Abstract

This document contains a preliminary, incomplete, draft for the MAC State Diagrams. It was posted on the FTP server in the beginning of January 1997 and is superseded by doc.: IEEE P802.11-97/001.
System Station

MAC_Data_Service

MAC_SAP

MAC_Management_Service

Station_MLME

MMGT

Includes request validation and add/remove MAC headers.

Includes power save queuing, fragmentation, encryption, and Tx/Rx timing.

Includes DCF, DCF-C很少Ack handling, retries, CF-poll response, CF-ack detect, Atn handling, and reassembly.

Includes backoff and timestamp insertion.

MAC_Control

PDU_Generation

PDU

Pdurequest

MCTL

MMTX

Includes filtering of MLME requests and MIB access. MIB attributes are remote variables, exported by their source processes.

Includes scan, join, beacon/dwell and awake/doze timing, (re/dis)associate, (de)authenticate, start IBSS, and monitor of station & power save state.

TX

TxConfirm, TxDone

PHY_SAP_TX

PHY_TXRequestSignals

PHY_TXConfirmSignals

RX

RxConfirm, RxDone, RxFailed, RxMed

PHY_SAP_RX

PHY_RXRequestSignals

RXIndicate, NeedAck, RxCFack

RXIndicate

contains validate, decrypt, address & duplicate filter, channel state (physical and virtual carrier sense), and IFS & slot timing.
System Station

```
signal Busy,
  ChangeNav(Time,Duration,NavSrc),
  DoBackoff(Natural,Natural),
  Idx(Octetstring,MacAddr,MacAddr),
  MlmeUnitdata.indication(MacAddr,MacAddr),
  Routing(Octetstring,RxStatus,CIPriority,ServiceClass),
  MlmeUnitdata.request(MacAddr,MacAddr),
  Routing(Octetstring,CIPriority,ServiceClass),
  MlminiData.indication(MacAddr),
  MlminiData.indication(MacAddr,MacAddr,MacAddr, TxStatus,CIPriority,ServiceClass),
  MlmeAssociate.confirm(Success),
  MlmeAssociate.indicate(MacAddr,MacAddr), AP only */
  MlmeAssociate.request(MacAddr,MacAddr),
  MlmeAuthenticate.confirm(MacAddr,AuthType,Success),
  MlmeAuthenticate.indicate(MacAddr,AuthType),
  MlmeDiscover.confirm(MacAddr,Success),
  MlmeDiscover.indicate(MacAddr),
  MlmeDiscover.request(MacAddr),
  MlmeGet.confirm(MibStatus,MibAttrib,MibValue),
  MlmeGet.request(MacAddr),
  MlmeJoin.confirm(Success),
  MlmeJoin.indicate(Success),
  MlmePowermgmt.confirm(Success),
  MlmePowermgmt.request(PwrSave,Natural),
  MlmeReassociate.confirm(Success),
  MlmeReassociate.indicate(MacAddr,AP only */
  MlmeReassociate.request(MacAddr,Natural),
  MlmeReset.confirm(Success),
  MlmeReset.indicate(Success),
  MlmeScan.confirm(BssDesc),
  MlmeScan.request(BssDesc,ScanType,MacAddr, Octetstring,ScanType),
  MlmeSet.confirm(MibStatus,MibAttrib),
  MlmeSet.request(MibAttrib,MibValue),
  MlmeStart.confirm(Success),
  MlmeStart.indicate(Octetstring,BootTime, Natural,CIFillme),
```
System Station

use macsorts
use macomb;

signallist
MlmeRequestSignals=
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeJoin.request,
MlmePowermgmt.request,
MlmeReassociate.request,
MlmeReset.request,
MlmeScan.request,
MlmeSet.request,
MlmeStart.request;

MlmeConfirmSignals=
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeJoin.confirm,
MlmePowermgmt.confirm,
MlmeReassociate.confirm,
MlmeReset.confirm,
MlmeScan.confirm,
MlmeSet.confirm,
MlmeStart.confirm;

signallist
SmRequestSignals=
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeJoin.request,
MlmeReassociate.request,
MlmeScan.request,
MlmeStart.request;

SmConfirmSignals=
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeJoin.confirm,
MlmeReassociate.confirm,
MlmeScan.confirm,
MlmeStart.confirm;

signallist
PhyTxRequestSignals=
PhyTxStart.request,
PhyTxEnd.request,
PhyData.request;

PhyTxConfirmSignals=
PhyTxStart.confirm,
PhyTxEnd.confirm,
PhyData.confirm;

signallist
PlmeRequestSignals=
PlmeSet.request,
PlmeReset.request;

PlmeConfirmSignals=
PlmeGet.confirm,
PlmeSet.confirm,
PlmeReset.confirm;

signallist
MlmeIndicateSignals=
MlmeAuthenticate.indicate,
MlmeDeauthenticate.indicate,
MlmeDisassociate.indicate;
/* The signals named below are
only generated at APs: 
MlmeAssociate.indicate,
MlmeReassociate.indicate */

signallist
SmIndicateSignals=
MlmeAuthenticate.indicate,
MlmeDeauthenticate.indicate,
MlmeDisassociate.indicate;
/* The signals named below are
only generated at APs: 
MlmeAssociate.indicate,
MlmeReassociate.indicate */

signallist
PhyRxSignals=
PhyRxStart.indicate,
PhyRxEnd.indicate,
PhyData.indicate,
PhyCca.indicate,
PhyCcast.confirm;
Process MSDU_to_LLC

1. Remove MAC header from beginning of MSDU to obtain the LLC data octet string:

- Reception status always successful because any error would prevent the Msdulndicate from reaching this process.

2. To_LLC

3. Msdulndicate (sdru, period)

4. da:= add1(sdu)

5. sa:= if frDe(sdu)=1 then add2(sdu) else add2(sdu) fi

6. serv:= if orderBit(sdu)>1 then strictlyOrdered else reorderable fi

7. LLCdata:= substr (sdu, sMacHdrLng, length(sdu)-sMacHdrLng)

8. MaUnitdata:= indication(sa, da, null, rt, LLCdata, successful_rx, period, serv)

9. del LLCdata Octetstring

del period CIPriority

del sa, da MacAddr

del sdu Frame

del serv ServiceClass
Process MSDU_from_LL

TxResult reports reasons for failure, including noBss, noAuth, noAssoc, retryLimit, txLifeTimeout. However, the only status code for Tx failure in 6.2.1.3 is 'undeliverable'.

route, LLCdata, period, serv

status = unsupported ServiceClass

status = unsupported Priority

status = excessive DataLength

make msdu

if BSS has no point coordinator, inform LLC, change priority to contention, and send MSDU

MsgSvcConfi rm (sdu, period, result)

status = if result is successful

period = contention

import (mPsm)

(sda_active)

import (mPoAvail)

(reorderable)

(strictly Ordered)

reject

MaUnidata status indication

malUnidata request (sa, da, serv)

null rt

else

serv

(type:Octetstring length (LLCdata))

make msdu

(serv, period, serv)

import (mBssld)

LLCdata

import(mBssld).

Build frame with 24-octet MAC header and LLCdata:

type := data

addr1 := da

addr2 := aMacAddress

(addr3(sdu) = other header fields) := 0

import(mBssld), LLCdata

sdu := msFrame (sda, da, serv)

else

setOrderBit (sdu, 1)

MsgSvcRequest (sdu, period)

addr1 (sdu)

addr2 (sdu)

addr3 (sdu)
This block is a summary of MIB access and update. MIB attributes are defined in ASN.1 code.

MIB(1,1)

MIMEGet(1,1), MIMESet(1,1)

ResetMAC

GetSet

MIMEGet.request, MIMESet.request

ReqConf

MIMEAssociate.request, MIMEAuthenticate.request, MIMEDeauthenticate.request, MIMEDisassociate.request, MIMEJoin.request, MIMEReassociate.request, MIMEReset.request, MIMEScan.request, MIMEStart.request

SMT_Requests (1,1)

MIMEAssociate.confirm, MIMEAuthenticate.confirm, MIMEDeauthenticate.confirm, MIMEDisassociate.confirm, MIMEJoin.confirm, MIMEReassociate.confirm, MIMEReset.confirm, MIMEScan.confirm, MIMEStart.confirm

SMT_Indications (1,1)

MIMEAssociate.indicate, MIMEAuthenticate.indicate, MIMEDeauthenticate.indicate, MIMEDisassociate.indicate, MIMEReassociate.indicate

FromMgt

mgMg

ReqConf

MIMEAssociate.request, MIMEAuthenticate.request, MIMEDeauthenticate.request, MIMEDisassociate.request, MIMEJoin.request, MIMEReassociate.request, MIMEScan.request, MIMEStart.request, ResetMAC

This block handles requests sequentially. Start, join, powermgmt, scan, redis/associate and deauthenticate must be sequential. Authenticate allows the possibility of multiple requests outstanding. To do this, MIME_Req_Resp would need to cache challenge text and match responses to outstanding requests.

MIMEAssociate.confirm, MIMEAuthenticate.confirm, MIMEDeauthenticate.confirm, MIMEDisassociate.confirm, MIMEJoin.confirm, MIMEReassociate.confirm, MIMEReset.confirm, MIMEScan.confirm, MIMEStart.confirm

SM_MLME_SAP
Process MIB

imported \{Read-Only\} MIB attribute values exported from other processes
Import of \{Read-Only\} MIB attribute values exported from other processes

imported aAckFailureCount, aFailedCount,
aFcsErrorCount, aFrameDuplicateCount,
aMulticastReceivedFrameCount,
aMultipleRetryCount, aRetryCount,
aReceivedFrameCount, aRtsFailureCount,
aRtsSuccessCount, aTransmittedFragmentCount,
exampleCounter::

Declarations of internal MAC variables
(updated from multiple sources using mSet)
dcl exported mBssld MacAddr := nullAddr,
mBssld Boolean := false,
mSsld Octetstring := null

Declarations of MIB attributes exported from this process
// Read-Write attributes
dcl exported aAuthenticationType Integer := 1,
aExcludeUnencrypted Boolean := false,
aFragmentationThreshold Integer := 2346,
aGroupAddresses MacAddrSet := empty,
aLongRetryLimit Integer := 4,
aMaxReceiveLifetime usec := 512,
aMaxTransmitMediaLifetime usec := 512,
aMediumOccupancyLimit usec := 100,
aPrivacyInvoked Boolean := false,
aReceiveDTIMs Boolean := true,
aRtsThreshold Integer := 3000,
aShortRetryLimit Integer := 7,
aWepDefault KeyIndex := 1,
aCurrentChannelNumber Integer,
aCurrentDwellTime usec := 390,
aCurrentSet Integer,
aCurrentPattern Integer,
aCurrentIndex Integer;
/* Read-Only attributes */
dcl exported aAuthenticationAlgorithm
AuthAlgSel := (open_system or shared_key);
/* Write-Only attributes */
dcl exported aDefaultWepKeys KeyVector;
dcl exported aWepKeyMapping KeyMapArray := (., nullAddr, false, null) ;
Process SMT_Requests

```plaintext
newtype SmtRqState
literals idle, bss, ibss, ap;
endnewtype SmtRqState;
```

```plaintext
MimeStart_request(sta, algo); 
MimeAuthenticate_confirm(sta); 
MimeDeauthenticate_request(sta); 
```
Process SMT_Requests

Wait for MAC management to process request.

- MmStart_ _confirm(ok)
- MmJoin_ _confirm(ok)
- MmAuthentiinate_confirm(sta, alg, ok)
- MmDeauthenicate_confirm(sta, alg, ok)
- MmScan_ _confirm(bss)

Save new (request) signals while awaiting response from SMT.

Return to the state prior to Wait_SM.

IDLE

泉State= idle

Import (mbss)

泉State= active

AP

IBSS

BSS

MmJoin_ _request(bss)

MmScan_ _request(typeSet)

MmScan_ _request(emptySet)

MmJoin_ _confirm(false)

MmScan_ _confirm(false)

Wait_SM

Reject Join and Scan at active AP.

- MmAssociate_confirm(ck)
- MmReassociate_confirm(ck)
- MmDisassociate_confirm(ck)

Approach allowed at AP and sta in bss.

Disassociate rejected if idle in bss.

泉State= ibss

泉State= bss

泉State= ap

泉State= idle

泉State= IBSS

泉State= AP

泉State= IDLE

泉State= IBSS
This state machine passes indications through, unmodified, from SMT to the MLME SAP. MimeAssociate.indicate and MimeReassociate.indicate are only generated by SMT at APs.
Block PDU_Generation

\[\text{signal:} \]
\[\text{FragConfirm(FragSdu,TxResult),} \]
\[\text{FragRequest(FragSdu):} \]

[Diagram]

- PDU
- PM_Filter (1,1)
- Mmsdu
- MmsduRequest
- MnRequest, SqResponse
- Mmpdu
- Mmpdu
- MnConfirm, SqInquiry
- PsInquiry
- PwrMgt
- PaResponse, PaChange
- PdlConfirm, Polled, Ready, ResetMAC
- Mmpdu
- PdlRequest
Process Prepare_MPDU

Mmpdu requests assumed to not need validation.

Min_ Request (sdu, pn)

requester := sender

Use Wep := if

Mmpdu Confirm (sdu, pn)

sdu := setAddr1 (sdu, may send all Msdus

Import(mBssld) , with toDs=1 when in infrastructure Bss.

Criteria are not specified for when to use toDs=0 in an infrastructure BSS.

The RC4 PRNG is accessed as an external procedure:
prnString := call RC4(key, length)

The placeholder for the RC4 remote procedure is in MIB. ,

imported procedure RC4;

fpar PrngKey, Integer;
returns Octetsting ;
Process Prepare_MPDU

1. Initialize FragSdu structure
   - fsdu/frag: = type: MacAddr, MacAddr: group (addr) (sdu),
   - mpa/ovhd: = sMaxHdrLrg + sCrcLrg + phmax = aMaxMPduLength,

2. The length of group-addressed frames is not checked because they cannot be fragmented. aMaxMPduLength must be >= sMaxMPduLrg.

