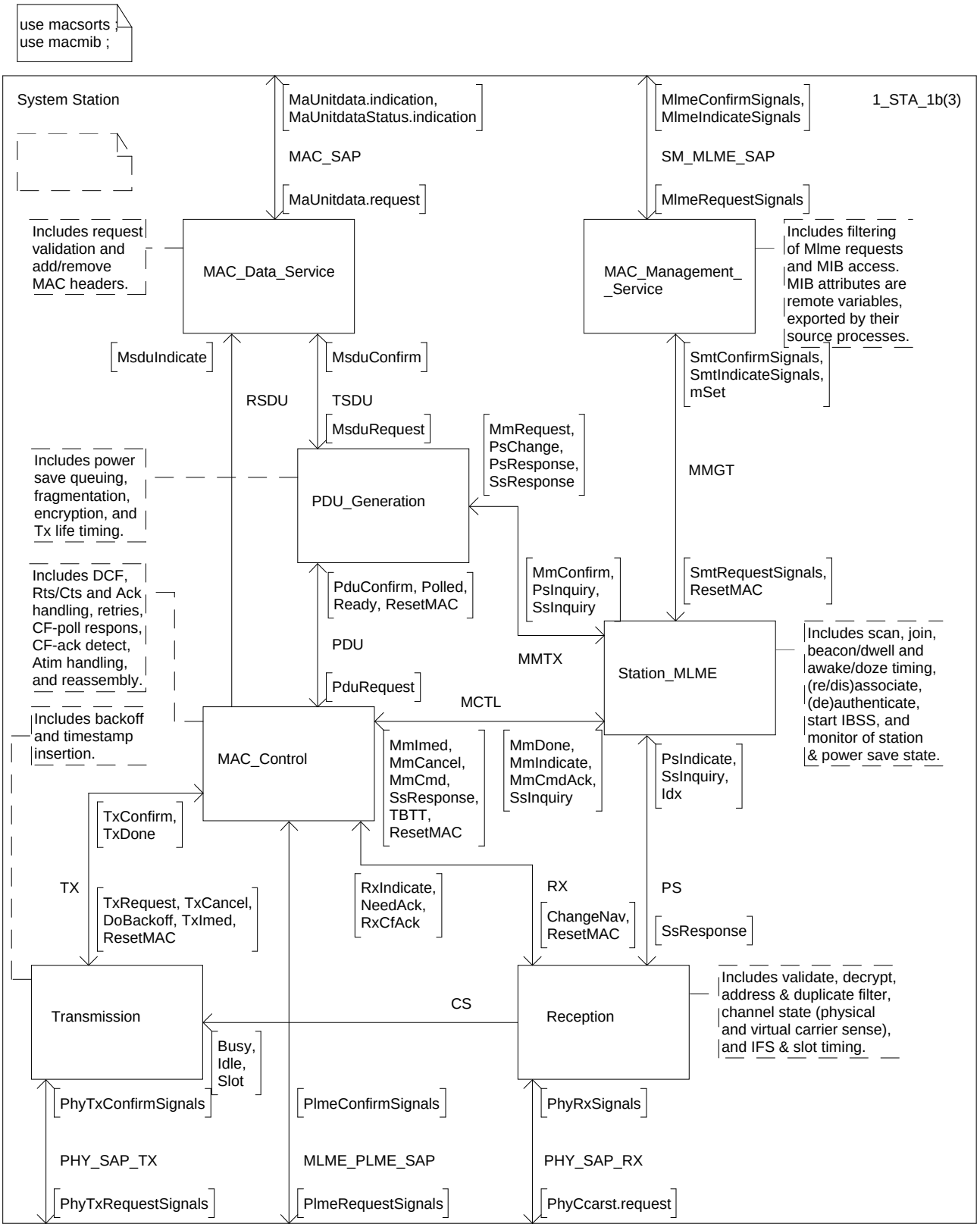

use macsorts ;
use macmib ;
System Station
1_STA_1b(3)
MaUnitdata.indication,
MaUnitdataStatus.indication
MAC_SAP
MaUnitdata.request
MlmeConfirmSignals,
MlmeIndicateSignals
SM_MLME_SAP
MlmeRequestSignals
Includes request validation and add/remove MAC headers.
MAC_Data_Service
MAC_Management_
_Service
Includes filtering of Mlme requests and MIB access. MIB attributes are remote variables, exported by their source processes.
MsduIndicate
MsduConfirm
SmtConfirmSignals,
SmtIndicateSignals,
mSet
RSDU
TSDU
MsduRequest
MmRequest,
PsChange,
PsResponse,
SsResponse
MMGT
Includes power save queuing, fragmentation, encryption, and Tx life timing.
PDU_Generation
Includes DCF, Rts/Cts and Ack handling, retries, CF-poll respons, CF-ack detect, Atim handling, and reassembly.
PduConfirm, Polled,
Ready, ResetMAC
MmConfirm,
PsInquiry,
SsInquiry
SmtRequestSignals,
ResetMAC
PDU
MMTX
Station_MLME
Includes scan, join, beacon/dwell and awake/doze timing, (re/dis)associate, (de)authenticate, start IBSS, and monitor of station & power save state.
PduRequest
MCTL
Includes backoff and timestamp insertion.
MAC_Control
MmImed,
MmCancel,
MmCmd,
SsResponse,
TBTT,
ResetMAC
MmDone,
MmIndicate,
MmCmdAck,
SsInquiry
PsIndicate,
SsInquiry,
Idx
TxConfirm,
TxDone
TX
TxRequest, TxCancel,
DoBackoff, TxImed,
ResetMAC
RxIndicate,
NeedAck,
RxCfAck
RX
ChangeNav,
ResetMAC
PS
SsResponse
Transmission
CS
Reception
Includes validate, decrypt, address & duplicate filter, channel state (physical and virtual carrier sense), and IFS & slot timing.
Busy,
Idle,
Slot
PhyTxConfirmSignals
PlmeConfirmSignals
PhyRxSignals
PHY_SAP_TX
MLME_PLME_SAP
PHY_SAP_RX
PhyTxRequestSignals
PlmeRequestSignals
PhyCcarst.request

```
use macsorts ;
use macmib ;
```

System Station

1_STA_2b(3)

```
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(BssTypeSet,MacAddr,
    Octetstring,ScanType,Intstring,Intstring),
  MlmeSet.confirm(MibStatus,MibAtrib),
  MlmeSet.request(MibAtrib,MibValue),
  MlmeStart.confirm(Success),
  MlmeStart.request(Octetstring,BssType,
    Natural,CfParms,PhyParms) ;
```

```
signal
  MmCancel,
  MmCmd(CtlCmd,Intstring),
  MmCmdAck(Boolean,Intstring),
  MmConfirm(Frame,TxResult),
  MmDone(TxResult),
  MmImed(Frame),
  MmIndicate(Frame,Time,Time,StateErr),
  MmRequest(Frame,CfPriority) ,
  MsduConfirm(Frame,CfPriority,TxResult),
  MsduIndicate(Frame,CfPriority),
  MsduRequest(Frame,CfPriority),
  mSet(MibAtrib,MibValue),
  NeedAck(MacAddr,Time,Duration),
  PduConfirm(FragSdu,TxResult),
  PduRequest(FragSdu),
  PhyCca.indicate(Ccastatus),
  PhyCcarst.confirm,
  PhyCcarst.request,
  PhyData.confirm,
  PhyData.indicate(Octet),
  PhyData.request(Octet),
  PhyRxEnd.indicate(PhyRxStat),
  PhyRxStart.indicate(Natural),
  PhyTxEnd.confirm,
  PhyTxEnd.request,
  PhyTxStart.confirm,
  PhyTxStart.request(Natural),
  PlmeGet.confirm(MibStatus,
    MibAtrib,MibValue),
  PlmeGet.request(MibAtrib),
  PlmeReset.confirm(Success),
  PlmeReset.request,
  PlmeSet.confirm(MibStatus,MibAtrib),
  PlmeSet.request(MibAtrib,MibValue),
  Polled(Boolean,MacAddr,StationId),
  PsChange(MacAddr,PsMode),
  PsIndicate(MacAddr,PsMode),
  PsInquiry(MacAddr,PsMode),
  PsResponse(MacAddr,PsMode),
  Ready(MacAddr,StationId),
  ResetMAC,
  RxCfAck,
  RxIndicate(Frame,Time,Time,Duration),
  Slot,
  SsInquiry(MacAddr),
  SsResponse(MacAddr,
    StationState,StationState),
  TBTT(Duration,Duration,
    Duration,Duration),
  TxCancel,
  TxConfirm(BackoffStatus,Natural),
  TxDone,
  TxImed(Frame),
  TxRequest(Frame,Boolean,
    Natural,Natural) ;
```

```
use macsorts ;
use macmib ;
```

System Station

1_STA_3b(3)

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

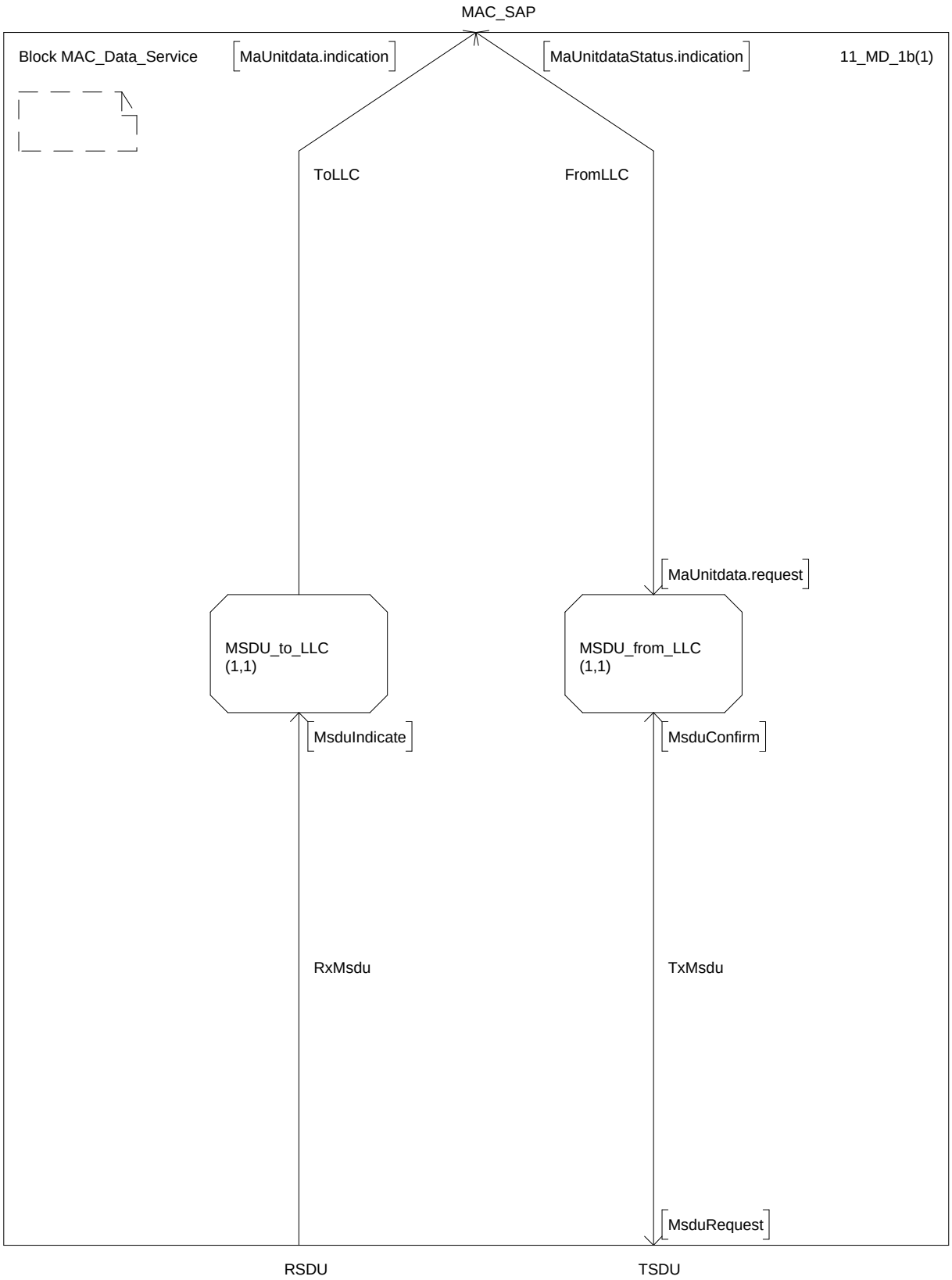
```
signallist
PhyTxRequestSignals=
   PhyTxStart.request,
   PhyTxEnd.request,
   PhyData.request ;
```

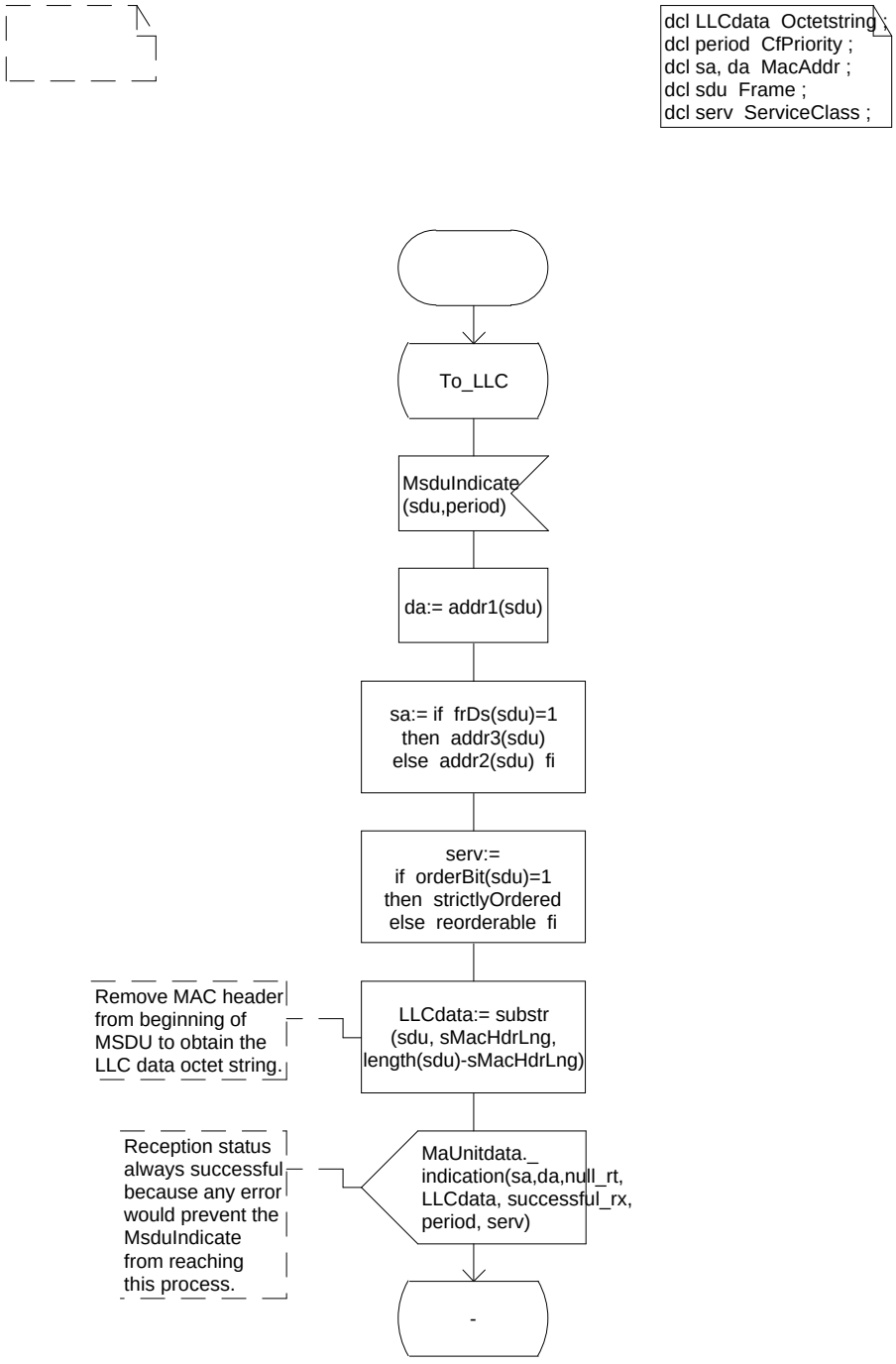
```
signallist
PhyTxConfirmSignals=
   PhyTxStart.confirm,
   PhyTxEnd.confirm,
   PhyData.confirm ;
```

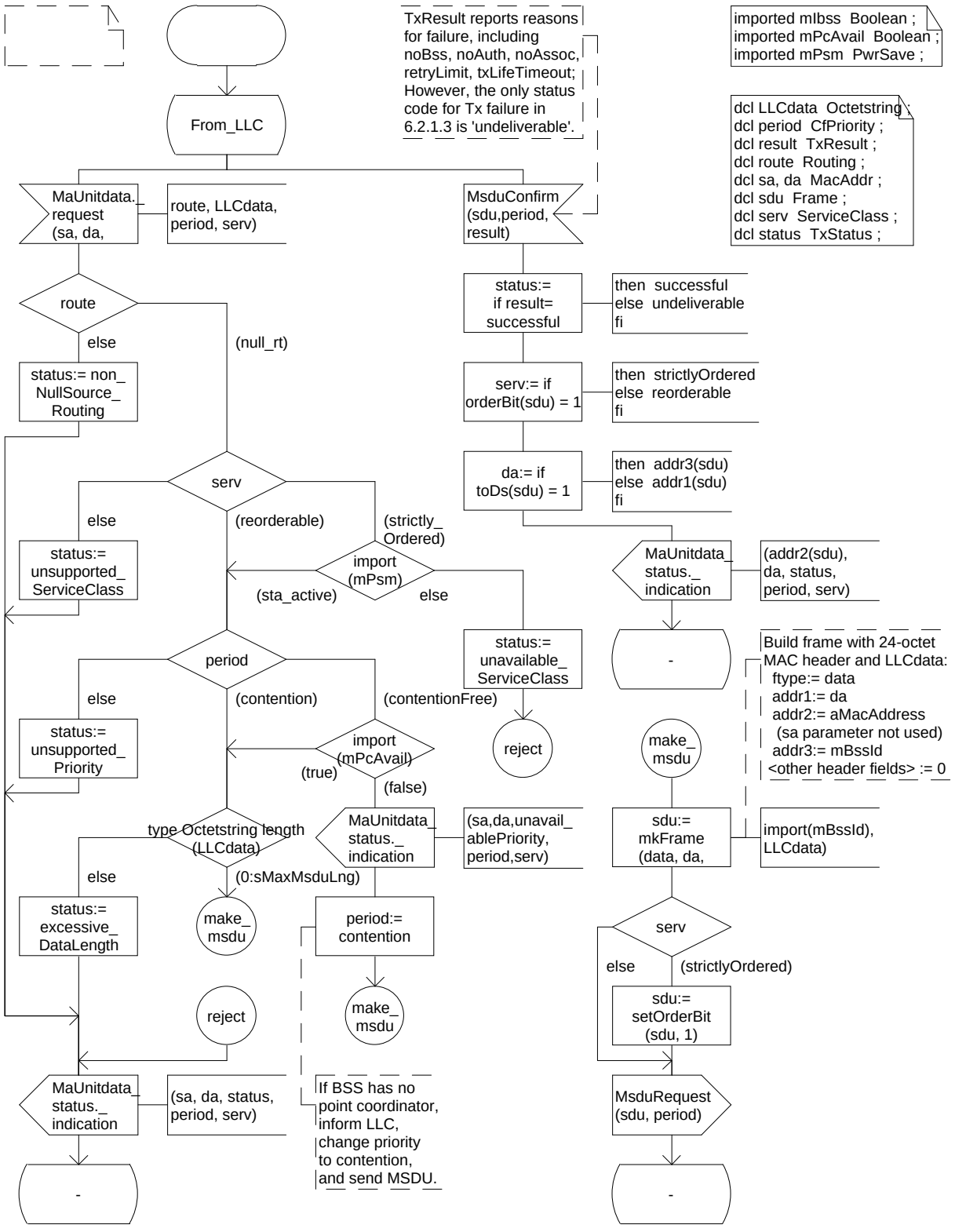
```
signallist
PhyRxSignals=
   PhyRxStart.indicate,
   PhyRxEnd.indicate,
   PhyData.indicate,
   PhyCca.indicate,
   PhyCcarst.confirm ;
```

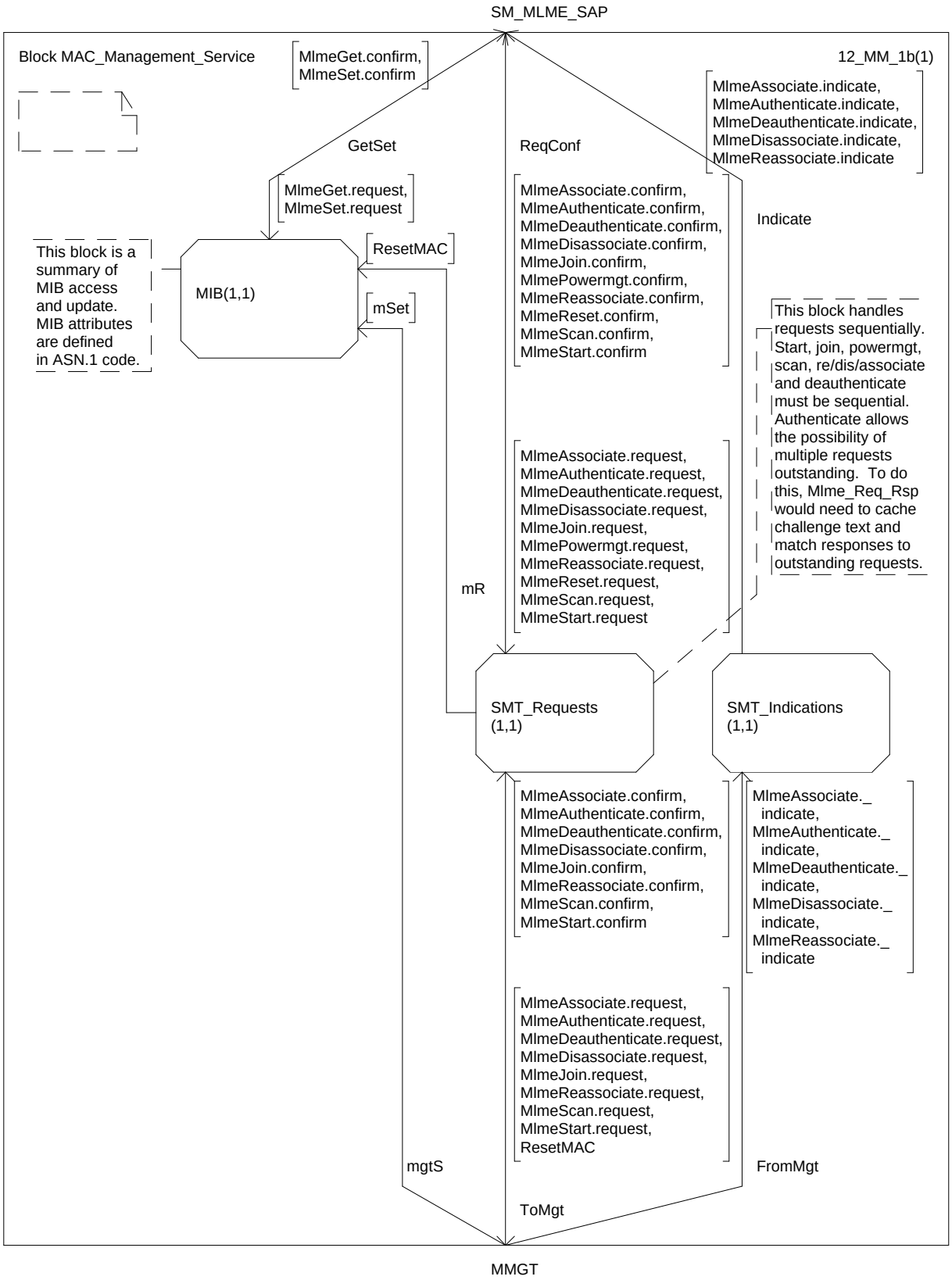
```
signallist
PlmeRequestSignals=
   PlmeGet.request,
   PlmeSet.request,
   PlmeReset.request ;
```

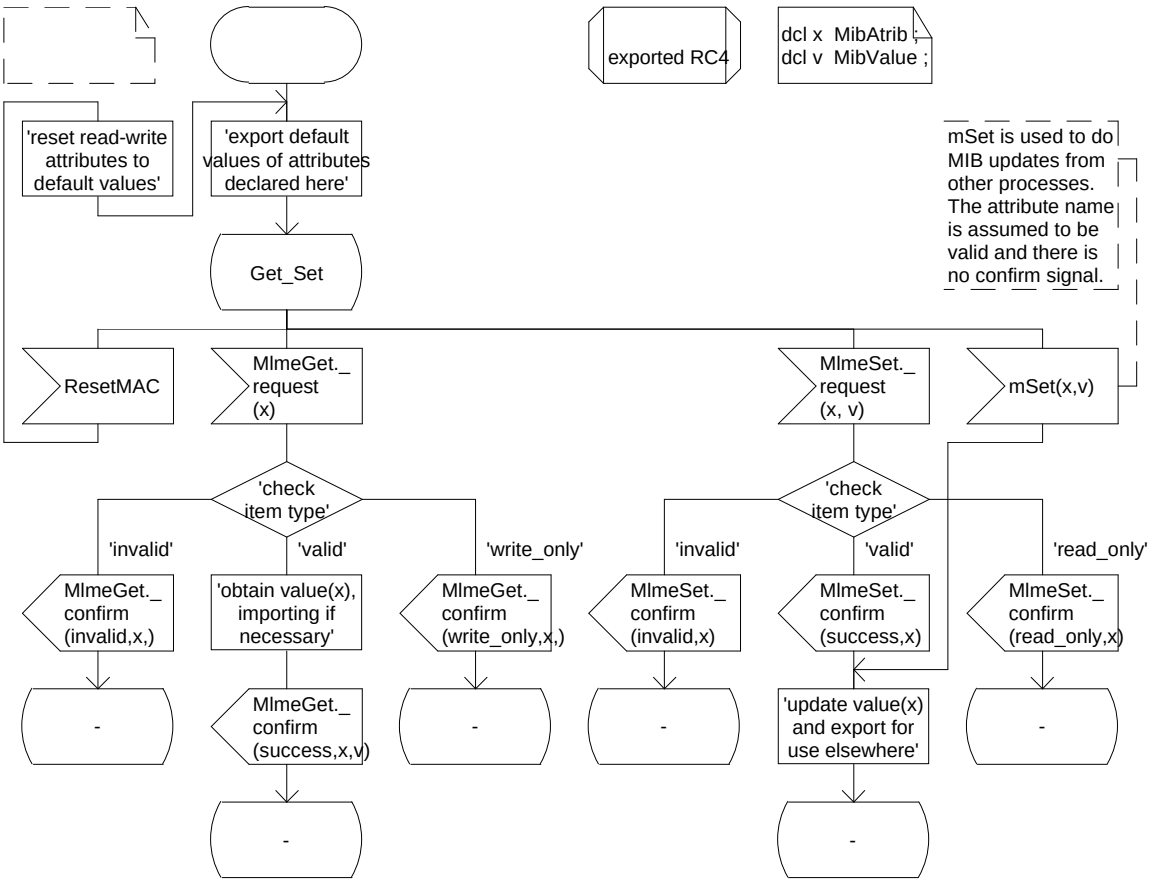
```
signallist
PlmeConfirmSignals=
   PlmeGet.confirm,
   PlmeSet.confirm,
   PlmeReset.confirm ;
```


MAC_SAP
Block MAC_Data_Service
MaUnitdata.indication
MaUnitdataStatus.indication
11_MD_1b(1)
ToLLC
FromLLC
MaUnitdata.request
MSDU_to_LLC
(1,1)
MSDU_from_LLC
(1,1)
MsduIndicate
MsduConfirm
RxMsdu
TxMsdu
MsduRequest
RSDU
TSDU


dcl LLCdata Octetstring ;
dcl period CfPriority ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl serv ServiceClass ;
To_LLC
MsduIndicate (sdu,period)
da:= addr1(sdu)
sa:= if frDs(sdu)=1 then addr3(sdu) else addr2(sdu) fi
serv:= if orderBit(sdu)=1 then strictlyOrdered else reorderable fi
Remove MAC header from beginning of MSDU to obtain the LLC data octet string.
LLCdata:= substr (sdu, sMacHdrLng, length(sdu)-sMacHdrLng)
Reception status always successful because any error would prevent the MsduIndicate from reaching this process.
MaUnitdata._ indication(sa,da,null_rt, LLCdata, successful_rx, period, serv)
-


TxResult reports reasons for failure, including noBss, noAuth, noAssoc, retryLimit, txLifeTimeout; However, the only status code for Tx failure in 6.2.1.3 is 'undeliverable'.
imported mIbss Boolean ;
imported mPcAvail Boolean ;
imported mPsm PwrSave ;
dcl LLCdata Octetstring ;
dcl period CfPriority ;
dcl result TxResult ;
dcl route Routing ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl serv ServiceClass ;
dcl status TxStatus ;
From_LLC
MaUnitdata._ request (sa, da,
route, LLCdata, period, serv)
MsduConfirm (sdu,period, result)
route
else
(null_rt)
status:= non_ NullSource_ Routing
status:= if result= successful
then successful
else undeliverable
fi
serv:= if orderBit(sdu) = 1
then strictlyOrdered
else reorderable
fi
da:= if toDs(sdu) = 1
then addr3(sdu)
else addr1(sdu)
fi
serv
else
(reorderable)
(strictly_ Ordered)
status:= unsupported_ ServiceClass
import (mPsm)
(sta_active)
else
MaUnitdata._ status._ indication
(addr2(sdu), da, status, period, serv)
status:= unavailable_ ServiceClass
-
period
else
(contention)
(contentionFree)
status:= unsupported_ Priority
import (mPcAvail)
(true)
(false)
reject
make_ msdu
Build frame with 24-octet MAC header and LLCdata:
ftype:= data
addr1:= da
addr2:= aMacAddress
(sa parameter not used)
addr3:= mBssId
<other header fields> := 0
type Octetstring length (LLCdata)
else
(0:sMaxMsduLng)
MaUnitdata._ status._ indication
(sa,da,unavail_ablePriority, period,serv)
sdu:= mkFrame (data, da,
import(mBssId), LLCdata)
status:= excessive_ DataLength
make_ msdu
period:= contention
make_ msdu
If BSS has no point coordinator, inform LLC, change priority to contention, and send MSDU.
serv
else
(strictlyOrdered)
sdu:= setOrderBit (sdu, 1)
reject
MaUnitdata._ status._ indication
(sa, da, status, period, serv)
MsduRequest (sdu, period)
-
-

