System Station

signal
Busy,
ChangeNav(Time,Duration,NavSrc),
DoBackoff(Natural,Natural),
Idle,
Idx(Octetstring,MacAddr,MacAddr),
MaUnitdata.indication(MacAddr,MacAddr,
Routing,Octetstring,RxStatus,
CfPriority,ServiceClass),
MaUnitdata.request(MacAddr,MacAddr,
Routing,Octetstring,CfPriority,ServiceClass),
MaUnitdataStatus.indication(MacAddr,
MacAddr,TxStatus,CfPriority,ServiceClass),
MlmeAssociate.confirm(Success),
MlmeAssociate.indicate(MacAddr), AP only */
MlmeAssociate.request(MacAddr,Natural),
MlmeAuthenticate.confirm(MacAddr,AuthType,Success),
MlmeAuthenticate.indicate(MacAddr,AuthType),
MlmeAuthenticate.request(MacAddr,AuthType,Natural),
MlmeDeauthenticate.confirm(MacAddr,Success),
MlmeDeauthenticate.indicate(MacAddr),
MlmeDeauthenticate.request(MacAddr),
MlmeDisassociate.confirm(Success),
MlmeDisassociate.indicate(MacAddr),
MlmeDisassociate.request(MacAddr),
MlmeGet.confirm(MibStatus,MibAtrib,MibValue),
MlmeGet.request(MibAtrib),
MlmeJoin.confirm(Success),
MlmeJoin.request(BssDscr),
MlmePowermgt.confirm(Success),
MlmePowermgt.request(PwrSave,Natural),
MlmeReassociate.confirm(Success),
MlmeReassociate.indicate(MacAddr), AP only */
MlmeReassociate.request(MacAddr,Natural),
MlmeReset.confirm(Success),
MlmeReset.request,
MlmeScan.confirm(BssDscrSet),
MlmeScan.request(BssDscrSet,MacAddr,
Ocetstring,ScanType,Intstring,Intstring),
MlmeSet.confirm(MibStatus,MibAtrib),
MlmeSet.request(MibAtrib,MibValue),
MlmeStart.confirm(Success),
MlmeStart.request(Octetstring,BssType,
Natural,CfParms,PhyParms) ;
use macsorts
use macmib;

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

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

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

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

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

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

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

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

signallist
PhyRxSignals=
  PhyRxStart.indicate,
  PhyRxEnd.indicate,
  PhyData.indicate,
  PhyCca.indicate,
  PhyCcarst.confirm;

signallist
PlmeRequestSignals=
  PlmeGet.request,
  PlmeSet.request,
  PlmeReset.request;

signallist
PlmeConfirmSignals=
  PlmeGet.confirm,
  PlmeSet.confirm,
  PlmeReset.confirm;
Process MSDU_to_LLCC

MsduIndicate(sdu, period)

da := addr1(sdu)

if frDs(sdu) = 1 then addr3(sdu) else addr2(sdu) fi

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

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

MaUnitdata_indication(sa, da, null_rt, LLCdata, successful_rx, period, serv)

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

Reception status always successful because any error would prevent the MsduIndicate from reaching this process.
This block is a summary of MIB access and update. MIB attributes are defined in ASN.1 code.

MIB(1,1)

ReqConf

MImeAssociate.confirm, MImeAuthenticate.confirm, MImeDeauthenticate.confirm, MImeDisassociate.confirm, MImeJoin.confirm, MImePowermgmt.confirm, MImeReassociate.confirm, MImeReset.confirm, MImeScan.confirm, MImeStart.confirm

Indicate

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

SMT_Requests (1,1)

MImeAssociate.confirm, MImeAuthenticate.confirm, MImeDeauthenticate.confirm, MImeDisassociate.confirm, MImeJoin.confirm, MImePowermgmt.confirm, MImeReassociate.confirm, MImeReset.request, MImeScan.request, MImeStart.request

MImeAssociate.request, MImeAuthenticate.request, MImeDeauthenticate.request, MImeDisassociate.request, MImeJoin.request, MImePowermgmt.request, MImeReassociate.request, MImeScan.request, MImeStart.request

ReqConf

MImeAssociate.confirm, MImeAuthenticate.confirm, MImeDeauthenticate.confirm, MImeDisassociate.confirm, MImeJoin.confirm, MImePowermgmt.confirm, MImeReassociate.confirm, MImeReset.confirm, MImeScan.confirm, MImeStart.confirm

MImeAssociate.indicate, MImeAuthenticate.indicate, MImeDeauthenticate.indicate, MImeDisassociate.indicate, MImeJoin.indicate, MImePowermgmt.indicate, MImeReassociate.indicate

FromMgt

ResetMAC

ToMgt

MMGT

mR

mSet

MImeAssociate.confirm, MImeAuthenticate.confirm, MImeDeauthenticate.confirm, MImeDisassociate.confirm, MImeJoin.confirm, MImePowermgmt.confirm, MImeReassociate.confirm, MImeReset.confirm, MImeScan.confirm, MImeStart.confirm

MImeAssociate.request, MImeAuthenticate.request, MImeDeauthenticate.request, MImeDisassociate.request, MImeJoin.request, MImePowermgmt.request, MImeReassociate.request, MImeScan.request, MImeStart.request

MImeAssociate.indicate, MImeAuthenticate.indicate, MImeDeauthenticate.indicate, MImeDisassociate.indicate, MImeJoin.indicate, MImePowermgmt.indicate, MImeReassociate.indicate
/* Import of {Read-Only} MIB attribute values exported from other processes */
imported aAckFailureCount, aFailedCount,
aFcsErrorCount, aFrameDuplicateCount,
aMulticastReceivedFrameCount,
aMulticastTransmittedFrameCount,
aMultipleRetryCount,
aReceivedFrameCount, aRetryCount,
aRtsFailureCount, aRtsSuccessCount,
aTransmittedFragmentCount,
aIcvErrorCount Counter32 ;

/* Declarations of internal MAC variables (updated from multiple sources using mSet) */
dcl exported mKeyId MacAddr:= nullAddr,
mBss Boolean:= false,
mSeld 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 Kusec:= 512,
aMaxTransmitMsduLifetime Kusec:= 512,
aMediumOccupancyLimit Kusec:= 100,
aPrivacyInvoked Boolean:= false,
aReceiveDTIMs Boolean:= true,
aRtsThreshold Integer:= 3000,
aShortRetryLimit Integer:= 7,
aWepDefault KeyIndex:= 1,
aCurrentChannelNumber Integer,
aCurrentDwellTime Kusec:= 390,
aCurrentSet Integer,
aCurrentPattern Integer,
aCurrentPattern Integer ;
/* Write-Only attributes */
dcl exported aDefaultWepKeys KeyVector ;
dcl exported aWepKeyMapping KeyMapArray := (. nullAddr, false, null .) ;
Process SMT_Requests

IDLE

Reject Authenticate and Deauthenticate, allow Start if idle

MlmeStart._request(ss, btype, n, cf, ph)

MlmeAuthenticate._request(sta, alg, n)

MlmeAuthenticate._confirm(sta, false)

MlmeDeauthenticate._request(sta)

MlmeDeauthenticate._confirm(false)

Reset and power management always accepted.

ResetMAC is propagated to all proc. with multiple states.

ResetMAC to MIB

BSS

Allow Associate and Reassociate if joined bss.

MlmeAssociate._request(sta, n)

MlmeReassociate._request(sta, n)

MlmeAssociate._confirm(false)

MlmeReassociate._confirm(false)

Wait_SMT

Wait_SMT

(IDLE)

Allow Authenticate and Deauthenticate, reject Start if not idle.

MlmeStart._request(ss, btype, n, cf, ph)

MlmeAuthenticate._request(sta, alg, n)

MlmeAuthenticate._confirm(sta, alg, false)

MlmeDeauthenticate._request(sta)

MlmeDeauthenticate._request(sta)

ResetMAC to MIB

ResetMAC to MibReq_RSP

ResetMAC to Synchronization

IDLE
Process SMT_Requests

Wait_SMT

- Wait for MAC management to process request.

MlmeStart_.confirm(ok)

MlmeJoin_.confirm(ok)

MlmeAuthenticate_.confirm(sta, alg, ok)

MlmeDeauthenticate_.confirm(ok)

MlmeScan_.confirm(bssSet)

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

- Return to the state prior to Wait_SMT.

rqState:= idle

IDLE

import(mlbss)

AP

IBSS

BSS

rqState:= ap

rqState:= ibss

rqState:= bss

IDLE

IBSS

BSS

AP

Allow Join and Scan if not AP.

Disassociate allowed at AP and sta in bss.

Disassociate rejected if idle in bss.

MlmeJoin_.request(bss)

MlmeJoin_.request(bss)

MlmeScan_.request(typeSet, sta, ss, scan, chlist, chtime)

MlmeScan_.request(typeSet, sta, ss, scan, chlist, chtime)

MlmeScan_.confirm(emptystring)

MlmeScan_.confirm(false)

MlmeJoin_.request(bss)

MlmeJoin_.request(bss)

MlmeAssoc_.iate.confirm(ok)

MlmeReassoc_.iate.confirm(ok)

MlmeDis_.associate_.request(sta)

MlmeDis_.associate_.request(sta)

MlmeDis_.associate_.confirm(false)

MlmeDis_.associate_.confirm(false)

MlmeAssoc_.iate.confirm(ok)

MlmeReassoc_.iate.confirm(ok)

MlmeDis_.associate_.confirm(ok)

MlmeDis_.associate_.confirm(ok)

MlmeScan_.request(typeSet, sta, ss, scan, chlist, chtime)

MlmeScan_.request(typeSet, sta, ss, scan, chlist, chtime)

MlmeScan_.confirm(emptystring)

MlmeScan_.confirm(false)

Wait_SMT

Wait_SMT

- Reject Join and Scan at active AP.
This state machine passes indications through, unmodified, from SMT to the MLME SAP. MlmeAssociate.indicate and MlmeReassociate.indicate are only generated by SMT at APs.
signal
FragsduConfirm(FragSdu, TxResult),
FragsduRequest(FragSdu) ;
Process Prepare_MPDU

- **Prepare_Idle**
  - **Msdu_Request** *(sdu, pri)*
    - **Mmpdu Requests** assumed to not need validation.
    - **Mm_Request** *(sdu, pri)*
      - **Mm_Confirm** *(sdu, pri, rsrl)*
        - **Frag_Confirm** *(fsdu, pri, rsrl)*
          - **rsdu** := substr *(fsdu!pdus(0), 0, sMacHdrLng)*
          - **pri** := fsdu!period
          - **basetype** *(fsdu!pdus(0))*
            - **else** *(management)*
              - **Mm_Confirm** *(rsdu, pri, rsrl)*
                - to fsdu!cnfTo

- **ResetMAC**
- **Encrypt**

**imported** mAssoc, mlbss, aPrivacyInvoked Boolean
imported mbssid MacAddr;
imported aFragmentationThreshold Integer;
" imported aPrefMaxMpwduFragmentLength Integer; */
imported aDefaultWepKeys KeyVector;
imported aWepDefault KeyIndex;
imported aWepKeyMapping KeyMapArray:

**dcl f FragNum ;**
**dcl fsdu FragSdu ;**
**dcl k, mpwduOvhd, p, pmax, pduSize, thld Integer ;**
**dcl pri ClPriority ;**
**dcl requester PId ;**
**dcl result, rsrl TxResult ;**
**dcl sst, assst StationState ;**
**dcl useDs Boolean:= true ;**
**dcl useWep Boolean:= false ;**