3. pduSize: = length(sdu) - aMaxMPduLength

4. pduSize = aPre/MaxPdu / FragmentSize
   - fsdu/frag/to: = requester, fsdu/frag/pri: = true
   - thld: = import aFragmentationThreshold

5. This equation is a common case. Selection of Pdu size is arbitrary, so long as the chosen size is:
   - >=256, even, and <= aMaxPduMaxSize.
   - if (length(sdu) mod pduSize) = 0 then 1 else 0 fi

6. if (pduSize > length(sdu)) then (length(sdu) - pduSize) else pduSize fi

7. Final fragment may be shorter than initial/intermediate fragments.
Use ageing function to force all buffers empty.

\[ \text{import(mCfp), true then contention_free else contention_false) \]

\[ \text{addr2(rpdu), seqNum(rpdu), fragNum(rpdu)} \]

\[ \text{buf(k)}!\text{rsdu := buf(k)}!\text{rsdu \text{1/substr(rpdu, sMacHdrLng, length(rpdu) · sMacHdrLng)}} \]

\[ \text{buf(k)}!\text{linUse}, \text{true, \text{buf(k)}}!\text{hrd} \]

\[ \text{buf(k)}!\text{cur} = \text{fragNum(rpdu), buf(k)}!\text{seq} = \text{seqNum(rpdu)} \]

\[ \text{buf(k)}!\text{rdru} = \text{rpdu} \]

\[ \text{buf(k)}!\text{linUse} = \text{false} \]

\[ \text{buf(k)}!\text{cur} = \text{fragNum(rpdu), buf(k)}!\text{seq} = \text{seqNum(rpdu)} \]

\[ \text{buf(k)}!\text{rdru} = \text{rpdu} \]

\[ \text{buf(k)}!\text{linUse} = \text{false} \]
96/162

Process Rx_Coordination

- (RxC_idle)
  + ResetMAC
  + setAddr1 (ackframe, ackTo)
    - dSifsDly := dUSec (aSifsTime -
    - dStartDly := dUSec (aSifsTime -
      (float(aPreambleLength) +
      (float(aPreambleLength) +
        float(sSifsCtsLns) +
          float(0.5 + (aBitTime)

NeedAck (ackTo, endRx, dAck)
  - if dAck = 0 then
    - dExp = 0 fi
  - send_sifs
  - CTS response to
    RTS only when
    the NAV is clear
    - RxCIack
    - susp := true
    - (end)
    - MIndicate (pdu, endRx, meta)
      - CTS (endRx)
      - strTS, noenr) to
        Synchronization
      (end)
      - CTS
      - MIndicate (pdu, endRx)
        (cts, endRx)

Txmed (pdu)
  - (RxC_idle)
  - (RxC_idle, endRx, dAck)
    - RxCIack

2.2.6.7 says the
RTS response
duration is based
on the data rate
of the RTS. This
is not currently
done because
rx data rate is
not reported by
the Phy.

Class 1 frames handled
on this page, class 2 and
3 frames on next page.

Bacon and probe_req
sent to Synchronization
directly. All others
sent via Reassembly.

None of these
flames should
have group DA.
Process Rx_Coordination

1. Rx with toDs=1 discarded by Filter_MPDU.
   toDs=1 never sent by STA, so explicit test for to/from
   not needed here.

2. lrDS=1 never sent by STA. so explicit test
   for lrDS=1 should not be sent to a station.

3. PS-Poll for to/from ~ should not be
   used here.

4. lrDS=1 never sent by STA. so explicit test
   for lrDS=1 should not be sent to a station.

5. RxPdu(pdu)

6. RxC_Idle

7. uc_ack

8. dAck=d Ack- 0 then
   dAck=0 else 0 8

9. RsLdu=setAddr1
   (ack frame, add2(pdu))

10. RsLdu=
    send

11. RsLdu=
    send

12. RxC_Idle

13. Response

14. RxPdu(pdu)

15. RxC_Idle
IOn this page are DISASSOCIATE Request and DEAUTHENTICATE Request.

The Mime request has no reason parm, use reason 1 (or 8).

Source of the key used to generate challenge text is not specified. */

dcl replyKey Octetstring ;

The Mime request has no reason parm, use reason 1.
Need means for SME to set capability bits.

(Import(mCap) // mkOS
    (import(mListenList,2) // apAddr // mkElem
        (eSsid,import(mSsid)) // mkElem(eSupRates, import(mSupRates)) ) )

See 7.2.3.6.

(Import(mCap) // mkOS
    (import(mListenList,2) // apAddr // mkElem
        (eSsid,import(mSsid)) // mkElem(eSupRates, import(mSupRates)) ) )
On this page is AUTHENTICATE Request.

Copy challenge text from auth seq #2 frame.

Mark shared key frame #3 for encryption.

An unsuccessful authentication attempt leaves the requester de-authenticated.
Process Mime_Req_Rsp

1. **RESPONSES TO INCOMING MANAGEMENT FRAMES**
   - Reply to non-authenticated sender of a class 2 frame.
   - Reply to un-associated sender of a class 3 frame.

2. **Replies (rpsdu, datagram)**
   - If rpsdu = mkFrame (auth, src, bss, class_2.err)
     - rstate = dis_auth
   - If rpsdu = mkFrame (disassoc, src, bss, class_2.err)
     - rstate = dis_assoc

3. **MinRequest (rpsdu, datagram)**
   - Response = true at arrival of frame with type=expect and source from.
     - Other directed, un-solicited mgmt frames ignored.
     - Other processes receive Beacon, Amin & Probe request.

4. **MinIndication (rpsdu, endRx, startTs.err)**
   - (Wait_Prto, Wait_Active, Wait_Passive)
   - bss = import(bssid)
   - if (type(rpsdu) != addr(rpsdu))
   - rstate = dis_auth

5. **Don't Handle responses while communication suspended for scan procedure**
   - if (disassoc or deauthenticate)

6. **MimeDeauth_indicate (src, rpsdu)**
   - rstate = de_auth

7. **MimeDisassociate (src, rpsdu)**
   - rstate = dis_assoc

8. **StaState (src, rpsdu)**
   - (True) and (response = false)
   - response = true, expect = null
     - (open_system) -> (shared_key)

9. **ra0g = alg in alg**
   - (true)
   - (false)

10. **Scm = unsav alg**
    - (true)

11. **Export mgmt (mAssoc = false)**
    - (false)

12. **rapdu = mkFrame (auth, src, bss, alg/mkOS(2, 3))**
    - (true)

13. **rstate = auth_seq_fail**
    - (false)

14. **auth_seq = auth_seq + 1**

15. **authSeq = authSeqNum (rpsdu)**
   - (false)
   - (true)

16. **ra0g = AuthAlg (rpsdu)**
   - (true)

17. **rstate = auth_seq + 1**
   - (false)

18. **StaState (src, auth_key)**

19. **cache (src, challenge) to check seq #3**

20. **authRsp = done**

An unsuccessful authentication attempt leaves the requester de-authenticated.
Sync waits until acceptable time, suspends DCF, then responds with HoldAck. Only start response timing after Probe_req sent, skip channel if tx attempt fails.

Sync waits until next TBTT, then resumes DCF on original channel.

Wait_PASS

Wait_PROBE

MmDone (result)

MmIndicate (npdu, err)

MmIndicate (npdu, endRx, beacon)

Need means for SME to set the capability bits.

mm_pdu := mkFrame (probe_req,)

bssAddr, sta, ss // import (eSupRates)

Switch to next channel on scan list.

Set (now+dDelay, Tresp)

MmCmd (chanw, chist(k),

Delay: mMax

Reset (Tresp, Tmax)

k := k+1

Wait for more responses on this channel until the dMax probe timeout.

Reset the dMin probe timeout upon detection of activity on this channel.

Rx activity flag, exported by Validate_Mmpdu.

Probe delay is local because eProbeDelay no longer in MIB. 

dt dDelay Duration

Return ScanSet, which may be empty.

Mime_Idle

MimeScan_req (types, sta,)

dMin := usec (ctime(1)), k := 1

dMax := usec (ctime(2)),

Wait_Tῆc

sync

NSync wait until

ns., scan,

St, 

Hold

Wait_ACTIVE

HoldAck

Response after probe

delay expires.

Wait_PASS

Release

MimeScan_conf (dor)

Mime_Idle
Each of these sets holds MAC addresses of stations with given operating state. Members are added to or removed from sets due to MLME requests and bits in received headers. The sets are not aged because the standard does not require periodic activity by a station in order to remain a member of a set, although aging to remove inactive stations is permitted.

- awake, detected in active mode
- asleep, detected in power_save mode
- unauth, stations detected, not authenticated
- authOs, authenticated by open system alg.
- authKey, authenticated using any other alg.
- deauth, deauthenticated or authenticate fail
- assoc, associated (<=1 member except AP)
- disassoc, disassociated or associate fail

MacAddrSet:

```
dcl awake, r
  detect in active mode

r

dcl asleep, r
  detect in power_save mode

r

dcl unauth, r
  stations detected, not authenticated

r

dcl authOs, r
  authenticated by open system alg.

r

dcl authKey, r
  authenticated using any other alg.

r

dcl deauth, r
  deauthenticated or authenticate fail

r

dcl assoc, r
  associated (<=1 member except AP)

r

dcl disassoc, r
  disassociated or associate fail
```

A state diagram for Power Save Mode and Station State monitoring is shown on the next page. The diagram includes transitions for awake, asleep, and authKey states, as well as association and disassociation events. The diagram also shows how authentication and deauthentication are handled, along with the handling of authenticated and unauthenticated stations at startup.
Process Power_Save_Monitor

Monitor_Idle

Power Save and Station State query and responsebelow, monitoring on previous page.

PsInquiry (sta)

PsInquiry returns PspResponse to report power mode awake, asleep, or unknown at the target station.

lsGroup (sta)

(isGroup (sta) (true) (false)

sta in awake (true) (false)

sta in asleep (true)

pm:= unknown

pm:= asleep

pm:= awake

PsInquiry (sta, pm) to sender

lp:= address

sta in assoc (true) (false)

sta in disassoc (false)

sta in auth (true) (false)

sta in authkey (true) (false)

asst:= auth_open

asst:= auth_key

asst:= de_auth

asst:= not_auth

When there is no association info, station state is identical to authentication state.

SspResponse (sta, asst) to sender

∗PsInquiry returns [asleep/awake/unknown]
Exchange of TxRequest and TxConfirm signals is synchronous between Blocks MAC Control and Transmission. MAC Control waits for a TxConfirm before sending another TxRequest. However, a Txmed/TxDone exchange may occur while a TxRequest is pending. An example is a Txmed to transmit an ACK in response to a frame received while a pending TxRequest is in backoff. TxCancel terminates the TxRequest if transmission has not yet started, and returns the residual backoff counter value with the TxConfirm.
Calculate PHY TX delay added to TSF time to get timestamp.

TX delay added to TSF time to get timestamp.

SIFS responses sent by Txmed, hence its priority.

Source = sender

k = 0, fcs = init_Crc

While length = Length(pdu)

PLCP length is longer than PDU due to CRC field.

Wait, TxStart

PhyTxStart, confirm

Send Frame

Upon confirmation of 24th octet, insert current time, offset by Phy Tx delay, into octets 24-31 of Beacon & Probe Response frames.

fcs = crc32 (fcs, pdu(k))

k := k + 1

k := a TxOctet (true)

else

Type(pdu)

(k := 0, fcs := mirror (not(fcs)))

Send_CRC

PhyData_confirm

pdu := setTs(pdu, now + dTx)

send1

If fcs CRC:

fcs CRC:

dc1 dTx Duration:
dcl k, txLength: Integer:
dcl pdu Frame:
dcl source Pld:

PhyData_confirm

send1

PhyData_request (pdu(k))

k := k + 1

PhyData_request (pdu(k))

PhyTxEnd_request

Send_CRC

PhyTxEnd_request

PhyTxEnd_request

TxDone to source

Tx Idle

Phy Tx End

ResetMAC

PhyTxEnd_request

TxDone is not generated if
Tx is halted by ResmiMAC.

k = sCRCLong

TxDone to source

PhyTxEnd_request
Block Reception

signal
  ClearNav(NavSrc),
  RtsTimeout,
  RxMpdu(Frame,Time,Time),
  SetNav(Time,Duration,NavSrc),
  UseDifs(Time),
  UseEifs(Time) :

FromCtl

Channel State (1,1)

setNav, ClearNav

RtsTimeout, UseDifs, UseEifs

UpdNav

filter MPDU (1,1)

RxIndicate, NeedAck,
RxClAck

Filter

ResetMAC

ValidatE MPDU (1,1)

phyRxStart.indicate,
phyRxEnd.indicate,
phyData.indicate

FromPHY

phyCca

ToCtl

Rval

SslnqulIY,
Idx

RxIndicate,
Gsinquiry,
Idx

ToPS

pslndicate,
SslnqulIY,
Idx

18_RX_1b(1)

ResetMAC

RespIesponse

ToCtl

ToPS

Busy, idle, Slot

PHY SAP RX
Process Filter_MPDU

1. **retryBit** := true
   - **dup** := searchTupleCache(cache, addr2(pdu), seq(pdu), frag(pdu))
   - **cDup** := inc(cDup)
   - **wepBit** := true
     - **RxIndicate** := updateTupleCache(cache, addr2(pdu), endRx, strTs, dAck)
   - **basetype** := data
   - **needAck** := encrypt(pdu, ivcOk)
   - **filterIdle**

2. **SIFS frames may be omitted from cache, see 9.2.9.**

3. **cache** := updateTupleCache(cache, addr2(pdu), seq(pdu), frag(pdu), endRx)

4. **filterIdle**

5. **create a new cache entry, or replace an old entry, if the (addr2,seq) pair is not cached. Update fragment number and time in existing entry if the (addr2,seq) pair is cached.**

