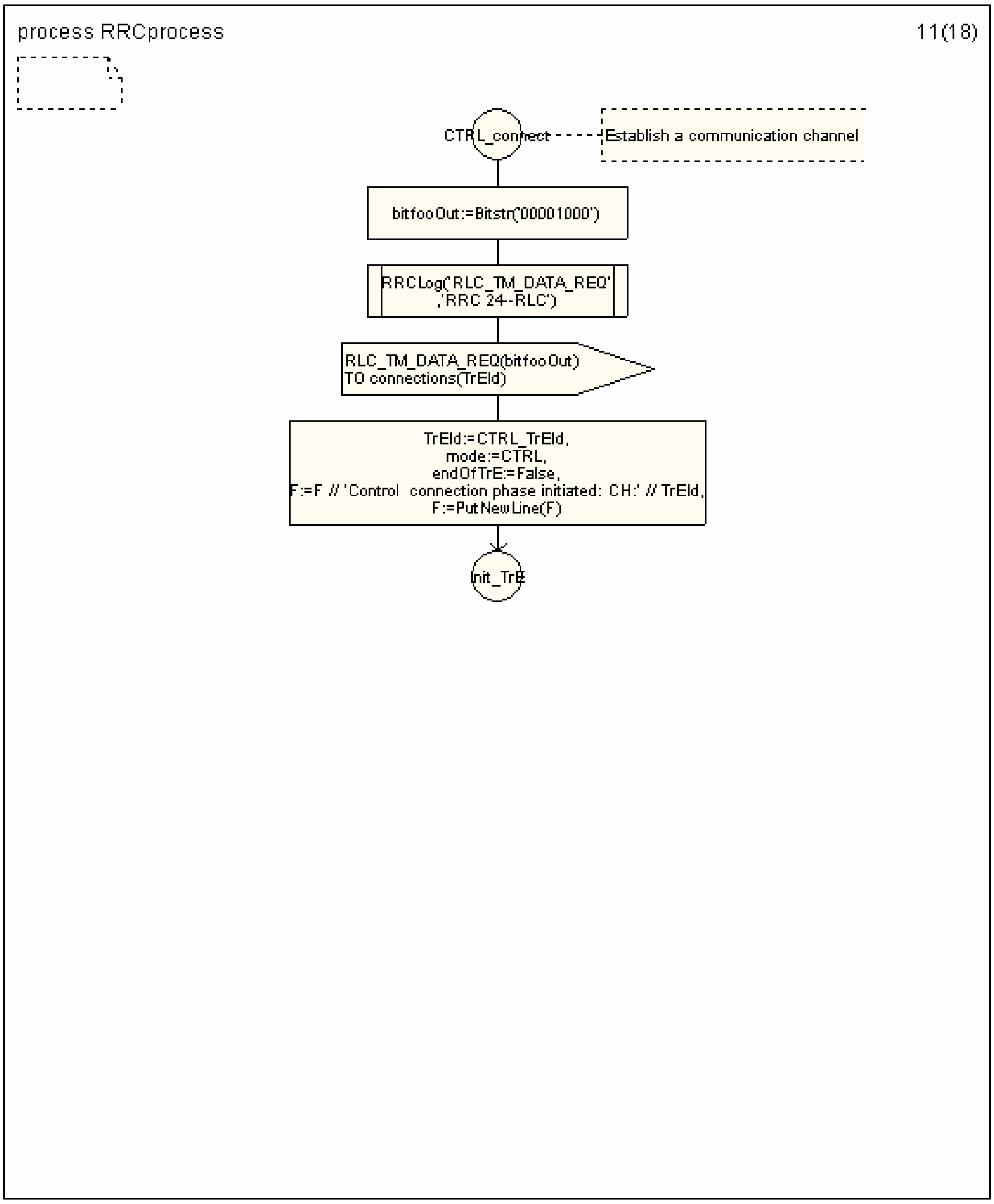

process RRCprocess
6(18)
wait_for_RRC_Connection_REQ_from_NAS
RR_EST_REQ (msgType)
msgType
else
=2
wait_for_RRC_Connection_REQ_from_NAS
=3
Shutdown
F:=F // ' Recv: RRC_Connection_REQ, CH:' // TrEId, F:=PutNewLine(F)
B
RLC_TM_DATA_IND(bitfoo)
bitfoo = Bitstr('00008000')
else
true
wait_for_RRC_Connection_REQ_from_NAS
F:=F // ' Recv: Paging, CH:' // TrEId, F:=PutNewLine(F)
bitfooOut := Bitstr('00000001')
RRCLog('RLC_TM_DATA_REQ' ,'RRC 7--RLC')
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEId)
wait_for_RRC_Connection_REQ_from_NAS