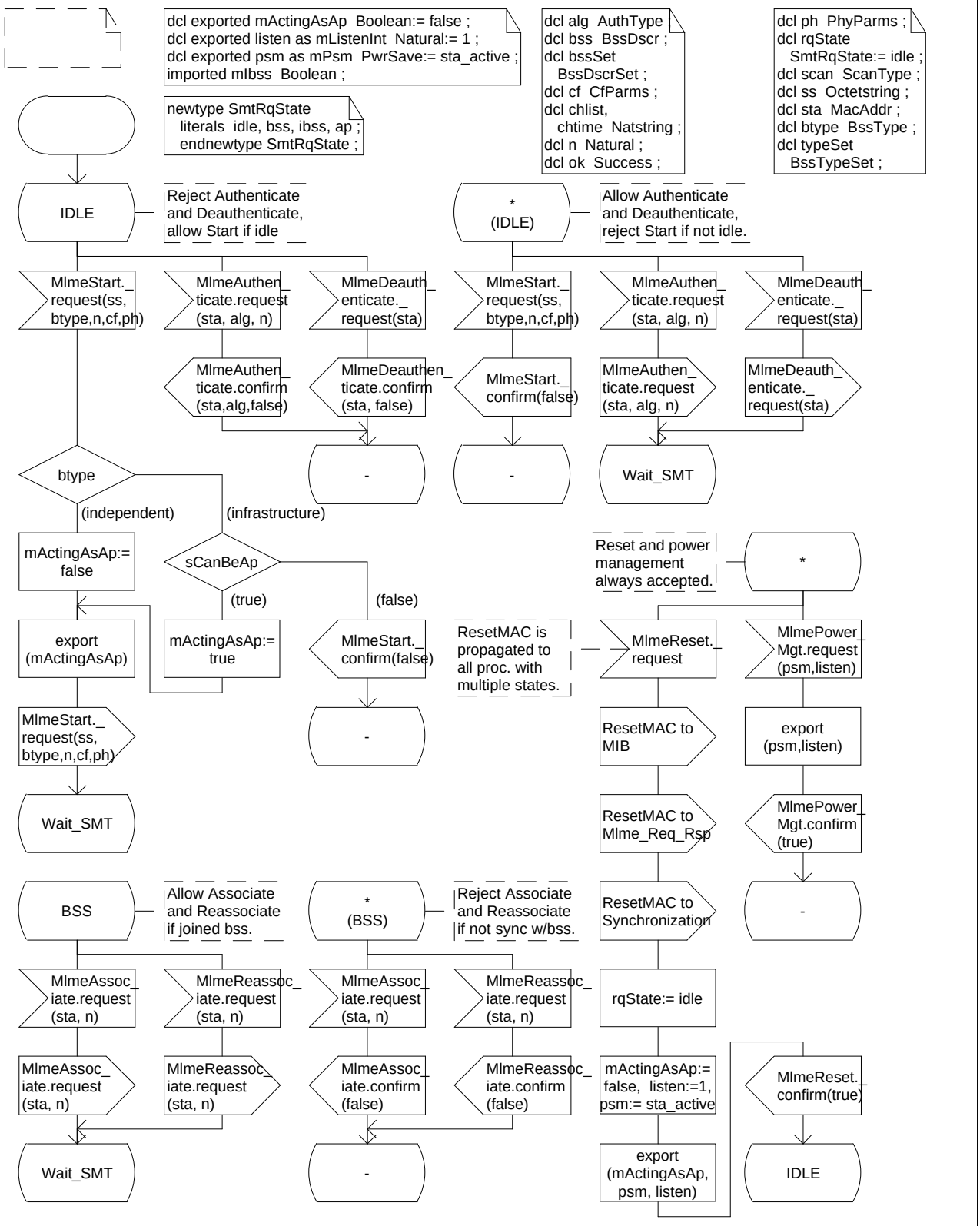

SM_MLME_SAP
Block MAC_Management_Service
MlmeGet.confirm,
MlmeSet.confirm
12_MM_1b(1)
MlmeAssociate.indicate,
MlmeAuthenticate.indicate,
MlmeDeauthenticate.indicate,
MlmeDisassociate.indicate,
MlmeReassociate.indicate
GetSet
ReqConf
MlmeGet.request,
MlmeSet.request
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeJoin.confirm,
MlmePowermgt.confirm,
MlmeReassociate.confirm,
MlmeReset.confirm,
MlmeScan.confirm,
MlmeStart.confirm
Indicate
This block is a summary of MIB access and update. MIB attributes are defined in ASN.1 code.
ResetMAC
MIB(1,1)
mSet
This block handles requests sequentially. Start, join, powermgt, scan, re/dis/associate and deauthenticate must be sequential. Authenticate allows the possibility of multiple requests outstanding. To do this, Mlme_Req_Rsp would need to cache challenge text and match responses to outstanding requests.
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeJoin.request,
MlmePowermgt.request,
MlmeReassociate.request,
MlmeReset.request,
MlmeScan.request,
MlmeStart.request
mR
SMT_Requests
(1,1)
SMT_Indications
(1,1)
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeJoin.confirm,
MlmeReassociate.confirm,
MlmeScan.confirm,
MlmeStart.confirm
MlmeAssociate._
indicate,
MlmeAuthenticate._
indicate,
MlmeDeauthenticate._
indicate,
MlmeDisassociate._
indicate,
MlmeReassociate._
indicate
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeJoin.request,
MlmeReassociate.request,
MlmeScan.request,
MlmeStart.request,
ResetMAC
mgtS
FromMgt
ToMgt
MMGT

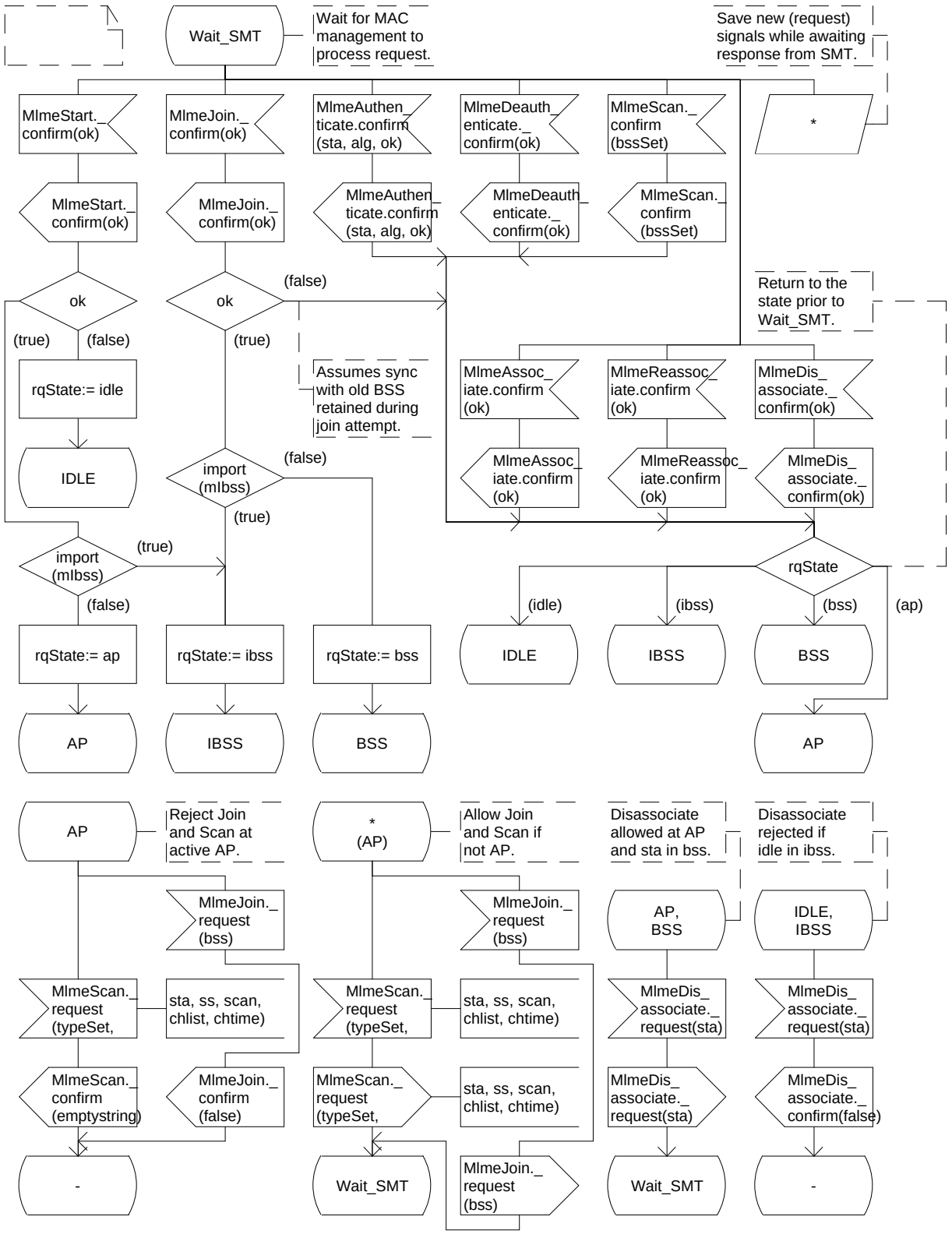


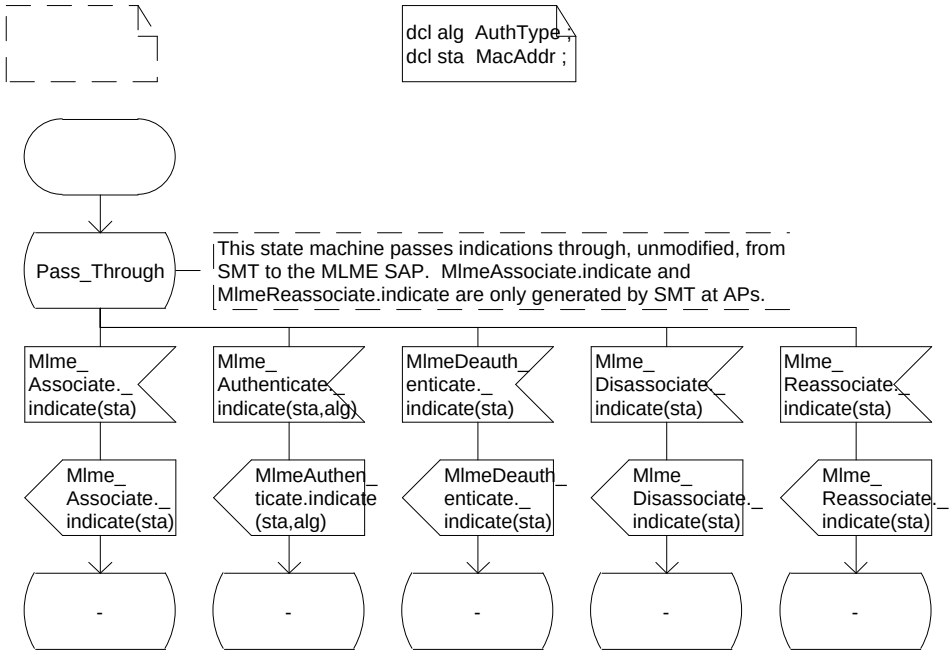

```
/* Import of {Read-Only} MIB attribute
   values exported from other processes */
imported  aAckFailureCount,  aFailedCount,
   aFcsErrorCount,  aFrameDuplicateCount,
   aMulticastReceivedFrameCount,
   aMulticastTransmittedFrameCount,
   aMultipleRetryCount,
   aReceivedFrameCount,  aRetryCount,
   aRtsFailureCount,  aRtsSuccessCount,
   aTransmittedFragmentCount,
   aIcvErrorCount  Counter32 ;
```

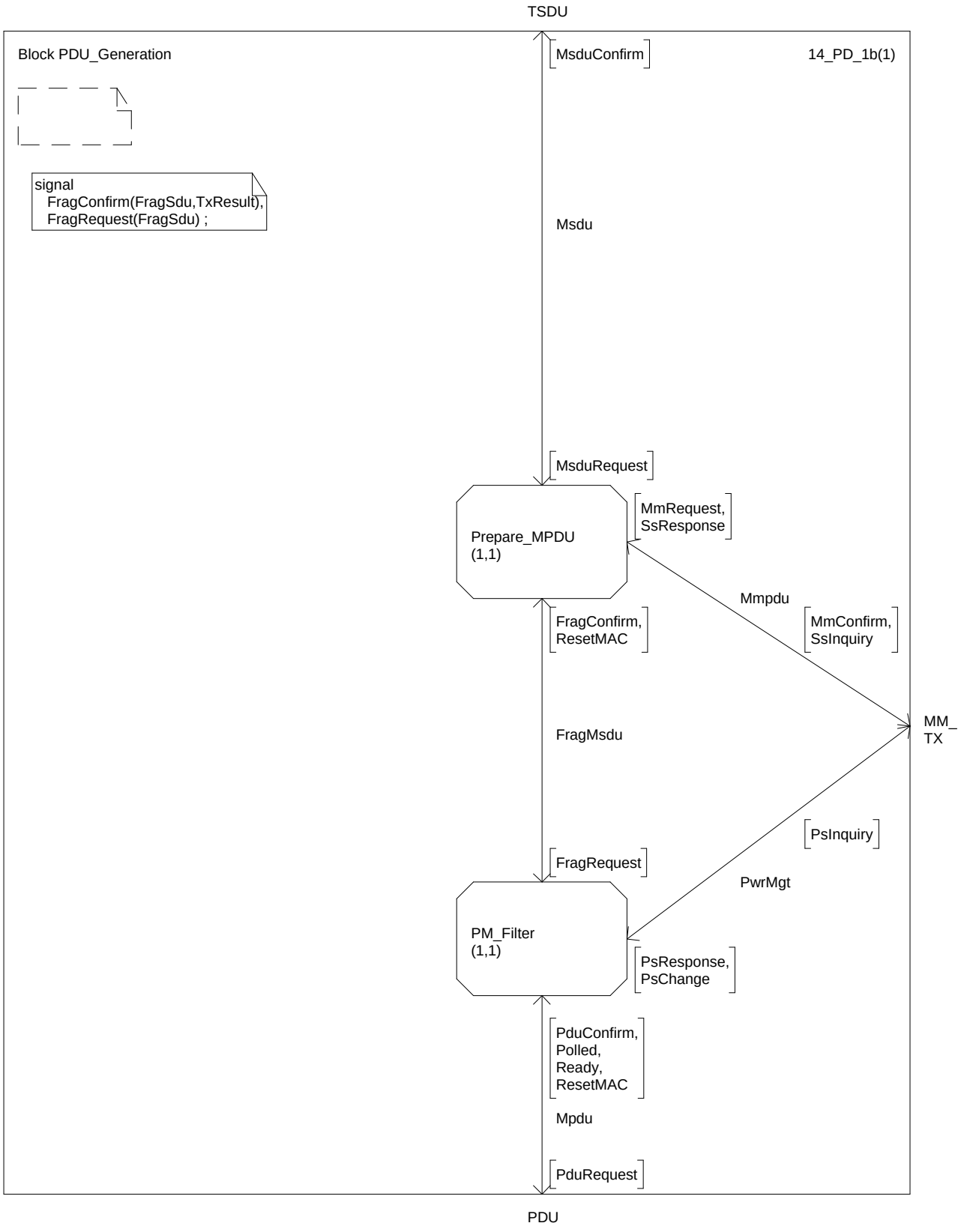
```
/* Declarations of internal MAC variables
   (updated from multiple sources using mSet) */
dcl exported
   mBssId  MacAddr:= nullAddr,
   mIbss  Boolean:= false,
   mSsId  Octetstring:= null ;
```

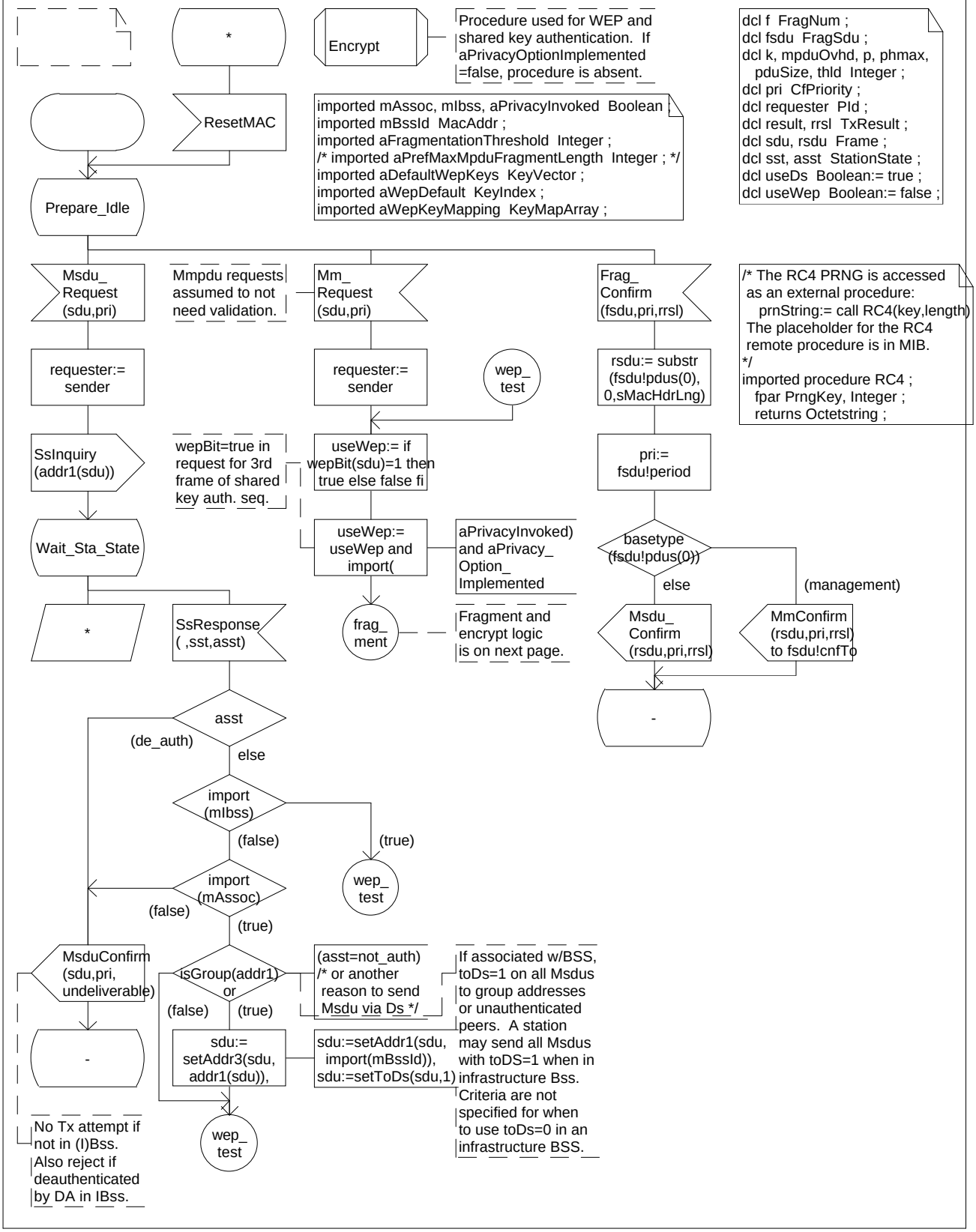
```
/* Declarations of MIB attributes exported from this process */
      /* Read-Write attributes */
dcl exported  aAuthenticationType  Integer:= 1,
   aExcludeUnencrypted  Boolean:= false,
   aFragmentationThreshold  Integer:= 2346,
   aGroupAddresses  MacAddrSet:= empty,
   aLongRetryLimit  Integer:= 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,
   aCurrentIndex  Integer ;
      /* Read-Only attributes */
dcl exported  aAuthenticationAlgorithms
     AuthAlgSet:= (open_system or shared_key) ;
      /* Write-Only attributes */
dcl exported  aDefaultWepKeys  KeyVector ;
dcl exported  aWepKeyMapping  KeyMapArray:= (. nullAddr, false, null .) ;
```

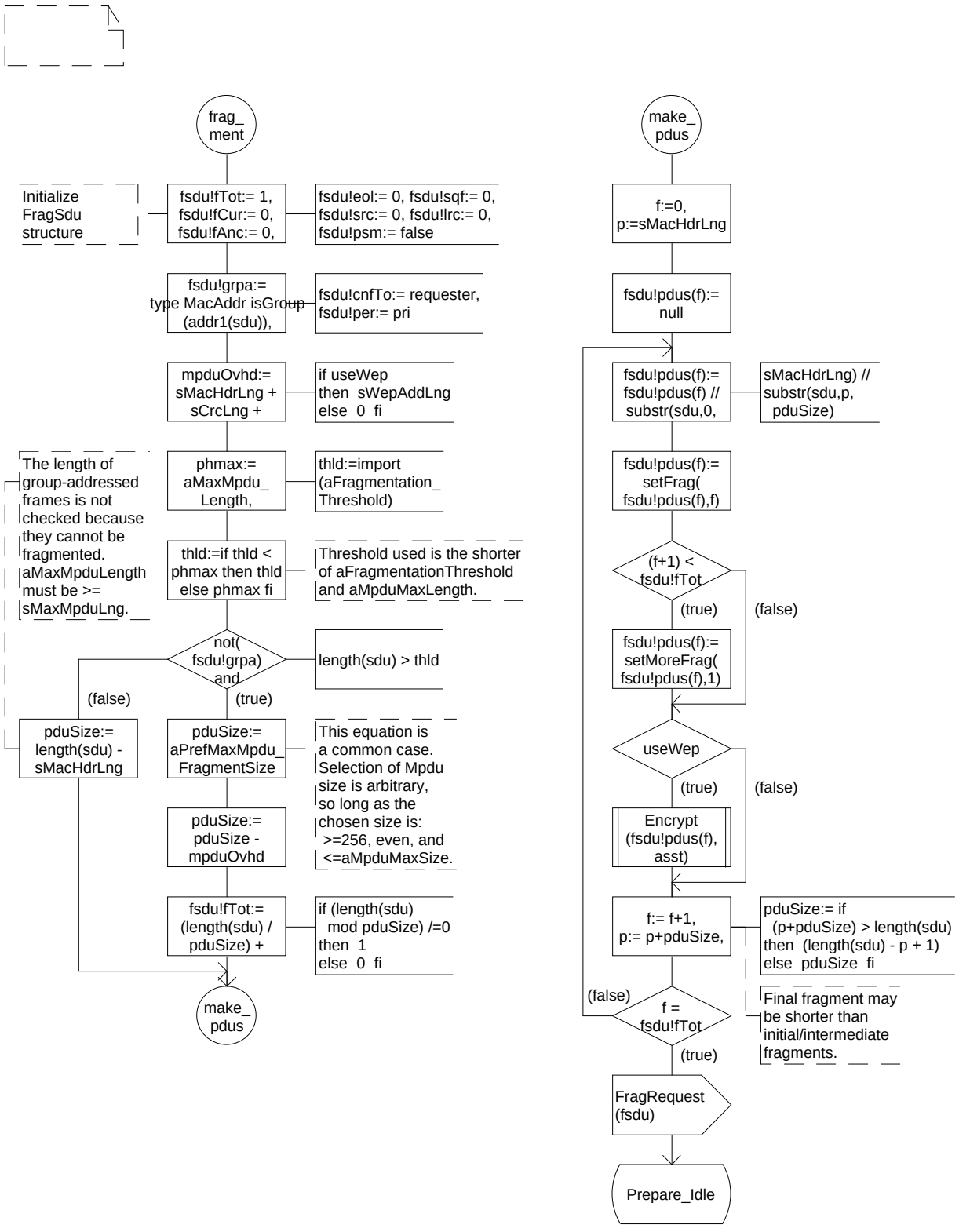

dcl exported mActingAsAp Boolean:= false ;
dcl exported listen as mListenInt Natural:= 1 ;
dcl exported psm as mPsm PwrSave:= sta_active ;
imported mIbss Boolean ;
dcl alg AuthType ;
dcl bss BssDscr ;
dcl bssSet
BssDscrSet ;
dcl cf CfParms ;
dcl chlist,
chtime Natstring ;
dcl n Natural ;
dcl ok Success ;
dcl ph PhyParms ;
dcl rqState
SmtRqState:= idle ;
dcl scan ScanType ;
dcl ss Octetstring ;
dcl sta MacAddr ;
dcl btype BssType ;
dcl typeSet
BssTypeSet ;
newtype SmtRqState
literals idle, bss, ibss, ap ;
endnewtype SmtRqState ;
IDLE
Reject Authenticate and Deauthenticate, allow Start if idle
*
(IDLE)
Allow Authenticate and Deauthenticate, reject Start if not idle.
MlmeStart._ request(ss, btype,n,cf,ph)
MlmeAuthen_ ticate.request (sta, alg, n)
MlmeDeauth_ enticate._ request(sta)
MlmeStart._ request(ss, btype,n,cf,ph)
MlmeAuthen_ ticate.request (sta, alg, n)
MlmeDeauth_ enticate._ request(sta)
MlmeAuthen_ ticate.confirm (sta,alg,false)
MlmeDeauthen_ ticate.confirm (sta, false)
MlmeStart._ confirm(false)
MlmeAuthen_ ticate.request (sta, alg, n)
MlmeDeauth_ enticate._ request(sta)
-
-
Wait_SMT
btype
(independent)
(infrastructure)
mActingAsAp:= false
sCanBeAp
(true)
(false)
Reset and power management always accepted.
*
export (mActingAsAp)
mActingAsAp:= true
MlmeStart._ confirm(false)
ResetMAC is propagated to all proc. with multiple states.
MlmeReset._ request
MlmePower_ Mgt.request (psm,listen)
MlmeStart._ request(ss, btype,n,cf,ph)
-
ResetMAC to MIB
export (psm,listen)
Wait_SMT
ResetMAC to Mlme_Req_Rsp
MlmePower_ Mgt.confirm (true)
BSS
Allow Associate and Reassociate if joined bss.
*
(BSS)
Reject Associate and Reassociate if not sync w/bss.
ResetMAC to Synchronization
-
MlmeAssoc_ iate.request (sta, n)
MlmeReassoc_ iate.request (sta, n)
MlmeAssoc_ iate.request (sta, n)
MlmeReassoc_ iate.request (sta, n)
rqState:= idle
MlmeAssoc_ iate.request (sta, n)
MlmeReassoc_ iate.request (sta, n)
MlmeAssoc_ iate.confirm (false)
MlmeReassoc_ iate.confirm (false)
mActingAsAp:= false, listen:=1, psm:= sta_active
MlmeReset._ confirm(true)
Wait_SMT
-
export (mActingAsAp, psm, listen)
IDLE

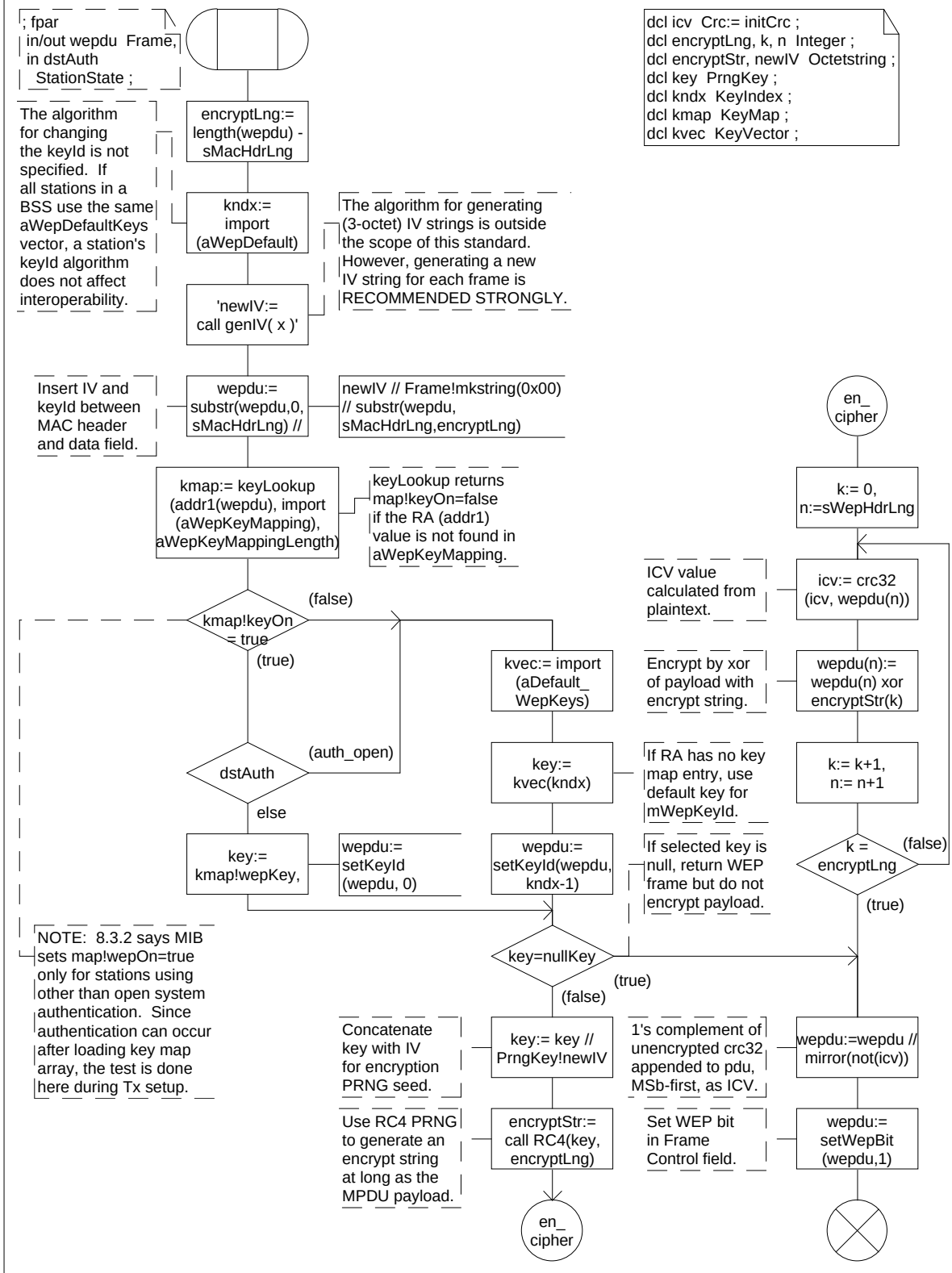

Wait_SMT
Wait for MAC management to process request.
Save new (request) signals while awaiting response from SMT.
MlmeStart._ confirm(ok)
MlmeJoin._ confirm(ok)
MlmeAuthen_ ticate.confirm (sta, alg, ok)
MlmeDeauth_ enticate._ confirm(ok)
MlmeScan._ confirm (bssSet)
*
MlmeStart._ confirm(ok)
MlmeJoin._ confirm(ok)
MlmeAuthen_ ticate.confirm (sta, alg, ok)
MlmeDeauth_ enticate._ confirm(ok)
MlmeScan._ confirm (bssSet)
ok
ok
(false)
(true)
(false)
(true)
Return to the state prior to Wait_SMT.
rqState:= idle
Assumes sync with old BSS retained during join attempt.
MlmeAssoc_ iate.confirm (ok)
MlmeReassoc_ iate.confirm (ok)
MlmeDis_ associate._ confirm(ok)
IDLE
import (mIbss)
(false)
MlmeAssoc_ iate.confirm (ok)
MlmeReassoc_ iate.confirm (ok)
MlmeDis_ associate._ confirm(ok)
(true)
import (mIbss)
(true)
rqState
(false)
(idle)
(ibss)
(bss)
(ap)
rqState:= ap
rqState:= ibss
rqState:= bss
IDLE
IBSS
BSS
AP
IBSS
BSS
AP
AP
Reject Join and Scan at active AP.
*
(AP)
Allow Join and Scan if not AP.
Disassociate allowed at AP and sta in bss.
Disassociate rejected if idle in ibss.
MlmeJoin._ request (bss)
MlmeJoin._ request (bss)
AP, BSS
IDLE, IBSS
MlmeScan._ request (typeSet,
sta, ss, scan, chlist, chtime)
MlmeScan._ request (typeSet,
sta, ss, scan, chlist, chtime)
MlmeDis_ associate._ request(sta)
MlmeDis_ associate._ request(sta)
MlmeScan._ confirm (emptystring)
MlmeJoin._ confirm (false)
MlmeScan._ request (typeSet,
sta, ss, scan, chlist, chtime)
MlmeDis_ associate._ request(sta)
MlmeDis_ associate._ confirm(false)
-
Wait_SMT
MlmeJoin._ request (bss)
Wait_SMT
-