6. **dup** := searchTupleCache(cache, addr2(pdu), seq(pdu), frag(pdu))
   - **cDup** := inc(cDup)
   - **export** := true
     - **wepBit** := true
       - **RxIndicate** := updateTupleCache(cache, addr2(pdu), endRx, strTs, dAck)
     - **basetype** := management
     - **needAck** := encrypt(pdu, ivcOk)
     - **filterIdle**

7. **increment ICV error count on unsuccessful decryption.**

8. **clearNav** := dUseInd(clfDurRem(pdu))
   - **filterIdle**

9. **clearNav** := dUseInd(clfDurRem(pdu))
   - **filterIdle**
dSifs := dUsec (aSifsTime),

\[ \text{dSlot} = \text{dUsec (aSlotTime)} \]

EIFS duration calculated using the formula in clause 9.2.9.

```
dEifs := \text{c}\text{i}\text{a}\text{l}\text{c}_{\text{Eifs}} (2 \text{dSlot})
```

reset(Tnav)

```
tNavEnd = \text{now}
```

```
\text{PhyCcast}_{\text{request}}
```

```
cs := \text{busy}
```

```
\text{curSrc} = \text{noSrc}
```

```
\text{Busy}
```

```
\text{Cs}_{\text{noNav}}
```

Idle

```
\text{noCS}_{\text{noNav}} */ IDLE */
```

Idle signal is sent at end of the M2 interval (Figure 47).

```
\text{Cs}_{\text{noNav}} */ BUSY */
```

ClearNav, RtsTimeout, Tnav, Tslot ignored in Cs_noNav state.

```
\text{SetNav}_{\text{(Ref, dNav, curSrc)}}
```

```
\text{tNavEnd} = \text{tRef+4Nav}
```

```
\text{export(tNavEnd)}
```

Cs_NAV

ResetMAC

Cs_noNav
Process Channel _ State

- **noCs_Nav; Cs_Nav */ BUSY */
  - Tslot and Tifs ignored in noCs_Nav state.
  - PhyCcra_ indicate(cs)
  - cs
    - (idle)
    - set (now+difs, Tifs)
    - Wait_IFS

- **noCs_Nav, Cs_Nav */ all NAV */
  - SetNav (tRef, dNav, newSrc)
    - (false)
    - tNavEnd:=true
    - (true)
      - tNavEnd:=tNavEnd, curSrc:=newSrc
      - set(tNavEnd, Tnav)
      - export(tNavEnd)

- **Cs_Nav */ BUSY */
  - Tslot and Tifs ignored in Cs_Nav state.
  - PhyCcra_ indicate(cs)
  - cs
    - (idle)
    - curSrc:= null
    - set (now+difs, Tifs)

- **Cs_NoNav */
  - Tslot and Tifs ignored in Cs_NoNav state.
  - PhyCcra_ indicate(cs)
  - cs
    - (idle)
    - curSrc:= null
    - set (now+difs, Tifs)
    - Cs_NoNav

Creating the NAV on RTS timeout is optional. See last paragraph of 9.2.5.4.

- **newSrc = cswtch**
  - (false)
  - curSrc:= rts
  - (true)

- **difs:= dNav-dRxT x**
  - (false)
  - (true)

- **tNavEnd:=now, curSrc:= nosrc**
  - (false)

- **UseDifs (tRxEnd)**
  - difs:= dDifs-dRxT x
  - set (tRxEnd+difs, Tifs)

- **UseDifs (tRxEnd)**
  - difs:= dDifs-dRxT x
  - set (tRxEnd+difs, Tifs)

- **Nav is cleared by setting Tnav to now. This causes immediate Tnav signal to enable exit from noCs_Nav or Cs_Nav state.**
```
dcl exported cErr as aFcsErrorCount Counter32:= 0;
dcl exported mRxA Boolean:=false;
time: Trts;

// Calculate PHY Rx delay to be subtracted from now at indicated reference points.
\( D1 := \text{dUsec} (aRxDelay - aRxFpDelay) \)

// Save arrival time of first octet of (what may be a) timestamp field.
startTs := now - D1

// Save time of Rx end as reference for start of IFS.
endRx := now - D1

// Indicate that a reception is in progress.
phyRxStart := now - D1

// Indicate that reception is not in progress.
phyRxEnd := now - D1

// Accumulate octet into Mpdo and CRC check.
\( k := k + 1 \)

// Initialize CRC & clear pdu buffer (length(pdu)=0).
k := 0;
fcs := initFcs;
pdu := empty

// UseEis (endRx)
if status (no_error)
  mRxA := false
else
  cErr := ino(cErr)

// Drop CRC field from valid frame before passing up for filtering.
dropFcs := fcs

// Process Validate_MPDU
```

```
dcl fcs Crc;
dcl D1, dRts Duration;
dcl endRx, startTs Time;
dcl k, nLength Integer;
dcl pdu Frame;
dcl status PhyRxStat;
dcl v Octet;

// Save time of Rx end as reference for start of IFS.
endRx := now - D1

// ResetMAC
reset(Trts)

// Export(mRxA)
mRxA := false

// Rx_idle
```
Procedure Decrypt

\[ \text{decryptLn} = \text{length(pdu)} \]

\[ \text{key} = \text{key with IV from frame for decrypt} \]

\[ \text{Use RC4(key) to generate a decrypt string at long as the encrypted payload.} \]

\[ \text{if selected key is null, report success without decrypting.} \]

\[ \text{If calculated ICV not valid, discard frame body, and report error.} \]
This package contains definitions of the custom sorts (data types), operators, literals, and synonyms (named constants) used by the MAC state machines.

/**
 * ENUMERATED TYPES for the MAC State Machines */

/*
 * BackoffStatus -- indicates status of the Backoff process when a TxConfirm is generated */
newtype BackoffStatus literals
  // done, 1/ indicates completion of a transmission requested with TxRequest 1/
  cancelled, 1/ indicates cancellation of an unstarted transmission attempt due to TxCancel 1/
  inactive; 1/ indicates that the Backoff process was inactive when a TxCancel was received */
endnewtype BackoffStatus ;

/*
 * ChangeType - indicates the type of change in communication at the upcoming boundary */
newtype ChangeType literals dwell, mocp, bss ;
endnewtype ChangeType ;

/*
 * CtlCmd -- identifies the function in MmCntl signals */
newtype CtlCmd literals
  suspend, 1/ halts transmission attempts to permit scanning */
  resume, 1/ restarts transmission after suspend */
  sleep, 1/ enters doze state after confirmation of doze from PHY */
  wake, 1/ leaves doze state and reactivates PHY */
  chanSw ; 1/ requests change of PHY channel */
endnewtype CtlCmd ;

/*
 * NavSrc -- identifies the source of a duration value for SetNav and ClearNav signals */
newtype NavSrc literals
  rts, cfpBss, cfpOther, cfpmisc, cfpmisc ;
endnewtype NavSrc ;

/*
 * PsMode -- identifies the power save state of a station in PsResponse */
newtype PsMode literals sta_active, power_save, unknown ;
endnewtype PsMode ;

/*
 * StateErr -- sent to MLME with Mmlndicate to initiate class 2 or class 3 error response */
newtype StateErr literals noerr, class2, class3 ;
endnewtype StateErr ;

/*
 * StationState -- identifies association/authentication state of a station in SsResponse */
newtype StationState literals
  unknown, 1/ no information is available about the subject station */
  not_auth, 1/ subject station known but not authenticated */
  de_auth, 1/ subject station explicitly deauthenticated */
  auth_open, 1/ subject station authenticated using open system algorithm */
  auth_key, 1/ subject station authenticated using any algorithm except open system */
  assoc, 1/ subject station associated */
  dis_assoc ; 1/ subject station explicitly disassociated */
endnewtype StationState ;

/*
 * TxResult - identifies (detailed) result of transmission attempt */
newtype TxResult literals successful, noBss, noAuth, noAssoc, retryLimit, txLifeTimeout ;
endnewtype TxResult ;
**Enumerated Types and Type Sets for MAC_SAP and MLME_SAP Parameters**

- **AuthType** -- used for `<authentication type>` parameter of MlmeAuthentication primitives

```
newtype AuthType inherits Octetstring operators all;
  adding literals open_system, shared_key;
  axioms open_system == mkOS(0, 2); shared_key == mkOS(1, 2);
newtype AuthType;
newtype AuthTypeSet powerset(AuthType);
```

- **BssType** -- used for `<BSS type>` parameter of MlmeScan, Join, and Start primitives

```
newtype BssType literals infrastructure, independent;
newtype BssTypeSet powerset(BssType);
```

- **GfPriority** -- used for `<priority>` parameter of MAG data service primitives

```
newtype GfPriority literals contention, contentionFree, datagram;
newtype GfPriority;
```

- **MibStatus** -- used for `<status>` parameter of MlmeGet.confirm and MlmeSet.confirm primitives

```
newtype MibStatus literals success, invalid, read_only, write_only;
newtype MibStatus;
```

- **PwrSave** -- used for `<power management mode>` parameter of MlmePowerMgt.request primitives

```
newtype PwrSave literals sta_active, power_save;
newtype PwrSaveSet powerset(PwrSave);
```

- **Routing** -- used for `<routing information>` parameter of MAC data service primitives

```
newtype Routing literals null_rt, other;
newtype Routing;
```

- **RxStatus** -- used for `<reception status>` parameter of MlmeGetData.indication

```
newtype RxStatus literals rx_success, failed_rx;
newtype RxStatusSet powerset(RxStatus);
```

- **ScanType** -- used for `<scan type>` parameter of MlmeScan.request primitives

```
newtype ScanType literals active_scan, passive_scan;
```

- **ServiceClass** -- used for `<service class>` parameter of MAC data service primitives

```
newtype ServiceClass literals reorderable, strictlyOrdered;
```

- **Success** -- used for `<operation successful>` parameter of MlmeXyz.confirm primitives

```
synthese Success = Boolean endsynthese Success;
```

- **TxStatus** -- used for `<transmission status>` parameter of MlmeGet.confirm and MlmeSet.confirm primitives

```
newtype TxStatus literals successful, undeliverable, excessiveDataLength, nonNullSourceRouting, unsupportedPriority, unsupportedServiceClass, unavailableServiceClass;
newtype TxStatus;
```

**Enumerated Types for PHY_SAP Parameters**

- **CcaStatus** -- used for `<status>` parameter of PhyCca.indicate primitives

```
newtype CcaStatus literals busy, idle;
```

- **PhyRxStat** -- used for `<error>` parameter of PhyEnd.indicate primitives

```
newtype PhyRxStat literals no_error, fmt_violation, carrier_lost, unsupported_rate;
```
INTRA-MAC REMOTE VARIABLES (not part of the MIB) -- names begin with "m"

- mActingAsAp Boolean nodelay;
- mAssoc =true if this station is associated with an AP (from Mlme_Req_Rsp);
- mAtimW =true if the ATIM window is in progress (from Synchronization);
- mCap holds capability info (ADDED TO) join request (from Synchronization);
- mCfp =true when a contention free period is in progress (from Synchronization);
- mlbss =true when this station is a member of an independent BSS (from MIB);
- mNavEnd Time nodelay;
- mNextBdry Time nodelay;
- mNextTbtt Time nodelay;
- mPcAvail =true if a point coordinator is operating in this BSS (from Mlme_Req_Rsp);
- mPcDlvr =true if the CF-period is used for delivery only (from Mlme_Req_Asp);
- mPcPoll =true if the CF-period is used for delivery and polling (from Mlme_Req_Rsp);
- mPsm =sta_active I power_save) for the current power save mode (from SMT_Requests);
- mRxA =true when the PHY indicates that a reception is in progress (from Validate_Mpdu);
- mRoA Boolean nodelay;
- mSsld Ocletstring nodelay;  */ maximum length=32 */

/* PLACEHOLDERS FOR MLME/PLME GET/SET PARAMETER VALUES */

- MibAtrib is a placeholder until integration of the MIB using Z.105 */
- MibValue is a placeholder until integration of the MIB using Z.105 */

remote procedure RC4 nodelay; fpar Ocletstring, Integer; returns Ocletstring;
PACKAGE macsorts

/* NAMES STATIC DATA VALUES — names begin with "s" in the form "sNameOfItem" */

/* Maximum number of octets in an MSDU passed to or from LLC */
synonym sMaxMsduLng Integer = 2304 ;
/* Number of octets in the basic MAC header for Data and Mgmt frames (without WEP) */
synonym sMaxHdrLng Integer = 24 ;
/* Number of octets in the basic MAC header plus IV/keyID for Data and Mgmt frames with WEP */
synonym sWepHdrLng Integer = 28 ;
/* Number of octets added to a PDU when using WEP (both IV/keyID and ICV fields) */
synonym sWepAddLng Integer = 6 ;
/* Number of octets added to the MAC header for Wireless Distribution System transfers */
synonym sWdsAddLng Integer = 6 ;
/* this is the length of the add4 field */
/* Number of octets in a CRC (or ICV) field */
synonym sCrcLng Integer = 4 ;
/* Maximum number of octets in an MPDU, and the corresponding index range */
synonym sMaxMpdulng Integer = (sMaxMsduLng+sMaxHdrLng+sWdsAddLng+sWepAddLng+sCrcLng) ;
synonym sFrameIndexRange = integer constants 1:sMaxMpdulng endsynonym sFrameIndexRange ;
/* Index of the first octet of the Timestamp field of Beacon and Probe Response frames */
synonym sTsOctet Integer = 24 ;
/* this value for use with the 0-ORIGIN OctetString defined herein */
/* Minimum allowed value for sMpdulMaxLength */
synonym sMinFragLng Integer = 256 ;
/* Maximum fragment number and corresponding index range */
synonym sMaxFragNum Integer = (sMaxMsduLng / (sMinFragLng-sMaxHdrLng-sWepAddLng-sCrcLng)) ;
/* Number of bits in ACK and CTS control frames */
synonym sAckCtlng Integer = 112 ;

*/ STATION CONFIGURATION FLAGS (supplementary to MIB) — names begin with "s" */

/* Protocol version number supported by this version of the MAC */
synonym sVerslon Integer = 0 ;
/* must be =0 for the current MAC */
synonym sCanBeAp Boolean = false ;
/* set to correct value at each kind of station */
synonym sCanBePc Boolean = false ;
/* set to correct value at each kind of station */
synonym sCfPollable Boolean = true ;
/* set to correct value at each kind of station */

** THE FOLLOWING IS A TEMPORARY DEFINITION **

sBitTime is the time (in microseconds) needed to transfer 1 bit over the wireless medium
at the minimum PHY data rate for the active (i)BSS. sBitTime is defined for use
in lieu of the minimum value in aBssBasicRateSet, which is no longer in the MIB.
synonym sBitTime Real = 1.0 ;
/* this value is for PHYs with 1 Mbps minimum basic rate */
DISCRETE, MICROSECOND-UNIT TIME AND DURATION SORTS

SDL does not define the relationship between its concept of Time and physical time in the system being described. An abstraction is needed to establish this relationship, because Time in SDL uses the semantics of Real, whereas time in the MAC protocol is discrete. The MAC uses intervals with specific, integral relationships, distributes time synchronization information as a 64-bit count of microsecond, and defines many actions relative to this 1MHz timebase. Therefore, time for the MAC is modeled using Natural.