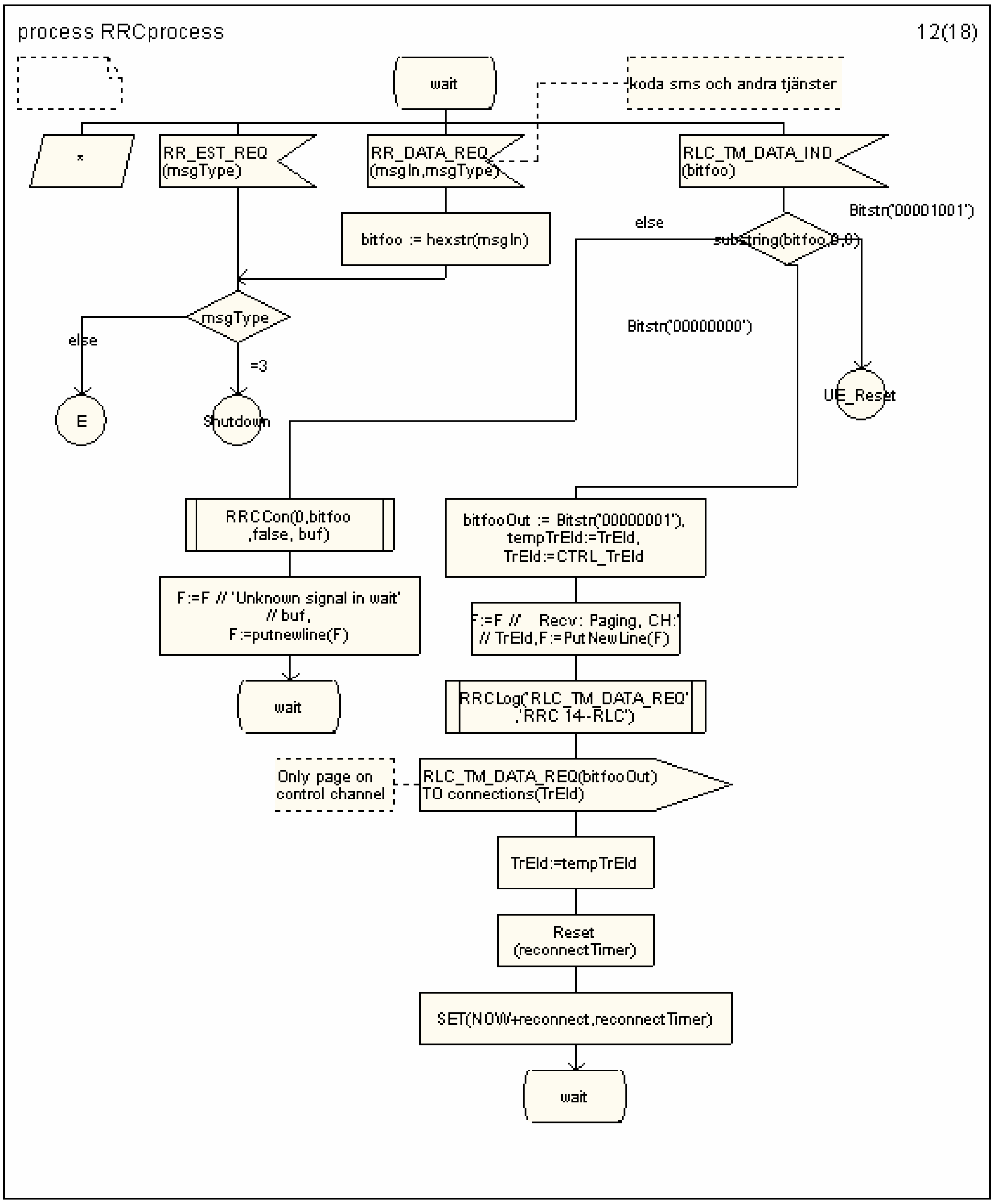

process RRCprocess
7(18)
B
SET(NOW+1, aTimer)
Delay
aTimer
bitfooOut := Bitstr('00000010')
RRCLog('RLC_TM_DATA_REQ' ,'RRC 8--RLC')
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEId)
Send a RRC_Connection_REQ to UTRAN
CELL_DCH
RR_EST_REQ (msgType)
else
msgType
=3
CELL_DCH
Shutdown