**imported** mIbss

**/* 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 Octetstring ;**

**reset MAC**
**Prepare_Idle**

**Msdu_Request** *(sdu, pri)*
- **Mmpdu Requests** assumed to not need validation.
  - **Mm_Request** *(sdu, pri)*
    - **Mm_Confirm** *(sdu, pri, rsrl)*
      - **Frag_Confirm** *(fsdu, pri, rsrl)*
        - **rsdu** := substr *(fsdu!pdus(0), 0, sMacHdrLng)*
        - **pri** := fsdu!period
        - **basetype** *(fsdu!pdus(0))*
          - **else** *(management)*
            - **Mm_Confirm** *(rsdu, pri, rsrl)*
              - to fsdu!cnfTo

**imported** mAssoc, mlbss, aPrivacyInvoked Boolean
imported mbssid MacAddr;
imported aFragmentationThreshold Integer;
" imported aPrefMaxMpwduFragmentLength Integer; */
imported aDefaultWepKeys KeyVector;
imported aWepDefault KeyIndex;
imported aWepKeyMapping KeyMapArray:

**dcl f FragNum ;**
**dcl fsdu FragSdu ;**
**dcl k, mpwduOvhd, p, pmax, pduSize, thld Integer ;**
**dcl pri ClPriority ;**
**dcl requester PId ;**
**dcl result, rsrl TxResult ;**
**dcl sst, assst StationState ;**
**dcl useDs Boolean:= true ;**
**dcl useWep Boolean:= false ;**

**imported** mIbss

**/* 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 Octetstring ;**

**reset MAC**
**Prepare_Idle**

**Msdu_Request** *(sdu, pri)*
- **Mmpdu Requests** assumed to not need validation.
  - **Mm_Request** *(sdu, pri)*
    - **Mm_Confirm** *(sdu, pri, rsrl)*
      - **Frag_Confirm** *(fsdu, pri, rsrl)*
        - **rsdu** := substr *(fsdu!pdus(0), 0, sMacHdrLng)*
        - **pri** := fsdu!period
        - **basetype** *(fsdu!pdus(0))*
          - **else** *(management)*
            - **Mm_Confirm** *(rsdu, pri, rsrl)*
              - to fsdu!cnfTo
Process Prepare_MPDU

fragments

Initialize FragSdu structure

fsdu!lTot:= 1, fsdu!lCur:= 0, fsdu!lAnc:= 0, fsdu!loe:= 0, fsdu!lsqf:= 0, fsdu!lsrc:= 0, fsdu!lrc:= 0, fsdu!lpsm:= false

fsdu!grp:= type MacAddr is Group (addr1(sdu)),

mpduOvhd:= sMacHdrLng + sCrcLng +

if useWep then sWepAddLng else 0 fi

phmax:= aMaxMpduLength,

thld:=import (aFragmentation Threshold)

Threshold used is the shorter of aFragmentationThreshold and aMpduMaxLength.

not(fsdu!grp) and

length(sdu) > thld

pduSize:= length(sdu) - sMacHdrLng

This equation is a common case. Selection of Mpdu size is arbitrary, so long as the chosen size is: >=256, even, and <=a MpduMaxSize.

pduSize:= aPrefMaxMpduFragmentSize

pduSize:= mpduOvhd

fsdu!lTot:= (length(sdu) / pduSize) +

make pdus

make pdus

f:=0, p:= sMacHdrLng

fsdu!pdus(f):= null

fsdu!pdus(f):= fsdu!pdus(f) // substr(sdu,p,

sMacHdrLng) // substr(sdu,p,

pduSize)

fsdu!pdus(f):= setFrag(fsdu!pdus(f), f)

(f+1) < fsdu!lTot

(fs|true)

(fs|false)

useWep

Encrypt (fsdu!pdus(f), asst)

(true)

(false)

f:= f+1, p:= p+pduSize,

pduSize:= if (p+pduSize) > length(sdu) then (length(sdu) - p + 1) else pduSize fi

f = fsdu!lTot

(fs|true)

(fs|false)

Final fragment may be shorter than initial/intermediate fragments.

FragRequest (fsdu)

Prepare_Idle
Procedure Encrypt

1. In/Out wepdu, Frame, dstAuth, StationState:

The algorithm for changing the keyId is not specified. If all stations in a BSS use the same aWepDefaultKeys vector, a station's keyId algorithm does not affect interoperability.

2. encryptLng := length(wepdu) - sMacHdrLng

The algorithm for generating (3-octet) IV strings is outside the scope of this standard. However, generating a new IV string for each frame is RECOMMENDED STRONGLY.

3. newIV := call genIV(x)

The algorithm for generating (3-octet) IV strings is outside the scope of this standard. However, generating a new IV string for each frame is RECOMMENDED STRONGLY.

4. newIV := substr(wepdu, 0, sMacHdrLng) // Insert IV and keyId between MAC header and data field.

5. kndx := import(aWepDefault)

The algorithm for changing the keyId is not specified. If all stations in a BSS use the same aWepDefaultKeys vector, a station's keyId algorithm does not affect interoperability.

6. n := sWepHdrLng

7. icv := initCrc

8. k := 0, n := sWepHdrLng

9. k := k+1, n := n+1

10. ICV value calculated from plaintext.

11. Encrypt by xor of payload with encrypt string.

12. Set WEP bit in Frame Control field.

13. 8.3.2 says MIB sets map!wepOn=true only for stations using other than open system authentication. Since authentication can occur after loading key map array, the test is done here during Tx setup.

14. 8.3.2 says MIB sets map!wepOn=true only for stations using other than open system authentication. Since authentication can occur after loading key map array, the test is done here during Tx setup.

15. Concatenate key with IV for encryption PRNG seed.

16. Use RC4 PRNG to generate an encrypt string at long as the MPDU payload.

17. Encrypt by xor of payload with encrypt string.

18. Set WEP bit in Frame Control field.

19. wepdu := wepdu // mirror(not(icv))

20. encryptStr := call RC4(key, encryptLng)

21. Set WEP bit in Frame Control field.

22. wepdu := setWepBit(wepdu, 1)
Block MAC_Control

signal
Ack(Time),
Cfpoll(Time),
Change(ChangeType),
Cts(Time),
Reply,
Resume,
RxFlush,
RxPdu(Frame),
Suspend,
TxCfAck(Time):

PDU

MsduIndicate
PduConfirm,
ResetMAC

RSDU

toLLC

ResetMAC

RxPdu, RxFlush

Tdat

Tenb

UcMgt

MmIndicate

Tmgt

MmDone, MmCmdAck

MmIndicate,
SsInquiry

MmIndicate,
ResetMAC

MsduIndicate
PduConfirm,
ResetMAC

PduRequest

TxConfirm, TxDone

RxPdu, RxFlush

RxPdu,
RxFlush

MmIndmed,
MmCancel,
MmCmd, TBTT,
ResetMAC

MsduIndicate
MsduIndicate
MsduIndicate

RxI
NeedAck,
RxCfAck,
RxIndicate

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX

MLME_PLME_SAP

RX
Process Reassembly

imported mCfp Boolean;
imported aMaxReceiveLifetime Kusec;

dcl buf ReasmArray;
dcl k Integer;
dcl pri CfPriority;
dcl rpdu,rsdu Frame;
dcl bufcnt, fnbr, ageTarget Boolean;

buf:=
ArAge(buf,
now + kUsec( import(aMaxReceiveLifetime)+1)
)

Asm_Idle

RxPdu (rpdu)

RxFlush

ResetMAC

Use ageing function to force all buffers empty.

basetype (rpdu)
else (data)

else contention free

else contention fi)

Msdu_Indicate (rpdu, if
import(mCfp)=true then contention_free
else contention fi)

fragNum (rpdu)=0

fragNum (rpdu)=0

fragNum (rpdu)

buf:=
ArAge(buf,
now + kUsec( import(aMaxReceiveLifetime)+1)
)

buf:=
ArAge(buf,
now + kUsec( import(aMaxReceiveLifetime)+1)
)

buf(!inUse:=false
buf(!rta:=addr2(rpdu),
buf(!rsn:=seqNum(rpdu)

buf(!rCur:=fragNum(rpdu),
buf(!reol:=now + kUsec( import(aMaxReceiveLifetime))

buf(!rsdu:=rpdu

buf(!inUse:=false
process Rx_Coordination

chk_sst

SsInquiry (addr2(pdu))

Wait_Response

Rx with tD=1 discarded by Filter_MPDU. tD=1 never sent by Sta. so explicit test for tD/tF not needed here.

However, PS-Poll should not be sent to a station.

SsResponse (sas,sau)

ftype(pdu)

else

RxC_Idle

MmIndicate (pdu, class2) to Mime_Req_Rsp

uc_ack

RxPdu(pdu)

CfPoll (endRx)

RxC_Idle

MmIndicate (pdu, noerr) to Synchronization

RxPdu(pdu)

if dAck>0 then dRsp else 0 fi

RxC_Idle

RxC_Idle

RxC_Idle

RxIndicate (pdu,endRx,strTs,dAck)

RxC_Idle

RxIndicate (pdu, endRx, strTs, dAck)

MmIndicate (pdu, endRx, strTs, dAck)

Beacon and probe_rsp sent to Mime_Req_Rsp while scanning, other types acknowledged (if unicast to this station) but ignored.

MmIndicate (pdu, class2) to Mime_Req_Rsp

uc_ack

uc_ack
On this page are DISASSOCIATE Request and DEAUTHENTICATE Request.

**DISASSOCIATE Request**
- The Mlme request has no reason parm, use reason 1 (or 8).
- **MmRequest** (mmpdu, contention)
- **Wait_Disassoc_Confirm**
- **MmConfirm** (, result)
- ok:= (result=successful), sst:= dis_assoc

**DEAUTHENTICATE Request**
- The Mlme request has no reason parm, use reason 1.
- **MmRequest** (mmpdu, contention)
- **Wait_Deauth_Confirm**
- **MmConfirm** (, result)
- ok:= (result=successful), sst:= de_auth

* Source of the key used to generate challenge text is not specified. *
- dcl ctxtKey Octetstring
- dcl n, k, seq, rseq Integer
- dcl ok Success
- dcl psn PsMode
- dcl response Boolean= false
- dcl result TxResult
- dcl sc, challenge StatusCode
- dcl scan ScanType
- dcl ss Octetstring
- dcl sst, rst, asst StationState
- dcl sta, peer, src, apAddr, bss MacAddr
- dcl types BssTypeSet
- timer Tresp, Tmax

**Switches**
- dcl n, k, seq, rseq Integer
- dcl ok Success
- dcl psn PsMode
- dcl response Boolean= false
- dcl result TxResult
- dcl sc, challenge StatusCode
- dcl scan ScanType
- dcl ss Octetstring
- dcl sst, rst, asst StationState
- dcl sta, peer, src, apAddr, bss MacAddr
- dcl types BssTypeSet
- timer Tresp, Tmax

**Context Key**
- dcl ctxtKey Octetstring
- dcl n, k, seq, rseq Integer
- dcl ok Success
- dcl psn PsMode
- dcl response Boolean= false
- dcl result TxResult
- dcl sc, challenge StatusCode
- dcl scan ScanType
- dcl ss Octetstring
- dcl sst, rst, asst StationState
- dcl sta, peer, src, apAddr, bss MacAddr
- dcl types BssTypeSet
- timer Tresp, Tmax

**Operation**
- dcl n, k, seq, rseq Integer
- dcl ok Success
- dcl psn PsMode
- dcl response Boolean= false
- dcl result TxResult
- dcl sc, challenge StatusCode
- dcl scan ScanType
- dcl ss Octetstring
- dcl sst, rst, asst StationState
- dcl sta, peer, src, apAddr, bss MacAddr
- dcl types BssTypeSet
- timer Tresp, Tmax
On this page are ASSOCIATE Request and REASSOCIATE Request.

See 7.2.3.6. Need means for SME to set capability bits.

See 7.2.3.4. Need means for SME to set capability bits.