In these MAC state machines, a change of 1.0 in Time (or Duration) is assumed to represent one microsecond of physical time. To avoid issues with roundoff and repeatable tests for equality, the time and duration calculations are generally done using the subtypes of Integer defined below, with explicit conversion to SDL Time (using the uTime operator), SDL Duration (using the dUsee operator), or from SDL Time (using the uTime operator) only when needed to set timers or to comply with SDL’s strong type checking. These operators are defined with each time sort to allow conversion functions to be changed as needed for future applications (e.g., simulation of the MAC protocol).

Microsecond sort — also provides selection operators min and max

newtype Usee inherits Integer operators all;

adding operators

dUsee : Usee -> Duration;
tUsee : Usee -> Time;
uTime : Time -> Usee;
min : Usee,Usee -> Usee;
axioms

for all u, w in Usee {
  u >= w ==> max(u,w) == u;
  u < w ==> max(u,w) == w;
  u > w ==> min(u,w) == w;
  u < w ==> min(u,w) == u;
for all t in Time ( for all r in Real {
  r = float(u) ==> tUsee(u) == time!(duration!(r));
  r = float(u) ==> uTime(time!(duration!(r))) == u; 
});
for all d in Duration ( for all r in Real {
  r = float(u) ==> dUsee(u) == duration!(r); 
});
for all u in Usee ( u2K(u) == u / 1024; )
;
constants >=0 /* constrain value range to non-negative, as with Natural */
endnewtype Usee;

Kmicrosecond sort — (Kusee) = 1024 * Usee

newtype Kusee inherits Integer operators all;

adding operators
dKusee : Kusee -> Duration;
tKusee : Kusee -> Time;
kuTime : Time -> Kusee;
ku2K : Usee -> Kusee;
K2U : Kusee -> Usee;
axioms

for all k in Kusee ( for all r in Real {
  r = float(k) ==> tKusee(k) == time!(duration!(1024*r));
  r = float(k) ==> kuTime(time!(duration!(1024*r))) == k; 
});
for all d in Duration ( for all r in Real {
  r = float(k) ==> dKusee(k) == duration!(1024*r); }}
for all u in Usee ( u2K(u) == u / 1024;
  k2U(k) == k * 1024; )
;
constants >=0 /* constrain value range to non-negative, as with Natural */
endnewtype Kusee;
0-ORIGIN STRING GENERATOR

String0 generator (derived from Z.105, Annex A) creates strings of any sort, indexed starting with 0 rather than 1.

Package macsorts

literals Emptystring:

operators

MkString : Item -> String0 ; /* make a string from an item */
Length : String0 -> Integer ; /* length of string */
First : String0 -> Item ; /* first item in string */
Tail : String0 -> String0 ; /* all but first item in string */
Last : String0 -> Item ; /* last item in string */
Head : String0 -> String0 ; /* all but the last item in string */
/\ String0, String0 -> String0 ; /* concatenation */
Extract! : String0, Integer -> Item ; /* get item from string */
Modify! : String0, Item, Integer -> String0 ; /* modify value of string */
SubStr : String0, Integer, Integer -> String0 ; /* string0 of length j starting at i-th item */
S2 : Item, Item -> String0 ;
S3 : Item, Item, Item -> String0 ;
S4 : Item, Item, Item, Item -> String0 ;
S5 : Item, Item, Item, Item, Item -> String0 ;
S6 : Item, Item, Item, Item, Item, Item -> String0 ;
S8 : Item, Item, Item, Item, Item, Item, Item, Item -> String0 ;

for all s, s1, s2 in String0 (for all i, j in Integer)

/" constructors are Emptystring, MkString, and /\; equalities between constructor terms /" s // Emptystring == 8 ; Emptystring // s == 8 ;
(s1 // s2) // s3 == s1 // (s2 // s3) ;

/" definition of Length by applying it to all constructors /"

type String Length(Emptystring) == 0 ;
type String Length(MkString(item0)) == 1 ;
type String Length(s1 // s2) == Length(s1) + Length(s2) ;

/" definition of Extract by applying it to all constructors, Error! cases handled separately /"
Extract(MkString(item0),i) == item0 ;
i < Length(s1) ==> Extract(s1 // s2,i) == Extract(s1 ,i) ;
i >= Length(s1) ==> Extract(s1 // s2,i) == Extract(s2,i-Length(s1)) ;
i < 0 or i >= Length(s) ==> Extract(s,i) == Error! ;

/" definition of First and Last by other operations /"
First(s) == Extract(s,0) ; Last(s) == Extract(s,Length(s)-1) ;

/" definition of substr(s,i,j) by induction on j, Error! cases handled separately /"
i >= 0 and i <= Length(s) ==> Substr(s,i,j) == Substr(s,i,j-1) //
MkString(Extract(s,i-1,j));
i > 0 and j <= Length(s) ==> Substr(s,i,j) == Substr(s,i,j-1) //
MkString(Extract(s,i,j-1)) ;
i < 0 or j < 0 or i+j > Length(s) ==> Substr(s,i,j) == Error! ;

/" definition of Modify!, Head, Tail, and Sx by other operations /"
Modify!(s,i,item0) == Substr(s,i) // MkString(item0) // Substr(s,i+1,Length(s)-i-1) ;
Head(s) == Substr(s,0,Length(s)-1) ; Tail(s) == Substr(s,1,Length(s)-1) ;
S2(item0,item1) == MkString(item0) // MkString(item1) ;
S3(item0,item1,item2) == MkString(item0) // MkString(item1) // MkString(item2) ;
S4(item0,item1,item2,item3) == MkString(item0) // MkString(item1) // MkString(item2) // MkString(item3) ;
S5(item0,item1,item2,item3,item4) == MkString(item0) // MkString(item1) // MkString(item2) // MkString(item3) // MkString(item4) ;
S6(item0,item1,item2,item3,item4,item5) == MkString(item0) // MkString(item1) // MkString(item2) // MkString(item3) // MkString(item4) // MkString(item5) ;
S7(item0,item1,item2,item3,item4,item5,item6) == MkString(item0) // MkString(item1) // MkString(item2) // MkString(item3) // MkString(item4) // MkString(item5) // MkString(item6) ;
S8(item0,item1,item2,item3,item4,item5,item6,item7) == MkString(item0) // MkString(item1) // MkString(item2) // MkString(item3) // MkString(item4) // MkString(item5) // MkString(item6) // MkString(item7) ;

endgenerator String0 ;
\* -- \* ASN.1-style BIT SORT (identical to definition of Bit in Z.105, Annex A) \*/
\* Bit is a subtype of Boolean -- bit values 0 and 1 cannot be used with Integer operators \*/
newtype Bit inherits Boolean literals 0 = FALSE, 1 = TRUE; operators all; endnewtype Bit;

\* \* ASN.1-style BITSTRING SORT (derived from Z.105, Annex A) \*/
\* Bitstring is a 0-origin string of Bit. Z.105 provides binary (1011'B) and hexadecimal (D3'H) literals,
but parsing these requires relaxation of a Z.100 rule regarding the use of apostrophes.
Therefore, this version of Bitstring provides hexadecimal literals 0x00:0xFF. Bitstrings of
non-octet length can be constructed by concatenating individual bits using MkString.
Operators \"not\", \"and\", \"or\", \"xor\", and \"implies\" act bitwise on Bitstring operands.
For dyadic operators, the length of the result is equal to the longer of the source operands. \*/
newtype Bitstring StringO(Bit.)
adding literals
macro Hex_literals; /* macro Binary_literals; */
operators
\"not\" : Bitstring -> Bitstring;
\"and\" : Bitstring, Bitstring -> Bitstring;
\"or\" : Bitstring, Bitstring -> Bitstring;
\"xor\" : Bitstring, Bitstring -> Bitstring;
\"implies\" : Bitstring, Bitstring -> Bitstring;

axioms
macro Hex_axioms; /* macro Binary_axioms; */
for all x, y in Bitstring (x = y == True;

\* connection to the String generator \*/
for all b1, b2 in Bit literals (Spelling(b1) = ""| b1 | bs2 | "", Spelling(b2) = ""| bs1 | | b2 | "", Spelling(b) = bs1 == b1 == MkString(b1) // b2 );
endnewtype Bitstring;

\* connection to the String generator \*/
for all b1, b2 in Bit literals (Spelling(b1) = ""| b1 | bs2 | "", Spelling(b2) = ""| bs1 | | b2 | "", Spelling(b) = bs1 == b1 == MkString(b1) // b2 );
endnewtype Bitstring;
Z.105 defines Octet as "syntype Octet = Bitstring constants size (8) endsyntype Octet;" unfortunately, "size" is an extension to the abstract grammar of SDL, unavailable in Z.100. Therefore, Octet is defined here as a subtype of Bitstring, and relies on proper usage to establish and maintain lengths which are integral multiples of 8. The easiest way to create octet lengths is to use mkOctet or the hexadecimal literals defined for Bitstring (e.g. 0x00).

This definition of Octet includes the following operators:

- o := mkOctet(i) converts a non-negative Integer (mod 256) to an Octet (always exactly 8 bits)
- o := octetVal(o) converts an Octet to an Integer (0-255)
- o := flip(o) reverses the order of the bits within the octet (0<>->7, 1<>->6, 2<>->5, 3<>->4)

newtype Octet inherits Bitstring operators all;

adding operators

mkOctet : Integer -> Octet;
octet Val : Octet -> Integer;
flip : Octet -> Octet;

axioms

for all i in Integer ( for all z in Octet ( i = 0 ==> mkOctet(i) == S8(0,0,0,0,0,0,0,0) ; i = 1 ==> mkOctet(i) == S8(1,0,0,0,0,0,0,0) ; i > 1 and i <= 255 ==> mkOctet(i) == substr( first(mkOctet(i mod 2)) // mkOctet(i/2) ) , 0, 8 ) ; i > 255 ==> mkOctet(i) == mkOctet(i mod 256) ) ;

OctetVal(z) = octetVal(first(z)) + (2*(octetVal(substr(z,1,length(z)-1)))) ;
length(z) > 8 ==> octetVal(z) == error!
flip(z) = S8(z(7),z(6),z(5),z(4),z(3),z(2),z(1),z(0) ) ;

endnewtype Octet;

Octetstring is a 0-ORIGIN string of Octets (UNLIKE the 1-origin Octet string of ASN.1).

Conversion ops to and from Bitstring, plus integer to Octetstring, but only literals are "null" and 1-4 position 0x00 strings O1, O2, O3, and O4. Octetstring constants are created using these literals and aggregation operators S2, S3, S4, S6, and S8. The following newtype Octetstring defines these literals and aggregates:

newtype Octetstring StringO (Octet,null)

adding literals

O1, O2, O3, O4;

operators

Bit_String : Octetstring -> Bitstring;
Octet_String : Bitstring -> Octetstring;
mkOS : Integer,Integer -> Octetstring; /* mkstring(mkOctet(i1)) 0-extended to length i2 */

axioms

for all b,b1,b2 in Bitstring ( for all s in Octetstring ( for all o in Octet ( Bit_String(null) == ; Bit_String( MkString(o) // s ) == o // Bit_String(s) ; Octet_String( ) == null ;

Length(b1) > 0, Length(b1) < 8 ==> Octet_String(b1) == MkString(b1 or 0x00) ; /* expand b1 to 8 bits */
b == b1 or b2, Length(b1) == 8 ==> Octet_String(b1) == MkString(b1) // Octet_String(b2) ;

for all i, k in Integer ( k = 1 ==> mkOS(i,k) == MkString(mkOctet(i)) ; k = 0 ==> error! ;
O1 == mkstring(0x00) ; O2 == S2(0x00,0x00) ; O3 == O1 or O2 ; O4 == O2 or O2 ;

map for all o1, o2 in Octetstring literals ( for all b1, b2 in Bitstring literals ( Spelling( o1 ) = Spelling( b1 ) , Spelling( o2 ) = Spelling( b2 ) ) ;

endnewtype Octetstring;
package macsorts

-- MAC ADDRESS SORTS
--
-- MacAddr is a subtype of Octetstring with added operators
-- isGroup(m), which returns true if given a group address,
-- isBcast(m), which returns true if given the broadcast address, and
-- isLocal(m), which returns true if given a locally-administered address.
-- MAC addresses must be defined such that they are exactly 6 octets long. The preferred
-- ways to achieve this are to use the S6 aggregation operator or nullAddr synonym. */

newtype MacAddr inherits Octetstring operators all ;

adding operators
isGroup : MacAddr -> Boolean;
isBcast : MacAddr -> Boolean;
isLocal : MacAddr -> Boolean;
adrOs : MacAddr -> Octetstring ;

axioms for all m in MacAddr (
(length(m) = 6) and (extract!(m,0) and OxOl = OxOl) ==> isGroup(m) == true ;
(length(m) = 6) and (extract!(m,0) and OxOl = OxOl0) ==> isGroup(m) == false ;
(length(m) = 6) and (m = S6(OxFF, OxFF, OxFF, OxFF, OxFF, OxFF)) ==> isBcast == true ;
(length(m) = 6) and (m /= S6(OxFF, OxFF, OxFF, OxFF, OxFF, OxFF)) ==> isBcast == false ;
(length(m) = 6) and (extract!(m,0) and Ox02 = Ox02) ==> isLocal == true ;
(length(m) = 6) and (extract!(m,0) and Ox02 = Ox020) ==> isLocal == false ;