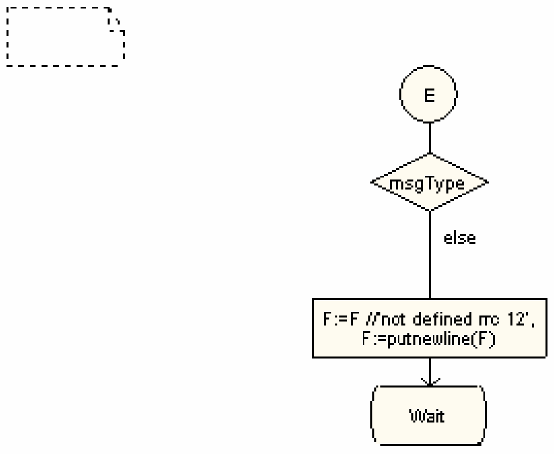

process RRCprocess
8(18)
CELL_DCH
Wait for Pagings, PowerOff and RRC_Connection_Setup
RLC_TM_DATA_IND (bitfoo)
else
bitfoo
Bitstr('00000000')
Bitstr('00000011')
F:=F// ' Recv: Paging, CH:' // TrEld, F:=PutNewLine(F)
F:=F// ' Recv: RRC_Connection_Setup, CH:' // TrEld, F:=PutNewLine(F)
bitfooOut := Bitstr('00000001')
bitfooOut := Bitstr('00001100')
RRCLog('RLC_RM_DATA_REQ' ,'RRC 10--RLC')
RRCLog('RLC_TM_DATA_REQ' ,'RRC 11--RLC')
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEld)
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEld)
Send RRC_Connection_Setup_Complete
CELL_DCH
wait_for_RRC_Connection_Established_from_UTRAN