'\textit{do something with supported rates element}'

\texttt{A\textsubscript{BssBasicRateSet}} is no longer in the MIB. Without the basic rates it is unclear what to do with supported rates.
On this page is AUTHENTICATE Request.

```
dResp := dUse(n)
set (now+dResp, Tresp)
(alg in algset)
  (true)
  (false)
and (not groupAddr(sta))

mmpdu := mkFrame
  (auth, sta, import(mBssid),
   alg // mkOS(1,2) // successful),
peer := sta,
expect := auth

MmRequest
  (mmpdu, contention)

Wait/Auth/_Confirm_1

Tresp

result = successful
  (false)
  (true)
seq := 2,
response := false

Wait/Auth/_Seq_2

Tresp

result = successful
  (false)
  (true)
seq := 4,
response := false

Wait/Auth/_Seq_4

Tresp

expect := null,
response := false

authStat (rxpdu)
authKey (true, sst := auth_key)
ok := true,
sst := de_auth
reset(Tresp)

Mlme_Idle
```

Copy challenge text from auth seq #2 frame.

```
mmpdu := mkFrame
  (auth, sta, import(mBssid),
   alg // mkOS(3,2) // successful // substr(rxpdu,31,128))

mmpdu := setWepBit
  (mmpdu, 1)

MmRequest
  (mmpdu, contention)

Wait/Auth/_Confirm_3

Tresp

result = successful
  (false)
  (true)
seq := 4,
response := false

Wait/Auth/_Seq_4

Tresp

expect := null,
response := false

ok := false,
sst := de_auth
reset(Tresp)

Mlme_Idle
```

Mark shared key frame #3 for encryption.

```
mmpdu := mkFrame
  (auth, sta, import(mBssid),
   alg // mkOS(1,2) // successful),
peer := sta,
expect := auth

MmRequest
  (mmpdu, contention)

Wait/Auth/_Confirm_1

Tresp

result = successful
  (false)
  (true)
seq := 2,
response := false

Wait/Auth/_Seq_2

Tresp

result = successful
  (false)
  (true)
seq := 4,
response := false

Wait/Auth/_Seq_4

Tresp

expect := null,
response := false

ok := false,
sst := de_auth
reset(Tresp)

Mlme_Idle
```

An unsuccessful authentication attempt leaves the requester de-authenticated.
Responses to incoming management frames:

- Don’t handle responses while communication suspended for scan procedure.
- If disassociate or deauthenticate is from current AP, clear Bssid.

- Don’t handle responses while communication suspended for scan procedure.
- If disassociate or deauthenticate is from current AP, clear Bssid.

- Reply to non-authenticated sender of a class 2 frame.
- Reply to un-associated sender of a class 3 frame.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- Do not handle responses while communication suspended for scan procedure.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.

- MmIndicate
  - (rxpdu.endRs, startTs.err)
  - ft:=ftype(rxpdu),
  - src:=addr2(rxpdu),
  - bss:=import(mBssid)

- If dissassociate or deauthenticate is from current AP, clear Bssid.
On this page is SCAN request

**Probe delay is local because aProbeDelay no longer in MIB */
dcl dDelay Duration;

Mlme_Idle

MlmeScan_request (types, sta, ss, scan, clist, ctime)

dMin:= usec (ctime(1)),
dMax:= usec(ctime(2)),
dscr:= Empty

Hold

Wait_Hlda

HoldAck

nx_chan

Switch to next channel on scan list.

Scan

(passive_scan)

(active_scan)

MmCmd (chansw, clist(1))

set(now+dDelay, Tresp)

Wait_Probe

Tresp

MmDone (result)

result

(successful)

Send Probe_req after probe delay expires.

Wait_Passive

Tresp

MmIndicate (rxpdu, endRx, startTs, err)

if(type(rxpdu) = response info into dscr (BssDscrSet))

else

'or response into dscr (BssDscrSet)'

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

Result BssDscrSet, which may be empty.

MlmeScan_confirm (dscr)

Mlme_Idle

MmCmd (chansw, clist(k))

MmIndicate (mmpdu)

MmImed (mmpdu)

Set (now+dMin, Tresp), (now+dMax, Tmax)

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

Release

MlmeScan_confirm (dscr)

Mlme_Idle

nx_channel

k:= k+1

k > length(clist)

(true)

(false)

Rx activity flag, exported by Validate_Mpdu.

reset(Tresp)

k:= k+1

k > length(clist)

(true)

(false)

Scan done

nx_probe

Scan done

nx_probe

nx_probe

nx_probe

nx_probe
/* Each of these sets holds MAC addresses of stations with given operating state. Members are added/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. */

dcl awake, /* detected in active mode */
asleep, /* detected in power_save mode */
umauth, /* 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 psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sta,
sta_map
StationState ;
dcl sta
MacAddr ;

dcl psm
PsMode ;
dcl psquery
Boolean ;

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

SsInquiry returns SsResponse to report station state as not_auth, de_auth, assoc, or dis_assoc; and authentication state as not_auth, de_auth, auth_open, or auth_key at the target station.

^{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 TxImed/TxDone exchange may occur while a TxRequest is pending. An example is a TxImed 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.
Process Backoff

Idle and Slot ignored in this state.

TxRequest (pdu, cw, slotCnt)

txPending:= true

slotCnt:= (slotCnt=0)

DoBackoff (cw, slotCnt)

Busy

txPending:= false

Channel_Idle

Channel.Busy

Busy (and Slot) ignored in this state.

TxRequest (pdu, cw, slotCnt)

txPending:= true

slotCnt:= (slotCnt=0)

Calculate random backoff slot count according to 9.2.4. 0 <= slotCnt < cw

DoBackoff (cw, slotCnt)

Channel_Idle

Channel.Busy

Channel_Idle, Channel.Busy

TxCancel

Channel_Idle

Channel.Busy

Wait for Idle

Idle, Slot

Idle

Busy

Idlev

send

TxCancel

TxStart(pdu)

Wait TxDone

TxConfirm (txdone,0)

ResetMAC

(=0)

(=/0)
Calculate PHY Tx delay added to TSF time to get timestamp.

\[ dTx := dUsec (aTxRfDelay + aTxPlcpDelay) \]

SiFS responses sent by TxImed, hence its priority.

**Source:**

\[ source := sender \]

Save sender's PId for address of TxDone.

\[ k := 0, \quad fcs := initCrc \]

\[ txLength := \text{Length(pdu)} \]

**PlcpTxStart**

**request(txLength + sCrcLng)**

Wait TxStart

**PhyTxStart**

**confirm**

\[ send1 \]

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

\[ pdu := setTs(pdu, now + dTx) \]

\[ send1 \]

Check for start of timestamp field in probe response and beacon frames.

\[ k := 0, \quad fcs := \text{mirror} (not(fcs)) \]

\[ k := k + 1 \]

**Send_CRC**

**PhyTxEnd**

**request**

**PhyTxEnd**

**confirm**

\[ (true) \]

\[ (false) \]

\[ k = sTsOctet \]

**Send_Frame**

**PhyData**

**confirm**

\[ send1 \]

\[ (beacon, probe_rsp) \]

1's complement of calculated CRC value is transmitted, MSb to LSb.

**Send_CRC**

**PhyData**

**request**

\[ (fcs(k)) \]

\[ k := k + 1 \]

Wait TxEnd

**PhyTxEnd**

**confirm**

TxDone to source

**PhyTxEnd**

**request**

\[ (false) \]

\[ (true) \]

\[ k = sCrcLng \]

24th octet Check for start of timestamp field in probe response and beacon frames.

**Send_Frame**

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := crc32 (fcs, pdu(k)) \]

**PhyData**

**request**

\[ (fcs.pdu(k)) \]

**PhyData**

**confirm**

\[ send1 \]

\[ (true) \]

\[ (false) \]

**PhyData**

**request**

\[ (pdu(k)) \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

\[ (true) \]

\[ (false) \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]

**PhyData**

**request**

\[ (pdu(k)) \]

\[ fcs := initCrc \]

**PhyData**

**confirm**

\[ send1 \]
signal
ClearNav(NavSrc),
RtsTimeout,
RxMpdu(Frame,Time,Time),
SetNav(Time,Duration,NavSrc),
UseDifs(Time),
UseEifs(Time) ;
Retry bit tested before WEP to avoid decrypting duplicate frames.

If duplicate frames are detected, the frame is discarded.

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if dup:= searchTupleCache (cache, addr2(pdu), seq(pdu), frag(pdu))

Then dup is true and false.

if dup:= true

Then cDup:= inc(cDup)

If the frame is an ATIM frame, it may be omitted from the cache, see 9.2.9.

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.

if dup:= false

Then cDup:= inc(cDup)

Export(cDup)

Import(aExclude_Unencrypted)

NeedAck(addr2(pdu), endRx, dAck)

RxIndicate(pdu, endRx, strTs, dAck)

Decrypted(pdu, endRx, civOk)

Error Count on unsuccessful decryption.

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if cRx:= inc(cRx), cMc:= inc(cMc)

Then cRx is true and false.

Export(cRx, cMc)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.

if ftype(pdu)

Then ftype is true and false.

fType:= (true)

Export(fType)

If the frame is a broadcast or multicast, the appropriate actions are taken.

If the frame is an unicast frame, further processing is performed.

If the frame type is beacon, the frame is discarded.

If the frame type is data or management, further processing is performed.

If the frame type is other, further processing is performed.
Process Channel_State

dcl exported tNavEnd as mNavEnd.Time;

dcl cs CcaStatus;
dcl rtx, slot, sifs Natural;
dcl dDifs, dEifs, difs, dNav, dRxTx, dSifs, dSlot Duration;
dcl tNew, tRef, tRxEnd, Time;
dcl newSrc, curSrc NavSrc;

dSifs:= dUsec(aSifsTime),

dRxTx:= dUsec(aRxTxTurnaroundTime)

dSlot:= dUsec(aSlotTime),
dDifs:= dSifs+ (2*dSlot)

dEifs:= call calc_Eifs ( ? )'

EIFS duration calculated using the formula in clause 9.2.9.

reset(Tnav)

tNavEnd:= now

PhyCcarst._request

cs:= busy

curSrc:= nosrc

PhyCca._indicate(cs)

not active(Tifs)

Tifs

PhyCca._indicate(cs)

set (now+dSlot, Tslot)

Tslot

Wait_IFS /* IDLE */

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

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

PhyCca._indicate(cs)

set (now+dSlot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (tRef+dNav, curSrc)

SetNav(tRef,dNav, curSrc)

set (tNavEnd,Tnav)

export(tNavEnd)

Busy

Cs_noNav

noCs.Nav

ResetMAC

Cs_noNav

Busy

Cs_noNav

noCs.Nav

export(tNavEnd)

(idle)

(cs)

(busy)

(busy)

(idle)

(set (nav+dSlot, Tslot))

Tslot

waitNav

/* IDLE */

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

PhyCca._indicate(cs)

set (tRef+dNav, curSrc)

SetNav(tRef,dNav, curSrc)

TnoopNav

slot

set (Tslot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (now+dSlot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (tRef+dNav, curSrc)

SetNav(tRef,dNav, curSrc)

TnoopNav

slot

set (Tslot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (now+dSlot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (tRef+dNav, curSrc)

SetNav(tRef,dNav, curSrc)

TnoopNav

slot

set (Tslot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.

PhyCca._indicate(cs)

set (now+dSlot, Tslot)

Tslot

Wait_IFS

Slot signal is generated at the end of each M2 interval
(Fig. 47) while channel is idle.

Tifs, RtsTimeout, Tnav, ClearNav ignored in TnoopNav state.
**Process Channel State**

- **noCs_Nav */* BUSY */*
  - PhyCca_indicate(cs)
  - Tnav
  - (busy)
  - (idle)
  - Cs_Nav

- **Tsot and Tifs ignored in noCs_Nav state.**

- **Cs_Nav */* BUSY */*
  - PhyCca_indicate(cs)
  - Tnav
  - (busy)
  - (idle)
  - Cs_Nav

- **Tslot and Tifs ignored in Cs_Nav state.**

- **PhyCca._indicate(cs)**
  - cs
  - set (now+dIfs, Tifs)
  - Wait_IFS
  - Clearing the NAV on RTS timeout is optional. See last paragraph of 9.2.5.4.