(length(m) /= 6) ==> Error! */

for all o in Octetstring (m = MacAddr!o == adrOs(m) = o ; )));
endnewtype MacAddr ;

/* set of Mac Addresses */
newtype MacAddrSet powerset(MacAddr) endnewtype MacAddrSet ;

/* Broadcast Address */
synonym BcstAddr MacAddr = S6(OxFF, OxFF, OxFF, OxFF, OxFF, OxFF);

/* Null Address (an all-zero, 6-Octet string for use as placeholder) */
synonym NullAddr MacAddr = S6(Ox.OO, OxOO, OxOO, OxOO, OxOO, OxOO) ;

/* BSS DESCRIPTION SORTS */

/* The BssDescr structure is used in parameters of MlmeScan.confirm and MlmeJoin.request */
newtype BssDescr Struct
bdBssIdd MacAddr ;
bdCap Capability ; /* capability information <ADDED -- NOT IN 10.3.3.1> */
bdSid Octetstring ; /* max length=32, not enforced by data structure */
bdType BssType;
bdBcnlnt Kusec ;
bdTstamp Octetstring ; /* 8 Octets from Probe Response or Beacon frame */
bdPhyParms PhyParms ; /* empty if inapplicable for active PHY */
bdCfParms CfParrns ; /* empty if no point coordinator in BSS */
bdbssParms IbasParms ; /* empty if infrastructure BSS */

endnewtype BssDescr ;

/* set of Bss Descriptors */
newtype BssDescrSet powerset(BssDescr) endnewtype BssDescrSet ;
/* DUPLICATE FILTERING SUPPORT STRUCTURES */
" Range of possible fragment numbers */
```
syntype FragNum = Integer constants 0..MaxFragNum endsytepe FragNum ;
```
" Range of possible sequence numbers */
```
syntype SeqNum = Integer constants 0..SeqNumMax endsytepe SeqNum ;
```
" Tuple structure (used for duplicate filtering and Msdu/Mmpdu reassembly.) */
```
newtype Tuple Struct
  full : Boolean; /* true if Tuple contains valid/current frame information */
  tMacAddr : MacAddr; /* transmitting station (Addr2) */
  tSeqNum : SeqNum; /* Msdu/Mmpdu sequence number */
  tFragNum : FragNum; /* Mpdru fragment number */
  tRx Time ; /* reception time (endRx of fragment) */
  default (. false, nullAddr, 0, 0, 0 .) ;
endnewtype Tuple ;
```

```
/* TUPLE CACHE SUPPORT */
" Number of entries in tuple cache at this station, and associated index range */
synonym tupleCacheSize Integer = 32 ; /* cache size (>2) is implementation dependent */
syntype CacheIndex = Integer constants 1..tupleCacheSize endsytepe CacheIndex ;
" Tuple cache array with search & update operators */
newtype TupleCache Array(CacheIndex,Tuple) ;
```
```
adding operators
  clearTupleCache : TupleCache -> TupleCache ;
  searchTupleCache : TupleCache, MacAddr, SeqNum, FragNum -> Boolean ;
  updateTupleCache : TupleCache, MacAddr, SeqNum, FragNum, Time -> TupleCache ;
```
```
operator clearTupleCache ; fpar cache TupleCache ;
returns TupleCache ; referenced ;
```
```
operator searchTupleCache ; fpar cache TupleCache, taddr MacAddr, tseq SeqNum, 
tfrag FragNum ; returns Boolean ; referenced ;
```
```
operator updateTupleCache ; fpar cache TupleCache, taddr MacAddr, tseq SeqNum, 
tfrag FragNum, tnow Time ; returns TupleCache ; referenced ;
```
endnewtype TupleCache ;
```

```
/* 32-BIT (unsigned) UP-COUNTER WITH WRAPAROUND */
" Used for MIB counters, inc(cntr) increments value by 1, with wraparound from (2^32)-1 to 0. */
newtype Counter32 inherits Integer operators all ;
```
```
adding operators
  inc : Counter32 -> Counter32 ;
```
```
axioms
for all c in Counter32 ( 
  c < 4294967295 ==> inc(c) == c + 1 ;
  c >= 4294967295 ==> inc(c) == 0 ;
);
```
endnewtype Counter32 ;
```

```
/* STRING OF INTEGER */
newtype Intstring String(Integer,nonlnt); endnewtype Intstring ;
```

49
package macsorts

""" QUEUE GENERATOR """

""" Queue generator (a variant of the StringO generator) creates Queues of any sort with operators"

Qfirst(queue, item) which adds item as the first queue element, and
Qlast(queue, item) which adds item as the last queue element.

Since Queue is derived from StringO, operators Length, First, Last, Head, Tail, etc. are available.
Since operators cannot modify source operands, removal of queue elements is a 2-step process:
dequeue first is Item := First(queue); queue := Tail(queue);
dequeue last is Item := Last(queue); queue := Head(queue); """

generator Queue (type Item, literal Emptyqueue)
literal Emptyqueue;

operators

MkO : Item -> Queue; /* make a queue from an item */
Length : Queue -> Integer; /* number of items on queue */
First : Queue -> Item; /* first item in string */
Qfirst : Queue, Item -> Queue; /* add item as first on queue */
Tail : Queue -> Queue; /* all but first item on queue */
Last : Queue -> Item; /* last item on queue */
Qlast : Queue, Item -> Queue; /* add item as last on queue */
Head : Queue -> Queue; /* all but the last item in string */
"""" : Queue, Queue -> Queue; /* concatenation */
Extract : Queue, Integer -> Item; /* get item from queue */
Modify : Queue, Integer, Item -> Queue; /* modify value of item in queue */
SubO : Queue, Integer, Integer -> Queue; /* queue of length j starting at i-th item */
Qsearch : Queue

axioms for all item0 in Item ( for all q, q1, q2, q3 in Queue ( for all i, j in Integer (
q == Emptyqueue = q ? Emptyqueue // q == q ;
(q1 // q2) // q3 == q1 // (q2 // q3) ;
/* definition of Length by applying it to all constructors */
type Queue Length(Emptyqueue) == 0 ;
type Queue Length(MkQueue(item0)) == 1 ;
type Queue Length(q1 // q2) == Length(q1) + Length(q2) ;
/* definition of Extract by applying it to all constructors, Error! cases handled separately */
Extract(MkQueue(item0), 0) == Item0 ;
i > Length(q1) ==> Extract(q1 // q2, i) == Extract(q1, i) ;
Length(q1) >= i > 0 ==> Extract(q1 // q2, i) == Extract(q2, i - Length(q1)) ;
/* definition of First by other operations */
First(q) == Extract(q, 0) ;
Last(q) == Extract(q, Length(q) - 1) ;
/* definition of SubO(q, i, j) by induction on j, Error! cases handled separately */
j < 0 or j > Length(q) ==> SubO(q, i, j) == Error ;
j >= 0 and j > 0 and j <= Length(q) => SubO(q, i, j) == Emtyqueue ;
MaxQueue(Extract(q, j+1)),
/* definition of Modify by other operations */
Modify(q, i, item0) == SubO(q, i, 0) // MkQueue(item0) // SubO(q, i+1, Length(q) - 1) ;
Head(q) == SubO(q, 0, 0, Length(q)-1),
Tail(q) == SubO(q, 1, Length(q)-1) ;
Qfirst(q, Item0) == MkQueue(Item0) // q ;
Qlast(q, Item0) == q // MkQueue(Item0) ;
endgenerator Queue ;

50
Package macsorts

The FragSdu structure is for OUTGOING MSDUs and MMPDUs (generically SDUs). Each SDU, even if not fragmented, is held in an instance of this structure, awaiting its (re)transmission attempt(s). Transmit queue(s) are ordered lists of FragSdu instances. Depending on station capabilities and BSS type, there may be one or more such queues.

newtype FragSdu Struct

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fTot</td>
<td>number of fragments in pdus FragArray</td>
</tr>
<tr>
<td>fCur</td>
<td>number of next fragment to be sent</td>
</tr>
<tr>
<td>fAnc</td>
<td>number of next fragment to announce in ATIM or TIM</td>
</tr>
<tr>
<td>eol</td>
<td>0 until first Tx attempt, then = (now+DUsec(aMaxTxMsduLifetime))</td>
</tr>
<tr>
<td>sqf</td>
<td>SDU sequence number (set, along with eol, at first Tx attempt)</td>
</tr>
<tr>
<td>src</td>
<td>short retry counter</td>
</tr>
<tr>
<td>src</td>
<td>long retry counter</td>
</tr>
<tr>
<td>dst</td>
<td>destination address</td>
</tr>
<tr>
<td>grpa</td>
<td>=true if RA (not DA) is group address</td>
</tr>
<tr>
<td>psm</td>
<td>=true if RA (not DA) is not known to be active</td>
</tr>
<tr>
<td>cnfTo</td>
<td>address of process to send confirmation</td>
</tr>
<tr>
<td>pfPriority</td>
<td>requested priority (from LLC)</td>
</tr>
<tr>
<td>pdus</td>
<td>array of Frame to hold fragments</td>
</tr>
</tbody>
</table>

newtype FragArray Array(FragNum,Frame); endnewtype FragArray;

newtype SduQueue Queue(FragSdu,emptyQ);

adding operators

Qsearch : SduQueue,MacAddr -> Integer;
Operator Qsearch;

index:= Qsearch(queue, addr)

where queue is an SduQueue and index is returned to identify the first queue entry at which entry!dst = addr; of as = 1 if no match found (or for and empty queue).
The PartialSdu structure is for INCOMPLETE MSDUs and MMPDUs (generically SDUs) for which at least 1 fragment has been received. Unfragmented SDUs are reported upward immediately, and are never stored in instances of this structure. */

newtype PartialSdu Struct
mute Boolean; /*true if >=1 fragments are buffered in this instance of PartialSdu */
rta MacAddr; /* transmitting station (Addr2) */
seq SeqNum; /* Mdud/Mmpdu sequence number */
rCur FragNum; /* Mpdud fragment number of most recent fragment */
rsn SeqNum; /* Msdu/Mmpdu sequence number */
rno SeqNum; /* Fsdu frame */
rtatime (now+Usec(aMaxReceiveLifetime)) at first fragment */
eol Time; /* ((now+Usec(aMaxReceiveLifetime)) at first fragment */
rsdu Frame; /* Msdu concatenated into this buffer, Mac header from first Mpdud */
endnewtype PartialSdu ;

} /*
   Number of entries in reassembly array at this station, and associated index range */
synonym reasmSize Integer = 6; /* number of reassembly buffers (minimum 3) is impl. dep */
synonym reasmIndex Integer constants 1:reasmSize endsynonym reasmIndex ;

} /*
   Array of PartialSdu for use reassembling fragmented Mdudus and Mmpdus.
   Searchable using the AsSearch operator

   index:=ArSearch(array, addr, seq, frag)
   where array is a ReasmArray and index is returned to identify the first element
   for which (inUse=true) and (entrylrta=addr) and (entrylrtn=seq) and (entrylrCur=frag-1);
   or as =1 if no match found.

   index:=ArFree(array) returns index of free entry, or -1 if none
   array:= ArAge(array, age) frees entry!eol < age (also used to clear array). */

newtype ReasmArray array(reasmIndex, PartialSdu) ;

adding operators

ArSearch : ReasmArray,MacAddr,SeqNum,FragNum -> Integer;
ArFree : ReasmArray -> Integer;
ArAge : ReasmArray,Time -> ReasmArray;

Operator ArSearch; fpar ar ReasmArray, adr MacAddr, seq SeqNum,
frag FragNum; returns Integer; referenced;
Operator ArFree; fpar ar ReasmArray; returns Integer; referenced;
Operator ArAge; fpar ar ReasmArray, age Time; returns ReasmArray; referenced;
endnewtype ReasmArray ;

/*
   SORTS for POWER MANAGEMENT SUPPORT */

/*
   define StationId */
synonym sMaxSld Integer = 2007 ;
synonym sStTableSize Integer = 2008 ;

} /*
   StationId table searchable by MacAddr
   index:= addrTold(table, addr)
   where table is a SldTable, returns the first index value where
   the table entry is equal to addr, or -1 if no match found. */

newtype SldTable Array(StationId,MacAddr);

adding operators

addrTold : SldTable,MacAddr -> Integer;
operator addrTold; fpar tbl SldTable, Val MacAddr; returns Integer;
endnewtype SldTable ;
Package macsorts

-- - l

~

(I

operator \[ 1

~

keyLookup

, . SORTS FOR CRC-32 (CRC and ICV values) */

newtype Crc inherits Octetstring operators all;

adding operators
crc32 : Crc, Octet -> Crc;
mirror : Crc -> Octetstring;
operator crc32 ;
    fpar crcin Crc, val Octet; returns Crc; referenced;
axioms for all c in Crc
    mirror(c) == S4(flip(c(3)), flip(c(2)), flip(c(1)), flip(c(0))) : ) ;
endnewtype Crc ;

/* Initial Crc value (all 1s) */
synonym initCrc Crc = S4(0xFF, 0xFF, 0xFF, 0xFF) ;

/* Valid Crc value after accumulation of Crc32 on Pdu including Crc field */
synonym goodCrc Crc = S4(0x7B, 0xDD, 0x04, 0x7C) ;

** SORTS for WIRED-EQUIVALENT PRIVACY (WEP) */

** define length of KeyVector and nullKey value */
syntype Keyindex = Integer constants 1:4 endsyntype KeyIndex ;
synonym nullKey Octetstring = 03 // 02 ;
syntype PrngKey = Octetstring default 03 // 02 endsyntype PrngKey ;