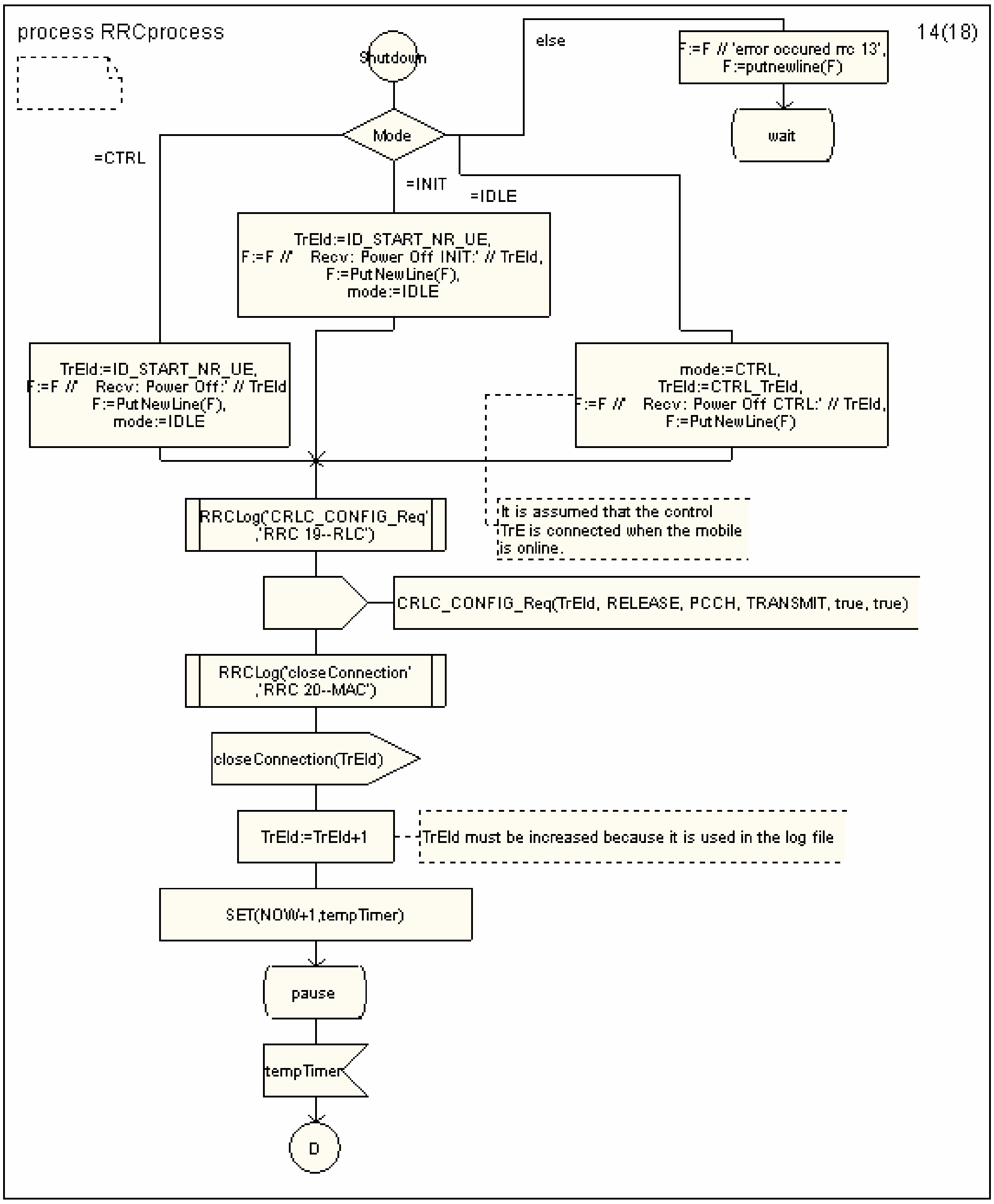

process RRCprocess
9(18)
wait_for_RRC_Connection_Established_from_UTRAN
RLC_TM_DATA_IND
(bitfoo)
RR_EST_REQ
(msgType)
else
msgType
-
=3
Shutdown
substring(bitfoo,0,8)
Bitstr('00000000')
Bitstr('00000101')
else
H
F:=F // 'Recv: Paging, CH:'
// TrEId,F:=PutNewLine(F)
bitfooOut := Bitstr('00000001')
RRCLog('RLC_TM_DATA_REQ'
,'RRC 12--RLC')
RLC_TM_DATA_REQ(bitfooOut)
TO connections(TrEId)
wait_for_RRC_Connection_Established_from_UTRAN
RRCCon(0, bitfoo,true, tempString)
F:=F // 'not defined rrc 19: '
// tempString,F:=putnewline(F)
CELL_DCH