- **SetNav (tRef,dNav, newSrc)**
  - RtsTimeout
  - ChangeNav (tRef,dNav, newSrc)
  - ClearNav (newSrc)
  - ChangeNav (tRef,dNav, curSrc)
  - ExportNav (tNavEnd)

- **UseDifs (tRefEnd)**
  - dDifs:= dDifs-dRxTx
  - dIfs:= dIfs-dRxTx

- **UseEifs (tRefEnd)**
  - dIfs:= dEifs-dRxTx
  - set (tRefEnd+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.**

- **ClearNav (newSrc)**
  - (newSrc=cfendBss) and (curSrc=cfpBss)
  - (newSrc=cfendOther) and (curSrc=cfpOther)

- **Nav is cleared by setting Tnav to now. This causes immediate Tnav signal to enable exit from noCs_Nav or Cs_Nav state.**
Process Validate_MPDU

1. dcl exported cErr as aFcsErrorCount Counter32:= 0;
   dcl exported mRxA Boolean:=false;
   timer Trts;

2. dcl fcs Crc;
   dcl D1, dRts Duration;
   dcl endRx, startTs Time;
   dcl k, rxLength Integer;
   dcl pdu Frame;
   dcl status PhyRxStat;
   dcl v Octet;

3. D1 := dUsec (aRxRfDelay + aRxPlcpDelay)
   Calculate PHY Rx delay to be subtracted from now at indicated reference points.

4. Rx_Idle
   Trts
   RtsTimeout

5. Rx_Frame
   k := 0,
   fcs := initCrc,
   pdu := null
   Initialize CRC & clear pdu buffer (length(pdu)==0).

6. PhyRxData.
   indi(v)
   Accumulate octet into Mpdu and CRC check.

7. PhyRxEnd.
   indicate(status)
   status
   k = rxLength
   protocolVer(pdu)
   if protocolVer(pdu) 'dRts := call calc_RtsTo(pdu)
   else
     UseEifs (endRx)

8. endRx := now-D1
   set(now+dRts, Trts)
   Save time of Rx end as reference for start of IFS.
   * (Rx_Idle)

9. mRxA := true
   Indicate that a reception is in progress.

10. mRxA := false
    Indicate that reception is not in progress.

11. Rx_Idle
    export(mRxA)

12. Rx_Idle
    export(mRxA)

13. UseDifs (endRx)
    cErr := inc(cErr)
    export(cErr)
    ftype(pdu)
    else (rts)

14. RxMpdu(pdu, endRx, startTs)
    Drop CRC field from valid frame before passing up for filtering.
    Rts Timeout formula is in 9.2.5.4. This definition is informal because PHYs do not report data rate in the RxVector.
Procedure Decrypt

1. fpar in/out pdu Frame
2. in/out icvOk Boolean

Procedure Decrypt

1. fpar in/out pdu Frame
2. in/out icvOk Boolean

kmap:= keyLookup
(addr2(pdu), import
(aWepKeyMapping),
aWepKeyMappingLength)

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.

SsInquiry
(addr1(pdu))

Wait_for
_Response

SsResponse
( , , asst)

asst

(true)

kmap!
keyOn

(false)

keyLookup
result
map!keyOn
=false if the
TA (addr2)
value not
found in
aWepKey
Mapping.

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.

SsInquiry
(addr1(pdu))

Wait_for
_Response

SsResponse
( , , asst)

asst

(true)

kmap!
keyOn

(false)

keyLookup
result
map!keyOn
=false if the
TA (addr2)
value not
found in
aWepKey
Mapping.

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.

SsInquiry
(addr1(pdu))

Wait_for
_Response

SsResponse
( , , asst)

asst

(true)

kmap!
keyOn

(false)

keyLookup
result
map!keyOn
=false if the
TA (addr2)
value not
found in
aWepKey
Mapping.

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.

SsInquiry
(addr1(pdu))

Wait_for
_Response

SsResponse
( , , asst)

asst

(true)

kmap!
keyOn

(false)

keyLookup
result
map!keyOn
=false if the
TA (addr2)
value not
found in
aWepKey
Mapping.

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.

SsInquiry
(addr1(pdu))

Wait_for
_Response

SsResponse
( , , asst)

asst

(true)

kmap!
keyOn

(false)

keyLookup
result
map!keyOn
=false if the
TA (addr2)
value not
found in
aWepKey
Mapping.

NOTE: 8.3.2 says the MIB
ensures map!wepOn=true only for
stations that use other than open
system authentication. However,
authentication may occur after
loading the key map table, so the
test is done here for Rx Mpdus.
/* PACKAGE MACSORTS */
/* 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 TxConfir is generated */
newtype BackoffStatus literals
txdone, /* indicates completion of a transmission requested with TxRequest */
cancelled, /* indicates cancellation of an unstarted transmission attempt due to TxCancel */
inactive; /* 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 MmCmd signals */
newtype CtlCmd literals suspend, /* halts transmission attempts (to permit scanning) */
resume, /* restarts transmission after suspend */
sleep, /* enters doze state after confirmation of doze from PHY */
wake, /* leaves doze state and reactivates PHY */
chanSw; /* 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, cfendBss, cfendOther, cswitch, misc, nosrc; 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 MmIndicate 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, /* no information is available about the subject station */
not_auth, /* subject station known but not authenticated */
de_auth, /* subject station explicitly deauthenticated */
auth_open, /* subject station authenticated using open system algorithm */
auth_key, /* subject station authenticated using any algorithm except open system */
assoc, /* subject station associated */
dis_assoc; /* 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) ;
endnewtype AuthType ;
newtype AuthTypeSet  powerset(AuthType) ; endnewtype AuthTypeSet ;

BssType -- used for <BSS type> parameter/description set element for MlmeScan, Join, and Start
newtype BssType  literals infrastructure, independent ; endnewtype BssType ;
newtype BssTypeSet  powerset(BssType) ; endnewtype BssTypeSet ;

CfPriority -- used for <priority> parameter of MAC data service primitives, and for intra-MAC
    signals MsdurRequest, MmRequest, and PduRequest. "datagram" is a special case of "contention"
    used with some [internal] Mmpdu transmissions to suppress the corresponding MmConfirm.
newtype CfPriority  literals contention, contentionFree, datagram ; endnewtype CfPriority ;

MibStatus -- used for <status> parameter of (Mlme|Plme)Get.confirm and (Mlme|Plme)Set.confirm
newtype MibStatus   literals success, invalid, write_only, read_only ; endnewtype MibStatus ;

PwrSave -- used for <power management mode> parameter of MlmePowerMgt.request
newtype PwrSave  literals sta_active, power_save ; endnewtype PwrSave ;

Routing -- used for <routing information> parameter of MAC data service primitives
    Only "null_rt" is accepted from LLC; "other" exists for testing of TxStatus=nonnullSourceRouting.
newtype Routing  literals null_rt, other ; endnewtype Routing ;

RxStatus -- used for <reception status> parameter of MlmeUnitdata.indication
    NOTE: successful and failed are defined, but there are currently no conditions which report failed.
newtype RxStatus  literals successful_rx, failed_rx ; endnewtype RxStatus ;

ScanType -- used for <scan type> parameter of MlmeScan.request
newtype ScanType  literals active_scan, passive_scan ; endnewtype ScanType ;

ServiceClass -- used for <service class> parameter of MAC data service primitives
    An Mpdu requiring strictlyOrdered service has the orderBit=1 in its Frame Control field.
newtype ServiceClass  literals reorderable, strictlyOrdered ; endnewtype ServiceClass ;

Success -- used for <operation successful> parameter of MlmeXyz.confirm primitives
syntype Success = Boolean  endsyntype Success ;

TxStatus -- used for <transmission status> parameter of MlmeUnitdataStatus.indication
    NOTE: All transmission failures (vs. Msdur validation failures) return "undeliverable." Several
    kinds of transmission failures could be distinguished, such as txLifeTimeout, retryLimit, and noBss.
newtype TxStatus  literals successful, undeliverable, excessiveDataLength, nonNullSourceRouting,
    unsupportedPriority, unavailablePriority, unsupportedServiceClass, unavailableServiceClass ;
endnewtype TxStatus ;

/* ENUMERATED TYPES for PHY_SAP Parameters */

CcaStatus -- used for <status> parameter of PhyCca.indicate
newtype CcaStatus  literals busy, idle ; endnewtype CcaStatus ;

PhyRxStat -- used for <error> parameter of PhyEnd.indicate
newtype PhyRxStat  literals no_error, fmt_violation, carrier_lost, unsupt_rate ; endnewtype PhyRxStat ;
/* INTRA-MAC REMOTE VARIABLES {not part of the MIB} -- names begin with "m" */
/* mActingAsAp =true when this station is operating as an Access Point (from SMT_Requests) */
remote mActingAsAp Boolean nodelay;

/* mAssoc =true if this station is associated with an AP (from Mlme_Req_Rsp) */
remote mAssoc Boolean nodelay;

/* mAtimW =true if the ATIM window is in progress (from Synchronization) */
remote mAtimW Boolean nodelay;

/* The identifier of the current (I)BSS; =nullAddr if not started/joined a BSS (from MIB) */
remote mBssId MacAddr nodelay;

/* mCap holds capability info <<ADDED TO>> join request (from Synchronization) */
remote mCap Octetstring nodelay;

/* mCfp =true when a contention free period is in progress (from Synchronization) */
remote mCfp Boolean nodelay;

/* mIbss =true when this station is a member of an independent BSS (from MIB) */
remote mIbss Boolean nodelay;

/* The current number of beacon intervals between waking up at TBTT (from SMT_Requests) */
remote mListenInt Integer nodelay;

/* Time that the current NAV setting will end, <now when NAV is clear (from Channel_State) */
remote mNavEnd Time nodelay;

/* Scheduled Time of the next medium-control boundary; =0 if none (from Synchronization) */
remote mNextBdry Time nodelay;

/* Time at which the next beacon interval is scheduled to begin (from Synchronization) */
remote mNextTbtt Time nodelay;

/* mPcAvail =true if a point coordinator is operating in this BSS (from Mlme_Req_Rsp) */
remote mPcAvail Boolean nodelay;

/* mPcDlvr =true if the CF-period is used for delivery only (from Mlme_Req_Rsp) */
remote mPcDlvr Boolean nodelay;

/* mPcPoll =true if the CF-period is used for delivery and polling (from Mlme_Req_Rsp) */
remote mPcPoll Boolean nodelay;

/* mPsm =[sta_active | power_save] for the current power save mode (from SMT_Requests) */
remote mPsm PwrSave nodelay;

/* mRxA =true when the PHY indicates that a reception is in progress (from Validate_Mpdu) */
remote mRxA Boolean nodelay;

/* The name identifying the current (I)BSS; =null if not started/joined (from MIB) */
remote mSsId Octetstring nodelay; /* maximum length=32 */

/* If aPrivacyOptionImplemented=true, the remote RC4 is available for WEP PRNG. The RC4 algorithm is not included in this formal description. For technical and licensing information on RC4, contact RSA Data Security, Inc. (see 8.2.4). */
remote procedure RC4 nodelay; ppar Octetstring, Integer; returns Octetstring;

/* PLACEHOLDERS FOR MLME/PLME GET/SET PARAMETER VALUES */
/* */
MibAtrib is a placeholder until integration of the MIB using Z.105 */
syntype MibAtrib = Charstring endsyntype MibAtrib;
/* */
MibValue is a placeholder until integration of the MIB using Z.105 */
syntype MibValue = Integer endsyntype MibValue;
Package macsorts