/* KeyVector for default WEP keys. Array of Octetstring indexed by KeyIndex. 
Length(KeyVector(n)) must =5, it is assumed that the MlmeSet.request enforces this. */
newtype KeyVector Array(KeyIndex, PrngKey) ; endnewtype KeyVector ;

/* Number of entries in aWepKeyMapping array at this station. 
Actual length of key map array (10 is the minimum per 8.3.2). */
synonym sWepKeyMappingLength Integer = 10 ;
syntype KeyMappingRange = Integer constants 1:sWepKeyMappingLength endsyntype KeyMappingRange ;

/* KeyMap structure -- used as elements of KeyMapArray 
Length(wepKey) must =5, it is assumed that the MlmeSet.request enforces this. */
newtype KeyMap struct
    mappedAddr MacAddr; keyOn Boolean; wepKey PrngKey; endnewtype KeyMap ;

/* KeyMapArray -- used for aWepKeyMapping table; an array of KeyMap indexed 
by KeyMappingRange. Procedural operator keyLookup searches the array for the 
mapping of a given address: keyMap := keyLookup(addr, keyMapArray, keyMapArrayLength) 
If an entry is found with mappedAddr=addr, keyMap is set to the value of this entry. 
If no entry is found with mappedAddr=addr, keyMap is set to (. addr, false, null) */
newtype KeyMapArray Array(KeyMappingRange, KeyMap) ;

adding operators
keyLookup : MacAddr, KeyMapArray, Integer -> KeyMap;
operator keyLookup ;
    fpar luadr MacAddr, kma KeyMapArray, kml Integer; returns KeyMap; referenced;
endnewtype KeyMapArray ;
Package macsorts

newtype Frame inherits Octetstring operators all:

adding operators
mkFrame : TypeSubtype, MacAddr, MacAddr, Octetstring -> Frame;
mkCtl : TypeSubtype, Octetstring, MacAddr -> Frame;
protocolVer : Frame -> Integer;
basetype : Frame -> BaseType;
 setType : Frame -> TypeSubtype;
toDs : Frame -> Bit;
mToDs : Frame, Bit -> Frame;
moreFrag : Frame -> Bit;
retbyBit : Frame -> Bit;
setRetyBit : Frame, Bit -> Frame;
pwrMgt : Frame -> Bit;
setPwrMgt : Frame, Bit -> Frame;
moreData : Frame -> Bit;
setMoreData : Frame, Bit -> Frame;
wepBit : Frame -> Bit;
setWepBit : Frame, Bit -> Frame;
orderBit : Frame -> Bit;
setOrderBit : Frame, Bit -> Frame;
durId : Frame -> Integer;
setDurId : Frame, Integer -> Frame;
add1 : Frame -> MacAddr;
setAdd1 : Frame, MacAddr -> Frame;
add2 : Frame -> MacAddr;
setAdd2 : Frame, MacAddr -> Frame;
add3 : Frame -> MacAddr;
setAdd3 : Frame, MacAddr -> Frame;
add4 : Frame -> MacAddr;
insAdd4 : Frame, MacAddr -> Frame;
seq : Frame -> SeqNum;
setSeq : Frame, SeqNum -> Frame;
frag : Frame -> FragNum;
setFrag : Frame, FragNum -> Frame;
ts : Frame -> Integer;
setTs : Frame, Integer -> Frame;
status : Frame -> StatusCode;
setStatus : Frame, StatusCode -> Frame;

axioms on next page
"Frame operator signatures continue */
authStat : Frame -> StatusCode; /* Status Code field in Authentication frame */
reason : Frame -> ReasonCode; /* Reason Code field (2 octets) */
authSeqNum : Frame -> Integer; /* Authentication Sequence Number field (2 octets) */
authAlg : Frame -> AuthType; /* Authentication Algorithm field (2 octets) */
beaconInt : Frame -> Kusec; /* Beacon Interval field (2 octets) */
listenInt : Frame -> Kusec; /* Listen Interval field (2 octets) */
setAsgnSId : Frame, Stationld -> Frame;
curApAddr : Frame -> MacAddr; /* Current AP Address field (6 octets) */
capA : Frame, Capability -> Bit; /* Capability Info field, AsoclReasoc (2 octets) */
setCapA : Frame, Capability, Bit -> Frame;
capB : Frame, Capability, Bit -> Frame; /* Capability Info field, BeaconProbeRsp (2 octets) */
setCapB : Frame, Capability, Bit -> Frame;
authSeqNum : Frame -> Integer; /* Authentication Sequence Number field (2 octets) */
keyId : Frame -> KeyIndex; /* Key ID subfield of ICV field (2 bits of 1 octet) */
setKeyId : Frame, KeyIndex -> Frame;
operator getElem ; fpar fr Frame, id ElementlD; returns Frame: referenced: */
axioms for Frame sort */
for all a, sa, ds, ra, ta, bssa in MacAddr (}
for all body, dur, sid, info in Octetstring |
addr1(f) == substr(f,4,6); /* setAddr1(f,a) == substr(f,0,4) // a // substr(f,10,length(f)-10) ;
addr2(f) == substr(f,10,6); /* setAddr2(f,a) == substr(f,0,10) // a // substr(f,16,length(f)-16) ;
addr3(f) == substr(f,16,6); /* setAddr3(f,a) == substr(f,0,16) // a // substr(f,22,length(f)-22) ;
addr4(f) == substr(f,22,6); /* insAddr4(f,a) == substr(f,0,22) // a // substr(f,28,length(f)-28) ;
curApAddr(f) == substr(f,28,6); /*
for all f in TypeSubtype (}
mkFrame(lt,da,bssa,body) == It // O3 // da // aMacAddress // bssa // O2 // body ;
(f = rm) ==> mkCmd(f,da,ra) == fl // O1 // ra // fr // aMacAddress ;
(f = ps-poll) ==> mkCmd(f,sid,bssa) == fl // O1 // sid // bssa // aMacAddress ;
(f = cts) or (f = ack) ==> mkCmd(f,da,ra) == fl // O1 // ra ;
(f = cts) or (f = ack) ==> mkCmd(f,sid,bssa) == fl // O3 // ra // bssa ;
setFtype(f,lt) == Modify!(f,O,mkstring«f(O) and Ox03) or It»; ;
for all bt in BasicType (}
basetype(f) == (O) and OxOC; ;
for all i in Integer (}
protocolVer(f) == octetVal(f(0) and Ox0O) ;
durId(f) == octetVal(f(2)) + octetVal(f(3)) * 256 ;
setDurld(f) == substr(f,0,2) // mkOS( mod 256,1) // mkOS(f / 256,1) // substr(f,4,length(f)-4) ;
authSeqNum(f) == octetVal(f(26)) + octetVal(f(27)) * 256 ;
for all e in ElementId (}
mkElem(e,info) == e // mkOS(length(info)+2 // info ) ;
for all b in Bit |
setToDs(f) = if (f(1) and Ox01) then 1 else 0 fi ;
setCapA(f) = Modify!(f(1),f(1) and Ox04) or Sb(0,0,0,0,0,0,0,0,0,0,b) ;
setFrDs(f) = if (f(1) and Ox02) then 1 else 0 fi ;
setFragl(f) = if (f(1) and Ox04) then 1 else 0 fi ;
setFrag2(f) = if (f(1) and Ox02) then 1 else 0 fi ;
retryBit(f) = if (f(1) and Ox0B) then 1 else 0 fi ;
setRetryBit(f) = Modify!(f(1),f(1) and Ox07) or Sb(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) ;
pwrMgt(f) = if (f(1) and Ox10) then 1 else 0 fi ;
/* axioms continue on next page */
axioms for Frame sort, continued

setPwrMgt(I,b) == Modify((l,l,(l(1) and OxFB) or SB(O,O,O,b,O,O,O,O»
moreData(I) == if ((l(1) and Ox20) then 1 else 0 fi;
setMoreData(l,b) == Modify((l,l,(l(1) and OxFB) or SB(O,O,O,b,O,O,O,O»
wepBit(l) == if ((l(1) and Ox40) then 1 else 0 fi;
setWepBit(l,b) == Modify((l,l,(l(1) and OxFB) or SB(O,O,O,b,O,O,O,O»
orderBit(l) == if ((l(1) and Ox80) then 1 else 0 fi;
setOrderBit(l,b) == Modify((l,l,(l(1) and OxFB) or SB(O,O,O,b,O,O,O,O»
for all c in Capability
  capA(f,c) == if (bitstring(substr(l,24,2)) and 8) then 1 else 0 fi;
  setCapA(f,c,b) == substr((l,24,2) + 8) subst((l,24,2) and (not 8)) or
  (if 9 then 8 else 0 fi) subst((l,26,length(f)-26) ;
capB(l,c) == if (bitstring(substr(l,34,2)) and 8) then 1 else 0 fi;
  setCapB(f,c,b) == substr((l,34,2) + 8) subst((l,34,2) and (not 8)) or
  (if 9 then 8 else 0 fi) subst((l,36,length(f)-36) ;
for all sq in SeqNum
  seq(l) == subst((l,0,22) subst((l,22,0) and OxFO) subst((l,23,0) and Ox0F)
  setSeq(l,sq) == subst((l,0,22) subst((l,22,0) and Ox0F) subst((l,22,0) and Ox0F)
  order(l) == subst((l,26) subst((l,26,0) and Ox80) subst((l,26,0) and Ox80)
  setOrder(l) == subst((l,26) subst((l,26,0) and Ox80) subst((l,26,0) and Ox80)
  for all fr in FragNum
    frag(l) == subst((l,0,22) subst((l,22,0) and OxFO) subst((l,22,0) and Ox0F)
    setFrag(l,fr) == subst((l,0,22) subst((l,22,0) and Ox0F) subst((l,22,0) and Ox0F)
    for all tm in Time
      tset(l) == subst((l,0,22) subst((l,22,0) and OxFO) subst((l,22,0) and Ox0F)
      setTset(l,tm) == subst((l,0,22) subst((l,22,0) and Ox0F) subst((l,22,0) and Ox0F)
      for all sta in StationId
        aSet(l) == subst((l,0,22) subst((l,22,0) and OxFO) subst((l,22,0) and Ox0F)
        setASet(l,sta) == subst((l,0,22) subst((l,22,0) and Ox0F) subst((l,22,0) and Ox0F)
        for all key in KeyIndexRange
          keyset(l) == subst((l,0,22) subst((l,22,0) and OxFO) subst((l,22,0) and Ox0F)
          setKeyset(l,key) == subst((l,0,22) subst((l,22,0) and Ox0F) subst((l,22,0) and Ox0F)
endnewtype Frame ;
```
package macsorts

addframe

3118_bIFrameTypes(23)

--. ADDITIONAL FRAME FORMAT SORTS *

newtype TypeSubtype inherits Octetstring operators all;
adding literals

dassoc, dassoc_rsp, dassoc_req, probe_req, 
probe_rsp, beacon, atim, disassoc, auth, 
dauth, ps_poll, rts, cts, ack, 
cfend, cfend_ack, data, data_ack, data_poll, 
data_poll_ack, null_frame, cfack, cfpoll, cfpoll_ack;
axioms

asoc_req == mkstring(S8(0,0,0,0,0,0,0,0)); 
asoc_rsp == mkstring(S8(0,0,0,0,0,1,0,0)); 
reasoc_req == mkstring(S8(0,0,0,0,0,0,1,0)); 
reasoc_rsp == mkstring(S8(0,0,0,0,0,1,1,0)); 
probe_req == mkstring(S8(0,0,0,0,0,0,0,1)); 
probe_rsp == mkstring(S8(0,0,0,0,0,1,0,1)); 
broadcast == mkstring(S8(0,0,0,0,0,0,0,0)); 
auth == mkstring(S8(0,0,0,0,0,0,0,0)); 
dauth == mkstring(S8(0,0,0,0,0,0,0,0));

data_ack == mkstring(S8(0,0,0,0,0,0,0,0)); 
data_poll == mkstring(S8(0,0,0,0,0,0,0,0)); 
data_poll_ack == mkstring(S8(0,0,0,0,0,0,0,0)); 
ack == mkstring(S8(0,0,0,0,0,0,0,1)); 
null_frame == mkstring(S8(0,0,0,0,0,0,0,0));

cfack == mkstring(S8(0,0,0,1,0,1,0,1)); 
cfpoll == mkstring(S8(0,0,0,1,1,1,1,1)); 
cfpoll_ack == mkstring(S8(0,0,0,1,1,1,1,1));

newtype BasicType inherits Bitstring operators all;
adding literals

control, data, management, reserved;
axioms

control == S8(0,0,0,1,0,0,0,0); 
data == S8(0,0,0,1,0,0,0,0); 
management == S8(0,0,0,1,0,0,0,0); 
reserved == S8(0,0,0,1,1,0,0,0);
endnewtype BasicType :

-- ELEMENT IDS *

newtype ElementId inherits Octetstring operators all;
adding literals

eSsid, eSupRates, eFhParms, eDsParms, 
eCfParms, eTim, eBfParms, eCtext;
axioms

eSsid == mkOS(0,1); /* service set identifier (length= 0-32) */
eSupRates == mkOS(1,1); /* supported rates (length= 1-8) */
eFhParms == mkOS(2,1); /* FH parameter set (length= 5) */
eDsParms == mkOS(3,1); /* DS parameter set (length= 1) */
eCfParms == mkOS(4,1); /* CF parameter set (length= 6) */
eTim == mkOS(5,1); /* Traffic Information Map (length= 4-254) */
eBfParms == mkOS(6,1); /* IBSS parameter set (length= 2) */
eCtext == mkOS(16,1); /* challenge text (length= 1-253 [7.3.2.8], 128 [8.1.2.2] */
endnewtype ElementId :
```

57
Package macsorts

REASON CODES

newtype ReasonCode inherits Octetstring operators all;
adding literals
  unspec_reason, auth_not_valid, deauth_lv_ss, inactivity, ap_overload,
  class2_err, class3_err, disass_lv_ss, assoc_not_auth;
axioms
  unspec_reason = mkOS(1,2); auth_not_valid = mkOS(2,2); deauth_lv_ss = mkOS(3,2);
  inactivity = mkOS(4,2); ap_overload = mkOS(5,2); class2_err = mkOS(6,2);
  class3_err = mkOS(7,2); disass_lv_ss = mkOS(8,2); assoc_not_auth = mkOS(9,2);
endnewtype ReasonCode;