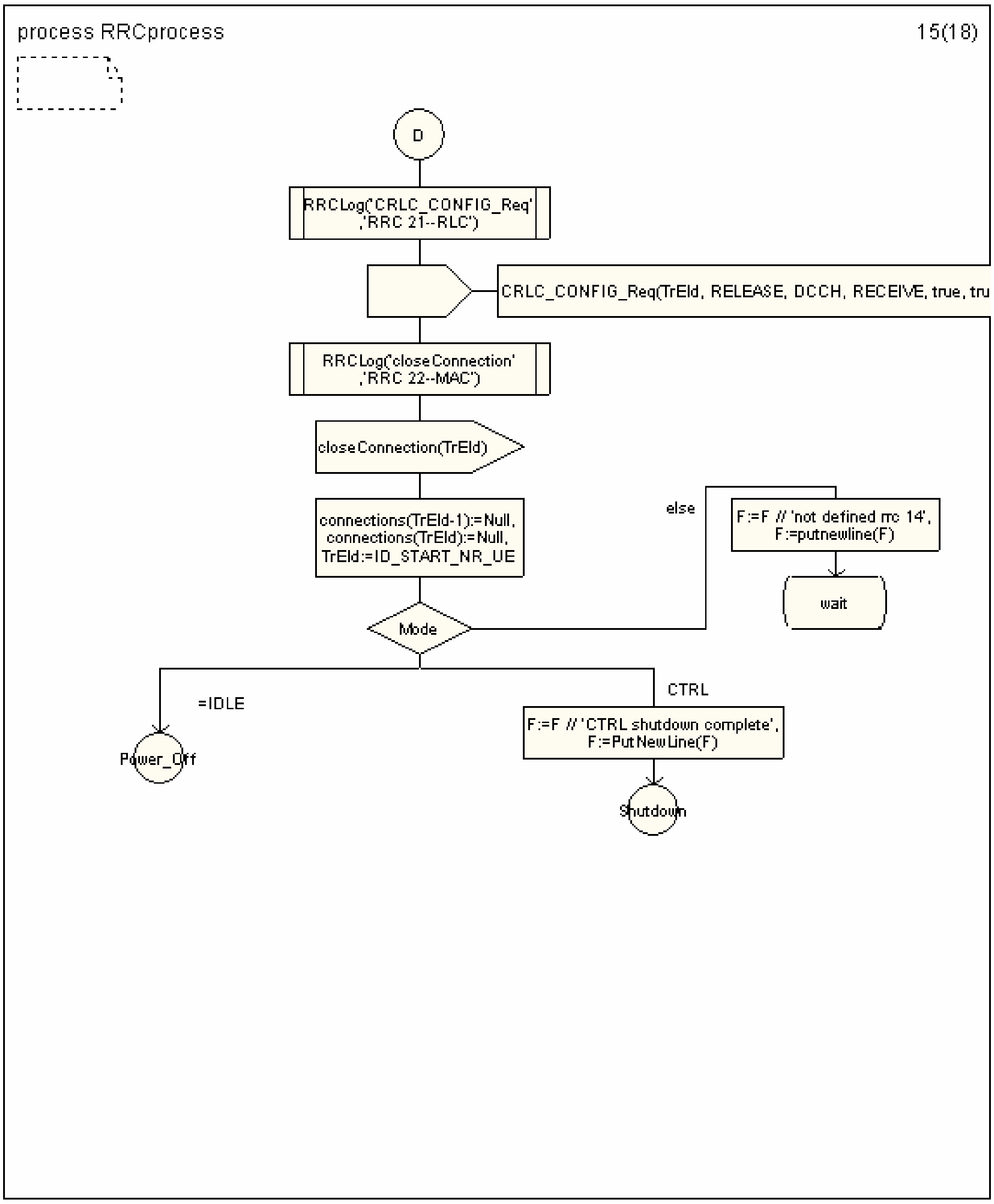

process RRCprocess
10(18)
H
F:=F // 'Recv: RRC Connection Established, CH:' // TrEId, F:=PutNewLine(F)
Reset (reconnectTimer)
SET(NOW+reconnect,reconnectTimer)
mode
=CTRL
=INIT
CTRL_connect
RRCLog('RR_EST_CNF' ,'RRC 13--Environment')
RR_EST_CNF ('2')
mode:=IDLE
F:=F // 'Paging phase completed', F:=putnewline(F)
wait
*
reconnectTimer
UE_Reset


process RRCprocess
11(18)
CTRL_connect
Establish a communication channel
bitfooOut:=Bitstr('00001000')
RRCLog('RLC_TM_DATA_REQ' ,'RRC 24--RLC')
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEId)
TrEId:=CTRL_TrEId,
mode:=CTRL,
endOfTrE:=False,
F:=F // 'Control connection phase initiated: CH:' // TrEId,
F:=PutNewLine(F)
Init_TrE