/* NAMED STATIC DATA VALUES -- names begin with "s" in the form "sNameOfItem" */
synonym sMaxMsduLng  Integer = 2304 ; /* Maximum number of octets in an MSDU passed to or from LLC */
synonym sMacHdrLng  Integer = 24 ; /* Number of octets in the basic MAC header for Data and Mgmt frames (without WEP) */
synonym sWepHdrLng  Integer = 28 ; /* Number of octets added to a PDU when using WEP (both IV/keyID and ICV fields) */
synonym sWdsAddLng  Integer = 6 ; /* this is the length of the addr4 field */
synonym sWepAddLng  Integer = 8 ; /* Number of octets added to the MAC header for Wireless Distribution System transfers */
synonym sWepAddLng  Integer = 8 ; /* Number of octets in the basic MAC header plus IV/keyID for Data and Mgmt frames with WEP */
synonym sWdsAddLng  Integer = 6 ; /* 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+sMacHdrLng+sWdsAddLng+sWepAddLng+sCrcLng) ;
syntax FrameIndexRange = Integer  constants 1:sMaxMpduLng  endsyntax FrameIndexRange ; /* 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 */
synonym sMinFragLng  Integer = 256 ; /* NOTE: Fix the MIB, which no longer lists this minimum. */
synonym sMaxFragNum  Integer = (sMaxMsduLng / (sMinFragLng-sMacHdrLng-sWepAddLng-sCrcLng)) ; /* Maximum fragment number and corresponding index range */
synonym sAckCtsLng  Integer = 123 ; /* Number of bits in ACK and CTS control frames */
synonym sCanBeAp  Boolean = false ; /* sCanBeAp =true if this station is capable of operating as an Access Point */
synonym sCanBePc  Boolean = true ; /* sCanBePc =true if this station is capable of serving as a Point Coordinator */
synonym sCfPollable  Boolean = true ; /* sCfPollable =true if this station is capable of responding to polls from a point coordinator */
synonym sCanBePc  Boolean = false ; /* sCanBePc =true if this station is capable of serving as a Point Coordinator */
synonym sCanBeAp  Boolean = false ; /* set to correct value at each kind of station */
synonym sMinFragLng  Integer = 256 ; /* NOTE: Fix the MIB, which no longer lists this minimum. */
synonym sMaxFragNum  Integer = (sMaxMsduLng / (sMinFragLng-sMacHdrLng-sWepAddLng-sCrcLng)) ; /* Maximum fragment number and corresponding index range */
synonym sAckCtsLng  Integer = 123 ; /* Number of bits in ACK and CTS control frames */
synonym sVersion  Integer = 0 ; /* Protocol version number supported by this version of the MAC */
synonym sCanBeAp  Boolean = false ; /* set to correct value at each kind of station */
synonym sCanBePc  Boolean = true ; /* set to correct value at each kind of station */
synonym sCanBePc  Boolean = false ; /* set to correct value at each kind of station */
synonym sCanBeAp  Boolean = false ; /* set to correct value at each kind of station */

/* STATION CONFIGURATION FLAGS {supplementary to MIB} -- names begin with "s" */
synonym sVersion  Integer = 0 ; /* must be =0 for the current MAC */
synonym sCanBeAp  Boolean = true ; /* true if this station is capable of operating as an Access Point */
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 */
synonym sBitTime  Real = 1.0 ; /* this value is for PHYs with 1Mbps minimum basic rate */
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 microseconds, 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 tUsec operator), SDL Duration (using the dUsec 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 Usec inherits Integer operators all;
adding operators
dUsec : Usec -> Duration;
tUsec : Usec -> Time;
uTime : Time -> Usec;
max : Usec,Usec -> Usec;
min : Usec,Usec -> Usec;
axioms
  for all u, w in Usec (  
    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) ==> tUsec(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) ==> dUsec(u) == duration!(1024*r) ;  
  ) ) ;
for all u in Usec (  
  u2K(u) == u / 1024 ;
  k2U(k) == k * 1024 ;  
) ;
axioms
  constants >=0 /* constrain value range to non-negative, as with Natural */
endnewtype Usec ;
```

Kmicrosecond sort -- (Kusec) = (1024 * Usec)

```
newtype Kusec inherits Integer operators all;
adding operators
dKusec : Kusec -> Duration;
tKusec : Kusec -> Time;
u2K : Usec -> Kusec;
k2U : Kusec -> Usec;
axioms
  for all k in Kusec (  
    for all t in Time ( for all r in Real (  
      r = float(k) ==> tKusec(k) == time!(duration!(1024*r)) ;
      r = float(k) ==> u2K(tUsec(time!(duration!(1024*r)))) == k ;  
    ) ) ;
  ) ;
for all d in Duration ( for all r in Real (  
    r = float(k) ==> dKusec(k) == duration!(1024*r) ;
  ) ) ;
for all u in Usec (  
  u2K(u) == u / 1024 ;
  k2U(k) == k * 1024 ;  
) ;
axioms
  constants >=0 /* constrain value range to non-negative, as with Natural */
endnewtype Kusec ;
```
/* 0-ORIGIN STRING GENERATOR */
/* String0 generator (derived from Z.105, Annex A) creates strings of any sort, indexed starting */
/* with 0 rather than 1. String0 includes all string operators, and adds Tail (all but first item), /*
/* Head (all but last item), and aggregators S2, S3, S4, S6, and S8 (String0 of listed length). */
generator String0(type Item, literal Emptystring)
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 */
  Extract!  :  String0, Integer  ->  Item ;  /* get item from string */
  Modify!   :  String0, Integer, Item     ->  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 ;
S6 : Item, Item, Item, Item, Item, Item  ->  String0 ;
S8 : Item, Item, Item, Item, Item, Item, Item, Item  ->  String0 ;
axioms   for all item0, item1, item2, item3, item4, item5, item6, item7 in Item {
  for all s, s1, s2, s3 in String0 ( for all i, j in Integer {
    s // Emptystring == s ;      Emptystring // s == s ;
    (s1 // s2) // s3 == s1 // (s2 // s3) ;
    type String Length(Emptystring) == 0 ;
    type String Length(MkString(item0)) == 1 ;
    type String Length(s1 // s2) == Length(s1) + Length(s2) ;
    i < Length(s) ==> Extract!(s,i) == Extract!(s,i+1) ;
    i >= Length(s)         ==>  Extract!(s,i) == Error! ;
    i < 0 or i >= Length(s) ==>  Extract!(s,i) == Error! ;
    First(s) == Extract!(s,0) ;      Last(s) == Extract!(s,Length(s)-1) ;
    i >= 0 and i <= Length(s-1) == Substr(s,i) == Substr(s,i,Length(s)-1) ;
    i > 0 and j > 0 and i+j <= Length(s) ==> Substr(s,i,j) == Substr(s,i,j-1) //
    MkString(Extract!(s,i+j-1)) ;
    i < 0 or i > Length(s) => Substr(s,i,j) == Error! ;
    Modify!(s,i,item0) ==
      Substr(s,0,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) ;
    S6(item0,item1,item2,item3,item4,item5) == MkString(item0) // MkString(item1) //
    MkString(item2) // MkString(item3) // MkString(item4) // MkString(item5) ;
    S8(item0,item1,item2,item3,item4,item5,item6,item7) ==
      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 ('1011B) 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 '=>' act bitwise on Bitstring operands. For dyadic operators, the length of the result is equal to the longer of the source operands. */
newtype Bitstring String0(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 ;
  "=>" : Bitstring, Bitstring -> Bitstring ;
noequality ;
axioms
  macro Hex_axioms ; /* macro Binary_axioms ; */
for all s, s1, s2, s3 in Bitstring (  
s = s == True ;
s1 = s2 == s2 = s1 ;
s1 /= s2 == not ( s1=s2 ) ;
s1 = s2 == True => s1 = s2 ;
((s1 = s2) and (s2 = s3)) => s1 = s3 == True ;
((s1 = s2) and (s2 /= s3)) => s1 = s3 == False ;
for all b, b1, b2 in Bit (  
  not("") == ";
  not(MkString(b) // s) == MkString( not(b) ) // not(s) ;
  " and " == ;
  Length(s) > 0 => " and s == MkString(0) and s ;
  Length(s) > 0 => s and " == s and MkString(0) ;
  (MkString(b1) // s1) and (MkString(b2) // s2)
  == MkString(b1 and b2) // ( s1 and s2 ) ;
  s1 or s2 == not (not s1 and not s2) ;
  s1 xor s2 == (s1 or s2) and not(s1 and s2) ;
  s1 => s2 == not (not s1 and s2) ;  ) ) ;
map for all b1,b2 in Bitstring literals (  
for all bs1, bs2 in Charstring literals (  
  /* connection to the String generator */
  for all b in Bit literals (  
    Spelling(b1) = "" // bs1 // bs2 // ", Spelling(b2) = "" // bs2 // "",  
    Spelling(b) = bs1 => b1 == MkString(b) // b2 ; ) ) ) ;
endnewtype Bitstring ;
/* OCTET and OCTETSTRING SORTS (somewhat influenced by Z.105, Annex A) */
/* 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. 0xD5). 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)
i:= 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 ;
    octetVal  :  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 < 0  ==>  mkOctet(i) == error! ;          z = MkString(0)  ==>  octetVal(z) == 0 ;          z = MkString(0)  ==>  octetVal(z) == 1 ;          length(z) > 1 and length(z) <= 8  ==>  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. */
newtype Octetstring String0 (Octet,null)
  adding literals
  O1, O2, O3, O4 ;
operators
  Bit_String    :  Octetstring  ->  Bitstring ;
  Octet_String  :  Bitstring    ->  Octetstring ;
  mkOS  :  Integer,Integer  ->  Octetstring ; /* mkstring(mkOctet(i)) 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
                                b1 == Octet_String(b)(1) == MkString(b1 or 0x00) ; /* expand b1 to 8 bits */
                                b == b1 // b2 ; Length(b1) == 8
                                b == Octet_String(b)(1) // Octet_String(b)(2) ;
  for all i, k in Integer (          k = 1  ==>  mkOS(i,k) == MkString(mkOctet(i)) ;          k > 1  ==>  mkOS(i,k) == MkString(mkOctet(i) // MkString(0x00)) ;          k < 0  ==>  error! ; ) ;
    O1 == mkstring(0x00) ; O2 == S2(0x00,0x00) ; O3 == O1 // O2 ; O4 == O2 // O2 ; ) ) ) ;
map for all o1, o2 in Octetstring literals ( for all b1, b2 in Bitstring literals ( Spelling(o1) = Spelling(b1) , Spelling(o2) = Spelling(b2) == o1 = o2 == b1 = b2 ; ) ) ) ;
endnewtype Octetstring ;
/* 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 ;
  isBcst   :  MacAddr  ->  Boolean ;
  isLocal  :  MacAddr  ->  Boolean ;
  adrOs    :  MacAddr  ->  Octetstring ;
axioms   for all m in MacAddr ( 
  (length(m) = 6) and (extract!(m,0) and 0x01 = 0x01)  ==>  isGroup(m) == true ;
  (length(m) = 6) and (extract!(m,0) and 0x01 = 0x00)  ==>  isGroup(m) == false ;
  (length(m) = 6) and (m = S6(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF))  ==>  isBcst == true ;
  (length(m) = 6) and (m /= S6(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF))  ==>  isBcst == false ;
  (length(m) = 6) and (extract!(m,0) and 0x02 = 0x02)  ==>  isLocal == true ;
  (length(m) = 6) and (extract!(m,0) and 0x02 = 0x00)  ==>  isLocal == false ;
  length(m) /= 6  ==>  Error!  /* common error! term */ ;
  for all o in Octetstring ( 
    m = MacAddr!o ==  adrOs(m) = o ; ) ) ;
endnewtype MacAddr ;

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

synonym BcstAddr  MacAddr = S6(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF) ;

/* Broadcast Address */

synonym NullAddr  MacAddr = S6(0x00, 0x00, 0x00, 0x00, 0x00, 0x00) ;

/* BSS DESCRIPTION SORTS */

/* The BssDscr structure is used in parameters of MlmeScan.confirm and MlmeJoin.request */

newtype BssDscr Struct
  bdBssId      MacAddr ;
  bdCap        Capability ;    /* capability information <<ADDED -- NOT IN 10.3.3.1>> */
  bdSsId       Octetstring ;   /* max length=32, not enforced by data structure */
  bdType       BssType ;
  bdBcnInt     Kusec ;
  bdTstamp     Octetstring ;   /* 8 Octets from Probe Response or Beacon frame */
  bdPhyParms   PhyParms ;      /* empty if inapplicable for active PHY */
  bdCfParms    CfParms ;       /* empty if no point coordinator in BSS */
  bdIbssParms  IbssParms ;     /* empty if infrastructure BSS */
endnewtype BssDscr ;

/* set of Bss Descriptors */
newtype BssDscrSet  powerset(BssDscr)   endnewtype BssDscrSet ;
/* DUPLICATE FILTERING SUPPORT STRUCTURES */
/* Range of possible fragment numbers */
synctype FragNum = Integer  constants 0:sMaxFragNum  endsynctype FragNum;
/* Range of possible sequence numbers */
synctype SeqNum = Integer  constants 0:4095  endsynctype SeqNum;
/* Tuple structure (used for duplicate filtering and Msdu/Mmpdu reassembly. */
newtype Tuple  Struct
  full  Boolean ;   /* =true if Tuple contains valid/current frame information */
  ta    MacAddr ;   /* transmitting station (Addr2) */
  sn    SeqNum  ;   /* Msdu/Mmpdu sequence number */
  fn    FragNum ;   /* Mpdu 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 */
synctype CacheIndex = Integer  constants 1:tupleCacheSize  endsynctype 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,noInt) ;  endnewtype Intstring ;
/* QUEUE GENERATOR */
/* Queue generator (a variant of the String0 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 String0, 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)
literals Emptyqueue;
operators
MkQ       :  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 */
SubQ      :  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 (    /* constructors are Emptyqueue, MkQueue, and "/"; equalities between constructor terms */
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) ;
i >= Length(q1) ===> Extract!(q1 // q2,i) == Extract!(q2,i-Length(q1)) ;
i < 0 or i >= Length(q) ===> Extract!(q,i) == Error! ;
/* definition of First and Last by other operations */
First(q) == Extract!(q,0) ;
Last(q) == Extract!(q,Length(q)-1) ;
/* definition of SubQ(q,i,j) by induction on j. Error! cases handled separately */
i > 0 and i <= Length(q) ===> SubQ(q,i,0) == Emptyqueue ;
i > 0 and j > 0 and i+j < Length(q) ===> SubQ(q,i,j) == SubQ(q,i,j-1) //
MkQueue(Extract!(q,i+j)) ;
i < 0 or j < 0 or i+j > Length(q) ===> SubQ(q,i,j) == Error! ;
/* definition of Modify!, Head, Tail, Qfirst, and Qlast by other operations */
Modify!(q,i,item0) == SubQ(q,0,i) // MkQueue(item0) // SubQ(q,i+1,Length(q)-i-1) ;
Head(q) == SubQ(q,0,Length(q)-1) ;
Tail(q) == SubQ(q,1,Length(q)-1) ;
MkQueue(Extract!(q,i+j)) ;
Qfirst(q,item0) == MkQueue(item0) // q ;
Qlast(q,item0) == q // MkQueue(item0) ;
)
endgenerator Queue ;
Package macsorts

newtype FragArray = Array(FragNum, Frame) ;

newtype FragSdu = Struct
  fTot    FragNum ; /* number of fragments in pdus FragArray */
  fCur    FragNum ; /* number of next fragment to be sent */
  fAnc    FragNum ; /* number of next fragment to announce in ATIM or TIM */
  eol     Time ; /* =0 until first Tx attempt, then =now+dusec(aMaxTxMsduLifetime) */
  sqf     SeqNum ; /* SDU sequence number (set, along with eol, at first Tx attempt) */
  src     Integer ; /* short retry counter */
  lrc     Integer ; /* long retry counter */
  dst     MacAddr ; /* destination address */
  grpa    Boolean ; /* =true if RA (not DA) is group address */
  psm     Boolean ; /* =true if RA (not DA) is not known to be active */
  cnfTo   PId ; /* address of process to send confirmation */
  per     CfPriority ; /* requested priority (from LLC) */
  pdus    FragArray ; /* array of Frame to hold fragments */
endnewtype FragSdu ;

Queue of FragSdu for power save buffering, etc.
Searchable using the Qsearch operator

newtype SduQueue = Queue(FragSdu, emptyQ) ;

adding operators
  Qsearch : SduQueue, MacAddr -> Integer ;
  Operator Qsearch ;
  Fpar que SduQueue, val MacAddr ; returns result Integer ; referenced ;
endnewtype SduQueue ;
/* REASSEMBLY DATA STRUCTURES */

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
  inUse Boolean ; /* =true if >=1 fragments are buffered in this instance of PartialSdu */
  rta MacAddr ; /* transmitting station (Addr2) */
  rsn SeqNum ; /* Msdu/Mmpdu sequence number */
  rCur FragNum ; /* Mpdus fragment number of most recent fragment */
  reol Time ; /* =now+dUsec(aMaxReceiveLifetime) at first fragment */
  rsdu Frame ; /* Mpdus concatenated into this buffer, Mac header from first Mpdus */
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. */
synotype reasmIndex Integer constants 1:reasmSize endsynotype reasmIndex ;

Array of PartialSdu for use reassembling fragmented Msdu and Mmpdu.
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 (entry!rta=addr) and (entry!rsn=seq) and (entry!rCur=fagr-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,
frg 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 sMaxSId Integer = 2007 ;
synotype StationId = Integer constants 0:sMaxSId endsynotype StationId ;

define size of StationId table (<= sMaxSId+1 ) */
synonym sSIdTableSize Integer = 2008 ;

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

newtype StdTable Array(StationId,MacAddr);
adding operators
  addrTold : StdTable,MacAddr -> Integer ;
  operator addrTold ; fpar tbl StdTable, val MacAddr ; returns Integer ;
endnewtype StdTable ;
SORTS FOR CRC-32 (CRC and ICV values)

- Crc is a subtype of Octetstring with added operators:
  - Crc32(crc,octet), which returns an updated Crc value including the new octet, and
  - Mirror(crc), which returns a Crc value with the order of the octets and of the bits in each octet reversed, suitable for transmission MSb-first.

Crc variables must be defined such that they are exactly 4 octets long. The preferred ways to achieve this are to use the S4 aggregation operator or initCrc synonym.