STATUS CODES

newtype StatusCode inherits Octetstring operators all;
adding literals
  successful, unspec_fail, unsup_cap, reassoc_no_asoc, fail_other, unsup_alg,
  auth_seq_fail, ching_fail, auth_timeout, ap_full, unsup_rate;
axioms
  successful = mkOS(0,2); unspec_failure = mkOS(1,2); unsup_cap = mkOS(10,2);
  reassoc_no_asoc = mkOS(11,2); fail_other = mkOS(12,2); unsup_alg = mkOS(13,2);
  auth_seq_fail = mkOS(14,2); ching_fail = mkOS(15,2); auth_timeout = mkOS(16,2);
  ap_full = mkOS(17,2); unsup_rate = mkOS(18,2);
endnewtype StatusCode;

CAPABILITY FIELD BITS

newtype Capability inherits Bitstring operators all;
adding literals
  cEss, cIBSS, cPollable, cPollReq, cPrivacy;
axioms
  cEss = S8(1,0,0,0,0,0,0,0,0,0,0,0); /* ESS capability */
  cIBSS = S8(0,1,0,0,0,0,0,0,0,0,0,0); /* IBSS capability */
  cPollable = S8(0,0,1,0,0,0,0,0,0,0,0,0); /* CF-pollable (sta), PC present (ap) */
  cPollReq = S8(0,0,0,1,0,0,0,0,0,0,0,0); /* (not) CF poll req (sta), PC polls (ap) */
  cPrivacy = S8(0,0,0,0,1,0,0,0,0,0,0,0); /* WEP required capability */
endnewtype Capability;
IBSS PARAMETER SET ELEMENT

newtype IbssParms inherits Octetstring operators all;

adding operators

atimWin : IbssParms -> Kusec;
setAtimWin : IbssParms, Kusec -> IbssParms;

axioms

for all ib in IbssParms ( for all u in Kusec ( 
  atimWin(ib) == octetVal(ib(0)) + (octetVal(ib(1)) * 256) ;
  setAtimWin(ib,u) == mkOS(u mod 256, 1) // mkOS(u / 256, 1) ; ) ) ;
endnewtype IbssParms ;

CF PARAMETER SET ELEMENT

newtype CfParms inherits Octetstring operators all;

adding operators

cfpCount : CfParms -> Integer; /* Cfp Count field, 1 octet */
setCfpCount : CfParms, Integer -> CfParms;
cfpPeriod : CfParms -> Integer; /* Cfp Period field, 1 octet */
setCfpPeriod : CfParms, Integer -> CfParms;
cfpMaxDur : CfParms -> Kusec; /* Cfp Max Duration field, 2 octets */
setCfpMaxDur : CfParms, Kusec -> CfParms;
cfpDurRem : CfParms -> Kusec; /* Cfp Dur Remaining field, 2 octets */
setCfpDurRem : CfParms, Kusec -> CfParms;

axioms

for all cf in CfParms ( for all i in Integer ( for all u in Kusec ( 
  cfpCount(cf) == octetVal(cf(0)) ;
  setCfpCount(cf,i) == mkOS(i,1) // tail(cf) ;
  cfpPeriod(cf) == octetVal(cf(1)) ;
  setCfpPeriod(cf,i) == cf(O) // mkOS(i,1) // substr(cf,2,4) ;
  cfpMaxDur(cf) == octetVal(cf(2)) + (octetVal(cf(3)) * 256) ;
  setCfpMaxDur(cf,u) == substr(cf,0,2) // mkOS(u mod 256, 1) // mkOS(u / 256, 1) // substr(cf,4,2) ;
  cfpDurRem(cf) == octetVal(cf(4)) + (octetVal(cf(5)) * 256) ;
  setCfpDurRem(cf,u) == substr(cf,0,4) // mkOS(u mod 256, 1) // mkOS(u / 256, 1) ; ) ) ;
endnewtype CfParms ;

FH PARAMETER SET ELEMENT

newtype FhParms inherits Octetstring operators all;

adding operators

dwellTime : FhParms -> Kusec; /* Dwell Time field, 2 octets */
sdTwellTime : FhParms, Kusec -> FhParms;
hopSet : FhParms -> Integer; /* Hop Set field, 1 octet */
setHopSet : FhParms, Integer -> FhParms;
hopPattern : FhParms -> Integer; /* Hop Pattern field, 1 octet */
setHopPattern : FhParms, Integer -> FhParms;
hopIndex : FhParms -> Integer; /* Hop Index field, 1 octet */
setHopIndex : FhParms, Integer -> FhParms;

axioms

for all fh in FhParms ( for all i in Integer ( for all u in Kusec ( 
  dwellTime(fh) == octetVal(fh(0)) + (octetVal(fh(1)) * 256) ;
  setDwellTime(fh,u) == mkOS(u mod 256, 1) // mkOS(u / 256, 1) // substr(fh,2,3) ;
  hopSet(fh) == octetVal(fh(2)) ;
  setHopSet(fh,i) == substr(fh,0,2) // mkOS(i,1) // substr(fh,3,2) ;
  hopPattern(fh) == octetVal(fh(3)) ;
  setHopPattern(fh,i) == substr(fh,0,3) // mkOS(i,1) // last(fh) ;
  hopIndex(fh) == octetVal(fh(4)) ;
  setHopIndex(fh,i) == substr(fh,0,4) // mkOS(i,1) ; ) ) ;
endnewtype FhParms ;
Operator clearTupleCache

\[ \text{clearTupleCache} \]
\[ \text{returns TupleCache} \]

\begin{align*}
\text{k} &:= 1 \\
\text{k} &:= k + 1 \\
\text{cache(k)} &:= \text{false} \\
\text{if} & \quad (\text{tupleCacheSize}) \\
\text{Mark all cache entries as empty.}
\end{align*}

\text{dot k CacheIndex}
Operator searchTupleCache

par
cache TupleCache,"
taddr MacAddr,
tseq SeqNum,
tfrag FragNum;
returns Boolean;

Search for exact
(TA,SeqNum,FragNum)
match at non-empty
cache entries.

result =
(cache(k)tta =
taddr) and
(result =
(cache(k)tseq =
seq) and
(cache(k)tfrag =
frag)
and cache(k)tlfull

k = 1
k = k + 1

(result =
true)

else
(result =
false)

k = tupleCacheSize
Operator updateTupleCache

del k, oldest CacheIndex;
dcl test Boolean;
dcl age Time;
dcl temp Tuple;

if k > oldest CacheIndex
    test = (temp.taddr = taddr) and (temp.tseq = tseq)
    if test
        if temp.fragNum = fragNum and temp.trae = tRx
            return TupleCache
    else
        k = k + 1
        age = now, k = 1, oldest = 1
        temp.full = true
        test = (temp.full = true) and (temp.full = true)
        if test
            oldest = k
            age = temp.age
        else
            oldest = k, age = 0
        =tupleCacheSize
        temp.full = true, temp.taddr = taddr, temp.tseq = tseq
        temp.full = true, temp.age = age
        temp.trae = temp.trae
        temp.fragNum = fragNum
        temp.trae = tRx
        cache(oldest) = temp

If a match is found with (TA, SeqNum) and FragNum
and TRe for that entry rather than creating a new
(redundant) entry:

Create new entry if no (TA, SeqNum) match.
If possible, use empty location, else replace
entry in least-recently
updated location.
Operator Crc32

`fpar
| crcin Crc;
| val Octet;
| returns Crc;
``

```c
int k, new, result, temp;

int bit_string(int new) {
    int head = (new & 1) ? 1 : 0;
    int tail = (new & 2) ? 1 : 0;
    return head ^ tail;
}

int crcin, val, next;

int Crc;

int calcl:

new = (new + 1) & 3;
next = (new + 2) & 3;

k = (k + 1) & 31;

new = val(k) xor last(temp);

result = (temp xor feedback) + bit_string(new);
```

```
// Bitstring with 1s at bit positions with feedback terms in CRC-32 polynomial */
synonym feedback Bitstring =
S8(0,1,1,0,1,1,0,1) //
S8(1,0,1,1,1,0,0,0) //
S8(1,0,0,0,0,0,1,1) //
S8(0,0,1,0,0,0,0,0) ;
```
Operator keyLookup

- [par luadr MapAddr;
  kma KeyMapArray;
  kmk Integer;
  returns KeyMap;
]

```plaintext
    (false)
    \[\text{result} = \text{keyMap}(k)\]
    \[\text{mappedAddr} := \text{luadr}[\text{kma}(k)]\]
    \[\text{lk} := \text{lk} + 1\]
    \[\text{idr} := \text{idr} + 1\]
    \[\text{result} := \text{idr} \text{KeyMap}\]
    \[\text{resultKeyOn} := \text{false}, \text{resultWepKey} := \text{nullKey}\]
    \(\text{result} \text{mappedAddr} := \text{luadr}\)
    \(\text{resultKeyOn} := \text{false}, \text{resultWepKey} := \text{nullKey}\)
```

- Return first KeyMap element with correct mappedAddr value.

- Return false KeyOn status and a null WepKey if the end of the array is reached with idr not found.
Operator \( \text{gelElem} \)

\[
\text{gelElem}(\text{Frame}, \text{id.ElementId}) \rightarrow \text{Frame}
\]

\[
\text{let}
\begin{align*}
  n &:= \text{length(Frame)} \\
  \text{type}(\text{Frame}) &\text{ else }
\end{align*}
\]

\[
\begin{align*}
  k &:= k + 6 \\
  k &:= k + 0 \\
  k &:= k + 12 \\
  k &:= k + 10 \\
  k &:= k + 4
\end{align*}
\]

\[
\begin{align*}
  \text{info} &:= \text{null} \\
  \text{info} &:= \text{substr(Frame)}(k, 1, n-k) \\
  \text{info} &:= \text{substr(frame)}(k, 1, n-k)
\end{align*}
\]

\[
\text{del Integer}:= sMacHdrLn
\]
Operator Qsearch

```scala
def Qsearch(q: SduQueue, macAddr: Int): Int = {
  def k: Integer =
  if (q.isEmpty) return -1
  val mac = q.dequeue()
  if (mac == macAddr) return k
  k = k + 1
  if (k < q.length) Qsearch(q, macAddr)
  result = k
```

Diagram:

- Decision: `q.isEmpty` (true) → `result = -1`
- Decision: `q.isEmpty` (false)
  - Decision: `mac = q.dequeue()` (true) → `k = 0`
  - Decision: `mac = q.dequeue()` (false)
    - Decision: `k < q.length` (true) → `k = k + 1`
    - Decision: `k < q.length` (false) → `result = k`
This SDL-92 rendition of the MAC MIB, plus portions of the PHY MIB, exists to permit analysis of the MAC state machines without having full Z.105 support in the SDL tool. However, some of the comments interspersed with the remote variable definitions are useful as explanations of the relationships between MIB attributes and state machine functions. Also, there are cases where attributes in this rendition differ from the ASN.1, and the reasons for the differences are explained in comments.

```
/* STATION_CONFIG_GROUP */
/*
 * WARNING: aStationID is not currently used in MAC state machines because, as defined, this attribute appears to serve no useful purpose, nor be able to alter MAC behavior. The stated purpose of aStationID is to allow renaming of a station, overriding aMacAddress. However, a read-only attribute with value=aMacAddress, does not achieve this.
 * Read-only; default=aMacAddress; declared in <nowhere>; used by <none>; */
remote aStationID MacAddr nodelay;
/*
 * Maximum time in Kusec before a point coordinator must relinquish the medium. Only used by PCF option at APs.
 * Read-write; default=100 [max=1000, not checked]; declared in MIB; used in <APM>Sync, [CTL/PCF]; */
remote eMediumOccupancyLimit Kusec nodelay;
/*
 * True if station wakes up to receive every Beacon frame which contains a DTIM element.
 * Read-write; default=true; declared in MIB; used in Synchronization; */
remote aReceiveDTIMs Boolean nodelay;
/*
 * Identifies authentication algorithm used during most recent, successful authentication sequence.
 * Written by the MlmeAuthenticate.Request handler in process Mlme_Req_Rsp, but never read by the
 * MAC state machines, so the benefit of writing to this attribute from an external entity are unclear.
 * Read-write; default=1 [different encoding than the Mime SAP AuthType]; declared in MIB; used in Mlme_Req_Rsp; */
remote aAuthenticationType AuthType nodelay;
/* end of StationConfigGroup */

/* AUTHENTICATION_ALGORITHMS_TABLE */
/*
 * Table of the authentication algorithms supported by this station. Since this table is used solely to determine whether
 * a requested algorithm is supported, the ordering of entries is irrelevant to MAC operation. Therefore, this table is
 * declared as an SDL set, which enhances the readability of the authentication requester and responder descriptions.
 * NOTE: Do NOT put shared_key in this list if aPrivacyOptionImplemented=false.
 * Read-only; default=(open_system (=1), shared_key (=2)); declared in MIB; used in Mlme_Req_Rsp; */
remote aAuthenticationAlgorithms AuthTypeSet nodelay;
/* end of AuthenticationAlgorithmsTable */

/* DEFAULT_WEP_KEY_TABLE -- only if aPrivacyOptionImplemented = true */
/*
 * 4-element vector of default WEP key values, corresponding to the 4 possible values in the keyid field (0-3).
 * WARNING: The ASN.1 keys are OctetString Size(8), whereas key length is specified in 8.2.3 as 40 bits (5 octets).
 * Also, to achieve privacy, WEP KEYS MUST BE WRITE-ONLY, not read-write as in the ASN.1.
 * Write-only; default=NULL; declared in MIB; used in Encrypt, Decrypt; */
remote aDefaultWepKeys KeyVector nodelay; /* name changed to plural because the attribute holds 4 keys */
/* end of DefaultWepKeyTable */
```