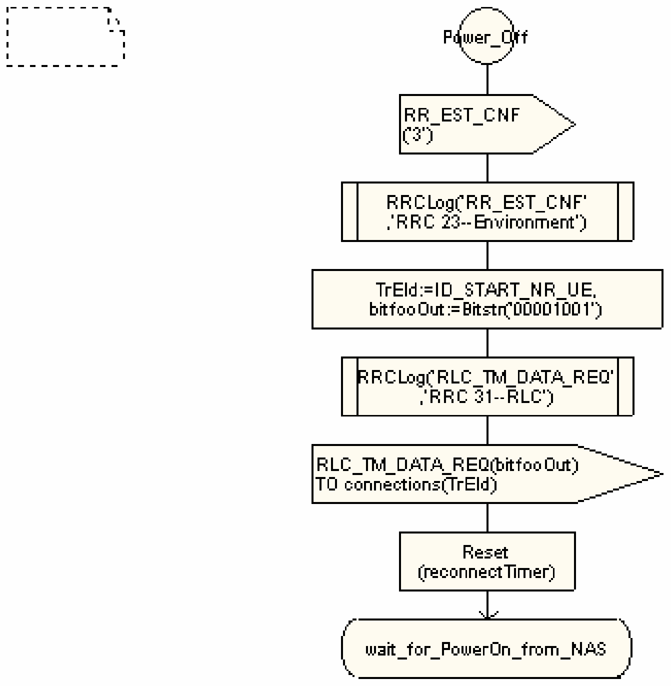

process RRCprocess
12(18)
wait
koda sms och andra tjänster
*
RR_EST_REQ (msgType)
RR_DATA_REQ (msgIn,msgType)
RLC_TM_DATA_IND (bitfoo)
bitfoo := hexstr(msgIn)
else
Bitstr('00001001')
substring(bitfoo,0,0)
msgType
else
=3
E
Shutdown
Bitstr('00000000')
UE_Reset
RRCCon(0,bitfoo ,false, buf)
F:=F // 'Unknown signal in wait' // buf, F:=putnewline(F)
wait
bitfooOut := Bitstr('00000001'), tempTrEId:=TrEId, TrEId:=CTRL_TrEId
F:=F // 'Recv: Paging, CH:' // TrEId,F:=PutNewLine(F)
RRCLog('RLC_TM_DATA_REQ' ,'RRC 14-RLC')
Only page on control channel
RLC_TM_DATA_REQ(bitfooOut) TO connections(TrEId)
TrEId:=tempTrEId
Reset (reconnectTimer)
SET(NOW+reconnect,reconnectTimer)
wait