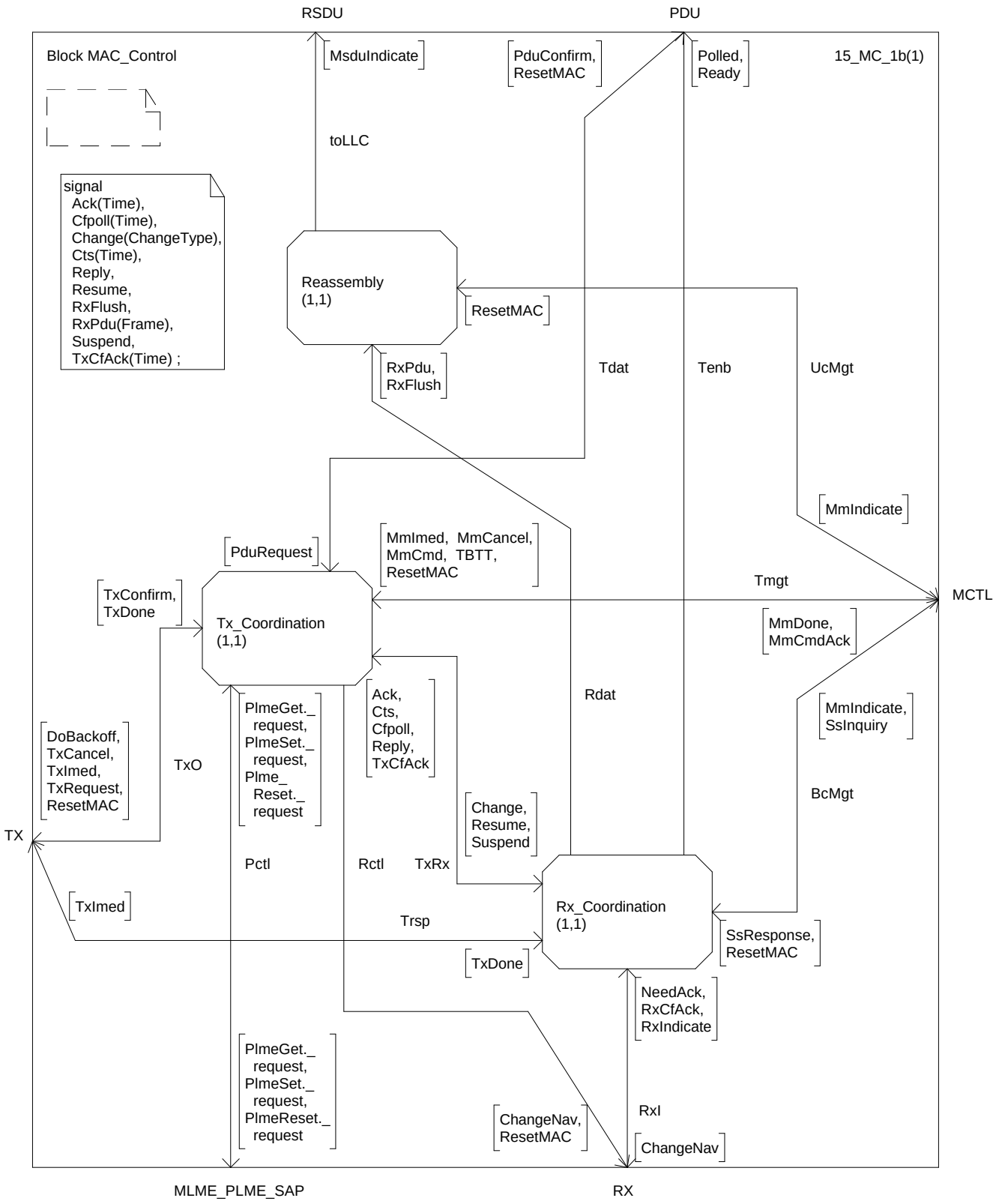

dcl alg AuthType ;
dcl sta MacAddr ;
Pass_Through
This state machine passes indications through, unmodified, from SMT to the MLME SAP. MlmeAssociate.indicate and MlmeReassociate.indicate are only generated by SMT at APs.
Mlme_ Associate._ indicate(sta)
Mlme_ Authenticate._ indicate(sta,alg)
MlmeDeauth_ enticate._ indicate(sta)
Mlme_ Disassociate._ indicate(sta)
Mlme_ Reassociate._ indicate(sta)
Mlme_ Associate._ indicate(sta)
MlmeAuthen_ ticate.indicate (sta,alg)
MlmeDeauth_ enticate._ indicate(sta)
Mlme_ Disassociate._ indicate(sta)
Mlme_ Reassociate._ indicate(sta)
-
-
-
-
-

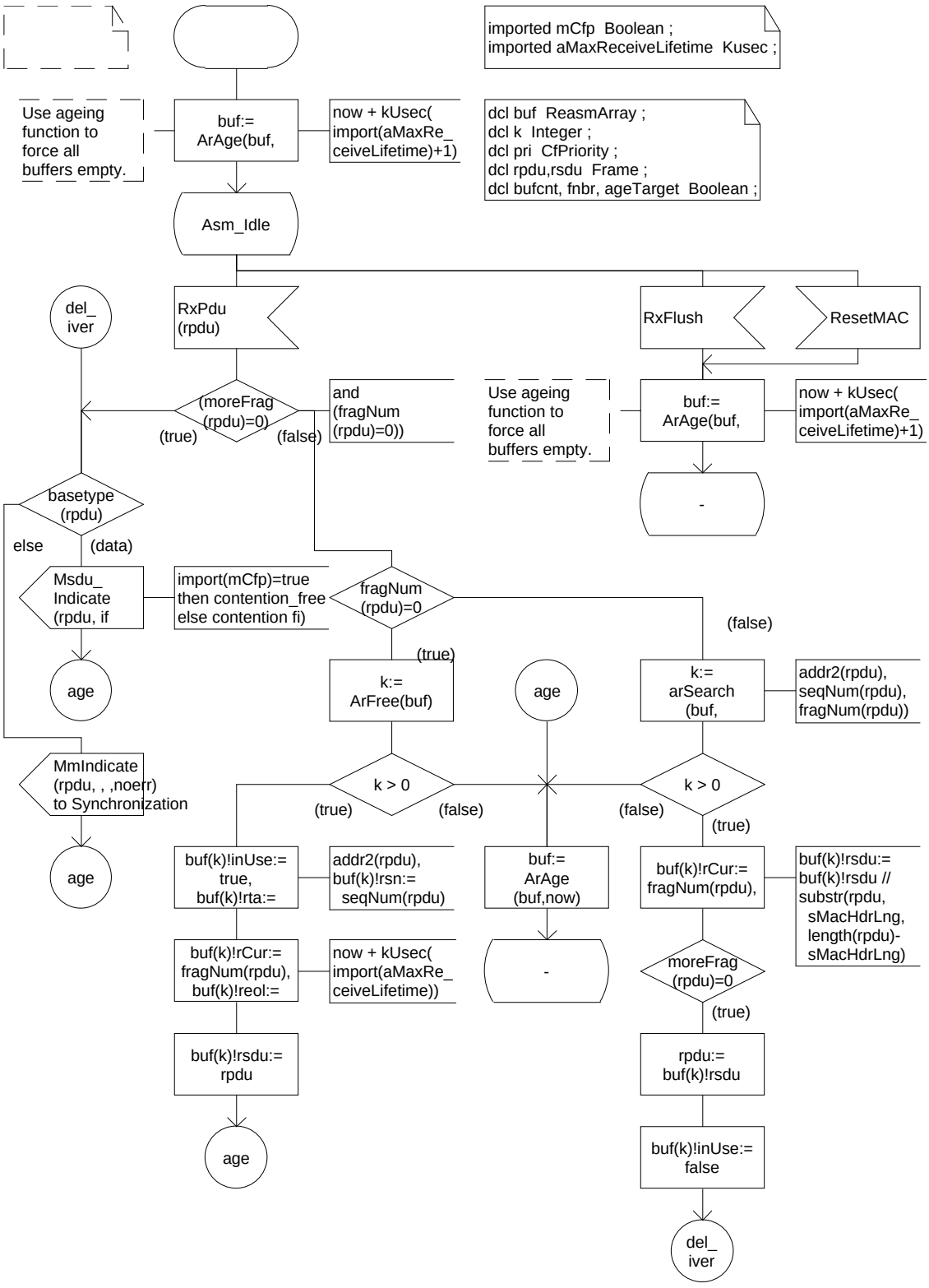

TSDU
Block PDU_Generation
14_PD_1b(1)
MsduConfirm
signal
FragConfirm(FragSdu,TxResult),
FragRequest(FragSdu) ;
Msdu
MsduRequest
Prepare_MPDU
(1,1)
MmRequest,
SsResponse
Mmpdu
FragConfirm,
ResetMAC
MmConfirm,
SsInquiry
MM_
TX
FragMsdu
PsInquiry
FragRequest
PwrMgt
PM_Filter
(1,1)
PsResponse,
PsChange
PduConfirm,
Polled,
Ready,
ResetMAC
Mpdu
PduRequest
PDU

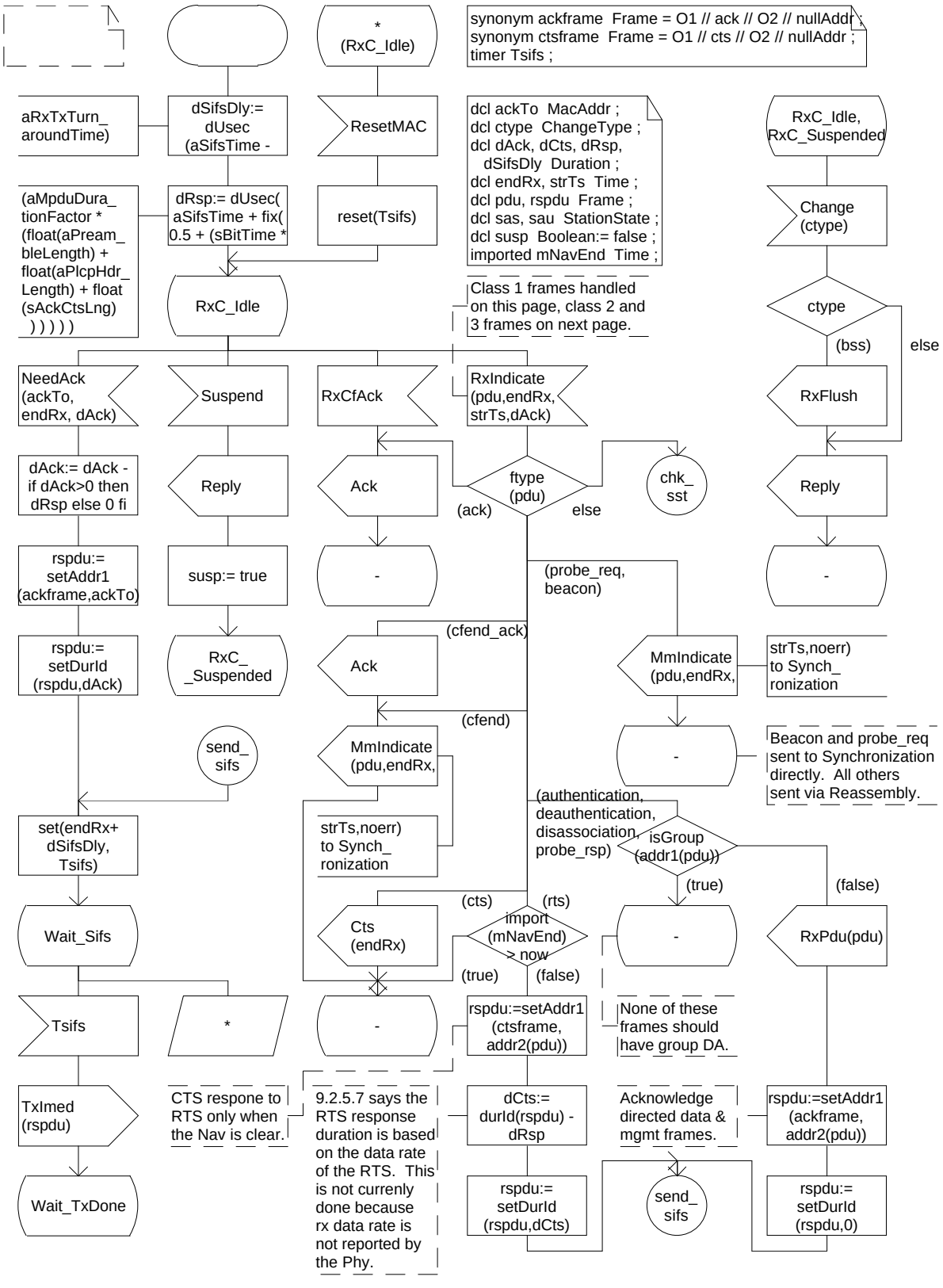

*
Encrypt
Procedure used for WEP and shared key authentication. If aPrivacyOptionImplemented =false, procedure is absent.
dcl f FragNum ;
dcl fsdu FragSdu ;
dcl k, mpduOvhd, p, phmax, pduSize, thld Integer ;
dcl pri CfPriority ;
dcl requester PId ;
dcl result, rrsl TxResult ;
dcl sdu, rsdu Frame ;
dcl sst, asst StationState ;
dcl useDs Boolean:= true ;
dcl useWep Boolean:= false ;
ResetMAC
imported mAssoc, mIbss, aPrivacyInvoked Boolean ;
imported mBssId MacAddr ;
imported aFragmentationThreshold Integer ;
/* imported aPrefMaxMpduFragmentLength Integer ; */
imported aDefaultWepKeys KeyVector ;
imported aWepDefault KeyIndex ;
imported aWepKeyMapping KeyMapArray ;
Prepare_Idle
Msdu_ Request (sdu,pri)
Mmpdu requests assumed to not need validation.
Mm_ Request (sdu,pri)
Frag_ Confirm (fsdu,pri,rrsl)
/* 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 ;
requester:= sender
requester:= sender
wep_ test
rsdu:= substr (fsdu!pdus(0), 0,sMacHdrLng)
SsInquiry (addr1(sdu))
wepBit=true in request for 3rd frame of shared key auth. seq.
useWep:= if wepBit(sdu)=1 then true else false fi
pri:= fsdu!period
Wait_Sta_State
useWep:= useWep and import(
aPrivacyInvoked) and aPrivacy_ Option_ Implemented
basetype (fsdu!pdus(0))
else
(management)
*
SsResponse (,sst,asst)
frag_ ment
Fragment and encrypt logic is on next page.
Msdu_ Confirm (rsdu,pri,rrsl)
MmConfirm (rsdu,pri,rrsl) to fsdu!cnfTo
-
asst
(de_auth)
else
import (mIbss)
(false)
(true)
wep_ test
import (mAssoc)
(false)
(true)
MsduConfirm (sdu,pri, undeliverable)
isGroup(addr1) or
(asst=not_auth) /* or another reason to send Msdu via Ds */
If associated w/BSS, toDs=1 on all Msdus to group addresses or unauthenticated peers. A station may send all Msdus with toDS=1 when in infrastructure Bss. Criteria are not specified for when to use toDs=0 in an infrastructure BSS.
(false)
(true)
-
sdu:= setAddr3(sdu, addr1(sdu)),
sdu:=setAddr1(sdu, import(mBssId)), sdu:=setToDs(sdu,1)
No Tx attempt if not in (I)Bss. Also reject if deauthenticated by DA in IBss.
wep_ test

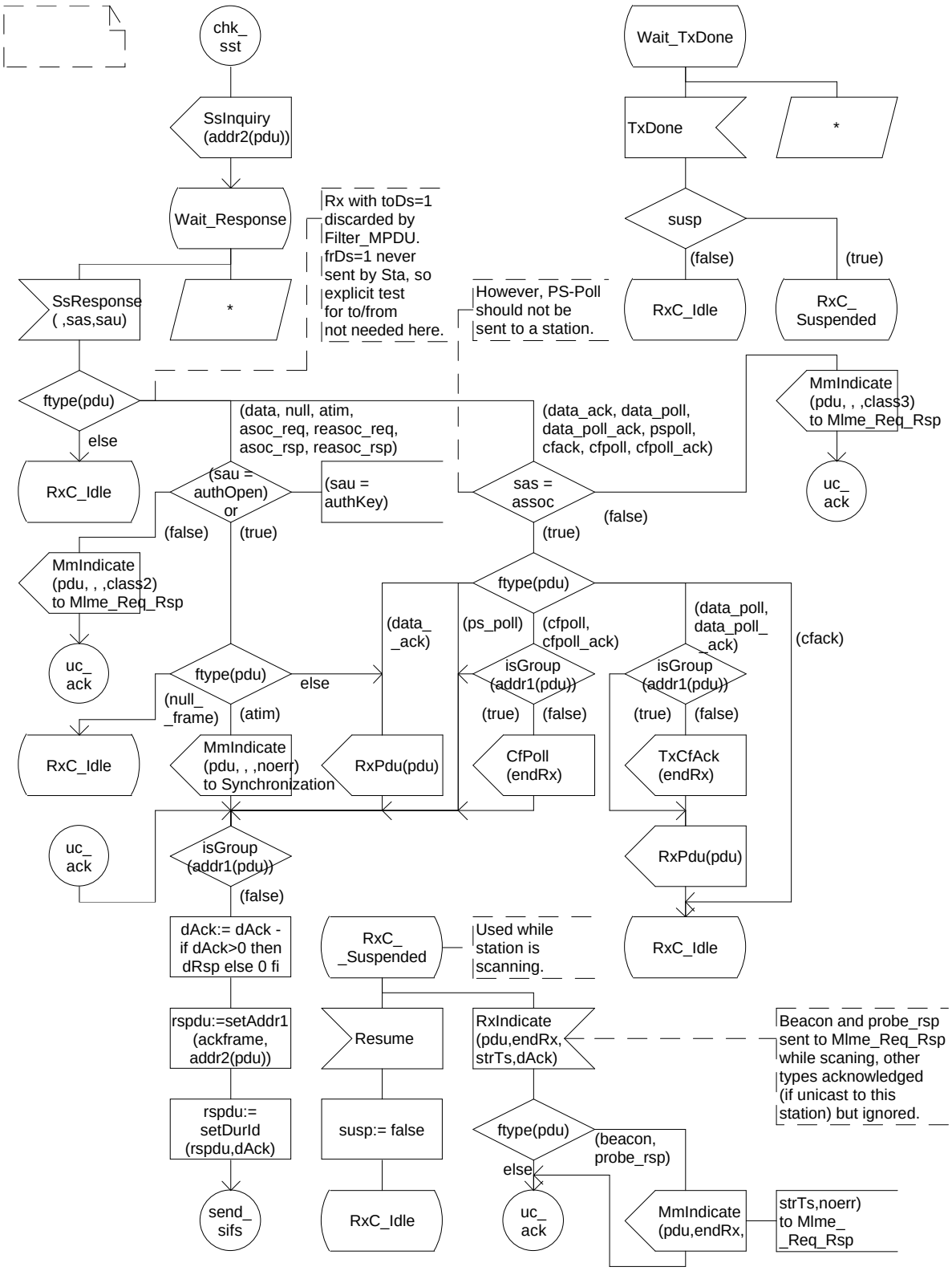

frag_ment
Initialize FragSdu structure
fsdu!fTot:= 1, fsdu!fCur:= 0, fsdu!fAnc:= 0,
fsdu!eol:= 0, fsdu!sqf:= 0, fsdu!src:= 0, fsdu!lrc:= 0, fsdu!psm:= false
fsdu!grpa:= type MacAddr isGroup (addr1(sdu)),
fsdu!cnfTo:= requester, fsdu!per:= pri
mpduOvhd:= sMacHdrLng + sCrcLng +
if useWep then sWepAddLng else 0 fi
The length of group-addressed frames is not checked because they cannot be fragmented. aMaxMpduLength must be >= sMaxMpduLng.
phmax:= aMaxMpdu_ Length,
thld:=import (aFragmentation_ Threshold)
thld:=if thld < phmax then thld else phmax fi
Threshold used is the shorter of aFragmentationThreshold and aMpduMaxLength.
not(fsdu!grpa) and
length(sdu) > thld
(false)
(true)
pduSize:= length(sdu) - sMacHdrLng
pduSize:= aPrefMaxMpdu_ FragmentSize
This equation is a common case. Selection of Mpdu size is arbitrary, so long as the chosen size is: >=256, even, and <=aMpduMaxSize.
pduSize:= pduSize - mpduOvhd
fsdu!fTot:= (length(sdu) / pduSize) +
if (length(sdu) mod pduSize) /=0 then 1 else 0 fi
make_ pdus
make_ pdus
f:=0, p:=sMacHdrLng
fsdu!pdus(f):= null
fsdu!pdus(f):= fsdu!pdus(f) // substr(sdu,0,
sMacHdrLng) // substr(sdu,p, pduSize)
fsdu!pdus(f):= setFrag(fsdu!pdus(f),f)
(f+1) < fsdu!fTot
(true)
(false)
fsdu!pdus(f):= setMoreFrag(fsdu!pdus(f),1)
useWep
(true)
(false)
Encrypt (fsdu!pdus(f), asst)
f:= f+1, p:= p+pduSize,
pduSize:= if (p+pduSize) > length(sdu) then (length(sdu) - p + 1) else pduSize fi
(false)
f = fsdu!fTot
Final fragment may be shorter than initial/intermediate fragments.
(true)
FragRequest (fsdu)
Prepare_Idle

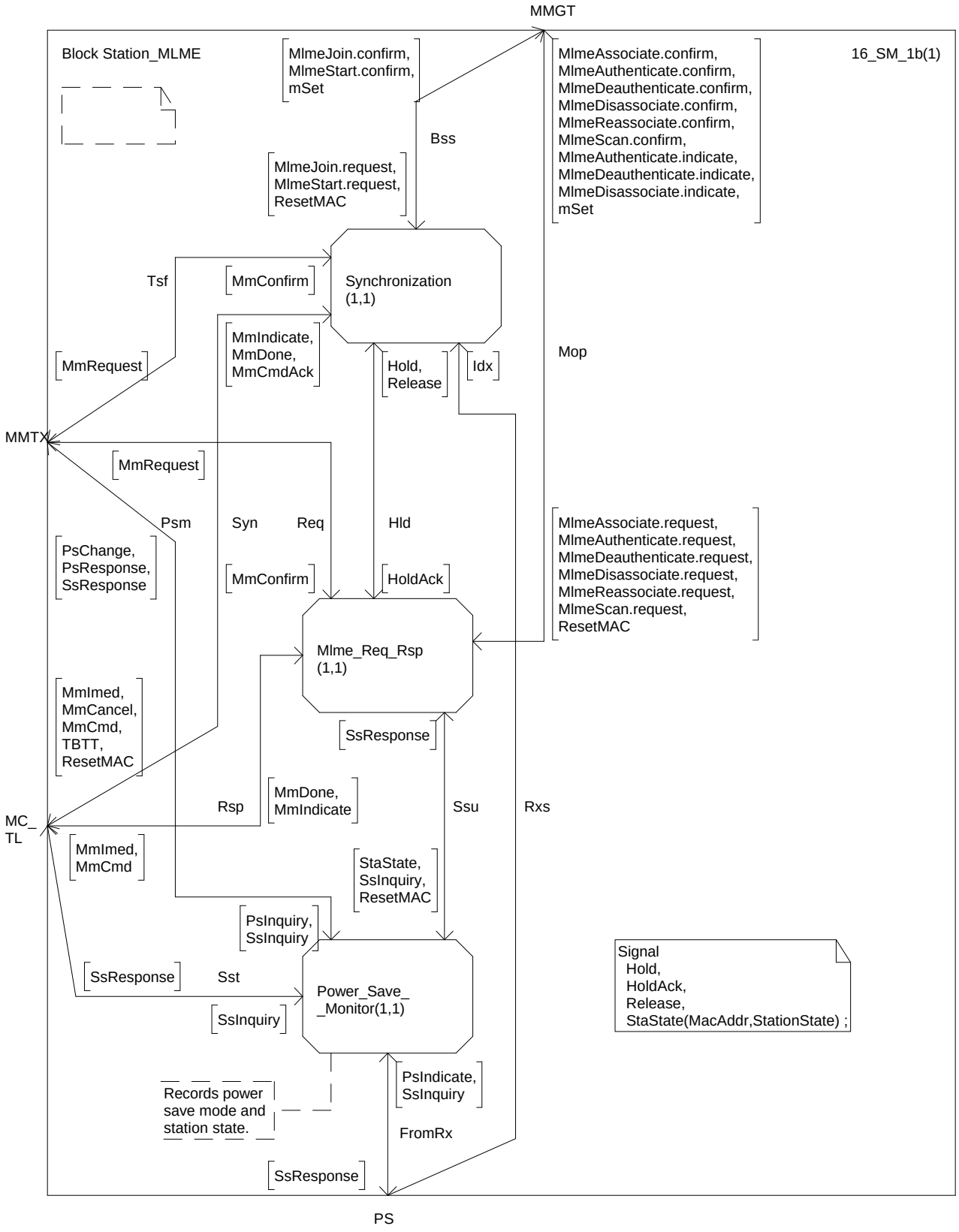

; fpar
in/out wepdu Frame,
in dstAuth
StationState ;
dcl icv Crc:= initCrc ;
dcl encryptLng, k, n Integer ;
dcl encryptStr, newIV Octetstring ;
dcl key PrngKey ;
dcl kndx KeyIndex ;
dcl kmap KeyMap ;
dcl kvec KeyVector ;
encryptLng:= length(wepdu) - sMacHdrLng
The algorithm for changing the keyId is not specified. If all stations in a BSS use the same aWepDefaultKeys vector, a station's keyId algorithm does not affect interoperability.
kndx:= import (aWepDefault)
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.
'newIV:= call genIV(x)'
Insert IV and keyId between MAC header and data field.
wepdu:= substr(wepdu,0, sMacHdrLng) //
newIV // Frame!mkstring(0x00) // substr(wepdu, sMacHdrLng,encryptLng)
kmap:= keyLookup (addr1(wepdu), import (aWepKeyMapping), aWepKeyMappingLength)
keyLookup returns map!keyOn=false if the RA (addr1) value is not found in aWepKeyMapping.
kmap!keyOn = true
(false)
(true)
dstAuth
(auth_open)
else
key:= kmap!wepKey,
wepdu:= setKeyId (wepdu, 0)
NOTE: 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.
kvec:= import (aDefault_ WepKeys)
key:= kvec(kndx)
If RA has no key map entry, use default key for mWepKeyId.
wepdu:= setKeyId(wepdu, kndx-1)
If selected key is null, return WEP frame but do not encrypt payload.
key=nullKey
(true)
(false)
Concatenate key with IV for encryption PRNG seed.
key:= key // PrngKey!newIV
Use RC4 PRNG to generate an encrypt string at long as the MPDU payload.
encryptStr:= call RC4(key, encryptLng)
en_ cipher
en_ cipher
k:= 0, n:=sWepHdrLng
ICV value calculated from plaintext.
icv:= crc32 (icv, wepdu(n))
Encrypt by xor of payload with encrypt string.
wepdu(n):= wepdu(n) xor encryptStr(k)
k:= k+1, n:= n+1
k = encryptLng
(false)
(true)
1's complement of unencrypted crc32 appended to pdu, MSb-first, as ICV.
wepdu:=wepdu // mirror(not(icv))
Set WEP bit in Frame Control field.
wepdu:= setWepBit (wepdu,1)

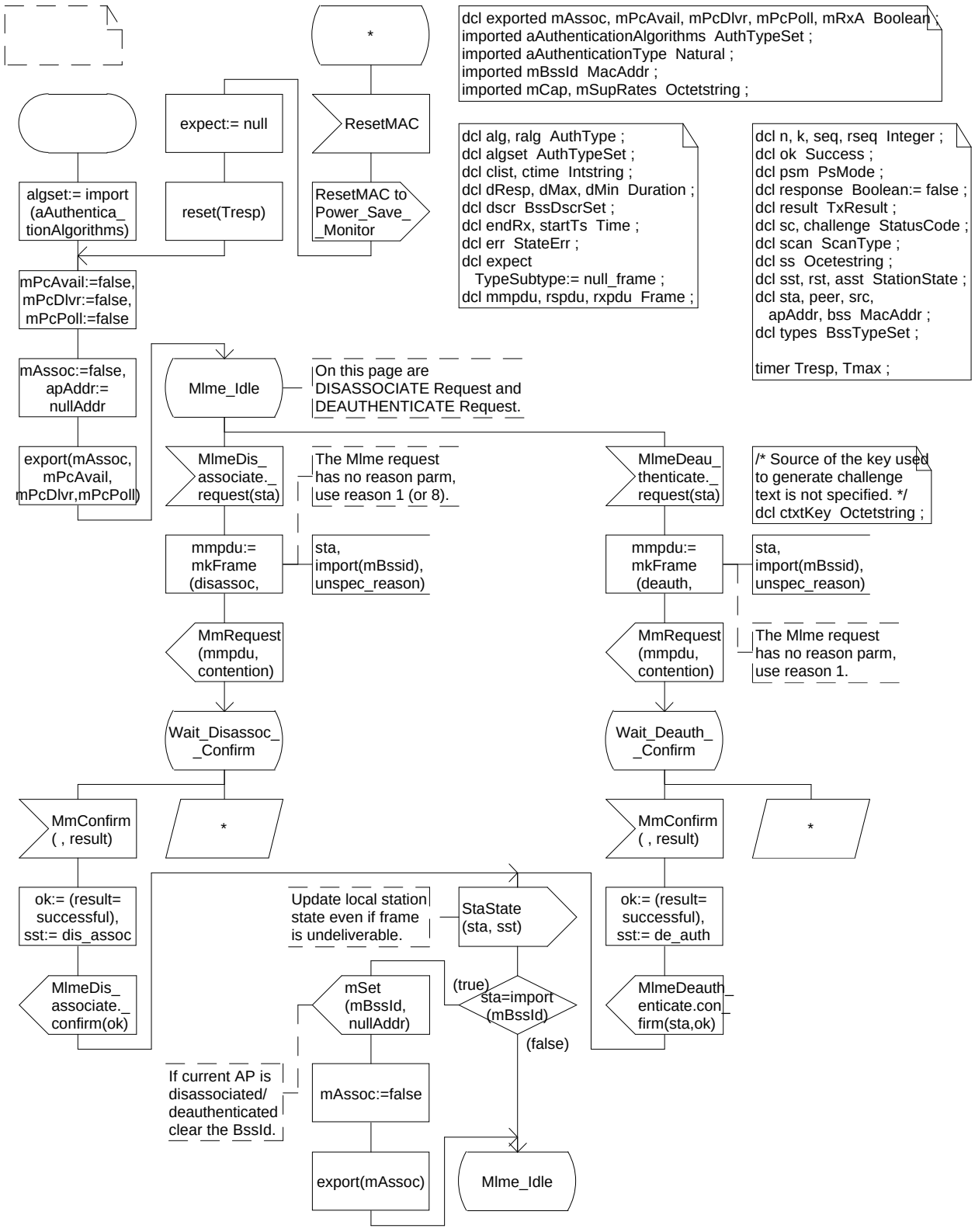

RSDU
PDU
Block MAC_Control
MsduIndicate
PduConfirm, ResetMAC
Polled, Ready
15_MC_1b(1)
toLLC
signal
Ack(Time),
Cfpoll(Time),
Change(ChangeType),
Cts(Time),
Reply,
Resume,
RxFlush,
RxPdu(Frame),
Suspend,
TxCfAck(Time) ;
Reassembly (1,1)
ResetMAC
RxPdu, RxFlush
Tdat
Tenb
UcMgt
MmIndicate
PduRequest
MmImed, MmCancel, MmCmd, TBTT, ResetMAC
Tmgt
MCTL
TxConfirm, TxDone
Tx_Coordination (1,1)
MmDone, MmCmdAck
Rdat
MmIndicate, SsInquiry
Ack, Cts, Cfpoll, Reply, TxCfAck
DoBackoff, TxCancel, TxImed, TxRequest, ResetMAC
TxO
PlmeGet._request, PlmeSet._request, Plme_Reset._request
BcMgt
Change, Resume, Suspend
TX
Pctl
Rctl
TxRx
TxImed
Trsp
Rx_Coordination (1,1)
SsResponse, ResetMAC
TxDone
NeedAck, RxCfAck, RxIndicate
PlmeGet._request, PlmeSet._request, PlmeReset._request
ChangeNav, ResetMAC
RxI
ChangeNav
MLME_PLME_SAP
RX