```haskell
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)

```haskell
synonym initCrc  Crc = S4(0xFF, 0xFF, 0xFF, 0xFF) ;
```

Valid Crc value after accumulation of Crc32 on Pdu including Crc field

```haskell
synonym goodCrc  Crc = S4(0x7B, 0xDD, 0x04, 0xC7) ;
```

SORTS for WIRED-EQUIVALENT PRIVACY (WEP)

- define length of KeyVector and nullKey value

```haskell
syntype KeyIndex = Integer  constants 1:4   endsyntype KeyIndex ;
synonym nullKey  Octetstring = O3 // O2 ;
syntype PrngKey = Octetstring  default O3 // O2  endsyntype PrngKey ;
```

- KeyVector for default WEP keys. Array of Octetstring indexed by KeyIndex.
- Number of entries in aWepKeyMapping array at this station.
- KeyMap structure -- used as elements of KeyMapArray

```haskell
newtype KeyVector  Array(KeyIndex,PrngKey) ;   endnewtype KeyVector ;
```

```haskell
newtype KeyMapArray  Array(KeyMappingRange, KeyMap) ;
adding operators
  keyLookup  :  MacAddr, KeyMapArray,  Integer  ->  KeyMap ;
operator keyLookup ;
  fpar luaadr MacAddr,  kma KeyMapArray,  kml Integer ;   returns KeyMap ;   referenced ;
endnewtype KeyMapArray ;
```
/*     FRAME SORT     */
/* Frame is a subtype of Octetstring with added operators for creating MAC headers, 
extracting each of the header fields and some management frame fields, and modifying 
most header fields and some other fields. There are no operators for the frame body, 
IV, ICV, and CRC fields, which are handled directly as Octetstrings. */
newtype Frame   inherits Octetstring   operators all ;
adding operators
  mkFrame      :  TypeSubtype,MacAddr,MacAddr,Octetstring  ->  Frame ;  /* Data or Mgt frame */
  mkCtl        :  TypeSubtype,Octetstring,MacAddr  ->  Frame ;   /* make Control frame */
  protocolVer  :  Frame  ->  Integer ;       /* Protocol version field (2 bits) */
  basetype     :  Frame  ->  BasicType ;   /* Type field (2 bits) */
  f$type        :  Frame  ->  TypeSubtype ;   /* Type & Subtype fields (6 bits) */
  setFtype     :  Frame, TypeSubtype  ->  Frame ;
  toDs         :  Frame  ->  Bit ;           /* To DS bit (1 bit) */
  setToDs      :  Frame, Bit  ->  Frame ;
  frDs         :  Frame  ->  Bit ;           /* From DS bit (1 bit) */
  setFrDs      :  Frame, Bit  ->  Frame ;
moreFrag     :  Frame  ->  Octetstring ;        /* More Fragments bit (1 bit) */
  setMoreFrag  :  Frame, Bit  ->  Frame ;
  retryBit     :  Frame  ->  Bit ;           /* Retry bit (1 bit) */
  setRetryBit  :  Frame, Bit  ->  Frame ;
  pwrMgt       :  Frame  ->  Bit ;           /* Power Management bit (1 bit) */
  setPwrMgt    :  Frame, Bit  ->  Frame ;
  moreData     :  Frame  ->  Bit ;           /* More Data bit (1 bit) */
  setMoreData  :  Frame, Bit  ->  Frame ;
wepBit       :  Frame  ->  Bit ;           /* WEP bit (1 bit) */
  setWepBit    :  Frame, Bit  ->  Frame ;
  orderBit     :  Frame  ->  Bit ;           /* Order [strictly ordered] bit (1 bit) */
  setOrderBit  :  Frame, Bit  ->  Frame ;
  durId        :  Frame  ->  Integer ;       /* Duration/ID field (2 octets) */
  setDurId     :  Frame, Integer  ->  Frame ;
  addr1        :  Frame  ->  MacAddr ;       /* Address 1 [RA/DA] field (6 octets) */
  setAddr1     :  Frame, MacAddr  ->  Frame ;
  addr2        :  Frame  ->  MacAddr ;       /* Address 2 [TA/SA] field (6 octets) */
  setAddr2     :  Frame, MacAddr  ->  Frame ;
  addr3        :  Frame  ->  MacAddr ;       /* Address 3 [BssId/SA/DA] field (6 octets) */
  setAddr3     :  Frame, MacAddr  ->  Frame ;
  addr4        :  Frame  ->  MacAddr ;       /* Address 4 [WDS-SA] field (6 octets) */
  insAddr4     :  Frame, MacAddr  ->  Frame ;
  seq          :  Frame  ->  SeqNum ;        /* Sequence Number field (12 bits) */
  setSeq       :  Frame, SeqNum  ->  Frame ;
  frag         :  Frame  ->  FragNum ;       /* Fragment Number field (4 bits) */
  setFrag      :  Frame, FragNum  ->  Frame ;
  ts           :  Frame  ->  Time ;          /* Timestamp field (beacon, probe_rsp; 8 octets) */
  setTs        :  Frame, Time  ->  Frame ;
  mkElem       :  ElementID, Octetstring  ->  Octetstring ;   /* make element, 1st parm is ID */
  getElem      :  Frame, ElementID  ->  Octetstring ;  /* obtain element from frame body */
  status       :  Frame  ->  StatusCode ;    /* Status Code field (2 octets) */
  setStatus    :  Frame, StatusCode  ->  Frame ;
/* 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) */
asgnSId : Frame -> StationId; /* Station ID field (2 octets) */
setAsgnSId : Frame, StationId -> Frame;
curApAddr : Frame -> MacAddr; /* Current AP Address field (6 octets) */
capA : Frame, Capability -> Bit; /* Capability Info field, Asoc/Reasoc (2 octets) */
setCapA : Frame, Capability, Bit -> Frame;
capB : Frame, Capability -> Bit; /* Capability Info field, Beacon/ProbeRsp (2 octets) */
setCapB : Frame, Capability, Bit -> Frame;
keyId : Frame -> KeyIndex; /* Key ID subfield of ICV field (2 bits of 1 octet) */
setKeyId : Frame, KeyIndex -> Frame;
Operator getElem; fpar fr Frame, id ElementID; returns Frame; referenced;