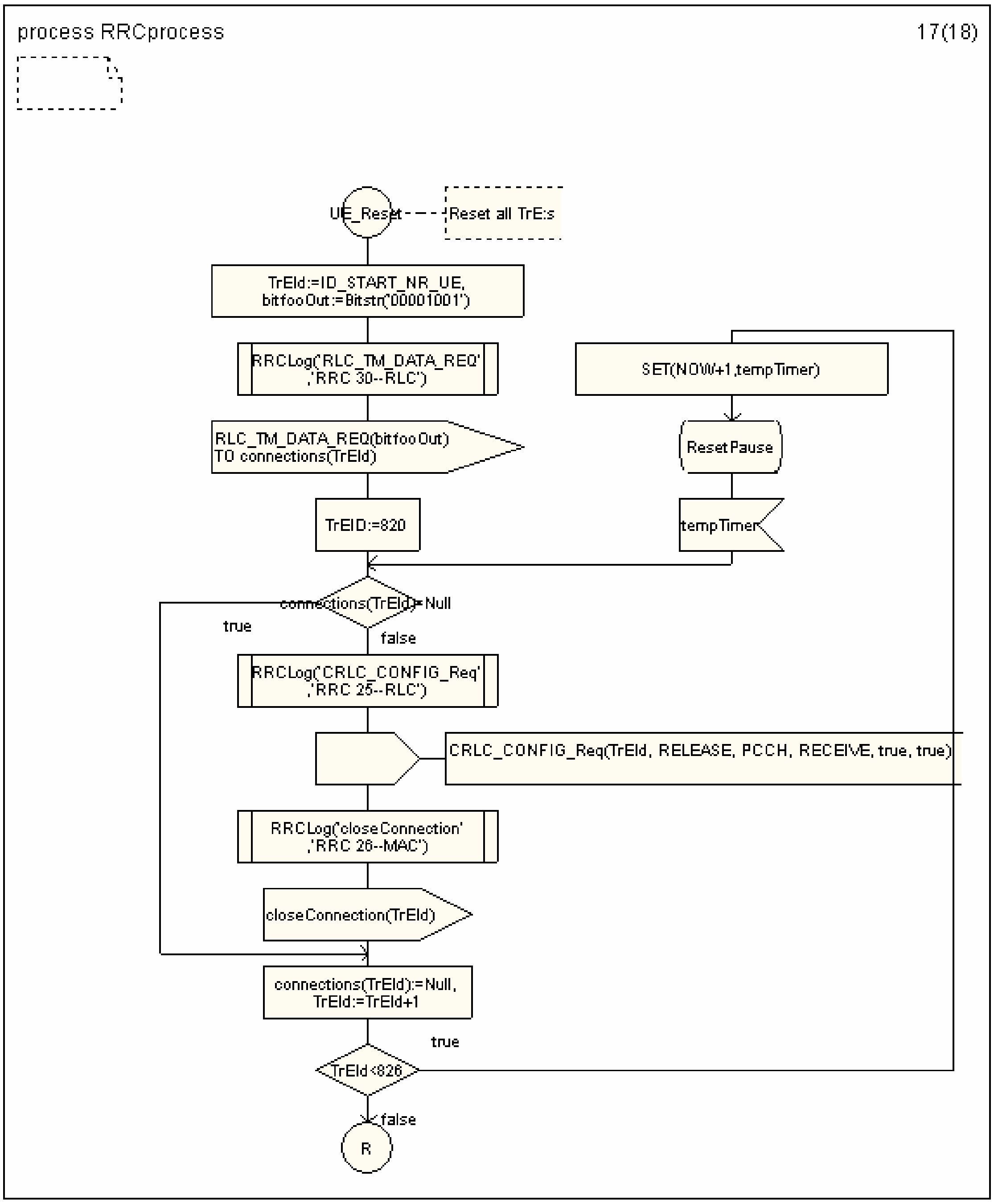

E
msgType
else
F:=F //'not defined rrc 12',
F:=putnewline(F)
Wait

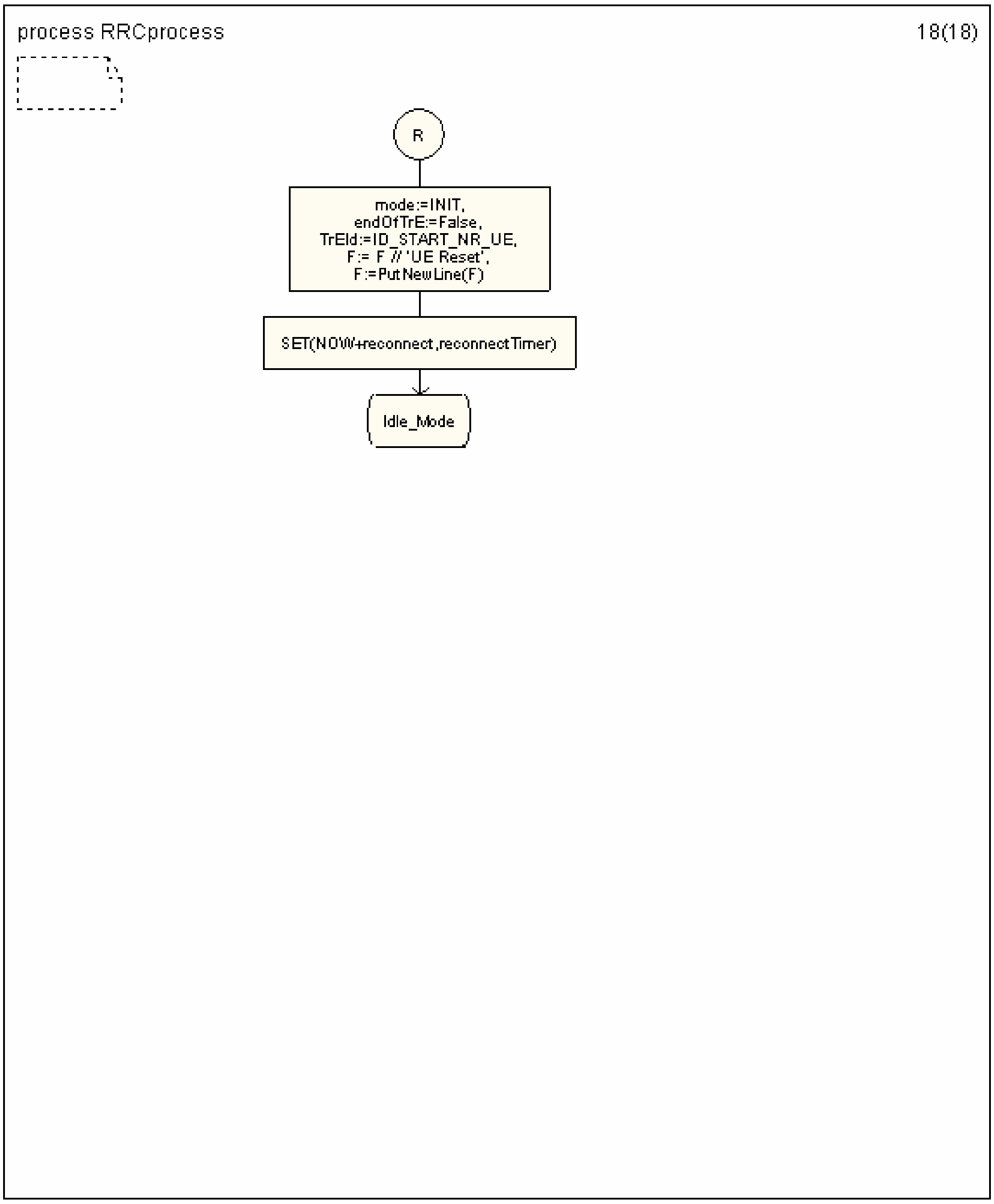

process RRCprocess
14(18)
Shutdown
Mode
=CTRL
=INIT
=IDLE
else
F:=F // 'error occured rrc 13',
F:=putnewline(F)
wait
TrEId:=ID_START_NR_UE,
F:=F // ' Recv: Power Off INIT:' // TrEId,
F:=PutNewLine(F),
mode:=IDLE
TrEId:=ID_START_NR_UE,
F:=F // ' Recv: Power Off:' // TrEId
F:=PutNewLine(F),
mode:=IDLE
mode:=CTRL,
TrEId:=CTRL_TrEId,
F:=F // ' Recv: Power Off CTRL:' // TrEId,
F:=PutNewLine(F)
It is assumed that the control
TrE is connected when the mobile
is online.
RRCLog('CRLC_CONFIG_Req'
,'RRC 19--RLC')
CRLC_CONFIG_Req(TrEId, RELEASE, PCCH, TRANSMIT, true, true)
RRCLog('closeConnection'
,'RRC 20--MAC')
closeConnection(TrEId)
TrEId:=TrEId+1
TrEId must be increased because it is used in the log file
SET(NOW+1,tempTimer)
pause
tempTimer
D


D
RRCLog('CRLC_CONFIG_Req' ,'RRC 21--RLC')
CRLC_CONFIG_Req(TrEId, RELEASE, DCCH, RECEIVE, true, tru