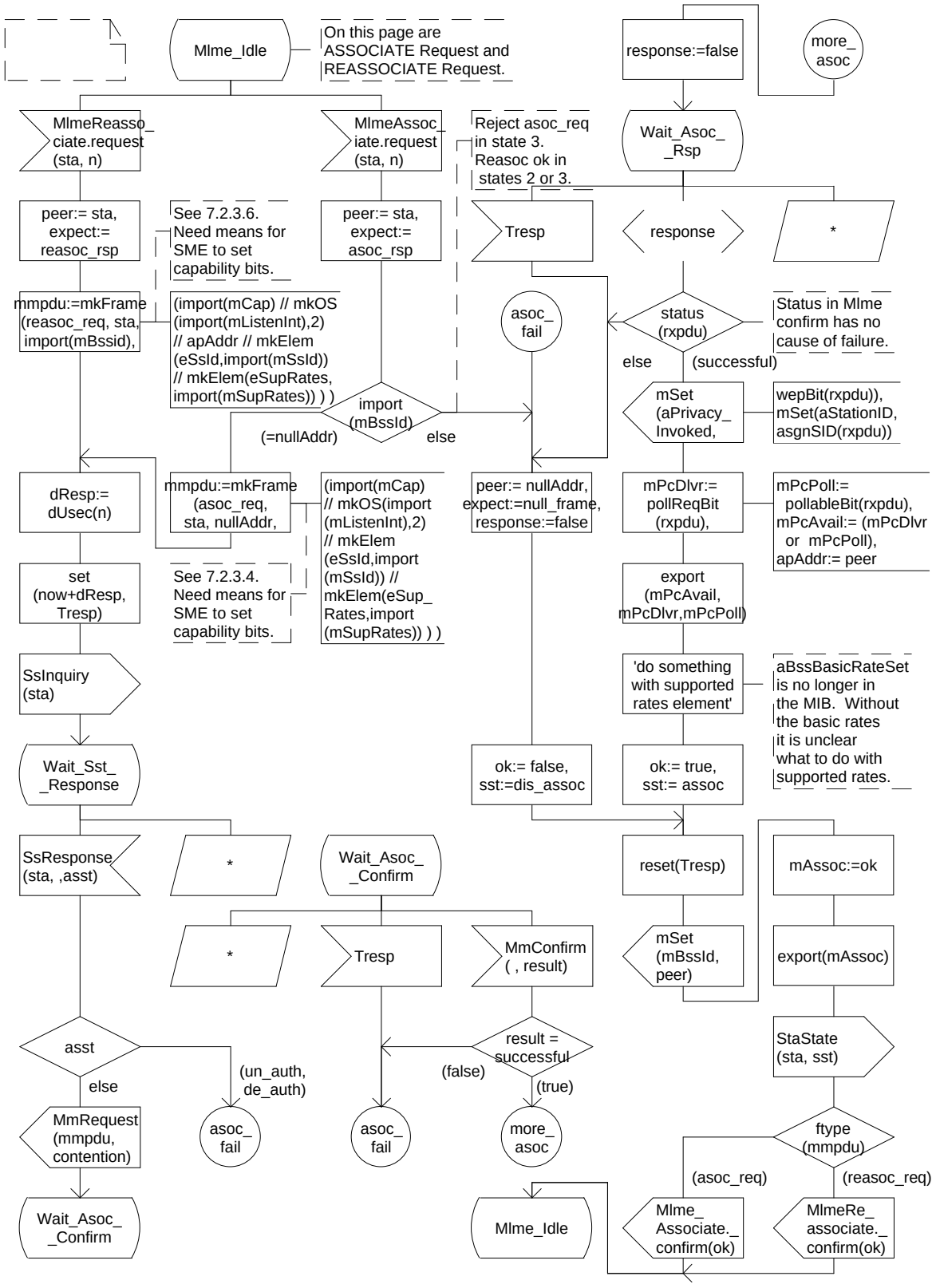

imported mCfp Boolean ;
imported aMaxReceiveLifetime Kusec ;
Use ageing function to force all buffers empty.
buf:= ArAge(buf,
now + kUsec(import(aMaxRe_ ceiveLifetime)+1)
dcl buf ReasmArray ;
dcl k Integer ;
dcl pri CfPriority ;
dcl rpdu,rsdu Frame ;
dcl bufcnt, fnbr, ageTarget Boolean ;
Asm_Idle
del_ iver
RxPdu (rpdu)
RxFlush
ResetMAC
(moreFrag (rpdu)=0)
(true)
(false)
and (fragNum (rpdu)=0))
Use ageing function to force all buffers empty.
buf:= ArAge(buf,
now + kUsec(import(aMaxRe_ ceiveLifetime)+1)
-
basetype (rpdu)
else
(data)
Msdu_ Indicate (rpdu, if
import(mCfp)=true then contention_free else contention fi)
fragNum (rpdu)=0
(true)
(false)
age
k:= ArFree(buf)
age
k:= arSearch (buf,
addr2(rpdu), seqNum(rpdu), fragNum(rpdu))
MmIndicate (rpdu, , ,noerr) to Synchronization
k > 0
(true)
(false)
k > 0
(false)
(true)
age
buf(k)!inUse:= true, buf(k)!rta:=
addr2(rpdu), buf(k)!rsn:= seqNum(rpdu)
buf:= ArAge (buf,now)
buf(k)!rCur:= fragNum(rpdu),
buf(k)!rsdu:= buf(k)!rsdu // substr(rpdu, sMacHdrLng, length(rpdu)- sMacHdrLng)
buf(k)!rCur:= fragNum(rpdu), buf(k)!reol:=
now + kUsec(import(aMaxRe_ ceiveLifetime))
-
moreFrag (rpdu)=0
(true)
buf(k)!rsdu:= rpdu
rpdu:= buf(k)!rsdu
age
buf(k)!inUse:= false
del_ iver

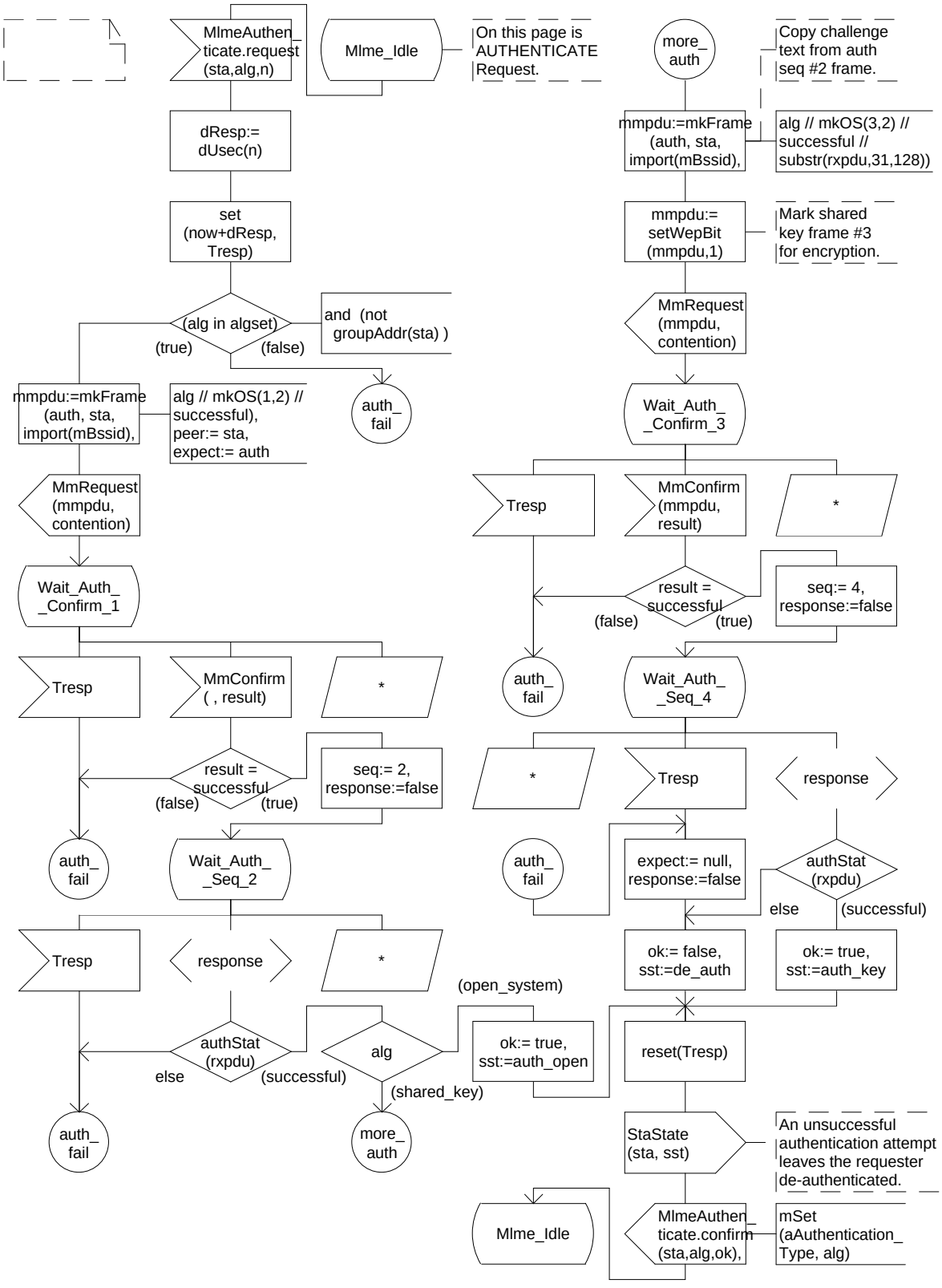

synonym ackframe Frame = O1 // ack // O2 // nullAddr ;
synonym ctsframe Frame = O1 // cts // O2 // nullAddr ;
timer Tsifs ;
*
(RxC_Idle)
ResetMAC
reset(Tsifs)
dcl ackTo MacAddr ;
dcl ctype ChangeType ;
dcl dAck, dCts, dRsp,
dSifsDly Duration ;
dcl endRx, strTs Time ;
dcl pdu, rspdu Frame ;
dcl sas, sau StationState ;
dcl susp Boolean:= false ;
imported mNavEnd Time ;
RxC_Idle,
RxC_Suspended
aRxTxTurn_
aroundTime)
dSifsDly:=
dUsec
(aSifsTime -
(aMpduDura_
tionFactor *
(float(aPream_
bleLength) +
float(aPlcpHdr_
Length) + float
(sSackCtsLng)
)))))
dRsp:= dUsec(
aSifsTime + fix(
0.5 + (sBitTime *
RxC_Idle
Class 1 frames handled
on this page, class 2 and
3 frames on next page.
Change
(ctype)
ctype
(bss)
else
NeedAck
(ackTo,
endRx, dAck)
Suspend
RxCfAck
RxIndicate
(pdu,endRx,
strTs,dAck)
RxFlush
dAck:= dAck -
if dAck>0 then
dRsp else 0 fi
Reply
Ack
ftype
(pdu)
(ack)
else
chk_
sst
Reply
rspdu:=
setAddr1
(ackframe,ackTo)
susp:= true
-
(probe_req,
beacon)
-
(cfend_ack)
rspdu:=
setDurId
(rspdu,dAck)
RxC_
_Suspended
Ack
MmIndicate
(pdu,endRx,
strTs,noerr)
to Synch_
ronization
(cfend)
send_
sifs
MmIndicate
(pdu,endRx,
-
Beacon and probe_req
sent to Synchronization
directly. All others
sent via Reassembly.
(authentication,
deauthentication,
disassociation,
probe_rsp)
set(endRx+
dSifsDly,
Tsifs)
strTs,noerr)
to Synch_
ronization
isGroup
(addr1(pdu))
(true)
(false)
(cts)
(rts)
Wait_Sifs
Cts
(endRx)
import
(mNavEnd)
> now
-
RxPdu(pdu)
(true)
(false)
Tsifs
*
-
rspdu:=setAddr1
(ctsframe,
addr2(pdu))
None of these
frames should
have group DA.
TxImed
(rspdu)
CTS respone to
RTS only when
the Nav is clear.
9.2.5.7 says the
RTS response
duration is based
on the data rate
of the RTS. This
is not currenly
done because
rx data rate is
not reported by
the Phy.
dCts:=
durId(rspdu) -
dRsp
Acknowledge
directed data &
mgmt frames.
rspdu:=setAddr1
(ackframe,
addr2(pdu))
Wait_TxDone
rspdu:=
setDurId
(rspdu,dCts)
send_
sifs
rspdu:=
setDurId
(rspdu,0)

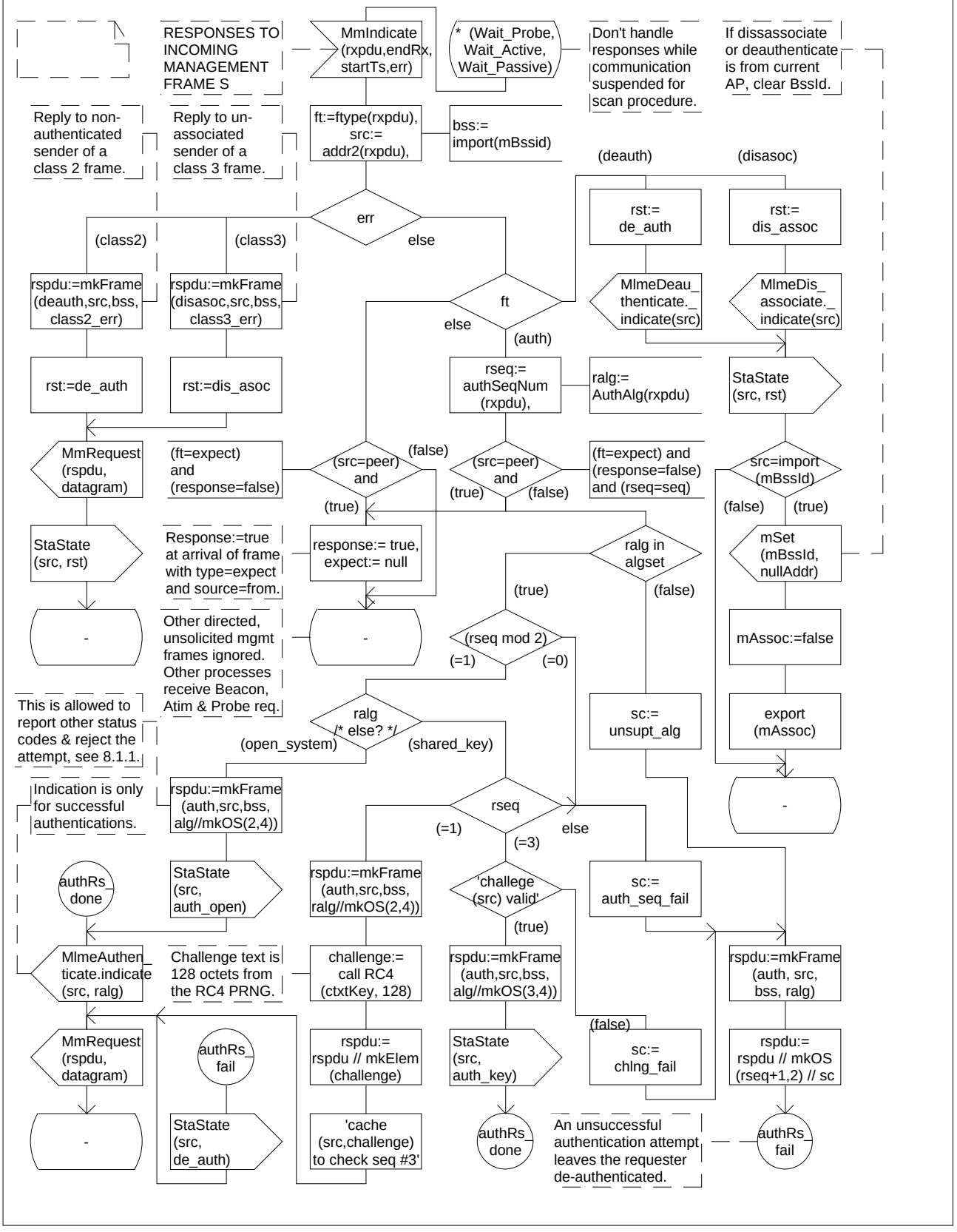

chk_
sst
SsInquiry
(addr2(pdu))
Wait_Response
SsResponse
(,sas,sau)
*
Rx with toDs=1
discarded by
Filter_MPDU.
frDs=1 never
sent by Sta, so
explicit test
for to/from
not needed here.
ftype(pdu)
else
RxC_Idle
(sau =
authOpen)
or
(sau =
authKey)
(false)
(true)
MmIndicate
(pdu, , ,class2)
to Mlme_Req_Rsp
uc_
ack
ftype(pdu)
else
(null_
_frame)
(atim)
RxC_Idle
MmIndicate
(pdu, , ,noerr)
to Synchronization
RxPdu(pdu)
uc_
ack
isGroup
(addr1(pdu))
(false)
dAck:= dAck -
if dAck>0 then
dRsp else 0 fi
rspdu:=setAddr1
(ackframe,
addr2(pdu))
rspdu:=
setDurId
(rspdu,dAck)
send_
sifs
(data, null, atim,
asoc_req, reasoc_req,
asoc_rsp, reasoc_rsp)
(data_ack, data_poll,
data_poll_ack, pspoll,
cfack, cfpoll, cfpoll_ack)
However, PS-Poll
should not be
sent to a station.
sas =
assoc
(false)
(true)
ftype(pdu)
(data_
_ack)
(ps_poll)
(cfpoll,
cfpoll_ack)
(data_poll,
data_poll_
_ack)
(cfack)
isGroup
(addr1(pdu))
isGroup
(addr1(pdu))
(true)
(false)
(true)
(false)
CfPoll
(endRx)
TxCfAck
(endRx)
RxPdu(pdu)
RxC_
_Suspended
Used while
station is
scanning.
RxC_Idle
Resume
susp:= false
RxC_Idle
RxIndicate
(pdu,endRx,
strTs,dAck)
ftype(pdu)
(beacon,
probe_rsp)
else
uc_
ack
Beacon and probe_rsp
sent to Mlme_Req_Rsp
while scaning, other
types acknowledged
(if unicast to this
station) but ignored.
MmIndicate
(pdu,endRx,
strTs,noerr)
to Mlme_
_Req_Rsp
Wait_TxDone
TxDone
*
susp
(false)
(true)
RxC_Idle
RxC_
Suspended
MmIndicate
(pdu, , ,class3)
to Mlme_Req_Rsp
uc_
ack

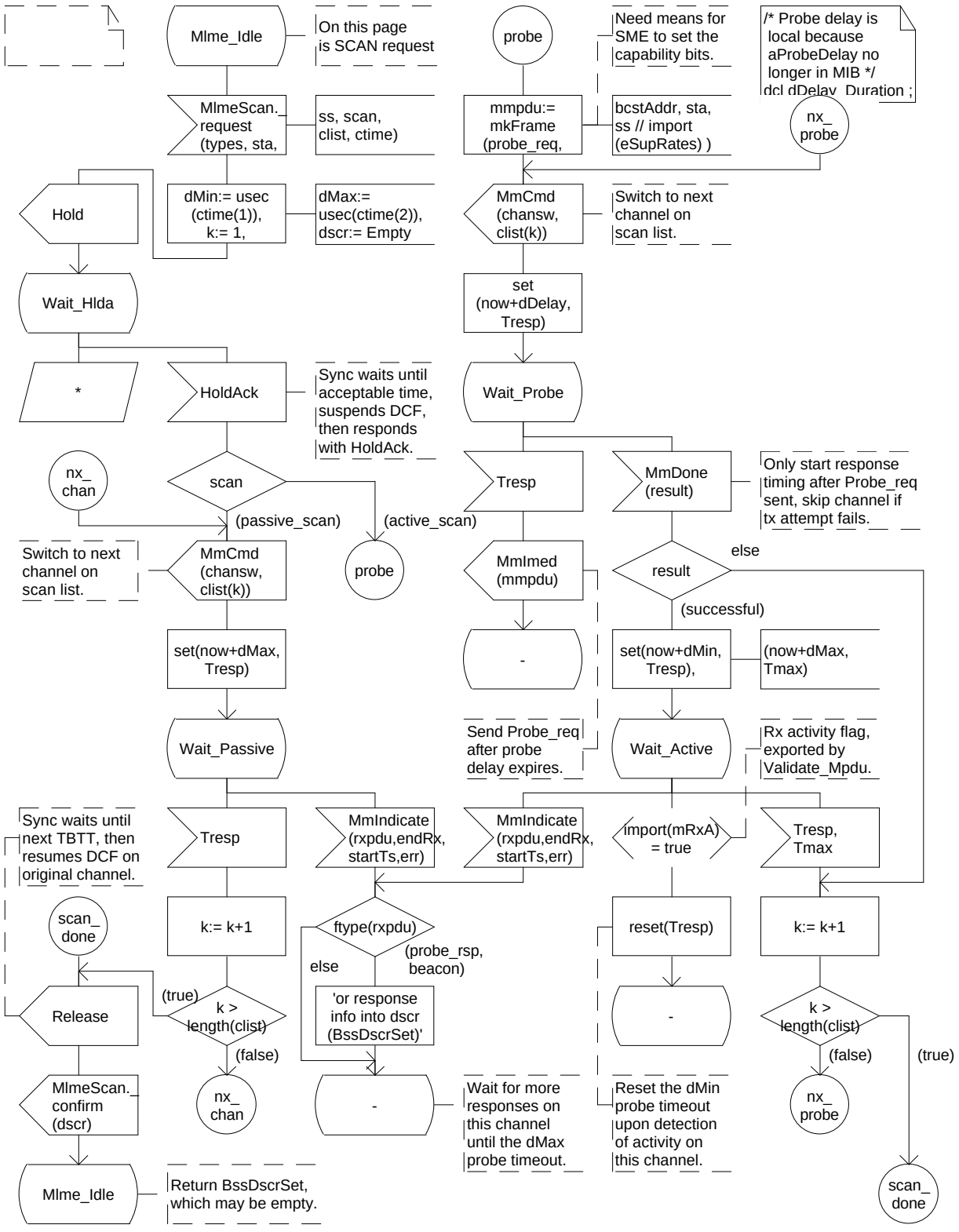

MMGT
Block Station_MLME
16_SM_1b(1)
MlmeJoin.confirm,
MlmeStart.confirm,
mSet
MlmeAssociate.confirm,
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeReassociate.confirm,
MlmeScan.confirm,
MlmeAuthenticate.indicate,
MlmeDeauthenticate.indicate,
MlmeDisassociate.indicate,
mSet
Bss
MlmeJoin.request,
MlmeStart.request,
ResetMAC
Tsf
MmConfirm
Synchronization
(1,1)
MmIndicate,
MmDone,
MmCmdAck
Hold,
Release
Idx
Mop
MmRequest
MMTX
MmRequest
Psm
Syn
Req
Hld
PsChange,
PsResponse,
SsResponse
MmConfirm
HoldAck
MlmeAssociate.request,
MlmeAuthenticate.request,
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeReassociate.request,
MlmeScan.request,
ResetMAC
Mlme_Req_Rsp
(1,1)
MmImed,
MmCancel,
MmCmd,
TBTT,
ResetMAC
SsResponse
MmDone,
MmIndicate
Rsp
Ssu
Rxs
MC_
TL
MmImed,
MmCmd
StaState,
SsInquiry,
ResetMAC
PsInquiry,
SsInquiry
SsResponse
Sst
Power_Save_
_Monitor(1,1)
SsInquiry
Signal
Hold,
HoldAck,
Release,
StaState(MacAddr,StationState) ;
PsIndicate,
SsInquiry
Records power
save mode and
station state.
FromRx
SsResponse
PS

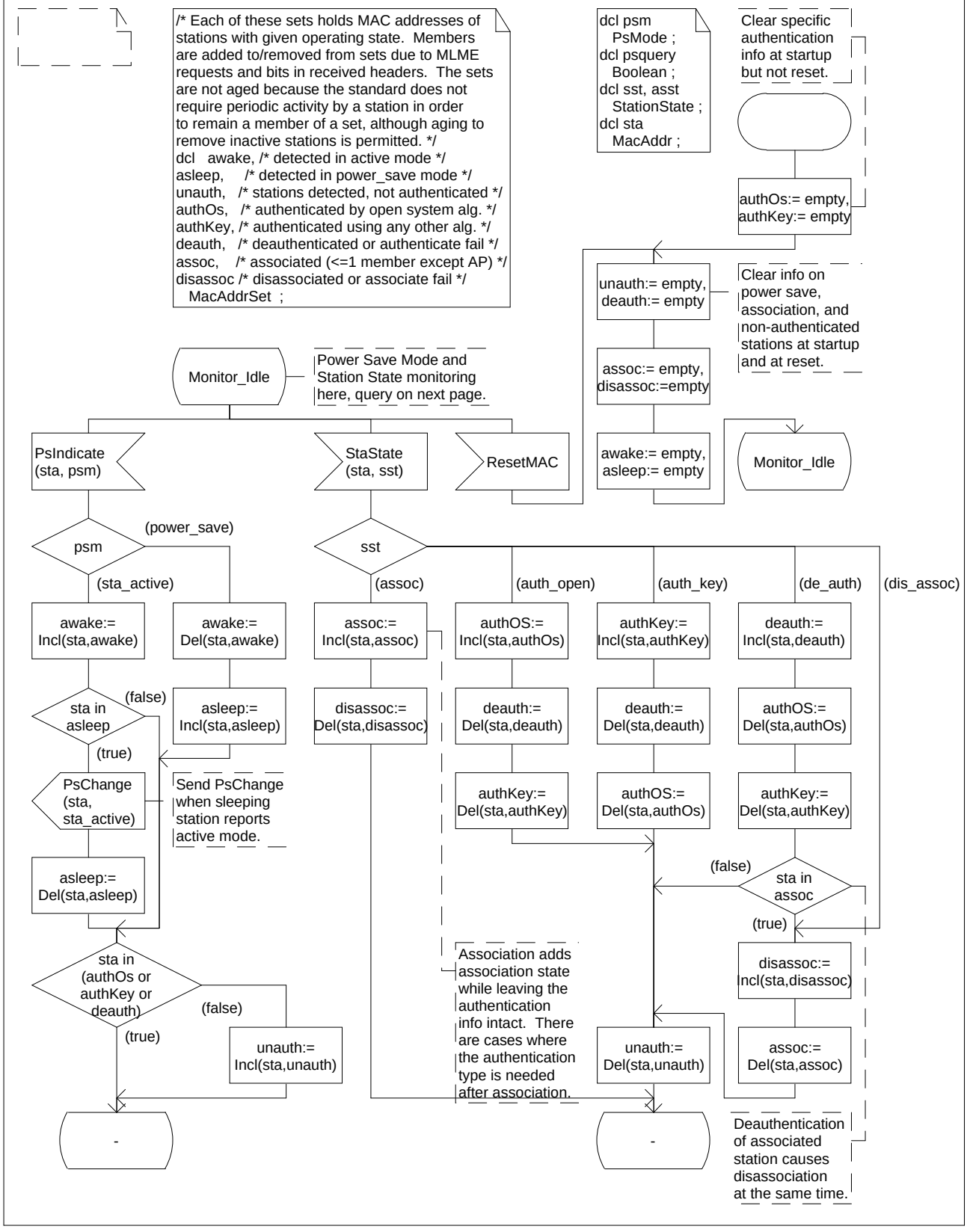

dcl exported mAssoc, mPcAvail, mPcDlvr, mPcPoll, mRxA Boolean ;
imported aAuthenticationAlgorithms AuthTypeSet ;
imported aAuthenticationType Natural ;
imported mBssId MacAddr ;
imported mCap, mSupRates Octetstring ;
*
ResetMAC
expect:= null
ResetMAC to Power_Save_ _Monitor
reset(Tresp)
algset:= import (aAuthentica_ tionAlgorithms)
dcl alg, ralg AuthType ;
dcl algset AuthTypeSet ;
dcl clist, ctime Intstring ;
dcl dResp, dMax, dMin Duration ;
dcl dscr BssDscrSet ;
dcl endRx, startTs Time ;
dcl err StateErr ;
dcl expect TypeSubtype:= null_frame ;
dcl mmpdu, rspdu, rxpdu Frame ;
dcl n, k, seq, rseq Integer ;
dcl ok Success ;
dcl psm PsMode ;
dcl response Boolean:= false ;
dcl result TxResult ;
dcl sc, challenge StatusCode ;
dcl scan ScanType ;
dcl ss Ocetestring ;
dcl sst, rst, asst StationState ;
dcl sta, peer, src, apAddr, bss MacAddr ;
dcl types BssTypeSet ;
timer Tresp, Tmax ;
mPcAvail:=false, mPcDlvr:=false, mPcPoll:=false
mAssoc:=false, apAddr:= nullAddr
Mlme_Idle
On this page are DISASSOCIATE Request and DEAUTHENTICATE Request.
export(mAssoc, mPcAvail, mPcDlvr,mPcPoll)
MlmeDis_ associate._ request(sta)
The Mlme request has no reason parm, use reason 1 (or 8).
MlmeDeau_ thenticate._ request(sta)
/* Source of the key used to generate challenge text is not specified. */
dcl ctxtKey Octetstring ;
mmpdu:= mkFrame (disassoc,
sta, import(mBssid), unspec_reason)
mmpdu:= mkFrame (deauth,
sta, import(mBssid), unspec_reason)
MmRequest (mmpdu, contention)
The Mlme request has no reason parm, use reason 1.
MmRequest (mmpdu, contention)
Wait_Disassoc_ _Confirm
Wait_Deauth_ _Confirm
MmConfirm (, result)
*
MmConfirm (, result)
*
ok:= (result= successful), sst:= dis_assoc
Update local station state even if frame is undeliverable.
StaState (sta, sst)
ok:= (result= successful), sst:= de_auth
MlmeDis_ associate._ confirm(ok)
mSet (mBssId, nullAddr)
(true)
sta=import (mBssId)
MlmeDeauth_ enticate.con_ firm(sta,ok)
(false)
If current AP is disassociated/ deauthenticated clear the BssId.
mAssoc:=false
export(mAssoc)
Mlme_Idle