/* axioms for Frame sort */

getElem from body of Management frame. If the target element is not present an Octetstring of length zero is returned.

getElem : Frame, id ElementID -> Frame;
/* axioms for Frame sort, continued */

setPwrMgt(f,b) == Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,0,0,0,0,0)) ;
moreData(f) == if (f(1) and 0x20) then 1 else 0 fi ;
setMoreData(f,b) == Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,0,0,0,0,0)) ;
wepBit(f) == if (f(1) and 0x40) then 1 else 0 fi ;
setWepBit(f,b) == Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,0,0,0,0,0)) ;
orderBit(f) == if (f(1) and 0x80) then 1 else 0 fi ;
setOrderBit(f,b) == Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,0,0,0,0,0)) ;
for all c in Capability {
  capA(f,c) == (bit_string(substr(f,24,2)) and c) then 1 else 0 fi ;
  setCapA(f,c,b) == substr(f,0,24) // ( bit_string(substr(f,24,2) and (not c)) or (if b then c else 02 fi) ) // substr(f,26,length(f)-26) ;
  capB(f,c) == (bit_string(substr(f,34,2)) and c) then 1 else 0 fi ;
  setCapB(f,c,b) == substr(f,0,34) // ( bit_string(substr(f,34,2) and (not c)) or (if b then c else 02 fi) ) // substr(f,36,length(f)-36) ;
for all sq in SeqNum {
  seq(f) == octetVal(f(22) and 0xF0) / 16 + (octetVal(f(23)) * 16) ;
  setSeq(f,sq) == substr(f,0,22) // mkstring((f(22) and 0x0F) or mkOctet((sq mod 16) * 16)) // mkOS(sq / 16, 1) // substr(f,24,length(f)-24) ;
for all fr in FragNum {
  frag(f) == octetVal(f(22) and 0xF0) ;
  setFrag(f,fr) == substr(f,0,22) // mkstring((f(22) and 0xF0) or mkOctet(f(24))) // substr(f,23,length(f)-23) ;
for all tm in Time {
  ts(f) == float (octetVal(f(24)) + (256*(octetVal(f(25))) + (256*(octetVal(f(25))) + (256*(octetVal(f(26))) + (256*(octetVal(f(27))) + (256*(octetVal(f(28))) + (256*(octetVal(f(29))) + (256*(octetVal(f(30))) + (256*(octetVal(f(31))) ))))))))) ;
  setTs(f,tm) == substr(f,0,24) // mkOS(fix(tm), 1) // mkOS((fix(tm)/256), 1) // mkOS((fix(tm)/65536), 1) // mkOS((fix(tm)/16777216), 1) // mkOS((fix(tm)/4294967296), 1) // mkOS((fix(tm)/4294967296), 1) // mkOS((fix(tm)/4294967296), 1) // mkOS((fix(tm)/4294967296), 1) // mkOS((fix(tm)/4294967296), 1) // substr(f,32,length(f)-32) ;
for all stat in StatusCode {
  status(f) == substr(f,26,2) ;
  setStatus(f,stat) == substr(f,0,26) // stat // substr(f,28,length(f)-28) ;
  authStat(f) == substr(f,28,2) ;
for all rea in ReasonCode {
  reason(f) == substr(f,24,2) ;
for all alg in AuthType {
  authType(f) == substr(f,24,2) ;
for all u in Kusec {
  beaconInt(f) == octetVal(f(32)) + (octetVal(f(33)) * 256) ;
  listenInt(f) == octetVal(f(26)) + (octetVal(f(27)) * 256) ;
for all sta in StationId {
  asgnSId(f) == octetVal(f(28)) + (octetVal(f(29)) * 256) ;
  setAsgnSId(f,sta) == substr(f,0,28) // mkOS(sta mod 256, 1) // mkOS(sta / 256, 1) // substr(f,30,length(f)-30) ;
for all kid in KeyIndexRange {
  keyId(f) == octetVal(f(27)) / 64 ;
  setKeyId(f,kid) == Modify!(f,27,mkOS(kid * 64)) ;
  )
}
endnewtype Frame ;
/* ADDITIONAL FRAME FORMAT SORTS */

TypeSubtype defines the 6-bit full frame type identifiers, used by the ftype operator of the Frame sort. */

class TypeSubtype inherits Octetstring operators all;
adding literals
  asoc_req, asoc_rsp, reasoc_req, reasoc_rsp, probe_req, probe_rsp, beacon, atim, disassoc, auth,
deauth, 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));  asoc_rsp == mkstring(S8(0,0,0,1,0,0));
  reasoc_req == mkstring(S8(0,0,0,0,1,0));  reasoc_rsp == mkstring(S8(0,0,0,1,1,0));
  probe_req == mkstring(S8(0,0,0,0,0,1));  probe_rsp == mkstring(S8(0,0,0,1,0,1));
  beacon == mkstring(S8(0,0,0,0,0,0));  atim == mkstring(S8(0,0,1,0,0,0));
  disassoc == mkstring(S8(0,0,0,0,0,1));  auth == mkstring(S8(0,0,0,1,1,0));
  deauth == mkstring(S8(0,0,0,0,0,1));  ps_poll == mkstring(S8(0,0,1,0,1,0));
  rts == mkstring(S8(0,0,1,0,1,1));  cts == mkstring(S8(0,0,1,0,0,1));
  ack == mkstring(S8(0,0,1,0,1,1));  cfend == mkstring(S8(0,0,1,0,1,1));
  cfend_ack == mkstring(S8(0,0,1,0,1,1));  data == mkstring(S8(0,0,0,1,0,0));
  data_ack == mkstring(S8(0,0,0,1,1,0));  data_poll == mkstring(S8(0,0,0,1,1,1));
  data_poll_ack == mkstring(S8(0,0,0,1,1,1));  null_frame == mkstring(S8(0,0,0,1,0,0));
  cfack == mkstring(S8(0,0,0,1,1,0));  cfpoll == mkstring(S8(0,0,1,0,1,0));
  cfpoll_ack == mkstring(S8(0,0,1,0,1,1));
endnewtype TypeSubtype;

BasicTypes defines the 2-bit frame type groups */

class BasicType inherits Bitstring operators all;
adding literals
  control, data, management, reserved;
axioms
  control == S8(0,0,1,0,0,0,0,0);  data == S8(0,0,0,1,0,0,0,0);
  management == S8(0,0,0,0,0,0,0,0);  reserved == S8(0,0,1,1,0,0,0,0);
endnewtype BasicType;

/* ELEMENT IDS */

class ElementId inherits Octetstring operators all;
adding literals
  eSsId, eSupRates, eFhParms, eDsParms, eCfParms, eTim, eIbParms, 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) */
  eIbParms == 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;
/* 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, disas_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) ; disas_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, reasoc_no_asoc, fail_other, unsupt_alg,
  auth_seq_fail, chlng_fail, auth_timeout, ap_full, unsup_rate;
axioms
  successful == mkOS(0,2) ; unspec_failure == mkOS(1,2) ; unsup_cap == mkOS(10,2) ;
  reasoc_no_asoc == mkOS(11,2) ; fail_other == mkOS(12,2) ; unsupt_alg == mkOS(13,2) ;
  auth_seq_fail == mkOS(14,2) ; chlng_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, clbss, cPollable, cPollReq, cPrivacy;
axioms
  cEss     == S8(1,0,0,0,0,0,0,0) // 0x00 ; /* ESS capability */
  clbss    == S8(1,0,0,0,0,0,0,0) // 0x00 ; /* IBSS capability */
  cPollable == S8(0,1,0,0,0,0,0,0) // 0x00 ; /* CF-pollable (sta), PC present (ap) */
  cPollReq  == S8(0,0,1,0,0,0,0,0) // 0x00 ; /* (not) CF poll req (sta), PC polls (ap) */
  cPrivacy  == S8(0,0,0,1,0,0,0,0) // 0x00 ; /* WEP required capability */
endnewtype Capability ;
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 ;

newtype CfParms inherits Octetstring operators all;
adding operators
  cfpCount : CfParms -> Integer ; /* CfpCount field, 1 octet */
  setCfpCount : CfParms, Integer  ->  CfParms ;
  cfpPeriod : CfParms -> Integer ; /* CfpPeriod field, 1 octet */
  setCfpPeriod : CfParms, Integer  ->  CfParms ;
  cfpMaxDur : CfParms -> Kusec ; /* CfpMaxDuration field, 2 octets */
  setCfpMaxDur : CfParms, Kusec -> CfParms ;
  cfpDurRem : CfParms -> Kusec ; /* CfpDurRemaining 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(0) // 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 ;

newtype FhParms inherits Octetstring operators all;
adding operators
  dwellTime : FhParms -> Kusec ; /* Dwell Time field, 2 octets */
  setDwellTime : 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

```plaintext
operator clearTupleCache
defines
   par
   cache TupleCache
   returns TupleCache :

   dcl k CacheIndex

   k := 1
   cache(k).full := false

   k := k + 1

   if (tupleCacheSize = k) then
     Mark all cache entries as empty.
   else
     cache
```

Mark all cache entries as empty.
Operator `searchTupleCache`

```plaintext
fpars
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum;
returns Boolean;

dcl k CacheIndex;
dcl result Boolean;

k := 1

result := (cache(k)!ta=taddr) and (cache(k)!sn=tseq) and (cache(k)!fn=tfrag) and cache(k)!full

k := k + 1

result = (false)

else

(result = (true))

k = (tupleCacheSize)

result
```

Search for exact \{TA, SeqNum, FragNum\} match at non-empty cache entries.
Operator updateTupleCache

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

age:= now,
k:= 1,
oldest:= 1

temp:=
cache(k)

tempfull
= true
(true)

(test:=(temp!ta=taddr) and
(temp!sn=tseq)

(test
=false
(true)

(oldest:= k,
age:= 0

else

k

(=tupleCacheSize)

tempfull:=true,
temp!ta=taddr,
temp!sn=tseq

tempfn:=tfrag,
temp!tRx=tnow

cache(oldest):= temp

cache

If a match is found with [TA,SeqNum], update FragNum and tRx 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 ;
```

dcl k Integer ;
dcl new Bit ;
dcl result Crc ;
dcl temp Bitstring ;

/* 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) ;

k:= 0

k:= k+1

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

temp:= mkstring(new) // head(temp)

new = 1

(temp xor feedback)

else

k

(=7)

result:= octet_string (temp)

result

```
Operator keyLookup

```pascal
if par luadr MacAddr, kma KeyMapArray, km1 Integer ; returns KeyMap ;

luadr = kma(lk)!mappedAddr
lk:= lk + 1
lk = (km1+1)
result!mappedAddr:= luadr,
result!keyOn:= false,
result!wepKey:= nullKey

Return false keyOn status and a null wepKey if the end of the array is reached with addr not found.

result:= kma(lk)
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 addr not found.
Operator getElem