RRCLog('closeConnection' ,'RRC 22--MAC')
closeConnection(TrEId)
connections(TrEId-1):=Null, connections(TrEId):=Null, TrEId:=ID_START_NR_UE
Mode
else
F:=F // 'not defined rrc 14', F:=putnewline(F)
wait
=IDLE
Power_Off
CTRL
F:=F // 'CTRL shutdown complete', F:=PutNewLine(F)
Shutdown

process RRCprocess




process RRCprocess
17(18)
UE_Reset
Reset all TrE:s
TrEId:=ID_START_NR_UE,
bitfooOut:=Bitstr('00001001')
RRCLog('RLC_TM_DATA_REQ'
,'RRC 30--RLC')
RLC_TM_DATA_REQ(bitfooOut)
TO connections(TrEId)
TrEID:=820
SET(NOW+1,tempTimer)
ResetPause
tempTimer
connections(TrEId)=Null
true
false
RRCLog('CRLC_CONFIG_Req'
,'RRC 25--RLC')
CRLC_CONFIG_Req(TrEId, RELEASE, PCCH, RECEIVE, true, true)
RRCLog('closeConnection'
,'RRC 26--MAC')
closeConnection(TrEId)
connections(TrEId):=Null,
TrEId:=TrEId+1
TrEId<826
true
false
R


process RRCprocess
18(18)
R
mode:=INIT,
endOfTrE:=False,
TrEId:=ID_START_NR_UE,
F:= F // 'UE Reset',
F:=PutNewLine(F)
SET(NOW+reconnect,reconnectTimer)
Idle_Mode