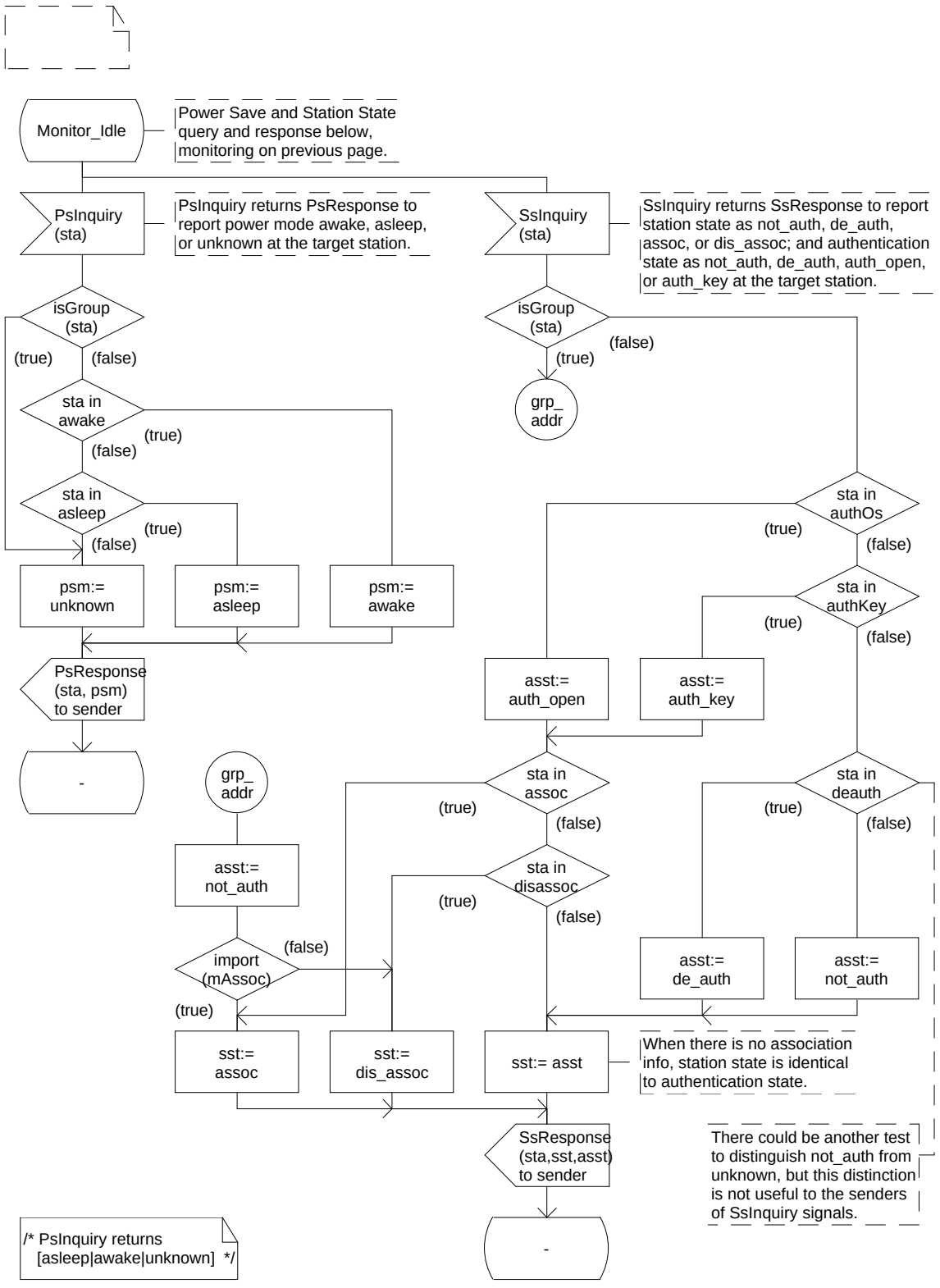

Mlme_Idle
On this page are ASSOCIATE Request and REASSOCIATE Request.
MlmeReasso_ciate.request (sta, n)
MlmeAssoc_iate.request (sta, n)
Reject asoc_req in state 3. Reasoc ok in states 2 or 3.
Wait_Asoc_ _Rsp
response:=false
more_ asoc
peer:= sta, expect:= reasoc_rsp
See 7.2.3.6. Need means for SME to set capability bits.
peer:= sta, expect:= asoc_rsp
Tresp
response
*
mmpdu:=mkFrame (reasoc_req, sta, import(mBssid),
(import(mCap) // mkOS (import(mListenInt),2) // apAddr // mkElem (eSsId,import(mSsId)) // mkElem(eSupRates, import(mSupRates))))
asoc_ fail
status (rxpdu)
Status in Mlme confirm has no cause of failure.
else
(successful)
import (mBssId)
(=nullAddr)
else
mSet (aPrivacy_ Invoked,
wepBit(rxpdu)), mSet(aStationID, asgnSID(rxpdu))
dResp:= dUsec(n)
mmpdu:=mkFrame (asoc_req, sta, nullAddr,
(import(mCap) // mkOS(import (mListenInt),2) // mkElem (eSsId,import (mSsId)) // mkElem(eSup_ Rates,import (mSupRates))))
peer:= nullAddr, expect:=null_frame, response:=false
mPcDlvr:= pollReqBit (rxpdu),
mPcPoll:= pollableBit(rxpdu), mPcAvail:= (mPcDlvr or mPcPoll), apAddr:= peer
set (now+dResp, Tresp)
See 7.2.3.4. Need means for SME to set capability bits.
export (mPcAvail, mPcDlvr,mPcPoll)
SsInquiry (sta)
'do something with supported rates element'
aBssBasicRateSet is no longer in the MIB. Without the basic rates it is unclear what to do with supported rates.
Wait_Sst_ _Response
ok:= false, sst:=dis_assoc
ok:= true, sst:= assoc
SsResponse (sta, ,asst)
*
Wait_Asoc_ _Confirm
reset(Tresp)
mAssoc:=ok
*
Tresp
MmConfirm (, result)
mSet (mBssId, peer)
export(mAssoc)
asst
result = successful
StaState (sta, sst)
else
(un_auth, de_auth)
(false)
(true)
MmRequest (mmpdu, contention)
asoc_ fail
asoc_ fail
more_ asoc
ftype (mmpdu)
(asoc_req)
(reasoc_req)
Wait_Asoc_ _Confirm
Mlme_Idle
Mlme_ Associate._ confirm(ok)
MlmeRe_ associate._ confirm(ok)

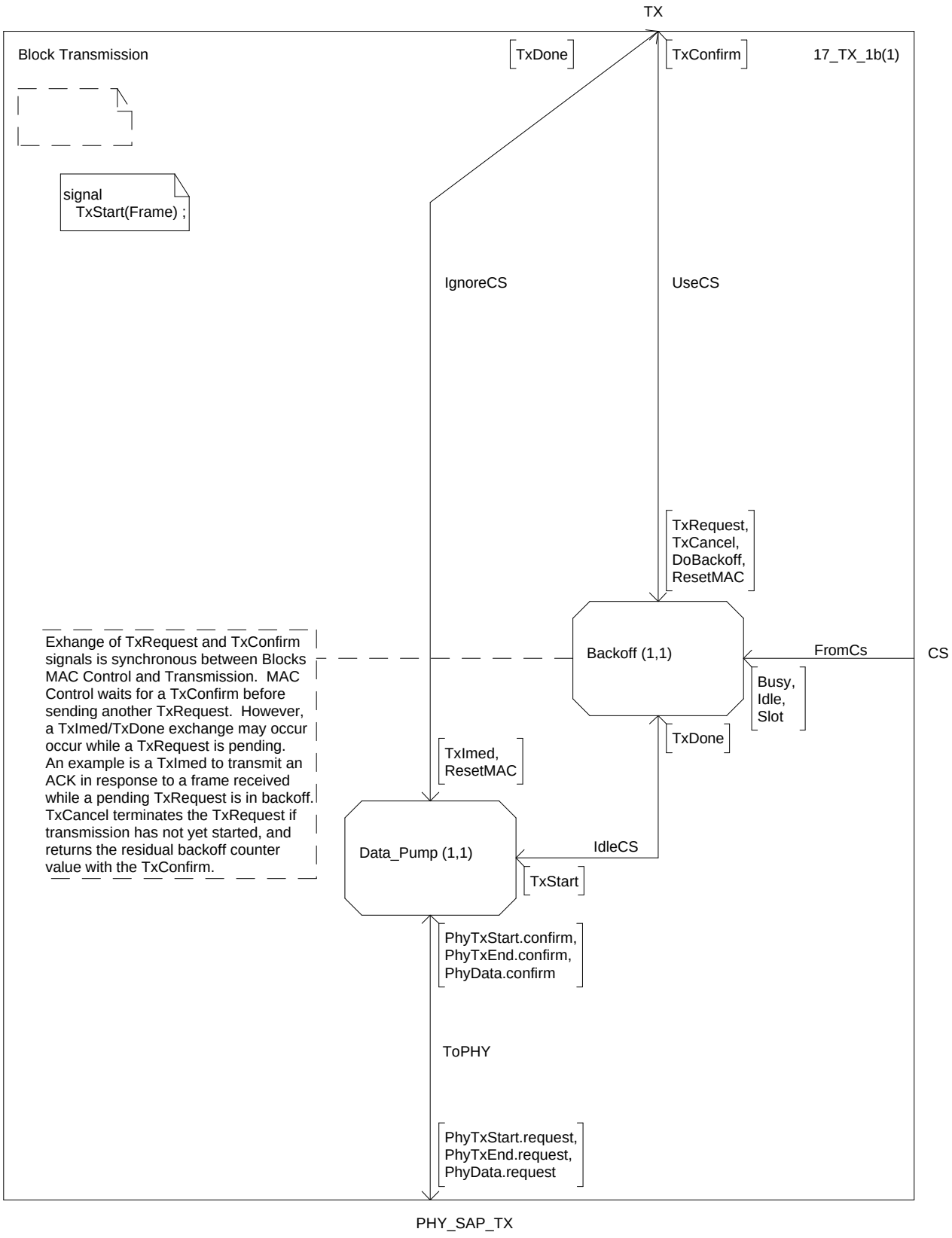

MlmeAuthen_
ticate.request
(sta,alg,n)
Mlme_Idle
On this page is
AUTHENTICATE
Request.
dResp:=
dUsec(n)
set
(now+dResp,
Tresp)
(alg in algset)
(true)
(false)
and (not
groupAddr(sta))
mmpdu:=mkFrame
(auth, sta,
import(mBssid),
alg // mkOS(1,2) //
successful),
peer:= sta,
expect:= auth
auth_
fail
MmRequest
(mmpdu,
contention)
Wait_Auth_
_Confirm_1
Tresp
MmConfirm
(, result)
*
result =
successful
(false)
(true)
seq:= 2,
response:=false
auth_
fail
Wait_Auth_
_Seq_2
Tresp
response
*
authStat
(rxpdu)
else
(successful)
alg
(open_system)
(shared_key)
ok:= true,
sst:=auth_open
auth_
fail
more_
auth
more_
auth
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)
Mark shared
key frame #3
for encryption.
MmRequest
(mmpdu,
contention)
Wait_Auth_
_Confirm_3
Tresp
MmConfirm
(mmpdu,
result)
*
result =
successful
(false)
(true)
seq:= 4,
response:=false
auth_
fail
Wait_Auth_
_Seq_4
*
Tresp
response
auth_
fail
expect:= null,
response:=false
authStat
(rxpdu)
else
(successful)
ok:= false,
sst:=de_auth
ok:= true,
sst:=auth_key
reset(Tresp)
StaState
(sta, sst)
An unsuccessful
authentication attempt
leaves the requester
de-authenticated.
Mlme_Idle
MlmeAuthen_
ticate.confirm
(sta,alg,ok),
mSet
(aAuthentication_
Type, alg)


RESPONSES TO INCOMING MANAGEMENT FRAME S
MmIndicate (rxpdu,endRx, startTs,err)
* (Wait_Probe, Wait_Active, Wait_Passive)
Don't handle responses while communication suspended for scan procedure.
If dissassociate 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.
ft:=ftype(rxpdu), src:= addr2(rxpdu),
bss:= import(mBssid)
(deauth)
(disasoc)
err
else
(class2)
(class3)
rst:= de_auth
rst:= dis_assoc
rspdu:=mkFrame (deauth,src,bss, class2_err)
rspdu:=mkFrame (disasoc,src,bss, class3_err)
ft
else
(auth)
MlmeDeau_ thenticate._ indicate(src)
MlmeDis_ associate._ indicate(src)
rst:=de_auth
rst:=dis_asoc
rseq:= authSeqNum (rxpdu),
ralg:= AuthAlg(rxpdu)
StaState (src, rst)
MmRequest (rspdu, datagram)
(ft=expect) and (response=false)
(src=peer) and
(false)
(true)
(src=peer) and
(ft=expect) and (response=false) and (rseq=seq)
src=import (mBssId)
(false)
(true)
StaState (src, rst)
Response:=true at arrival of frame with type=expect and source=from.
response:= true, expect:= null
ralg in algset
mSet (mBssId, nullAddr)
(true)
(false)
-
Other directed, unsolicited mgmt frames ignored. Other processes receive Beacon, Atim & Probe req.
-
(rseq mod 2)
(=1)
(=0)
mAssoc:=false
This is allowed to report other status codes & reject the attempt, see 8.1.1.
ralg /* else? */
(open_system)
(shared_key)
sc:= unsupt_alg
export (mAssoc)
Indication is only for successful authentications.
rspdu:=mkFrame (auth,src,bss, alg//mkOS(2,4))
rseq
(=1)
else
(=3)
-
authRs_ done
StaState (src, auth_open)
rspdu:=mkFrame (auth,src,bss, ralg//mkOS(2,4))
'challege (src) valid'
sc:= auth_seq_fail
(true)
MlmeAuthen_ ticate.indicate (src, ralg)
Challenge text is 128 octets from the RC4 PRNG.
challenge:= call RC4 (ctxtKey, 128)
rspdu:=mkFrame (auth,src,bss, alg//mkOS(3,4))
rspdu:=mkFrame (auth, src, bss, ralg)
(false)
MmRequest (rspdu, datagram)
authRs_ fail
rspdu:= rspdu // mkElem (challenge)
StaState (src, auth_key)
sc:= chlng_fail
rspdu:= rspdu // mkOS (rseq+1,2) // sc
-
StaState (src, de_auth)
'cache (src,challenge) to check seq #3'
authRs_ done
An unsuccessful authentication attempt leaves the requester de-authenticated.
authRs_ fail

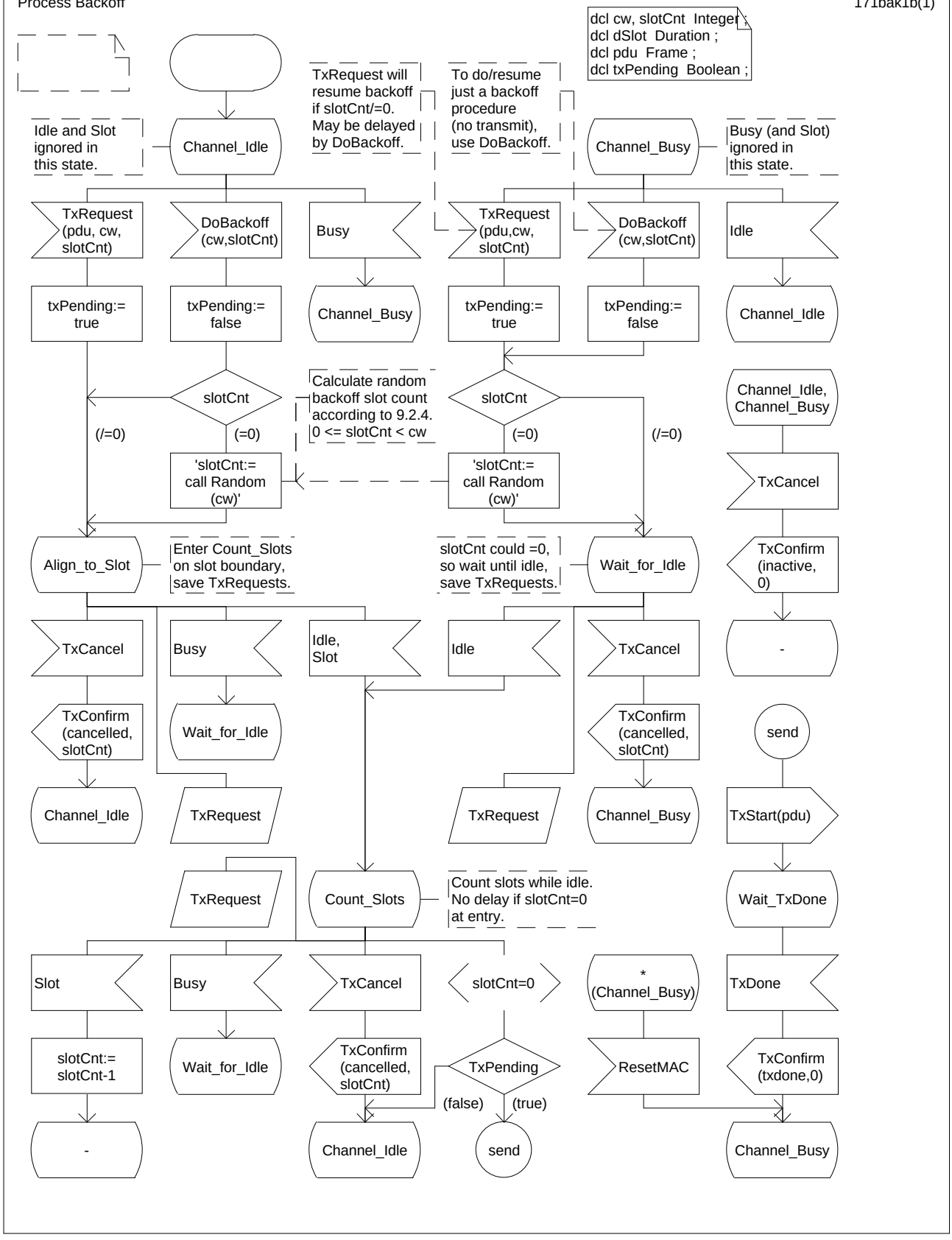

Mlme_Idle
On this page is SCAN request
MlmeScan. request (types, sta,
ss, scan, clist, ctime)
dMin:= usec (ctime(1)), k:= 1,
dMax:= usec(ctime(2)), dscr:= Empty
Hold
Wait_Hlda
*
HoldAck
Sync waits until acceptable time, suspends DCF, then responds with HoldAck.
nx_ chan
scan
(passive_scan)
(active_scan)
Switch to next channel on scan list.
MmCmd (chansw, clist(k))
probe
set(now+dMax, Tresp)
Wait_Passive
Sync waits until next TBTT, then resumes DCF on original channel.
Tresp
MmIndicate (rxpdu,endRx, startTs,err)
scan_ done
k:= k+1
ftype(rxpdu)
(probe_rsp, beacon)
else
(true)
k > length(clist)
'or response info into dscr (BssDscrSet)'
Release
(false)
nx_ chan
MlmeScan. confirm (dscr)
-
Wait for more responses on this channel until the dMax probe timeout.
Mlme_Idle
Return BssDscrSet, which may be empty.
probe
Need means for SME to set the capability bits.
mmpdu:= mkFrame (probe_req,
bcstAddr, sta, ss // import (eSupRates))
MmCmd (chansw, clist(k))
Switch to next channel on scan list.
set (now+dDelay, Tresp)
Wait_Probe
Tresp
MmImed (mmpdu)
-
Send Probe_req after probe delay expires.
MmIndicate (rxpdu,endRx, startTs,err)
/* Probe delay is local because aProbeDelay no longer in MIB */
dcl dDelay Duration ;
nx_ probe
MmDone (result)
Only start response timing after Probe_req sent, skip channel if tx attempt fails.
else
result
(successful)
set(now+dMin, Tresp),
(now+dMax, Tmax)
Wait_Active
Rx activity flag, exported by Validate_Mpdu.
import(mRxA) = true
reset(Tresp)
-
Reset the dMin probe timeout upon detection of activity on this channel.
Tresp, Tmax
k:= k+1
k > length(clist)
(false)
(true)
nx_ probe
scan_ done

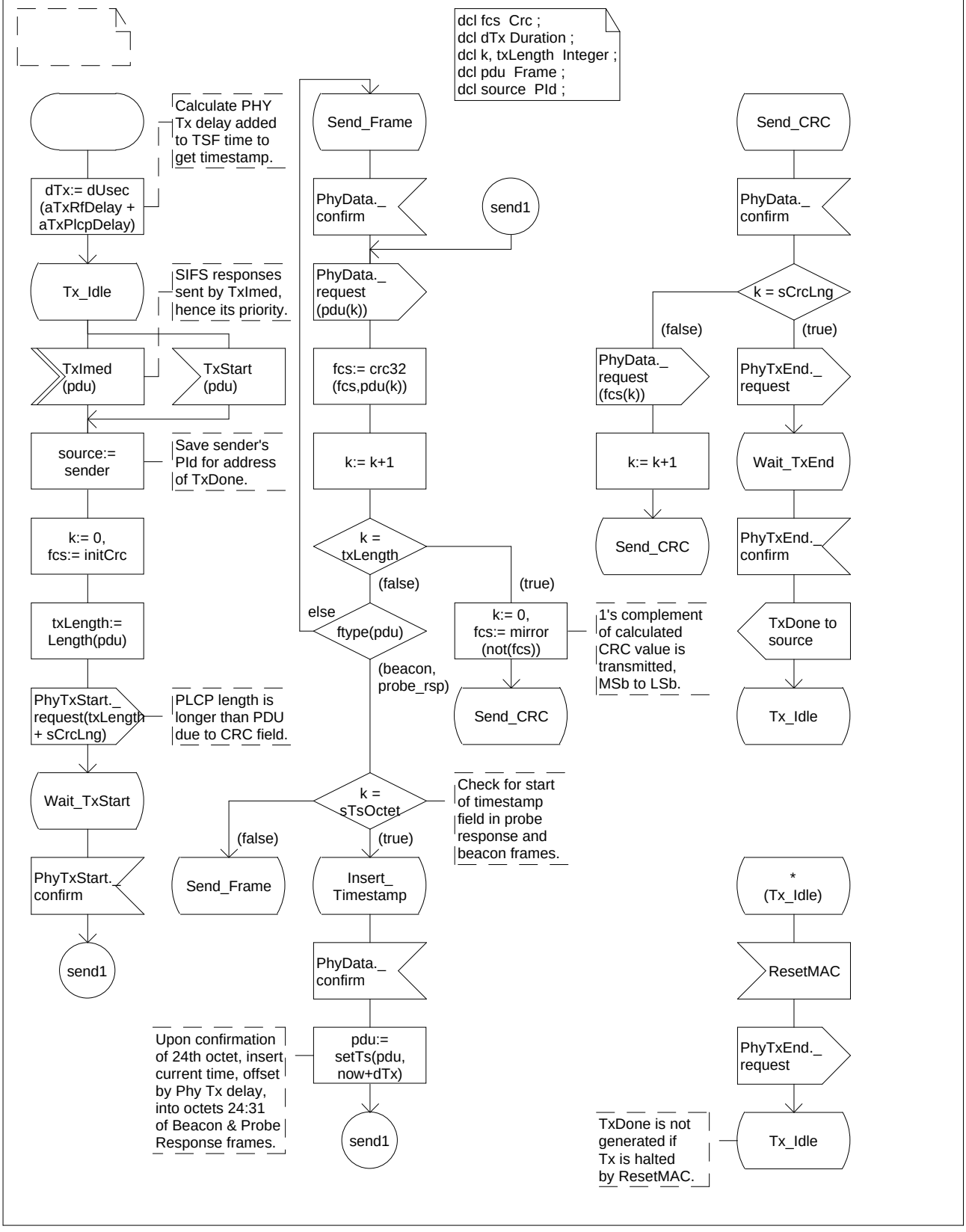

/* Each of these sets holds MAC addresses of stations with given operating state. Members are added to/removed from sets due to MLME requests and bits in received headers. The sets are not aged because the standard does not require periodic activity by a station in order to remain a member of a set, although aging to remove inactive stations is permitted. */
dcl awake, /* detected in active mode */
asleep, /* detected in power_save mode */
unauth, /* stations detected, not authenticated */
authOs, /* authenticated by open system alg. */
authKey, /* authenticated using any other alg. */
deauth, /* deauthenticated or authenticate fail */
assoc, /* associated (<=1 member except AP) */
disassoc /* disassociated or associate fail */
MacAddrSet ;
dcl psm
PsMode ;
dcl psquery
Boolean ;
dcl sst, asst
StationState ;
dcl sta
MacAddr ;
Clear specific authentication info at startup but not reset.
authOs:= empty, authKey:= empty
unauth:= empty, deauth:= empty
Clear info on power save, association, and non-authenticated stations at startup and at reset.
assoc:= empty, disassoc:=empty
awake:= empty, asleep:= empty
Monitor_Idle
Power Save Mode and Station State monitoring here, query on next page.
PsIndicate (sta, psm)
StaState (sta, sst)
ResetMAC
psm
(power_save)
(sta_active)
sst
(assoc)
(auth_open)
(auth_key)
(de_auth)
(dis_assoc)
awake:= Incl(sta,awake)
awake:= Del(sta,awake)
assoc:= Incl(sta,assoc)
authOS:= Incl(sta,authOs)
authKey:= Incl(sta,authKey)
deauth:= Incl(sta,deauth)
sta in asleep
(false)
(true)
asleep:= Incl(sta,asleep)
disassoc:= Del(sta,disassoc)
deauth:= Del(sta,deauth)
deauth:= Del(sta,deauth)
authOS:= Del(sta,authOs)
PsChange (sta, sta_active)
Send PsChange when sleeping station reports active mode.
authKey:= Del(sta,authKey)
authOS:= Del(sta,authOs)
authKey:= Del(sta,authKey)
asleep:= Del(sta,asleep)
sta in assoc
(false)
(true)
sta in (authOs or authKey or deauth)
(false)
(true)
unauth:= Incl(sta,unauth)
Association adds association state while leaving the authentication info intact. There are cases where the authentication type is needed after association.
disassoc:= Incl(sta,disassoc)
unauth:= Del(sta,unauth)
assoc:= Del(sta,assoc)
-
-
Deauthentication of associated station causes disassociation at the same time.