```vhdl
   fpar
   | fr Frame, id ElementId ;
   | returns Frame ;

   dcl k Integer := sMacHdrLn;
   dcl lng, n Integer ;
   dcl info Frame ;

   n := length(fr)

   if type(fr) then
      if (auth) then
         k := k + 6
      elsif (probe_req) then
         k := k + 0
      elsif (beacon, probe_rsp) then
         k := k + 12
      elsif (reasoc_req) then
         k := k + 10
      elsif (asoc_req, asoc_rsp, reasoc_rsp) then
         k := k + 4
      end if

      (n-k) > 0
      info := null
      info := substr(fr,k,1) = Frame!id
      lng := octetVal (fr(k+1))
      info := substr (fr,k+2,lng)
      info := octetVal (fr(k+1)) + 2

      info := substr (fr,k+2,lng)
      info
   else
      info := null
   end if
```

Operator Qsearch

\begin{verbatim}
dcl k Integer;
que SduQueue, val MacAddr;
returns result Integer;
\end{verbatim}

\begin{algorithm}
\begin{enumerate}
\item \textbf{true} \quad que = emptyQ
\item \textbf{false} \quad k := 0
\item \textbf{false} \quad val = que(k).dst
\item \textbf{false} \quad k := k + 1
\item \textbf{true} \quad result := k
\item \textbf{true} \quad k < length(que)
\item \textbf{false} \quad result := -1
\end{enumerate}
\end{algorithm}
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.

remote aStationID MacAddr nodelay;

Maximum time in Kusec before a point coordinator must relinquish the medium. Only used by PCF option at APs.

remote aMediumOccupancyLimit Kusec nodelay;

True if station wakes up to receive every Beacon frame which contains a DTIM element.

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 benefits of writing to this attribute from an external entity are unclear.

remote aAuthenticationType AuthType nodelay;

**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.

remote aAuthenticationAlgorithms AuthTypeSet nodelay;

**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).

remote aDefaultWepKeys KeyVector nodelay; /* name changed to plural because the attribute holds 4 keys */
use macsorts;

Package macmib


/* WEP_KEY_MAPPING_TABLE -- only if aPrivacyOptionImplemented=true */
/*
Array of KeyMap entries, each of which is (mappedAddr MacAddr, keyOn Boolean, wepKey Octetstring).
 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 this array 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.
 write-only; default entries (nullAddr, false, null); declared in MIB; used in Encrypt, Decrypt, KeyLookup; */
remote aWepKeyMapping  KeyMapArray nodelay ;

/* Number of entries in aWepKeyMapping array at this station.
 Actual length of key map array (10 is the minimum per 8.3.2). WARNING: This attribute is not in the ASN.1 MIB.
 read-only; default=10; declared package macsorts; used in Encrypt, Decrypt; */
synonym aWepKeyMappingLength  Integer = package macsorts sWepKeyMappingLength ;

/* end of WepKeyMappingTable */

/* 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.
 read-only; default=false; declared here; used by Filter_Mpdu, Prepare_Mpdu, Mlme_Req_Rsp; */
synonym aPrivacyOptionImplemented  Boolean = true ; /* turned on here for testing of WEP option */

/* True to turn on WEP encryption, false to turn off WEP encryption.
 read-write; default=false; declared in MIB; used in Filter_Mpdu, Prepare_Mpdu, Mlme_Req_Rsp, Sync; */
remote aPrivacyInvoked  Boolean nodelay ;

/* Selects an element of aDefaultWepKeys for encrypting frames to stations for which there is no defined key mapping.
 NOTE: The KeyIndexRange here is 1:4, vs. 0:4 in 8.3.2 and the ASN.1 MIB. At one time WEP was turned off by aWepDefault=0, now done by aPrivacyInvoked=false. There is no need for two WEP on/off mechanisms, so the so the state machines use aPrivacyInvoked, which simplifies converting KeyIndexRange to/from keyId values.
 read-write; default=1; declared in MIB; used in Encrypt; */
remote aWepDefault  KeyIndex nodelay ;

/* True to block acceptance of data frames sent without WEP. False to accept unencrypted data frames.
 read-write; default=false; declared in MIB; used in Filter_Mpdu, Synchronization; */
remote aExcludeUnencrypted  Boolean nodelay ;

/* Count of received WEP frames with acceptable address and CRC values, that are rejected due to ICV errors.
 read-only; default=<counts up from 0>; declared in Filter_MPDU; used in MIB (MlmeGet response); */
remote aIcvErrorCount  Counter32 nodelay ;

/* end of PrivacyGroup */
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.

```
*     OPERATION_GROUP    */
*/
RTS/CTS handshake not performed prior to transmitting and Mpdru or Mmpdu if length(pdu) <= aRtsThreshold.
read-write; default=3000; declared in MIB; used in Prepare_Mpdu, 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_Mpdu, 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_Mpdu, PM_Filter, Transmit_Control; */
remote aLongRetryLimit Integer nodelay ;
* /
The maximum number of octets in any Mpdru or Mmpdu delivered to the PHY. Fragmentation is required if
length(macHeader // {addr4} // {IV} // payload // {ICV} // crc) > aFragmentationThreshold.
read-write; default=2346 [min=256, not checked]; declared in MIB; used in Prepare_Mpdu, Tx_Ctl; */
remote aFragmentationThreshold Integer nodelay ;
* /
Time in Kmicroseconds between the initial attempt to transmit an Msdu, Mmpdu, or fragment thereof
and termination of transmission attempts with MaUnitdatastatus.indication of 'undeliverable' to LLC.
read-write; default=512; declared in MIB; used in PM_Filter, Transmit_Control; */
remote aMaxTransmitMsduLifetime Kusec nodelay ;
* /
This is the unique, 48-bit, individual address assigned to this station at the time of manufacture.
read-only; <each station address is unique>; declared here; used <almost everywhere>; */
synonym aMacAddr MacAddr = S6(0x00,0x11,0x22,0x33,0x44,0x55) ; */
must have exactly 6 octets */
* /
Strings include, at least, the name of the manufacturer and an identifier unique to the manufacturer. */
synonym aManufacturerId Charstring = 'name of manufacturer {etc.}' ;
synonym aProductId Charstring = 'identifier unique to manufacturer {etc.}' ;
*/
Time in Kmicroseconds between the receipt of the first fragment of an Msdu or Mmpdu and termination
of reassembly attempts and reclaiming of the buffers occupied by any received fragments.
read-write [read-only in ASN.1]; default=512; declared in MIB; used in Reassembly; */
remote aMaxReceiveLifetime Kusec nodelay ;*/
end of OperationGroup */
*/
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 GroupAddressesTable */
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.

/* 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):
read-only; default=<counts up from 0>; declared in Tx_Control; used in MIB (MlmeGet.confirm); */
remote aTransmittedFragmentCount 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 received successfully. Success requires protVer=0, addr1=aMacAddress or an acceptable group address plus correct BSSID in addr3 (addr2 if FromDS=0), length within Mpdu size limits, and valid CRC. Sequence, WEP, and reassembly are not considered, so this count includes pdus subsequently discarded as duplicates, ICV failures, or fragments of Msdus exceeding aMaxReceiveLifetime. This access/location information applies to all remaining counters except aFcsErrorCount:
read-only; default=<counts up from 0>; declared in Filter_MPDU; used in MIB (MlmeGet response); */
remote aReceivedFrameCount 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 (MlmeGet 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 + aMacPrcTime) ;
/*
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 = (aTxPlcpDelay + aRxTxSwitchTime + aRxRampOnTime + 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 | 40 for IR];  used in aRxTxTurnaroundTime equation;  */
synonym aTxPlcpDelay 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 */
/*
Maximum time in microseconds for the PMD to turn the transmitter on.
read-only;  default=[8 | 10 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 the 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 = (aRxRfDelay + aRxPlcpDelay + aMacPrcTime + aRxTxTurnaroundTime) ;
*/

end of Part 1, PhyOperationGroup continues . . .  */
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.

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={4 for FH, impl. dep. for DS, 4 for IR}; used in aSifsTime equation; */
synonym aRxRfDelay 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 aSifsTime equation; */
synonym aRxPlcpDelay 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, by definition, uniform for all PHYs}; used in aSifsTime equation; */
synonym aMacPrcTime Usec = 2; /* this value is uniform for all PHYs */

Nominal time in microseconds for the PMD to turn the transmit power amplifier off.
read-only; default={8 for FH, impl. dep. for DS, 0 for IR}; used in <not used>; */
synonym aTxRampOffTime Usec = 8; /* this value is for the FH PHY */

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: aMpduDurationFactor 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 Integer32 which holds the integral part of (MpduDurationFactor * 1e9).
read-only; default={1.03125 for FH, 1.0 for DS, 1.0 for IR}; used in Filter_MPDU, Tx_Control; */
synonym aMpduDurationFactor 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 <not used>; */
synonym aTempType Integer = 01; /* 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 Tx_Control; */
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 */

MAC state machines currently do not reference any attributes in:
PhyAntennaGroup, PhyTxPowerGroup, PhyDsssGroup, PhyStatusGroup,
PhyPowerSavingGroup, RegDomainsSupport, AntennasList, */
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.

*PHY_RATE_GROUP*

Define unsigned byte integer subtype and string thereof for SupportedRates lists.

```
syntype RateVal = Integer   constants 0:255   endsyntype rateVal;
newtype RateString  string(RateVal,noRates)  endnewtype RateString;
```

Null-terminated string of (8-bit) integers representing available PHY tx data rates in units of 100Kb/s. read-only [read-write in ASN.1]; default=<$PHY dep., includes 10>; used in Transmit_Control; */
synonym aSupportedRatesTx Ratestring = mkstring(10) // mkstring(0) ; /* this value for the FH PHY */
```
```
Null-terminated string of (8-bit) integers representing available PHY rx data rates in units of 100Kb/s. read-only [read-write in ASN.1]; default=<$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 = 4095 ; /* 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=<not specified>; used in <not used> ; */
synonym aPrefMaxMpdumaxFragmentLength Integer = aMpdumaxLength ;
```

*PHY_FHSS_GROUP*

Time in nanoseconds(? s/b usec per 14.8.2) for PMD to change from channel 2 to channel 80.
read-only; default=224 ; used in Transmit_Control; */
synonym aHopTime Usec = 224 ;
```
```
Channel number currently set for the frequency synthesizer
read-write; default <none, controlled by MAC> [range 2:80]; used in Transmit_Control; */
remote aCurrentChannelNumber Integer nodelay ;
```
```
Maximum time in ms (? s/b Kusec) the transmitter is permitted to operate on a single channel.
read-only; default=390 (for FCC); used in Synchronization; */
synonym aMaxDwellTime Kusec = 390 ;
```
```
Time in ms (? s/b Kusec) the transmitter currently operates on a single channel.
read-write; default=20 ; used in Synchronization; */
remote aCurrentDwellTime Integer nodelay ;
```
```
Currently selected set of hop patterns.
read-write; default=0 ?? [range 1:3]; used in Transmit_Control; */
remote aCurrentSet Integer nodelay ;
```
```
Currently selected pattern within current set.
read-write; default=<varies with regulatory domain>; used in Transmit_Control; */
remote aCurrentPattern Integer nodelay ;
```
```
Currently selected index into current pattern within current set.
read-write; default=<varies with regulatory domain>; used in Transmit_Control; */
remote aCurrentIndex Integer nodelay ;
```

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

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.

"start timer"
set(end_time, timer)

"call" procedure (parms)

Output symbol with wedge on right side used for signals from PHY & others logically below this process.

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

Output symbol with point to right side used for signals to PHY & others logically below this process.

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

Signals not listed in a state's input symbols are discarded unless they appear in a Save symbol for the state. * Save refers to all signals except those listed in the Input symbol(s).