**WEP_KEY_MAPPING_TABLE** -- only if aPrivacyOptionImplemented=true

```
Array of KeyMap entries, each of which is (mappedAddr MacAddr, keyOn Boolean, wepKey Octets!ln!l). aWepKeyMappingLength defines the number of entries. Conceptually array elements are (keyOn, wepKey), indexed by MAC address, but the formal description adds mappedAddr to each element, and defines procedure keyLookup to search the array. This is because octets in MAC addresses have no magnitude significance, so using MacAddr as an array index implies the existence of arithmetic properties unique to 802.11 MAC addresses. Also, the small required size array this indicates a clear intention that this be a sparse mapping from the MAC address space.

WARNING: The ASN.1 is incorrect, mapping addresses to default key index values, and allowing read-write access.
```

**PRIVACY_GROUP** -- only first attribute needed if aPrivacyOptionImplemented=false

```
A static indicator of whether the privacy option (WEP and shared key authentication) is implemented at the station. This attribute is true if the privacy option is implemented, default=false because privacy is optional.
```

**aWepDefault KeyIndex**

```
True to turn on WEP encryption, false to turn off WEP encryption.
```

**aExcludeUnencrypted**

```
Count of received WEP frames with acceptable address and CRC values, that are rejected due to ICV errors.
```

---

68

/* OPERATION_GROUP */

RTS/CTS handshake not performed prior to transmitting an Mpdru or Mmpdu if length(pdu) <= aRtsThreshold.
read-write; default=3000; declared in MIB; used in Prepare_Mpdru, Transmit Control; /* remote aRtsThreshold Integer nodelay; */

The maximum number of retransmission attempts for an Mpdru or Mmpdu whose length(pdu) <= aRtsThreshold.
read-write; default=7; declared in MIB; used in Prepare_Mpdru, PM_Filter, Transmit Control; /* remote aShortRetryLimit Integer nodelay; */

The maximum number of retransmission attempts for an Mpdru or Mmpdu whose length(pdu) > aRtsThreshold.
read-write; default=4; declared in MIB; used in Prepare_Mpdru, PM_Filter, Transmit Control; /* remote aLongRetryLimit Integer nodelay; */

remote aFragmentationThreshold Integer nodelay;

/* GROUP_ADDRESSES_TABLE */

Table of the group addresses to be accepted in frames received by this station. Since this table is used solely to determine whether a given group address is to be received, the ordering of entries is irrelevant to MAC operation. Therefore, this table is declared as a subtype of an SDL set, which enhances the readability of the receive filtering description. The subtype of set defines a modified 'in' operator which returns true for the broadcast address as well as for successful membership tests.
read-write; default=null; declared in MIB; used in Filter_MPDU; /* remote aGroupAddresses MacAddrSet nodelay; */

end of OperationGroup */

COUNTERS_GROUP

Count of Mpdus and Mmpdus successfully delivered (acknowledged or sent to a group address). This access and location information applies to the first 8 counters (through aAckFailureCount):

remote aTransmittedFrameCount Counter32 nodelay;

Count of Mpdus and Mmpdus transmitted with a group address in the addr1 field./*
remote aMulticastTransmittedFrameCount Counter32 nodelay;

Count of transmit attempts abandoned due to reaching either aShortRetryLimit or aLongRetryLimit. /*
remote aFailedCount Counter32 nodelay;

Count of Mpdus and Mmpdus successfully delivered (acknowledged) when transmitted with RetryBit=1. /*
remote aRetryCount Counter32 nodelay;

Count of Mpdus and Mmpdus successfully delivered (acknowledged) after more than 1 retransmission attempt. /*
remote aMultipleRetryCount Counter32 nodelay;

Count of instances when a CTS frame is received in response to transmission of an RTS frame. /*
remote aRtsSuccessCount Counter32 nodelay;

Count of instances when a CTS frame is not received in response to transmission of an RTS frame. /*
remote aRtsFailureCount Counter32 nodelay;

Count of instances when neither an ACK frame, nor a CF-Ack indication (during the contention free period) is received in response to transmission of a unicast Data-type frame. /*
remote aAckFailureCount Counter32 nodelay;

Count of Mpdus and Mmpdus with a group address in the addr1 field received successfully. See discussion under aReceivedFrameCount regarding other (non-address) criteria for successful reception. /*
remote aMulticastReceivedFrameCount Counter32 nodelay;

Count of receptions that failed due to detection of a CRC error, read-only; default=counts up from 0; declared in Validate_MPDU; used in MIB (MimeGet response); /*
remote aFcsErrorCount Counter32 nodelay;

Count of Mpdus and Mmpdus with the RetryBit=1 received successfully, then discarded as a duplicated frame. /*
remote aFrameDuplicateCount Counter32 nodelay;

end of CountersGroup

PHY_OPERATION_GROUP -- part 1 of 2

Identifies the type of PHY (FHSS 2.4GHz=01, DSSS 2.4GHz=02, IR baseband=03).
read-only; default=identifies type of PHY physically present; used in Sync, Transmit_Control; */
synonym aPHYType Integer = 01 ; /* 01 for FH PHY used here to exercise FH-specific MAC functions */
/*
Identifies regulatory domain the current instance of the PMD is supporting.
read-write; default=implementation dependent; used in <none>; */
remote aCurrentRegDomain Integer nodelay;
/*
Slot time in microseconds (static, PHY-dependent value). The slot time is the time which separates SIFS from PIFS and PIFS from DIFS. Also, the backoff time is decremented in units of slot time.
read-only; default=PHY-dependent; used in Filter_MPDU, Channel_State, Backoff; */
synonym aSlotTime package macsorts Usec = (aCcaTime + aRxTxTurnaroundTime + aAirPropagationTime + aMacProTime);
/*
Time for CCA mechanism to assess the medium during each slot.
read-only; default=27 for FH, 15 for DS, 5 for IR); used in aSlotTime equation; */
synonym aCcaTime Usec = 27 ; /* this value is for the FH PHY */
/*
Maximum time in microseconds for the PHY to change from receiving to the start of transmitting the first symbol (static, PHY-dependent value).
read-only; default=PHY-dependent; used in Validate_MPDU, Channel_State; */
synonym aRxTxTurnaroundTime Usec = (aTxPclpDelay + aRxTxSwitchTime + aTxRampOnTime + aTxRfDelay);
/*
Nominal time in microseconds for PLCP to deliver a symbol from the MAC interface to the PMD.
read-only; default=1 for FH, impl. dep. for DS, 60-140 for IR); used in aRxTxTurnaroundTime eqn; */
synonym aTxPclpDelay Usec = 1 ; /* this value is for the FH PHY */
/*
Nominal time in microseconds which the PMD takes to switch from receive to transmit.
read-only; default=10 for FH, 5 for DS, 0 for IR); used in aRxTxTurnaroundTime equation; */
synonym aRxTxSwitchTime Usec = 10 ; /* this value is for the FH PHY */
/*
Nominal time in microseconds to turn the transmitter on.
read-only; default=8 for FH, impl. dep. for DS, 0 for IR); used in aRxTxTurnaroundTime equation; */
synonym aTxRampOnTime Usec = 8 ; /* this value is for the FH PHY */
/*
Nominal time in microseconds between issuance of PmdData.request and the start of the corresponding symbol at the air interface. Start of a symbol is defined to be 1/2 symbol period prior to center of the symbol for FH, 1/2 chip period prior to the center of the first chip of the symbol for DS, or 1/2 PPM-slot time prior to the center of the corresponding PPM-slot for IR.
read-only; default=1 for FH, impl. dep. for DS, 1 for IR); used in aRxTxTurnaroundTime equation; */
synonym aRxRfDelay Usec = 1 ; /* this value is for the FH PHY */
/*
Time in microseconds for the MAC+PHY to receive the last symbol of a frame at the air interface, process the frame, and respond with the first symbol on the air interface of the earliest possible response frame.
read-only; default=PHY-dependent; used in Channel_State, Backoff; */
synonym aSifsTime Usec = (aRxDelay + aRxPclpDelay + aMacProTime + aRxTxTurnaroundTime);
*/
end of Part 1, PhyOperationGroup continues . . .

PHY_OPERATION_GROUP -- part 2 of 2 /
Nominal time in microseconds between the end of a symbol at the air interface and the issuance of a PdmData.indicate to the PLCP. End of a symbol is defined to be 1/2 symbol period after the center of the symbol for FH, 1/2 chip period after the center of the last chip of the symbol for DS, or 1/2 PPM-slot time after the center of the corresponding PPM-slot for IR.
read-only; default=14 for FH, impl. dep. for DS, 4 for IR); used in aSilence time equation; /*
synonym aPhRfDelay Usec = 4; /* this value is for the FH PHY */
Nominal time in microseconds for the PLCP to deliver a bit from Pmd receiver to the MAC interface.
read-only; default=2 for FH, impl. dep. for DS, 1 for IR); used in aSilence time equation; /*
synonym aPmdPlcpDelay Usec = 2; /* this value is for the FH PHY */
Nominal time in microseconds for the MAC to process the end of reception of a frame, and to decide whether to respond to the received frame.
read-only; default=(2 for FH, impl. dep. for DS, 0 for IR); used in aSilsTime equation; /*
synonym aMacProcTime Usec = 2; /* this value is for all PHYs */
Lengths of the PHY preamble and PHY PLCP header in bits.
read-only; default=(96,32 for FH, 144,48 for DS, unspecified for IR); used in Filter MPDU, /*
synonym aPreambleLength Integer = 96; /* this value is for the FH PHY */
synonym aPlcpHdrLength Integer = 32; /* this value is for the FH PHY */
The overhead (in bits) added by the PHY to the MPDU for transmission through the wireless medium.
WARNING: aMduDurationFactor is specified to be Integer32, but the FH PHY uses a default value of 1.03125. Temporarily, the data type is declared as Real. For SNMPv2, this attribute could be an Integer, but <code for temp range of PHY physically present>; used in aSilence time equation; /*
synonym aMduDurationFactor Real = 1.03125; /* this value is for the FH PHY */
The anticipated time for a transmitted signal to traverse the wireless medium to the receiving station.
read-only; default=1, by definition, uniform for all PHYs); used in aSlotTime equation; /*
synonym aAirPropagationTime Usec = 1; /* this value is uniform for all PHYs */
Identifies the operating temperature range specified for the PHY (commercial 0:+40 degC=01, industrial -20:+55 degC=02, extended_industrial -30:+70 degC=03).
read-only; default=<code for temp range of PHY physically present>; used in aSilence time equation; /*
synonym aTempType Integer = 1; /* this value is for the commercial temperature range */
Maximum and minimum sizes of the contention window, in units of aSlotTime.
read-only; default=(1023,15 for FH, 1023,31 for DS, 1023,63 for IR); used in aSilsTime equation; /*
synonym aCWmax Integer = 1023; /* this value is for the FH PHY */
synonym aCWmin Integer = 15; /* this value is for the FH PHY */
end of Part 2, end of PhyOperationGroup */
This SDL-92 rendition of the MAC MIB, plus portions of the PHY MIB, exists to permit analysis of the MAC state machines without having full Z.105 support in the SDL tool. However, some of the comments interspersed with the remote variable definitions are useful as explanations of the relationships between MIB attributes and state machine functions. Also, there are cases where data types in this rendition differ from the ASN.1, and the reasons for the differences are explained in comments.

```
use macsorts ;

package macmib

---

This SDL-92 rendition of the MAC MIB, plus portions of the PHY MIB, exists to permit analysis of the MAC state machines without having full Z.105 support in the SDL tool. However, some of the comments interspersed with the remote variable definitions are useful as explanations of the relationships between MIB attributes and state machine functions. Also, there are cases where data types in this rendition differ from the ASN.1, and the reasons for the differences are explained in comments.

```

```c
#define unsigned byte integer subtype and string thereof for SupportedRates lists.

typedef RateVal = integer constants 0..255 endtype rateVal;
newtype RateString = string(RateVal, noRates) endnewtype RateString;

Null-terminated string of (8-bit) integers representing available PHY tx data rates in units of 100Kbps.
read-only [read-write in ASN.1] default<int PHY dep., includes 10> used in Transmit_Control; */
synonym aSupportedRatesTx "Ratestring = mkstring(10) // mkstring(0) ; /* this value for the FH PHY */

Null-terminated string of (8-bit) integers representing available PHY rx data rates in units of 100Kbps. read-only [read-write in ASN.1] default<int PHY dep., includes 10> used in <not used>; */
synonym aSupportedRatesRx "Ratestring = mkstring(10) // mkstring(0) ; /* this value for the FH PHY */

The maximum number of octets in an MPDU which can be conveyed in a single PLCP PDU. read-only [read-write in ASN.1] default=(4095-FH, 8191-DS, 2500-IR) used in Prepare_Mpdu; */
synonym aMpdumaxLength Integer = 4096 ; /* this value for the FH PHY */

The recommended maximum number of octets in an MPDU conveyed in a single PLCP PDU. NOTE: Listed as read-only in ASN.1 and old clause 13, but no static value given, so probably should be read-write. ANOTHER NOTE: The name of this attribute is redundant and inconsistent with previous attribute.
read-only; default<int not specified>; used in <not used> */
synonym aPreMaxMpduFragmentLength Integer = aMpdumaxLength;
```

```

```

```
```
This is a text symbol, used to hold data type (sort) definitions, declarations, signal lists, and other SDL statements that have no graphical representation.

* in a state symbol means all states except those listed

- in a state symbol refers to the starting state of the transition

I This is a process start symbol (one per process, contains no text).

- State symbol, arrowhead indicates transition(s) entering the state.

- Input symbol with wedge on left side used for signals from LLC, SMT, self, and others logically above or parallel to this process.

- Task symbol for algorithmic process steps

- Signal symbol, text extension holds overflow text

- Decision symbol

- Process stop symbol

- Create request symbol used for dynamic creation of an instance of the specified process type.

- Output symbol with point to left side used for signals to PHY & others logically above or parallel to this process.

- Priority input symbol enables its transition if the named signal is anywhere in the process input queue.

- Label

- Error signal

- Actions to recover from error