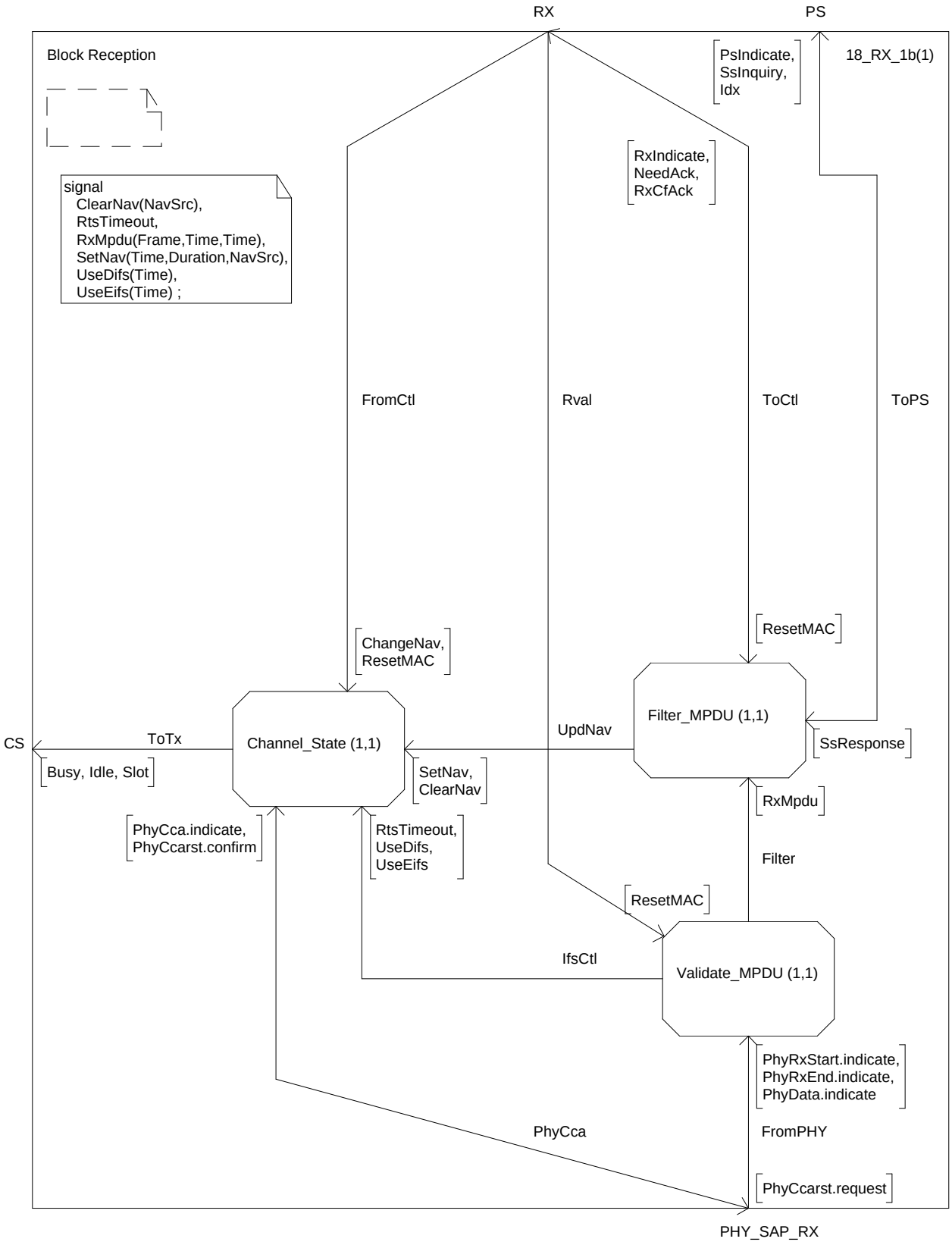

Monitor_Idle
Power Save and Station State query and response below, monitoring on previous page.
PsInquiry (sta)
PsInquiry returns PsResponse to report power mode awake, asleep, or unknown at the target station.
SsInquiry (sta)
SsInquiry returns SsResponse to report station state as not_auth, de_auth, assoc, or dis_assoc; and authentication state as not_auth, de_auth, auth_open, or auth_key at the target station.
isGroup (sta)
(true)
(false)
sta in awake
sta in asleep
psm:= unknown
psm:= asleep
psm:= awake
PsResponse (sta, psm) to sender
-
grp_ addr
sta in authOs
sta in authKey
asst:= auth_open
asst:= auth_key
sta in assoc
sta in disassoc
sta in deauth
asst:= not_auth
import (mAssoc)
asst:= de_auth
asst:= not_auth
sst:= assoc
sst:= dis_assoc
sst:= asst
When there is no association info, station state is identical to authentication state.
SsResponse (sta,sst,asst) to sender
There could be another test to distinguish not_auth from unknown, but this distinction is not useful to the senders of SsInquiry signals.
/* PsInquiry returns [asleep|awake|unknown] */

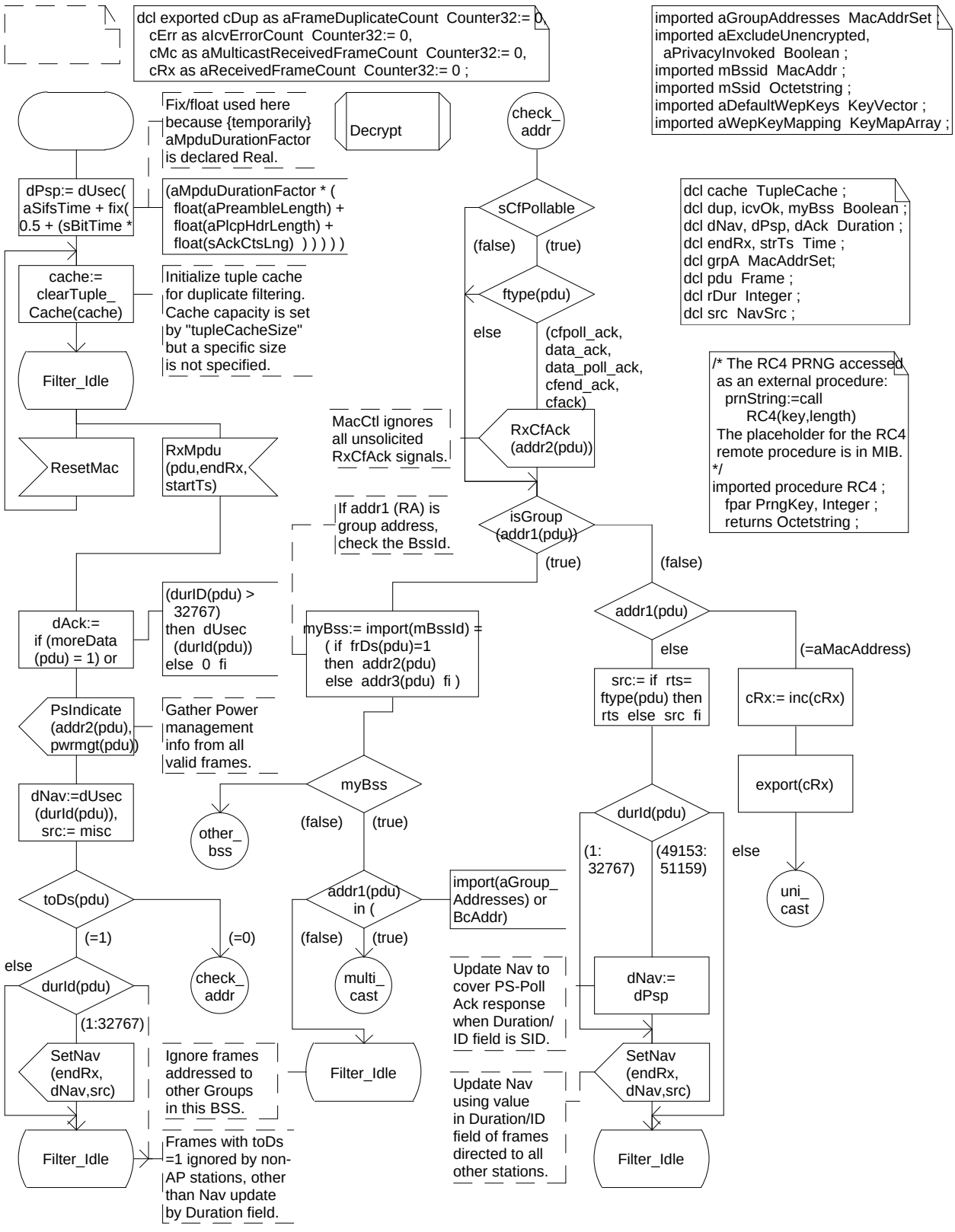

TX
Block Transmission
TxDone
TxConfirm
17_TX_1b(1)
signal
TxStart(Frame) ;
IgnoreCS
UseCS
TxRequest,
TxCancel,
DoBackoff,
ResetMAC
Exhange of TxRequest and TxConfirm signals is synchronous between Blocks MAC Control and Transmission. MAC Control waits for a TxConfirm before sending another TxRequest. However, a TxImed/TxDone exchange may occur 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.
Backoff (1,1)
FromCs
CS
Busy,
Idle,
Slot
TxDone
TxImed,
ResetMAC
Data_Pump (1,1)
IdleCS
TxStart
PhyTxStart.confirm,
PhyTxEnd.confirm,
PhyData.confirm
ToPHY
PhyTxStart.request,
PhyTxEnd.request,
PhyData.request
PHY_SAP_TX

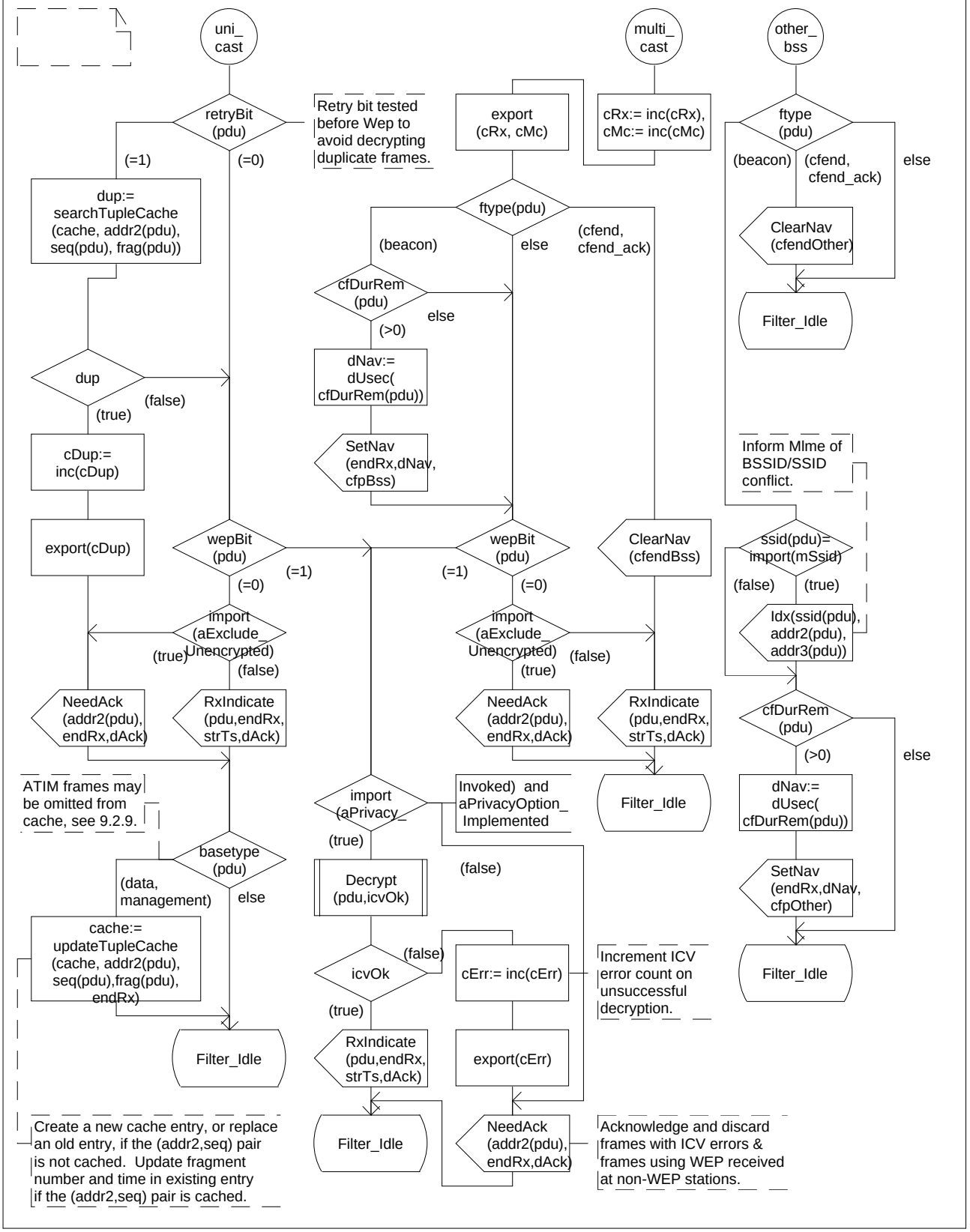
Process Backoff

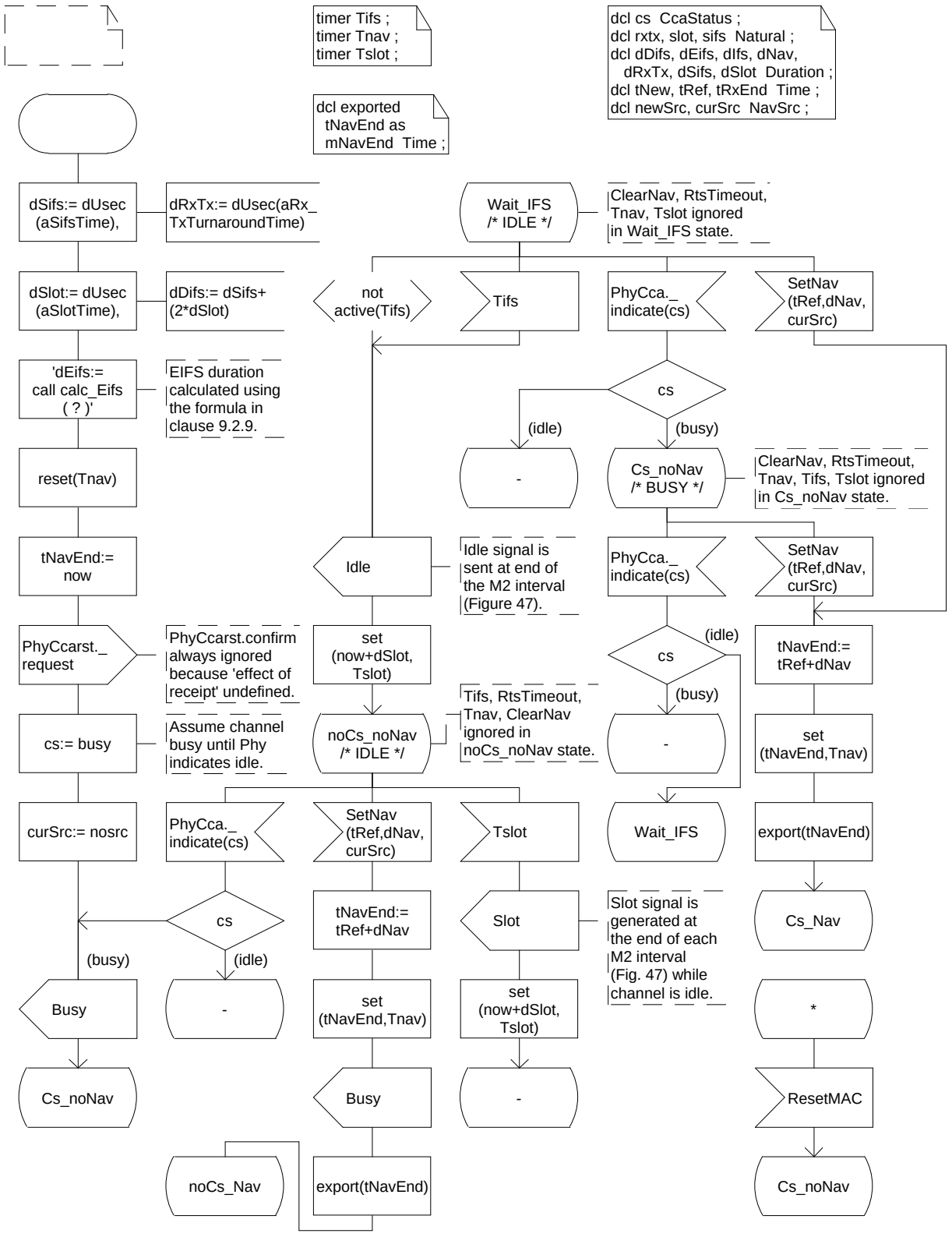


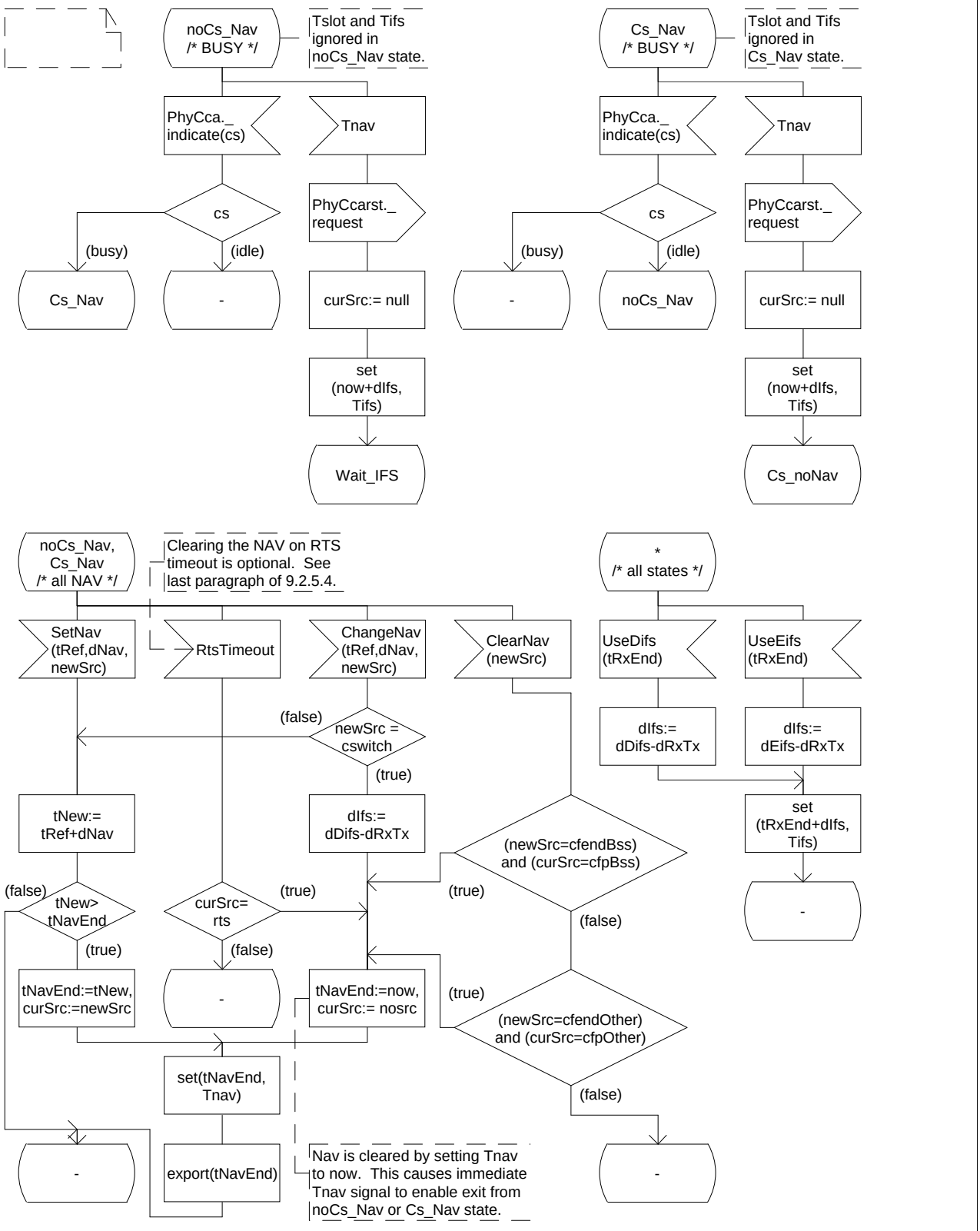

dcl fcs Crc ;
dcl dTx Duration ;
dcl k, txLength Integer ;
dcl pdu Frame ;
dcl source PId ;
Calculate PHY Tx delay added to TSF time to get timestamp.
dTx:= dUsec (aTxRfDelay + aTxPlcpDelay)
Tx_Idle
SIFS responses sent by TxImed, hence its priority.
TxImed (pdu)
TxStart (pdu)
source:= sender
Save sender's PId for address of TxDone.
k:= 0, fcs:= initCrc
txLength:= Length(pdu)
PhyTxStart._ request(txLength + sCrcLng)
PLCP length is longer than PDU due to CRC field.
Wait_TxStart
PhyTxStart._ confirm
send1
Send_Frame
PhyData._ confirm
send1
PhyData._ request (pdu(k))
fcs:= crc32 (fcs,pdu(k))
k:= k+1
k = txLength
(false)
(true)
else
ftype(pdu)
(beacon, probe_rsp)
k = sTsOctet
Check for start of timestamp field in probe response and beacon frames.
(false)
(true)
Send_Frame
Insert_ Timestamp
PhyData._ confirm
pdu:= setTs(pdu, now+dTx)
Upon confirmation of 24th octet, insert current time, offset by Phy Tx delay, into octets 24:31 of Beacon & Probe Response frames.
send1
k:= 0, fcs:= mirror (not(fcs))
1's complement of calculated CRC value is transmitted, MSb to LSb.
Send_CRC
Send_CRC
PhyData._ confirm
k = sCrcLng
(false)
(true)
PhyData._ request (fcs(k))
k:= k+1
Send_CRC
PhyTxEnd._ request
Wait_TxEnd
PhyTxEnd._ confirm
TxDone to source
Tx_Idle
*
(Tx_Idle)
ResetMAC
PhyTxEnd._ request
Tx_Idle
TxDone is not generated if Tx is halted by ResetMAC.

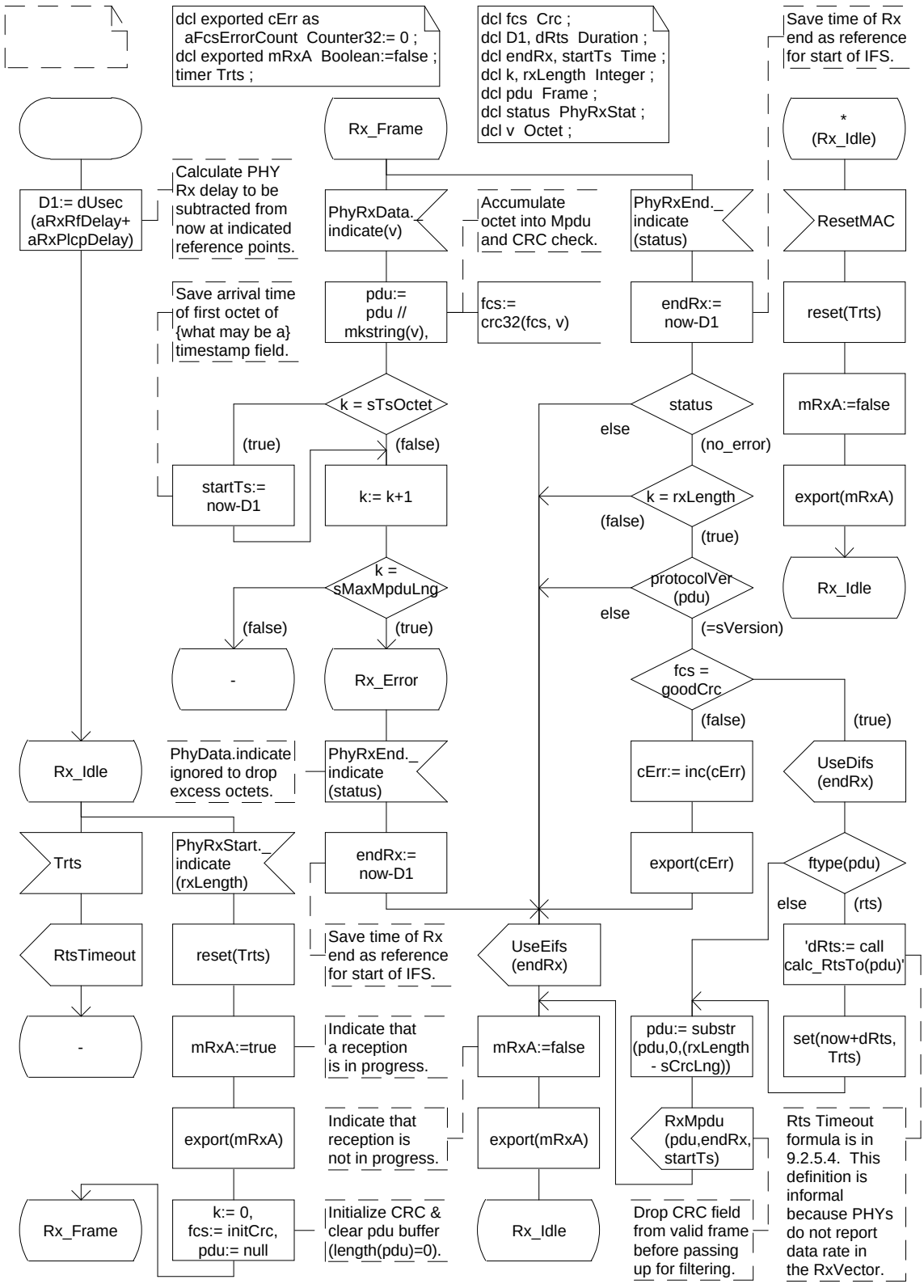

RX
PS
Block Reception
PsIndicate,
SsInquiry,
Idx
18_RX_1b(1)
RxIndicate,
NeedAck,
RxCfAck
signal
ClearNav(NavSrc),
RtsTimeout,
RxMpdu(Frame,Time,Time),
SetNav(Time,Duration,NavSrc),
UseDifs(Time),
UseEifs(Time) ;
FromCtl
Rval
ToCtl
ToPS
ResetMAC
ChangeNav,
ResetMAC
Filter_MPDU (1,1)
CS
ToTx
Channel_State (1,1)
UpdNav
SsResponse
Busy, Idle, Slot
SetNav,
ClearNav
RxMpdu
PhyCca.indicate,
PhyCcarst.confirm
RtsTimeout,
UseDifs,
UseEifs
Filter
ResetMAC
IfsCtl
Validate_MPDU (1,1)
PhyRxStart.indicate,
PhyRxEnd.indicate,
PhyData.indicate
PhyCca
FromPHY
PhyCcarst.request
PHY_SAP_RX

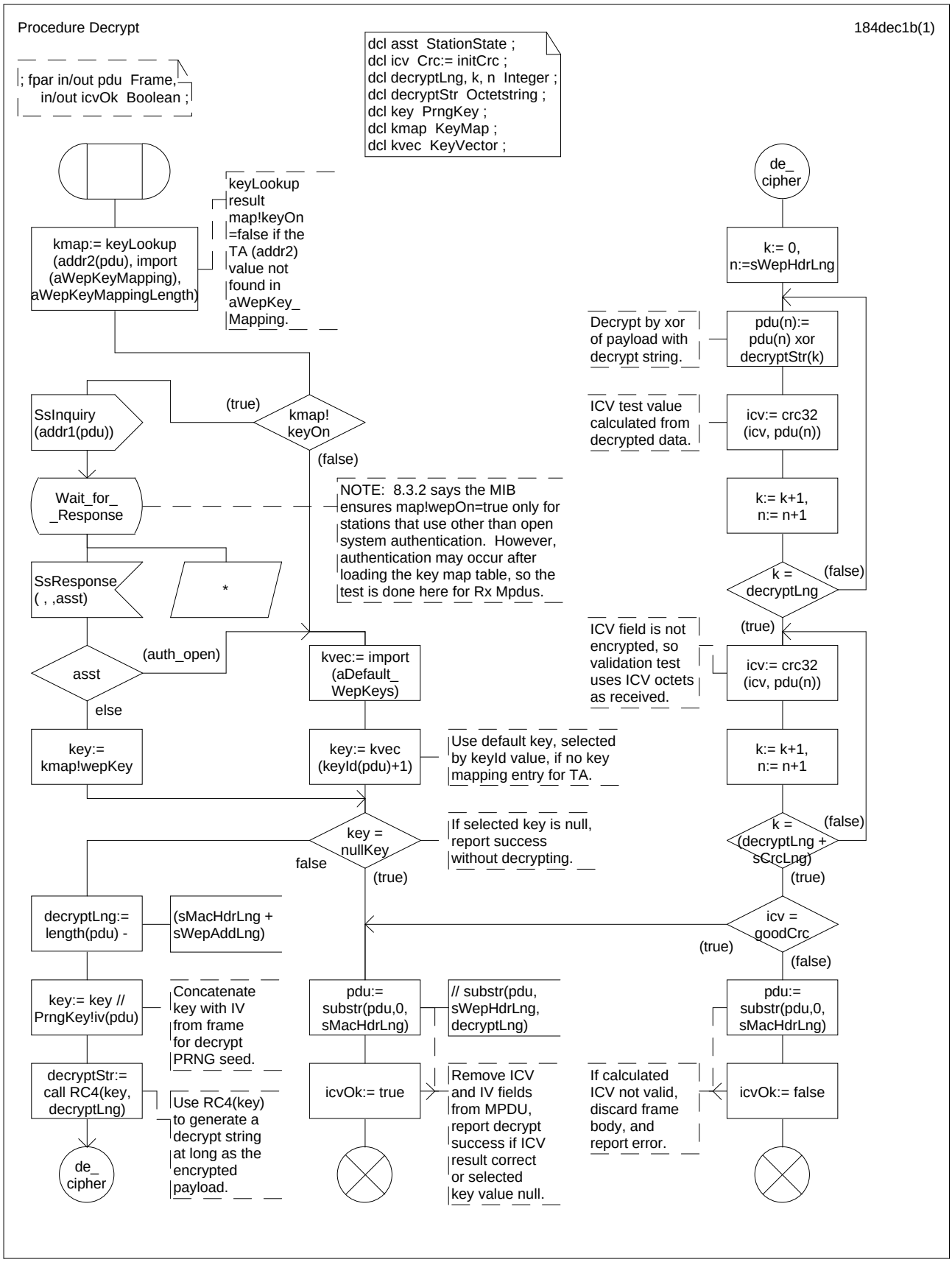

dcl exported cDup as aFrameDuplicateCount Counter32:= 0,
cErr as aIcvErrorCount Counter32:= 0,
cMc as aMulticastReceivedFrameCount Counter32:= 0,
cRx as aReceivedFrameCount Counter32:= 0 ;
imported aGroupAddresses MacAddrSet ;
imported aExcludeUnencrypted,
aPrivacyInvoked Boolean ;
imported mBssid MacAddr ;
imported mSsid Octetstring ;
imported aDefaultWepKeys KeyVector ;
imported aWepKeyMapping KeyMapArray ;
Fix/float used here because {temporarily} aMpduDurationFactor is declared Real.
Decrypt
check_ addr
dPsp:= dUsec(aSifsTime + fix(0.5 + (sBitTime *
(aMpduDurationFactor * (float(aPreambleLength) + float(aPlcpHdrLength) + float(sAckCtsLng)))))
sCfPollable
(false)
(true)
dcl cache TupleCache ;
dcl dup, icvOk, myBss Boolean ;
dcl dNav, dPsp, dAck Duration ;
dcl endRx, strTs Time ;
dcl grpA MacAddrSet;
dcl pdu Frame ;
dcl rDur Integer ;
dcl src NavSrc ;
cache:= clearTuple_ Cache(cache)
Initialize tuple cache for duplicate filtering. Cache capacity is set by "tupleCacheSize" but a specific size is not specified.
ftype(pdu)
else
(cfpoll_ack, data_ack, data_poll_ack, cfend_ack, cfack)
/* The RC4 PRNG 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 ;
Filter_Idle
MacCtl ignores all unsolicited RxCfAck signals.
RxCfAck (addr2(pdu))
ResetMac
RxMpdu (pdu,endRx, startTs)
If addr1 (RA) is group address, check the BssId.
isGroup (addr1(pdu))
(true)
(false)
(durID(pdu) > 32767) then dUsec (durId(pdu)) else 0 fi
dAck:= if (moreData (pdu) = 1) or
myBss:= import(mBssId) = (if frDs(pdu)=1 then addr2(pdu) else addr3(pdu) fi)
addr1(pdu)
else
(=aMacAddress)
src:= if rts= ftype(pdu) then rts else src fi
cRx:= inc(cRx)
PsIndicate (addr2(pdu), pwrmgt(pdu))
Gather Power management info from all valid frames.
export(cRx)
dNav:=dUsec (durId(pdu)), src:= misc
myBss
other_ bss
(false)
(true)
durId(pdu)
(1: 32767)
(49153: 51159)
else
uni_ cast
toDs(pdu)
addr1(pdu) in (
import(aGroup_ Addresses) or BcAddr)
(=1)
(=0)
(false)
(true)
else
durId(pdu)
check_ addr
multi_ cast
Update Nav to cover PS-Poll Ack response when Duration/ ID field is SID.
dNav:= dPsp
(1:32767)
SetNav (endRx, dNav,src)
Ignore frames addressed to other Groups in this BSS.
Filter_Idle
Update Nav using value in Duration/ID field of frames directed to all other stations.
SetNav (endRx, dNav,src)
Filter_Idle
Frames with toDs =1 ignored by non-AP stations, other than Nav update by Duration field.
Filter_Idle


uni_ cast
multi_ cast
other_ bss
retryBit (pdu)
Retry bit tested before Wep to avoid decrypting duplicate frames.
(=1)
(=0)
dup:= searchTupleCache (cache, addr2(pdu), seq(pdu), frag(pdu))
dup
(false)
(true)
cDup:= inc(cDup)
export(cDup)
wepBit (pdu)
(=1)
(=0)
import (aExclude_ Unencrypted)
(true)
(false)
NeedAck (addr2(pdu), endRx,dAck)
RxIndicate (pdu,endRx, strTs,dAck)
ATIM frames may be omitted from cache, see 9.2.9.
basetype (pdu)
(data, management)
else
cache:= updateTupleCache (cache, addr2(pdu), seq(pdu),frag(pdu), endRx)
Filter_Idle
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.
export (cRx, cMc)
ftype(pdu)
(beacon)
else
(cfend, cfend_ack)
cfDurRem (pdu)
else
(>0)
dNav:= dUsec(cfDurRem(pdu))
SetNav (endRx,dNav, cfpBss)
wepBit (pdu)
(=1)
(=0)
import (aExclude_ Unencrypted)
(true)
(false)
NeedAck (addr2(pdu), endRx,dAck)
import (aPrivacy_ Invoked) and aPrivacyOption_ Implemented
(true)
(false)
Decrypt (pdu,icvOk)
icvOk
(false)
(true)
cErr:= inc(cErr)
Increment ICV error count on unsuccessful decryption.
RxIndicate (pdu,endRx, strTs,dAck)
export(cErr)
Filter_Idle
NeedAck (addr2(pdu), endRx,dAck)
Acknowledge and discard frames with ICV errors & frames using WEP received at non-WEP stations.
cRx:= inc(cRx), cMc:= inc(cMc)
ClearNav (cfendBss)
RxIndicate (pdu,endRx, strTs,dAck)
Filter_Idle
ftype (pdu)
(beacon)
(cfend, cfend_ack)
else
ClearNav (cfendOther)
Filter_Idle
Inform Mlme of BSSID/SSID conflict.
ssid(pdu)= import(mSsid)
(false)
(true)
Idx(ssid(pdu), addr2(pdu), addr3(pdu))
cfDurRem (pdu)
(>0)
else
dNav:= dUsec(cfDurRem(pdu))
SetNav (endRx,dNav, cfpOther)
Filter_Idle


timer Tifs ;
timer Tnav ;
timer Tslot ;
dcl cs CcaStatus ;
dcl rxtx, slot, sifs Natural ;
dcl dDifs, dEifs, dIfs, dNav,
dRxTx, dSifs, dSlot Duration ;
dcl tNew, tRef, tRxEnd Time ;
dcl newSrc, curSrc NavSrc ;
dcl exported
tNavEnd as
mNavEnd Time ;
dSifs:= dUsec
(aSifsTime),
dRxTx:= dUsec(aRx_
TxTurnaroundTime)
Wait_IFS
/* IDLE */
ClearNav, RtsTimeout,
Tnav, Tslot ignored
in Wait_IFS state.
dSlot:= dUsec
(aSlotTime),
dDifs:= dSifs+
(2*dSlot)
not
active(Tifs)
Tifs
PhyCca._
indicate(cs)
SetNav
(tRef,dNav,
curSrc)
'dEifs:=
call calc_Eifs
(?)'
EIFS duration
calculated using
the formula in
clause 9.2.9.
cs
(idle)
(busy)
reset(Tnav)
-
Cs_noNav
/* BUSY */
ClearNav, RtsTimeout,
Tnav, Tifs, Tslot ignored
in Cs_noNav state.
tNavEnd:=
now
Idle
Idle signal is
sent at end of
the M2 interval
(Figure 47).
PhyCca._
indicate(cs)
SetNav
(tRef,dNav,
curSrc)
PhyCcarst._
request
PhyCcarst.confirm
always ignored
because 'effect of
receipt' undefined.
set
(now+dSlot,
Tslot)
(idle)
cs
tNavEnd:=
tRef+dNav
(busy)
cs:= busy
Assume channel
busy until Phy
indicates idle.
noCs_noNav
/* IDLE */
Tifs, RtsTimeout,
Tnav, ClearNav
ignored in
noCs_noNav state.
-
set
(tNavEnd,Tnav)
curSrc:= nosrc
PhyCca._
indicate(cs)
SetNav
(tRef,dNav,
curSrc)
Tslot
Wait_IFS
export(tNavEnd)
cs
tNavEnd:=
tRef+dNav
Slot
Slot signal is
generated at
the end of each
M2 interval
(Fig. 47) while
channel is idle.
Cs_Nav
(busy)
(idle)
Busy
-
set
(tNavEnd,Tnav)
set
(now+dSlot,
Tslot)
*
Cs_noNav
Busy
-
ResetMAC
noCs_Nav
export(tNavEnd)
Cs_noNav


noCs_Nav
/* BUSY */
Tslot and Tifs ignored in noCs_Nav state.
PhyCca._ indicate(cs)
Tnav
cs
PhyCcarst._ request
(busy)
(idle)
Cs_Nav
-
curSrc:= null
set (now+dIfs, Tifs)
Wait_IFS
Cs_Nav
/* BUSY */
Tslot and Tifs ignored in Cs_Nav state.
PhyCca._ indicate(cs)
Tnav
cs
PhyCcarst._ request
(busy)
(idle)
-
noCs_Nav
curSrc:= null
set (now+dIfs, Tifs)
Cs_noNav
noCs_Nav, Cs_Nav
/* all NAV */
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)
(false)
newSrc = cswitch
(true)
tNew:= tRef+dNav
dIfs:= dDifs-dRxTx
(false)
tNew> tNavEnd
(true)
curSrc= rts
(true)
(false)
tNavEnd:=tNew, curSrc:=newSrc
-
tNavEnd:=now, curSrc:= nosrc
set(tNavEnd, Tnav)
-
export(tNavEnd)
Nav is cleared by setting Tnav to now. This causes immediate Tnav signal to enable exit from noCs_Nav or Cs_Nav state.
*
/* all states */
UseDifs (tRxEnd)
UseEifs (tRxEnd)
dIfs:= dDifs-dRxTx
dIfs:= dEifs-dRxTx
set (tRxEnd+dIfs, Tifs)
-
(newSrc=cfendBss) and (curSrc=cfpBss)
(true)
(false)
(true)
(newSrc=cfendOther) and (curSrc=cfpOther)
(false)
-


dcl exported cErr as aFcsErrorCount Counter32:= 0 ;
dcl exported mRxA Boolean:=false ;
timer Trts ;
dcl fcs Crc ;
dcl D1, dRts Duration ;
dcl endRx, startTs Time ;
dcl k, rxLength Integer ;
dcl pdu Frame ;
dcl status PhyRxStat ;
dcl v Octet ;
Save time of Rx end as reference for start of IFS.
Rx_Frame
*
(Rx_Idle)
D1:= dUsec (aRxRfDelay+ aRxPlcpDelay)
Calculate PHY Rx delay to be subtracted from now at indicated reference points.
PhyRxData._ indicate(v)
Accumulate octet into Mpdu and CRC check.
PhyRxEnd._ indicate (status)
ResetMAC
Save arrival time of first octet of {what may be a} timestamp field.
pdu:= pdu // mkstring(v),
fcs:= crc32(fcs, v)
endRx:= now-D1
reset(Trts)
k = sTsOctet
status
mRxA:=false
(true)
(false)
else
(no_error)
startTs:= now-D1
k:= k+1
k = rxLength
export(mRxA)
(false)
(true)
k = sMaxMpduLng
protocolVer (pdu)
Rx_Idle
(false)
(true)
else
(=sVersion)
-
Rx_Error
fcs = goodCrc
(false)
(true)
Rx_Idle
PhyData.indicate ignored to drop excess octets.
PhyRxEnd._ indicate (status)
cErr:= inc(cErr)
UseDifs (endRx)
Trts
PhyRxStart._ indicate (rxLength)
endRx:= now-D1
export(cErr)
ftype(pdu)
else
(rts)
RtsTimeout
reset(Trts)
Save time of Rx end as reference for start of IFS.
UseEifs (endRx)
'dRts:= call calc_RtsTo(pdu)'
-
mRxA:=true
Indicate that a reception is in progress.
mRxA:=false
pdu:= substr (pdu,0,(rxLength - sCrcLng))
set(now+dRts, Trts)
export(mRxA)
Indicate that reception is not in progress.
export(mRxA)
RxMpdu (pdu,endRx, startTs)
Rts Timeout formula is in 9.2.5.4. This definition is informal because PHYs do not report data rate in the RxVector.
Rx_Frame
k:= 0, fcs:= initCrc, pdu:= null
Initialize CRC & clear pdu buffer (length(pdu)=0).
Rx_Idle
Drop CRC field from valid frame before passing up for filtering.


Procedure Decrypt
184dec1b(1)
; fpar in/out pdu Frame,
in/out icvOk Boolean ;
dcl asst StationState ;
dcl icv Crc:= initCrc ;
dcl decryptLng, k, n Integer ;
dcl decryptStr Octetstring ;
dcl key PrngKey ;
dcl kmap KeyMap ;
dcl kvec KeyVector ;
kmap:= keyLookup (addr2(pdu), import (aWepKeyMapping), aWepKeyMappingLength)
keyLookup result map!keyOn =false if the TA (addr2) value not found in aWepKey_ Mapping.
SsInquiry (addr1(pdu))
(true)
kmap! keyOn
(false)
Wait_for_ _Response
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.
SsResponse (, ,asst)
*
asst
(auth_open)
else
kvec:= import (aDefault_ WepKeys)
key:= kmap!wepKey
key:= kvec (keyId(pdu)+1)
Use default key, selected by keyId value, if no key mapping entry for TA.
key = nullKey
If selected key is null, report success without decrypting.
false
(true)
decryptLng:= length(pdu) -
(sMacHdrLng + sWepAddLng)
key:= key // PrngKey!iv(pdu)
Concatenate key with IV from frame for decrypt PRNG seed.
decryptStr:= call RC4(key, decryptLng)
Use RC4(key) to generate a decrypt string at long as the encrypted payload.
de_ cipher
pdu:= substr(pdu,0, sMacHdrLng)
// substr(pdu, sWepHdrLng, decryptLng)
icvOk:= true
Remove ICV and IV fields from MPDU, report decrypt success if ICV result correct or selected key value null.
de_ cipher
k:= 0, n:=sWepHdrLng
Decrypt by xor of payload with decrypt string.
pdu(n):= pdu(n) xor decryptStr(k)
ICV test value calculated from decrypted data.
icv:= crc32 (icv, pdu(n))
k:= k+1, n:= n+1
k = decryptLng
(false)
(true)
ICV field is not encrypted, so validation test uses ICV octets as received.
icv:= crc32 (icv, pdu(n))
k:= k+1, n:= n+1
k = (decryptLng + sCrcLng)
(false)
(true)
icv = goodCrc
(true)
(false)
pdu:= substr(pdu,0, sMacHdrLng)
If calculated ICV not valid, discard frame body, and report error.
icvOk:= false

Package macsorts

3101_b\MacEnum(23)

```
/*     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 TxConfirm 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, cfpOther, 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 MsduRequest, 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 MaUnitdata.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 MaUnitdataStatus.indication
      NOTE:  All transmission failures (vs. Msdu 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 <rxerror> 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 ;   fpar 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 ;
```

```
/*  NAMED STATIC DATA VALUES -- names begin with "s" in the form "sNameOfItem"  */
      /* Maximum number of octets in an MSDU passed to or from LLC */
synonym sMaxMsduLng  Integer = 2304 ;
      /* Number of octets in the basic MAC header for Data and Mgmt frames (without WEP) */
synonym sMacHdrLng  Integer = 24 ;
      /* Number of octets in the basic MAC header plus IV/keyID for Data and Mgmt frames with WEP */
synonym sWepHdrLng  Integer = 28 ;
      /* Number of octets added to a PDU when using WEP (both IV/keyID and ICV fields) */
synonym sWepAddLng  Integer = 8 ;
      /* Number of octets added to the MAC header for Wireless Distribution System transfers */
synonym sWdsAddLng  Integer = 6 ;     /* this is the length of the addr4 field */
      /* Number of octets in a CRC (or ICV) field */
synonym sCrcLng  Integer = 4 ;
      /* Maximum number of octets in an MPDU, and the corresponding index range */
synonym sMaxMpduLng  Integer = (sMaxMsduLng+sMacHdrLng+sWdsAddLng+sWepAddLng+sCrcLng) ;
syntype FrameIndexRange = Integer  constants 1:sMaxMpduLng   endsyntype 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 */
      /* Minimum allowed value for aMpduMaxLength */
synonym sMinFragLng  Integer = 256 ;   /* NOTE: Fix the MIB, which no longer lists this minimum. */
      /* Maximum fragment number and corresponding index range */
synonym sMaxFragNum  Integer = (sMaxMsduLng / (sMinFragLng-sMacHdrLng-sWepAddLng-sCrcLng)) ;
      /* Number of bits in ACK and CTS control frames */
synonym sAckCtsLng  Integer = 112 ;
```

```
/*  STATION CONFIGURATION FLAGS {supplementary to MIB} -- names begin with "s"  */
      /* Protocol version number supported by this version of the MAC */
synonym sVersion  Integer = 0 ;     /* must be =0 for the current MAC */
      /* sCanBeAp =true if this station is capable of operating as an Access Point */
synonym sCanBeAp  Boolean = false ;     /* set to correct value at each kind of station */
      /* sCanBePc =true if this station is capable of serving as a Point Coordinator */
synonym sCanBePc  Boolean = false ;     /* set to correct value at each kind of station */
      /* sCfPollable =true if this sttation is capable of responding to polls from a point coordinator */
synonym sCfPollable  Boolean = true ;     /* set to correct value at each kind of station */
      /*
   * * * THE FOLLOWING IS A TEMPORARY DEFINITION * * *
         sBitTime is the time (in microseconds) needed to transfer 1 bit over the wireless medium
         at the minimum PHY data rate for the active (I)BSS.  sBitTime is defined for use
         in lieu of the minimum value in aBssBasicRateSet, which is no longer in the MIB. */
synonym sBitTime  Real = 1.0 ;     /* this value is for PHYs with 1Mbps minimum basic rate */
```

```
/*    DISCRETE, MICROSECOND-UNIT TIME AND DURATION SORTS     */
  /*
       SDL does not define the relationship between its concept of Time and physical time in the
       system being described.  An abstraction is needed to establish this relationship, because
       Time in SDL uses the semantics of Real, whereas time in the MAC protocol is discrete.  The
       MAC uses intervals with specific, integral relationships, distributes time synchronization
       information as a 64-bit count of 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!(r) ; ) ) ) ;
  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 ;
    kuTime  :  Time   ->  Kusec ;
    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)  ==>  kuTime(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 ; ) ) ;
  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 */
    "//"      :  String0, String0  ->  String0 ;  /* concatenation */
    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 (
      /* constructors are Emptystring, MkString, and "//"; equalities between constructor terms */
        s // Emptystring == s ;      Emptystring // s == s ;
        (s1 // s2) // s3 == s1 // (s2 // s3) ;
      /* definition of Length by applying it to all constructors */
        type String Length(Emptystring) == 0 ;
        type String Length(MkString(item0)) == 1 ;
        type String Length(s1 // s2) == Length(s1) + Length(s2) ;
      /* definition of Extract! by applying it to all constructors, Error! cases handled separately */
        Extract!(MkString(item0),0) == item0 ;
        i < Length(s1)          ==>  Extract!(s1 // s2,i) == Extract!(s1,i) ;
        i >= Length(s1)         ==>  Extract!(s1 // s2,i) == Extract!(s2,i-Length(s1)) ;
        i < 0 or i >= Length(s) ==>  Extract!(s,i) == Error! ;
      /* definition of First and Last by other operations */
        First(s) == Extract!(s,0) ;      Last(s) == Extract!(s,Length(s)-1) ;
      /* definition of substr(s,i,j) by induction on j, Error! cases handled separately */
        i >= 0 and i <= Length(s) ==> Substr(s,i,0) == Emptystring ;
        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 j < 0 or i+j > Length(s) ==> Substr(s,i,j) == Error! ;
      /* definition of Modify!, Head, Tail, and Sx by other operations */
        Modify!(s,i,item0) ==
            Substr(s,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(item0) // MkString(item1) // MkString(item2) // MkString(item3) //
            MkString(item4) // MkString(item5) // Mkstring(item6) // MkString(item7) ; ) ) ) ;
endgenerator String0 ;
```

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

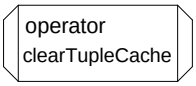
```
/*     ASN.1-style BITSTRING SORT  (derived from Z.105, Annex A)     */
  /* Bitstring is a 0-origin string of Bit.  Z.105 provides binary ('1011'B) and hexadecimal ('D3'H) literals,
     but parsing these requires relaxation of a Z.100 rule regarding the use of apostrophes.
     Therefore, this version of Bitstring provides hexadecimal literals 0x00:0xFF.  Bitstrings of
     non-octet length can be constructed by concatenating individual bits using MkString.
     Operators 'not', 'and', 'or', 'xor', and '=>' 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 ;
```

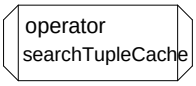
```
/*     OCTET and OCTETSTRING SORTS  (somewhat influenced by Z.105, Annex A)     */
  /* Z.105 defines Octet as   "syntype Octet = Bitstring constants size (8)  endsyntype Octet ;"
     Unfortunately, "size" is an extension to the abstract grammar of SDL, unavailable in Z.100.
     Therefore, Octet 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 > 255  ==>  mkOctet(i) == mkOctet(i mod 256) ;
      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(i1)) 0-extended to length i2 */
  axioms
    for all b,b1,b2 in Bitstring (      for all s in Octetstring (      for all o in Octet (
      Bit_String(null) == '' ;
      Bit_String( MkString(o) // s ) == o // Bit_String(s) ;
      Octet_String('') == null ;
      Length(b1) > 0, Length(b1) < 8
           ==>  Octet_String(b1) == MkString(b1 or 0x00) ;  /* expand b1 to 8 bits */
      b == b1 // b2, Length(b1) = 8
           ==>  Octet_String(b) == MkString(b1) // Octet_String(b2) ;
      for all i, k in Integer (
        k = 1  ==>  mkOS(i,k) == MkString(mkOctet(i)) ;
        k > 1  ==>  mkOS(i,k) == mkOS(i,k-1) // 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;
```

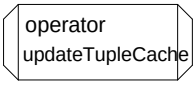
```
/*     MAC ADDRESS SORTS     */
  /* MacAddr is a subtype of Octetstring with added operators
       isGroup(m), which returns true if given a group address,
       isBcst(m), which returns true if given the broadcast address, and
       isLocal(m), which returns true if give a locally-administered 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 ;
      /* Broadcast Address */
synonym BcstAddr  MacAddr = S6(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF) ;
      /* Null Address (an all-zero, 6-Octet string for use as placeholder) */
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  */
syntype FragNum = Integer  constants 0:sMaxFragNum   endsyntype FragNum ;
      /* Range of possible sequence numbers  */
syntype SeqNum = Integer  constants 0:4095   endsyntype SeqNum ;
      /* Tuple strucure (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 */
syntype CacheIndex = Integer  constants 1:tupleCacheSize   endsyntype 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 ;
```

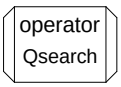
```
/*     32-BIT (unsigned) UP-COUNTER WITH WRAPAROUND     */
/* Used for MIB counters, inc(cntr) increments value by 1, with wraparound from (2^32)-1 to 0. */
newtype Counter32   inherits Integer   operators all ;
   adding operators
      inc  :  Counter32  ->  Counter32 ;
   axioms
      for all 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-1)) ;
      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) ;
      Qfirst(q,item0) == MkQueue(item0) // q ;    Qlast(q,item0) == q // MkQueue(item0) ;  )  )  ) ;
endgenerator Queue ;
```

Package macsorts

3112_b\Fragment(23)




```
/*     FRAGMENTATION DATA STRUCTURES     */
  /*
     Array to hold up to FragNum fragments of an Msdu or Mmpdu  */
newtype FragArray  Array(FragNum,Frame) ;   endnewtype FragArray ;
  /*
     The FragSdu structure is for OUTGOING MSDUs and MMPDUs (generically SDUs)
     Each SDU, even if not fragmented, is held in an instance of this structure, awaiting its
     (re)transmission attempt(s).  Transmit queue(s) are ordered lists of FragSdu instances.
     Depending on station capabilities and BSS type, there may be one or more such queues.  */
newtype FragSdu  Struct
  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 */
                        /* when fAnc>fCur, pdus(fCur)+ may be transmitted. */
  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 ;     /* destinaton 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
        index:= Qsearch(queue, addr)
     where queue is an SduQueue and index is returned to identify the first queue entry
     at which entry!dst = addr; of as =1 if no match found (or for and empty queue).  */
newtype SduQueue   Queue(FragSdu,emptyQ) ;
  adding operators
     Qsearch  :  SduQueue,MacAddr  ->  Integer ;
  Operator Qsearch ;
     fpar que  SduQueue, val  MacAddr ;   returns  result  Integer ;   referenced ;
endnewtype SduQueue ;
```

Package macsorts

operator
ArSearch

operator
ArFree

operator
ArAge

```
/*     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 ;     /* Mpdu 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 Mpdu */
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/ */
syntype reasmIndex = Integer  constants 1:reasmSize   endsyntype reasmIndex ;
   /*
      Array of PartialSdu for use reassembling fragmented Msdus and Mmpdus.
      Searchable using the AsSearch operator
         index:= ArSearch(array, addr, seq, frag)
      where array is a ReasmArray and index is returned to identify the first element
      for which (inUse=true) and (entry!rta=addr) and (entry!rsn=seq) and (entry!rCur=frag-1);
      or as =1 if no match found.
         index:= ArFree(array)   returns index of free entry, or -1 if none
         array:= ArAge(array, age)   frees entry!eol < age  (also used to clear array).  */
  */
newtype ReasmArray  array(reasmIndex, PartialSdu) ;
   adding operators
      ArSearch  :  ReasmArray,MacAddr,SeqNum,FragNum  ->  Integer ;
      ArFree    :  ReasmArray  ->  Integer ;
      ArAge     :  ReasmArray,Time  ->  ReasmArray ;
   Operator ArSearch ;   fpar ar  ReasmArray, adr  MacAddr, seq  SeqNum,
         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 ;
```

operator
addrToId

```
/*     SORTS for POWER MANAGEMENT SUPPORT     */
   /*
        define StationId  */
synonym sMaxSId  Integer = 2007 ;
syntype StationId = Integer   constants 0:sMaxSId   endsyntype StationId ;
   /*
        define size of StationId table (<= sMaxSId+1 )  */
synonym sSIdTableSize  Integer = 2008 ;
   /*   StationId table
        searchable by MacAddr
           index:= addrToId(table, addr)
        where table is a SIdTable, returns the first index vaue where
        the table entry is equal to addr, or -1 if no match found.  */
newtype SIdTable  Array(StationId,MacAddr);
   adding operators
      addrToId  :  SIdTable,MacAddr  ->  Integer ;
   operator addrToId ;   fpar tbl  SIdTable, val  MacAddr ;  returns Integer ;
endnewtype SIdTable ;
```

operator
crc32

```
/*     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. */
newtype Crc   inherits Octetstring   operators all ;
  adding operators
    crc32   :  Crc, Octet  ->  Crc ;
    mirror  :  Crc         ->  Octetstring ;
  operator crc32 ;
    fpar crcin  Crc,   val  Octet ;   returns Crc ;   referenced ;
  axioms   for all c in Crc (
    mirror(c) == S4(flip(c(3)), flip(c(2)), flip(c(1)), flip(c(0))) ;  ) ;
endnewtype Crc ;
  /* Initial Crc value (all 1s) */
synonym initCrc  Crc = S4(0xFF, 0xFF, 0xFF, 0xFF) ;
  /* Valid Crc value after accumulation of Crc32 on Pdu including Crc field */
synonym goodCrc  Crc = S4(0x7B, 0xDD, 0x04, 0xC7) ;
```

operator
keyLookup

```
/*     SORTS for WIRED-EQUIVALENT PRIVACY (WEP)     */
  /*   define length of KeyVector  and  nullKey value  */
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.
      Length(KeyVector(n)) must =5, it is assumed that the MlmeSet.request enforces this.  */
newtype KeyVector  Array(KeyIndex,PrngKey) ;   endnewtype KeyVector ;
  /*
      Number of entries in aWepKeyMapping array at this station.
      Actual length of key map array (10 is the minimum per 8.3.2).  */
synonym sWepKeyMappingLength  Integer = 10 ;
syntype KeyMappingRange = Integer  constants 1:sWepKeyMappingLength
  endsyntype KeyMappingRange ;
  /*
      KeyMap structure -- used as elements of KeyMapArray
      Length(wepKey) must =5, it is assumed that the MlmeSet.request enforces this.  */
newtype KeyMap  struct
  mappedAddr  MacAddr ;   keyOn  Boolean ;   wepKey  PrngKey ;   endnewtype KeyMap ;
  /*
      KeyMapArray -- used for aWepKeyMapping table; an array of KeyMap indexed
      by KeyMappingRange.  Procedural operator keyLookup searches the array for the
      mapping of a given address:
           keyMap := keyLookup(addr, keyMapArray, keyMapArrayLength)
           If an entry is found with mappedAddr=addr, keyMap is set to the value of this entry.
           If no entry is found with mappedAddr=addr, keyMap is set to (. addr, false, null .)  */
newtype KeyMapArray  Array(KeyMappingRange, KeyMap) ;
  adding operators
    keyLookup  :  MacAddr, KeyMapArray,  Integer  ->  KeyMap ;
  operator keyLookup ;
    fpar luadr  MacAddr,   kma  KeyMapArray,   kml  Integer ;   returns KeyMap ;   referenced ;
endnewtype KeyMapArray ;
```

```
/*     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) */
     ftype        :  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  ->  Bit ;           /* 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 ;
  /*
     axioms on next page  */
```

operator
getElem

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

```
/* 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 */
  axioms for all f in Frame (   for all a, sa, da, ra, ta, bssa in MacAddr (
    for all body, dur, sid, info in Octetstring (
      addr1(f)  ==  substr(f,4,6) ;      setAddr1(f,a)  ==  substr(f,0,4) // a // substr(f,10,length(f)-10) ;
      addr2(f)  ==  substr(f,10,6) ;      setAddr2(f,a)  ==  substr(f,0,10) // a // substr(f,16,length(f)-16) ;
      addr3(f)  ==  substr(f,16,6) ;      setAddr3(f,a)  ==  substr(f,0,16) // a // substr(f,22,length(f)-22) ;
      addr4(f)  ==  substr(f,24,6) ;      insAddr4(f,a)  ==  substr(f,0,24) // a // substr(f,24,length(f)-24) ;
      curApAddr(f)  ==  substr(f,28,6) ;
      for all ft in TypeSubtype (
        mkFrame(ft,da,bssa,body)  ==  ft // O3 // da // aMacAddress // bssa // O2 // body ;
        (ft = rts)  ==>  mkCtl(ft,dur,ra) == ft // O1 // dur // ra // aMacAddress ;
        (ft = ps_poll)  ==>  mkCtl(ft,sid,bssa) == ft // O1 // sid // bssa // aMacAddress ;
        (ft = cts) or (ft = ack)  ==>  mkCtl(ft,dur,ra) == ft // O1 // dur // ra ;
        (ft = cfend) or (ft = cfend_ack)  ==>  mkCtl(ft,bssa,ra) == ft // O3 // ra // bssa ;
        ftype(f)  ==  mkstring(f(0) and 0xFC) ;
        setFtype(f,ft)  ==  Modify!(f,0,mkstring((f(0) and 0x03) or ft)) ;  ) ;
      for all bt in BasicType (
        basetype(f)  ==  f(0) and 0x0C ;  ) ;
      for all i in Integer (
        protocolVer(f)  ==  octetVal(f(0) and 0x03) ;
        durId(f)  ==  octetVal(f(2)) + (octetVal(f(3)) * 256) ;
        setDurId(f,i)  ==  substr(f,0,2) // mkOS(i mod 256,1) // mkOS(i / 256,1) // substr(f,4,length(f)-4) ;
        authSeqNum(f)  ==  octetVal(f(26)) + (octetVal(f(27)) * 256) ; ) ;
      for all e in ElementId (
        mkElem(e,info)  ==  e // mkOS(length(info)+2 // info  ) ;
      for all b in Bit (
        toDs(f)  ==  if (f(1) and 0x01) then  1  else  0  fi ;
        setToDs(f,b)  ==  Modify!(f,1,(f(1) and 0xFE) or S8(0,0,0,0,0,0,0,b)) ;
        frDs(f)  ==  if (f(1) and 0x02) then  1  else  0  fi ;
        setFrDs(f,b)  ==  Modify!(f,1,(f(1) and 0xFD) or S8(0,0,0,0,0,0,b,0)) ;
        moreFrag(f)  ==  if (f(1) and 0x04) then  1  else  0  fi ;
        setMoreFrag(f,b)  ==  Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,0,0,b,0,0)) ;
        retryBit(f)  ==  if (f(1) and 0x08) then  1  else  0  fi ;
        setRetryBit(f,b)  ==  Modify!(f,1,(f(1) and 0xF7) or S8(0,0,0,0,b,0,0,0)) ;
        pwrMgt(f)  ==  if (f(1) and 0x10) then  1  else  0  fi ;

  /* axioms continue on next page */
```

```
/* axioms for Frame sort, continued */
        setPwrMgt(f,b)  ==  Modify!(f,1,(f(1) and 0xFB) or S8(0,0,0,b,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,b,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,b,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(b,0,0,0,0,0,0,0)) ;
        for all c in Capability (
          capA(f,c)  ==  if (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  O2  fi) ) // substr(f,26,length(f)-26) ;
          capB(f,c)  ==  if (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  O2  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 0x0F) ;
        setFrag(f,fr)  ==  substr(f,0,22) // mkstring((f(22) and 0xF0) or mkOctet(fr)) //
            substr(f,23,length(f)-23) ;  ) ;
      for all tm in Time (
        ts(f)  ==  float (octetVal(f(24)) + (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)/256), 1) //
            mkOS(((fix(tm)/4294967296)/65536), 1) //
            mkOS(((fix(tm)/4294967296)/16777216), 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. */
newtype TypeSubtype   inherits Octetstring   operators all ;
  adding literals
     asoc_req,     asoc_rsp,     reasoc_req,    reasoc_rsp,    probe_req,
     probe_rsp,    beacon,       atim,          disasoc,       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(0,0,0,0,0,0,0,0)) ;     asoc_rsp     ==  mkstring(S8(0,0,0,0,1,0,0,0)) ;
     reasoc_req   ==  mkstring(S8(0,0,0,0,0,1,0,0)) ;     reasoc_rsp   ==  mkstring(S8(0,0,0,0,1,1,0,0)) ;
     probe_req    ==  mkstring(S8(0,0,0,0,0,0,1,0)) ;     probe_rsp    ==  mkstring(S8(0,0,0,0,1,0,1,0)) ;
     beacon       ==  mkstring(S8(0,0,0,0,0,0,0,1)) ;     atim         ==  mkstring(S8(0,0,0,0,1,0,0,1)) ;
     disasoc      ==  mkstring(S8(0,0,0,0,0,1,0,1)) ;     auth         ==  mkstring(S8(0,0,0,0,1,1,0,1)) ;
     deauth       ==  mkstring(S8(0,0,0,0,0,0,1,1)) ;     ps_poll      ==  mkstring(S8(0,0,1,0,0,1,0,1)) ;
     rts          ==  mkstring(S8(0,0,1,0,1,1,0,1)) ;     cts          ==  mkstring(S8(0,0,1,0,0,0,1,1)) ;
     ack          ==  mkstring(S8(0,0,1,0,1,0,1,1)) ;     cfend        ==  mkstring(S8(0,0,1,0,0,1,1,1)) ;
     cfend_ack    ==  mkstring(S8(0,0,1,0,1,1,1,1)) ;     data         ==  mkstring(S8(0,0,0,1,0,0,0,0)) ;
     data_ack     ==  mkstring(S8(0,0,0,1,1,0,0,0)) ;     data_poll    ==  mkstring(S8(0,0,0,1,0,1,0,0)) ;
     data_poll_ack ==  mkstring(S8(0,0,0,1,1,1,0,0)) ;    null_frame   ==  mkstring(S8(0,0,0,1,0,0,1,0)) ;
     cfack        ==  mkstring(S8(0,0,0,1,1,0,1,0)) ;     cfpoll       ==  mkstring(S8(0,0,0,1,0,1,1,0)) ;
     cfpoll_ack   ==  mkstring(S8(0,0,0,1,1,1,1,0)) ;
endnewtype TypeSubtype ;
  /*
     BasicTypes defines the 2-bit frame type groups */
newtype 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    */
newtype 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,   asoc_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) ;   asoc_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 ;
```

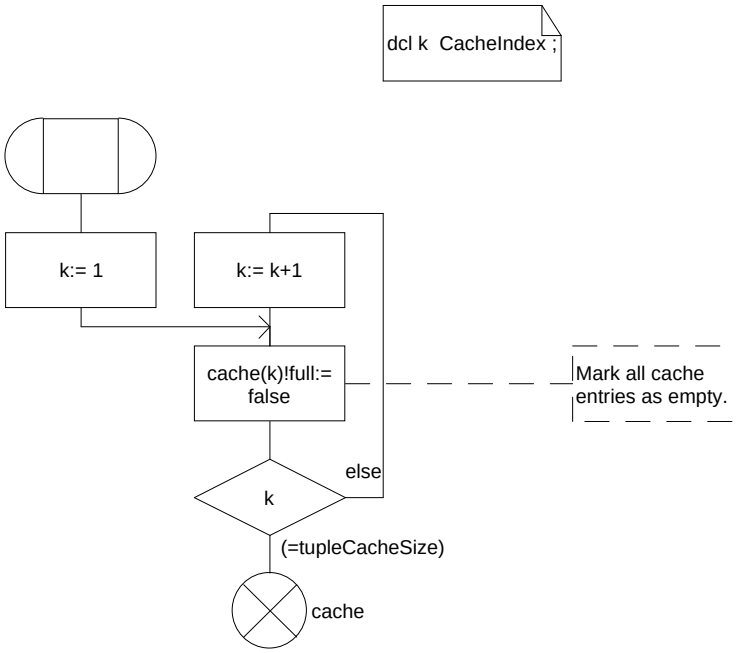
```
/*   CAPABILITY FIELD BITS     */
newtype Capability   inherits Bitstring   operators all ;
  adding literals
     cEss,   cIbss,   cPollable,   cPollReq,   cPrivacy ;
  axioms
     cEss       ==  S8(1,0,0,0,0,0,0,0) // 0x00 ;   /* ESS capability */
     cIbss      ==  S8(0,1,0,0,0,0,0,0) // 0x00 ;   /* IBSS capability */
     cPollable  ==  S8(0,0,1,0,0,0,0,0) // 0x00 ;   /* CF-pollable (sta), PC present (ap) */
     cPollReq   ==  S8(0,0,0,1,0,0,0,0) // 0x00 ;   /* {not} CF poll req (sta), PC polls (ap) */
     cPrivacy   ==  S8(0,0,0,0,1,0,0,0) // 0x00 ;   /* WEP required capability */
endnewtype Capability ;
```

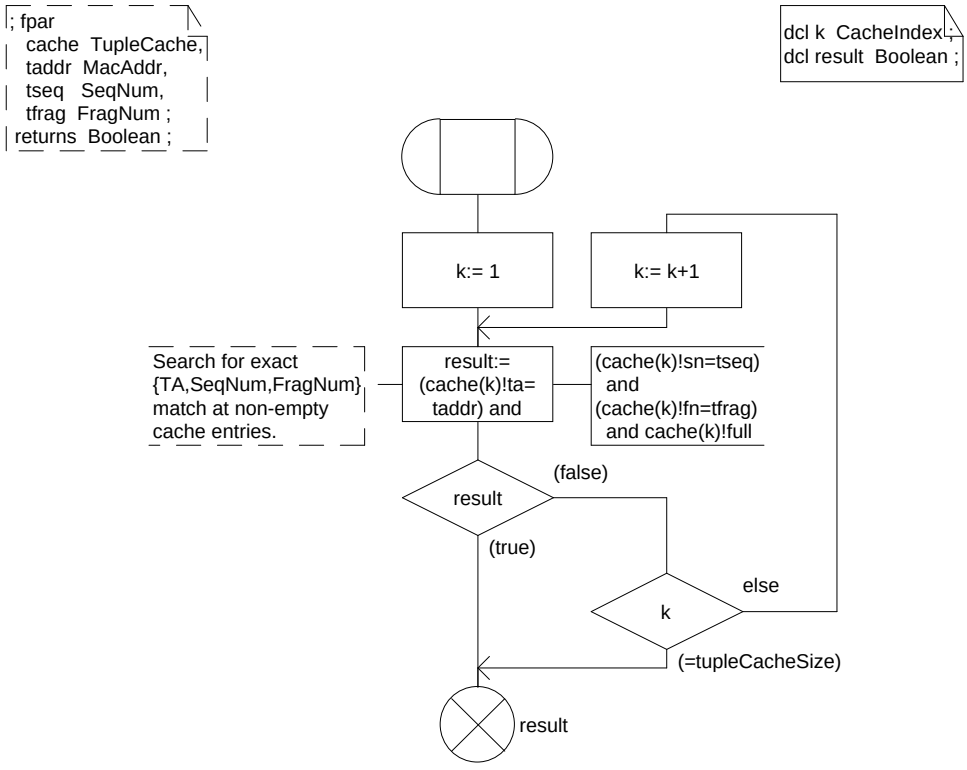
```
/*     IBSS PARAMETER SET ELEMENT     */
newtype IbssParms  inherits Octetstring   operators all ;
  adding operators
     atimWin     :  IbssParms  ->  Kusec ;
     setAtimWin  :  IbssParms, Kusec  ->  IbssParms ;
  axioms
     for all ib in IbssParms (  for all u in Kusec (
        atimWin(ib)  ==  octetVal(ib(0)) + (octetVal(ib(1)) * 256 ) ;
        setAtimWin(ib,u)  ==  mkOS(u mod 256, 1) // mkOS(u / 256, 1) ;  ) ) ;
endnewtype IbssParms ;
```

```
/*     CF PARAMETER SET ELEMENT     */
newtype CfParms   inherits Octetstring   operators all ;
  adding operators
     cfpCount      :  CfParms  ->  Integer ;   /* 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 ;
```

```
/*     FH PARAMETER SET ELEMENT     */
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 ;
```

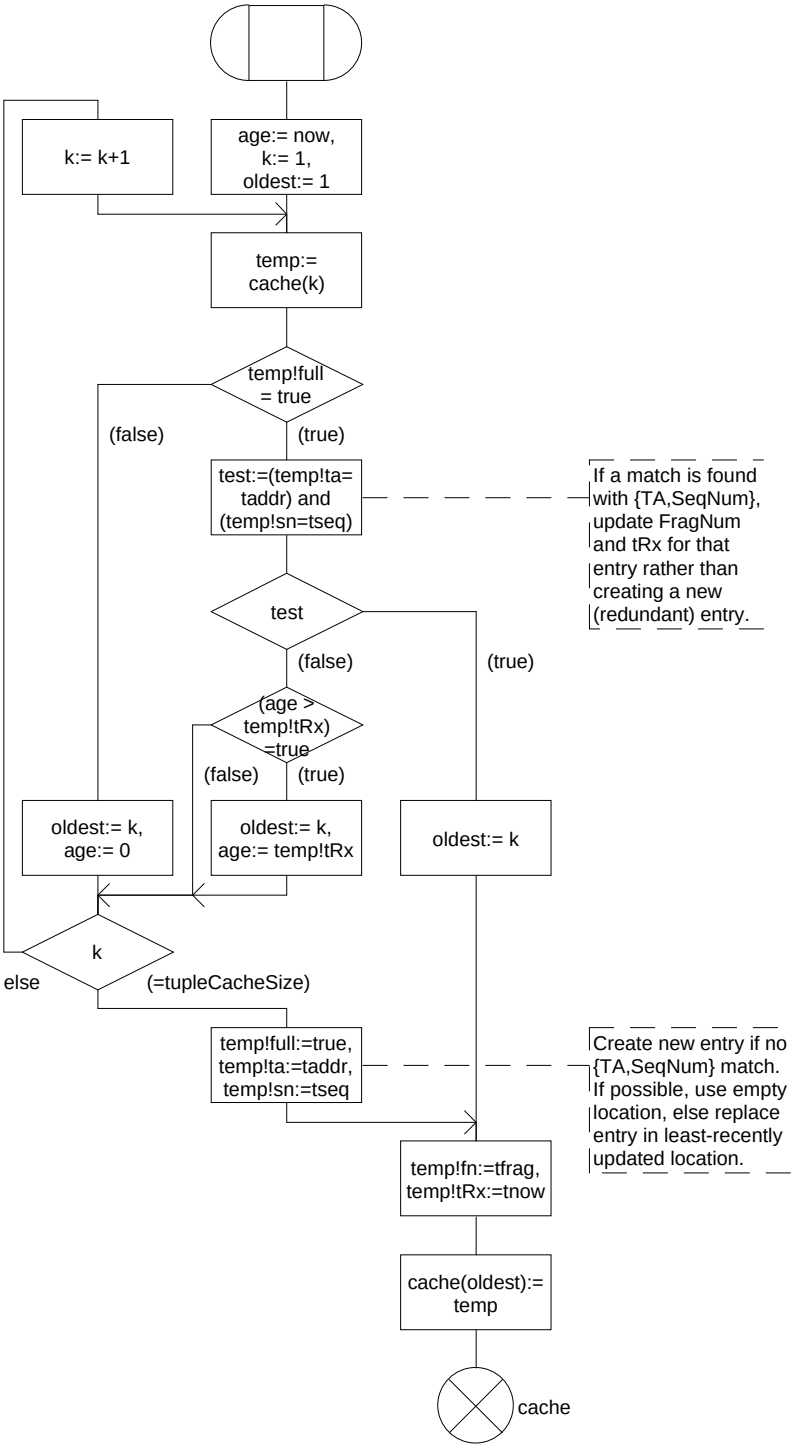
```
; fpar
   cache  TupleCache ;
returns TupleCache ;
```



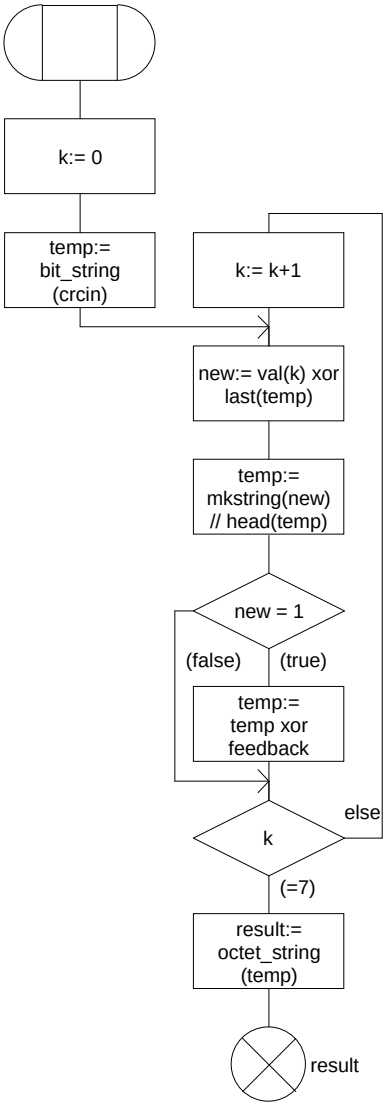

; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum ;
returns Boolean ;
dcl k CacheIndex ;
dcl result Boolean ;
k:= 1
k:= k+1
Search for exact {TA,SeqNum,FragNum} match at non-empty cache entries.
result:= (cache(k)!ta= taddr) and
(cache(k)!sn=tseq) and (cache(k)!fn=tfrag) and cache(k)!full
result
(false)
(true)
k
else
(=tupleCacheSize)
result

; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum,
tnow Time ;
returns TupleCache ;

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



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

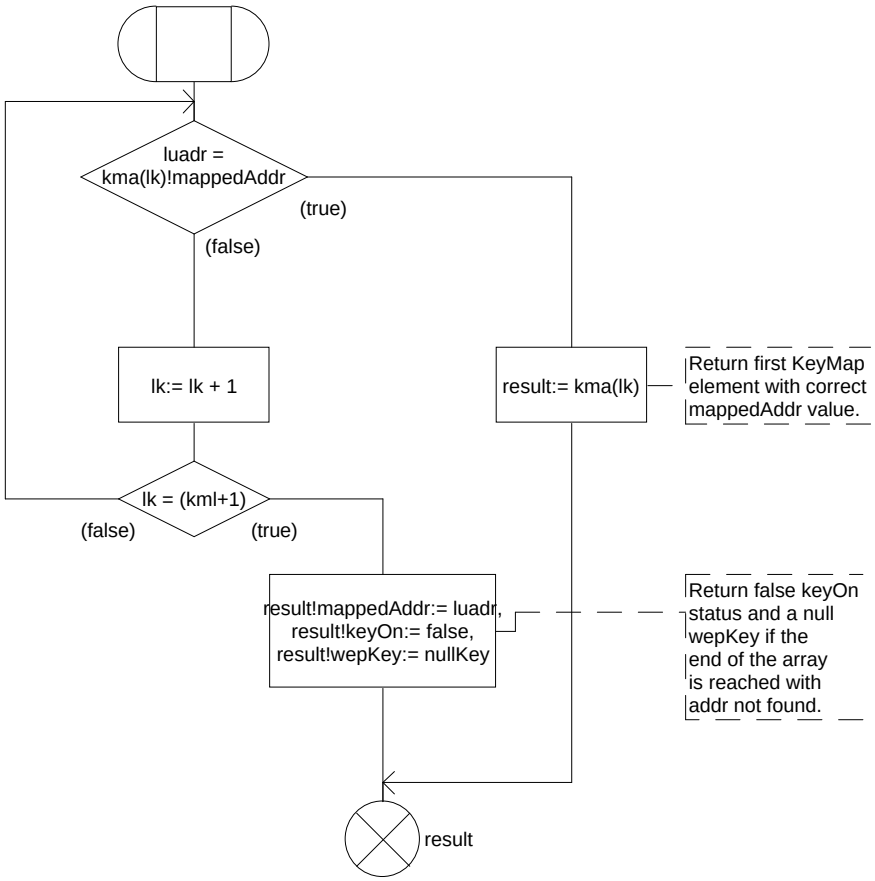


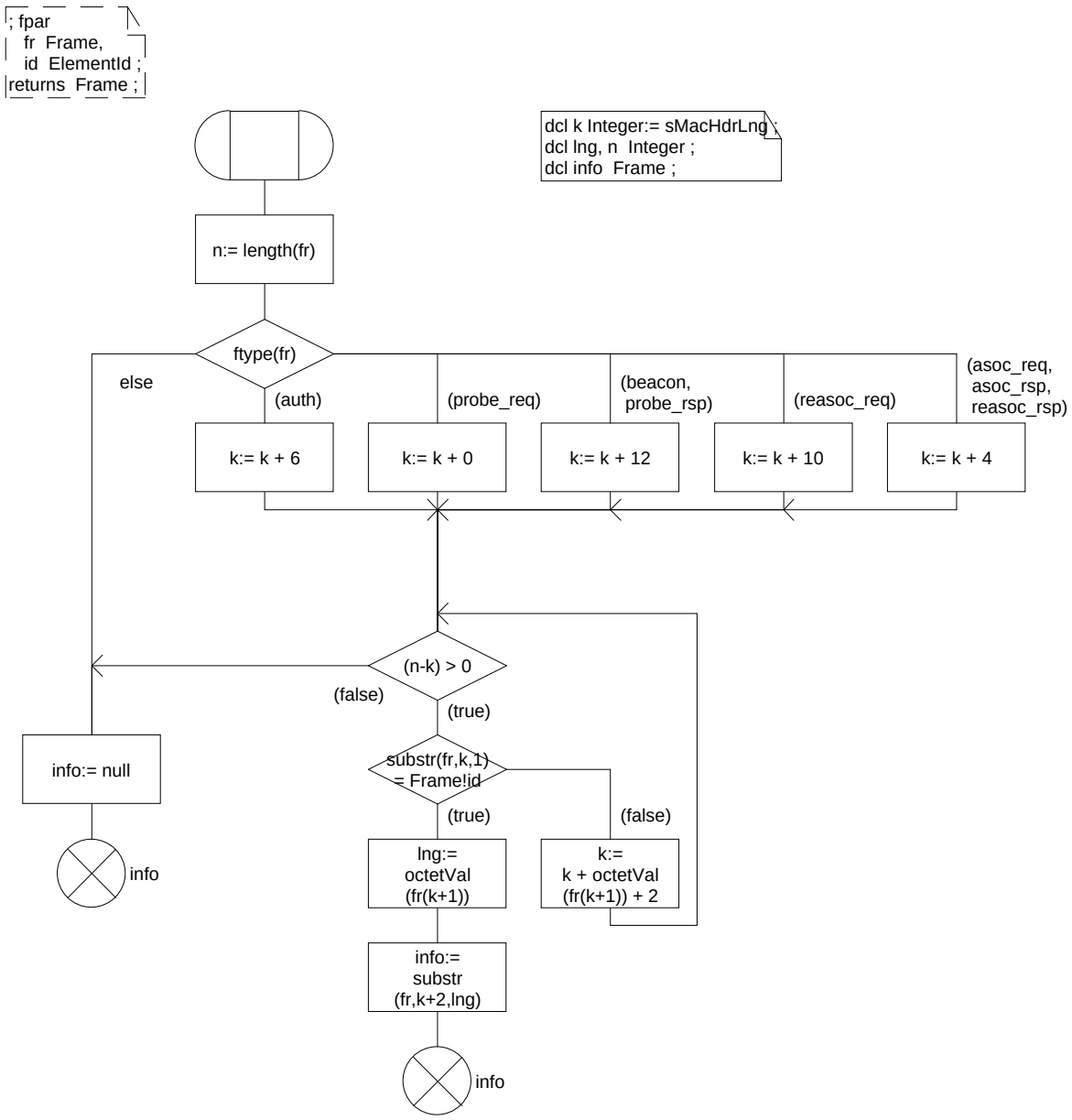

```
dcl k  Integer ;
dcl new  Bit ;
dcl result  Crc ;
dcl temp  Bitstring ;

/* 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) ;
```

```
; fpar luadr  MacAddr,
  kma  KeyMapArray,
  kml  Integer ;
returns KeyMap ;
```

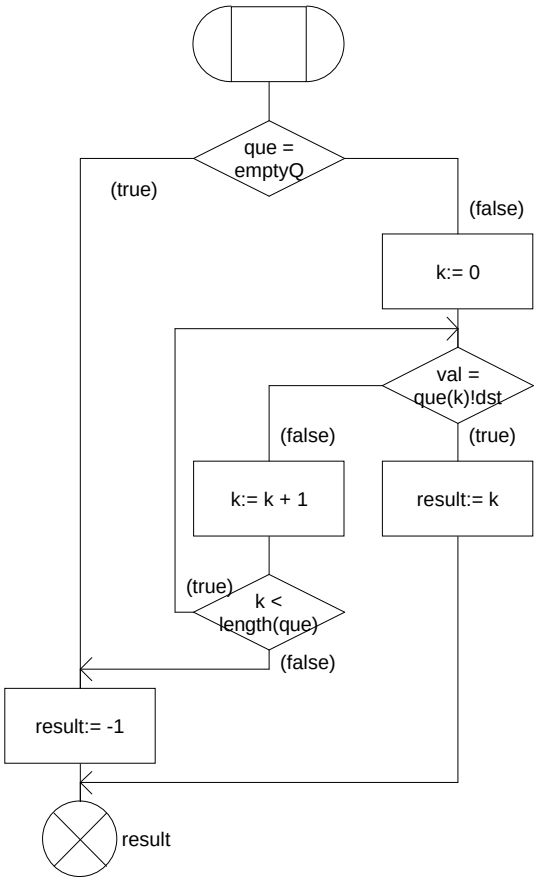
```
dcl lk  Integer := 1 ;
dcl result  KeyMap ;
```




; fpar
fr Frame,
id ElementId ;
returns Frame ;
dcl k Integer:= sMacHdrLng ;
dcl lng, n Integer ;
dcl info Frame ;
n:= length(fr)
ftype(fr)
else
(auth)
(probe_req)
(beacon, probe_rsp)
(reasoc_req)
(asoc_req, asoc_rsp, reasoc_rsp)
k:= k + 6
k:= k + 0
k:= k + 12
k:= k + 10
k:= k + 4
(n-k) > 0
(false)
(true)
info:= null
substr(fr,k,1) = Frame!id
(true)
(false)
info
lng:= octetVal (fr(k+1))
k:= k + octetVal (fr(k+1)) + 2
info:= substr (fr,k+2,lng)
info

```
; fpar
 que  SduQueue,
 val  MacAddr ;
returns  result  Integer ;
```

```
dcl k  Integer ;
```



```
use macsorts ;
```

Package macmib 3201_b\StationConfig(7)

```
/* THIS SDL-92 RENDITION OF THE MAC MIB, PLUS PORTIONS OF THE PHY MIB, EXISTS
   TO PERMIT ANALYSIS OF THE MAC STATE MACHINES WITHOUT HAVING FULL Z.105
   SUPPORT IN THE SDL TOOL.  HOWEVER, SOME OF THE COMMENTS INTERSPERSED
   WITH THE REMOTE VARIABLE DEFINITIONS ARE USEFUL AS EXPLANATIONS OF THE
   RELATIONSHIPS BETWEEN MIB ATTRIBUTES AND STATE MACHINE FUNCTIONS.
   ALSO, THERE ARE CASES WHERE ATTRIBUTES IN THIS RENDITION DIFFER FROM THE
   ASN.1, AND THE REASONS FOR THE DIFFERENCES ARE EXPLAINED IN COMMENTS.  */
```

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

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

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

```
use macsorts ;
```

Package macmib 3202_b\Privacy(7)

```
/* 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.  */
```

```
/*       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  */
```

```
use macsorts ;
```

Package macmib 3203_b\MacOperation(7)

```
/* 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 Mpdu 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 retansmission attempts for an Mpdu 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 retansmission attempts for an Mpdu 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 Mpdu 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 entires 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  */
```

use macsorts ;

Package macmib

/* THIS SDL-92 RENDITION OF THE MAC MIB, PLUS PORTIONS OF THE PHY MIB, EXISTS TO PERMIT ANALYSIS OF THE MAC STATE MACHINES WITHOUT HAVING FULL Z.105 SUPPORT IN THE SDL TOOL. HOWEVER, SOME OF THE COMMENTS INTERSPERSED WITH THE REMOTE VARIABLE DEFINITIONS ARE USEFUL AS EXPLANATIONS OF THE RELATIONSHIPS BETWEEN MIB ATTRIBUTES AND STATE MACHINE FUNCTIONS. ALSO, THERE ARE CASES WHERE DATA TYPES IN THIS RENDITION DIFFER FROM THE ASN.1, AND THE REASONS FOR THE DIFFERENCES ARE EXPLAINED IN COMMENTS. */

```
/*      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  */
```

use macsorts ;

Package macmib 3205_b\PhyOperation1(7)

/* 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 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 suporting.
          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 + aTxRampOnTime + aTxRfDelay) ;
  /*
      Nominal time in microseconds for PLCP to deliver a symbol from the MAC interface to the PMD.
          read-only;   default={1 for FH, impl. dep. for DS, 60 | 40 for IR};   used in aRxTxTurnaroundTime eqn;  */
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 for FH, impl. dep. for DS, 0 for IR};   used in aRxTxTurnaroundTime equation;  */
synonym aTxRampOnTime  Usec = 8 ;   /* this value is for the FH PHY */
  /*
      Nominal time in microseconds between issuance of PmdData.request and the start of the corresponding
      symbol at the air interface.  Start of a symbol is defined to be 1/2 symbol period prior to center of the
      symbol for FH, 1/2 chip period prior to the center of the first chip of the symbol for DS, or 1/2 PPM-slot
      time prior to the center of the corresponding PPM-slot for IR.
          read-only;   default={1 for FH, impl. dep. for DS, 1 for IR};   used in aRxTxTurnaroundTime equation;  */
synonym aTxRfDelay  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 . . .  */
```

use macsorts ;

Package macmib

/* THIS SDL-92 RENDITION OF THE MAC MIB, PLUS PORTIONS OF THE PHY MIB, EXISTS TO PERMIT ANALYSIS OF THE MAC STATE MACHINES WITHOUT HAVING FULL Z.105 SUPPORT IN THE SDL TOOL. HOWEVER, SOME OF THE COMMENTS INTERSPERSED WITH THE REMOTE VARIABLE DEFINITIONS ARE USEFUL AS EXPLANATIONS OF THE RELATIONSHIPS BETWEEN MIB ATTRIBUTES AND STATE MACHINE FUNCTIONS. ALSO, THERE ARE CASES WHERE DATA TYPES IN THIS RENDITION DIFFER FROM THE ASN.1, AND THE REASONS FOR THE DIFFERENCES ARE EXPLAINED IN COMMENTS. */

```
/*       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 transmision 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.  */
```

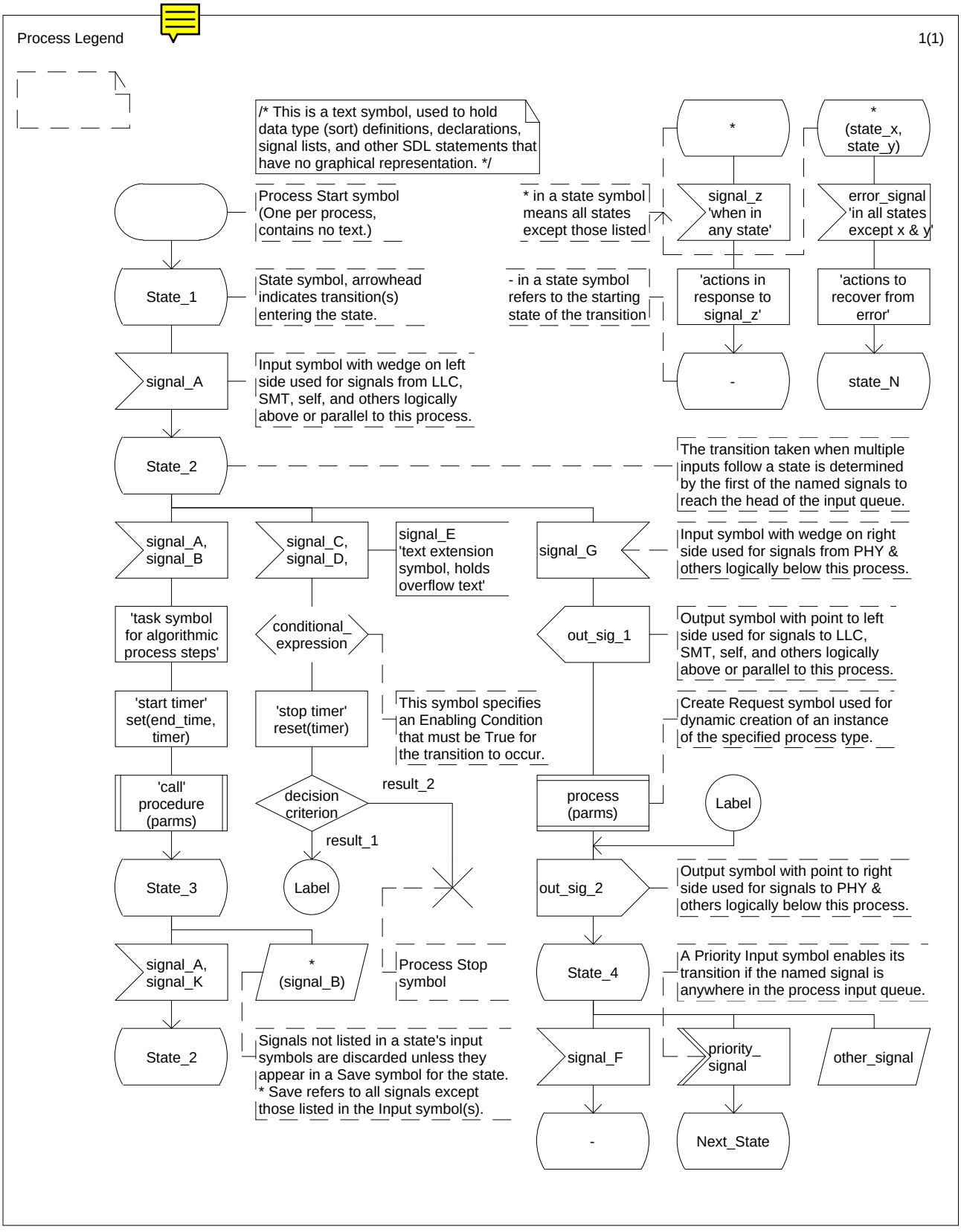
```
use macsorts ;
```

Package macmib 3207_b\PhyRate(7)

```
/* 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 specfied};   used in <not used>  */
synonym aPrefMaxMpduFragmentLength  Integer = aMpduMaxLength ;
/*  end of PhyRateGroup  */
```

```
/*       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_1
State symbol, arrowhead indicates transition(s) entering the state.
signal_A
Input symbol with wedge on left side used for signals from LLC, SMT, self, and others logically above or parallel to this process.
State_2
signal_A, signal_B
signal_C, signal_D,
signal_E 'text extension symbol, holds overflow text'
'task symbol for algorithmic process steps'
conditional_ expression
'start timer' set(end_time, timer)
'stop timer' reset(timer)
This symbol specifies an Enabling Condition that must be True for the transition to occur.
'call' procedure (parms)
decision criterion
result_2
result_1
State_3
Label
signal_A, signal_K
* (signal_B)
Process Stop symbol
State_2
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).
* in a state symbol means all states except those listed
- in a state symbol refers to the starting state of the transition
*
* (state_x, state_y)
signal_z 'when in any state'
error_signal 'in all states except x & y'
'actions in response to signal_z'
'actions to recover from error'
-
state_N
The transition taken when multiple inputs follow a state is determined by the first of the named signals to reach the head of the input queue.
signal_G
Input symbol with wedge on right side used for signals from PHY & others logically below this process.
out_sig_1
Output symbol with point to left side used for signals to LLC, SMT, self, and others logically above or parallel to this process.
Create Request symbol used for dynamic creation of an instance of the specified process type.
process (parms)
Label
out_sig_2
Output symbol with point to right side used for signals to PHY & others logically below this process.
State_4
A Priority Input symbol enables its transition if the named signal is anywhere in the process input queue.
signal_F
priority_ signal
other_signal
-
Next_State