Table 40—Requirements in Spain (values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
47	2.447	56	2.456	65	2.465
48	2.448	57	2.457	66	2.466
49	2.449	58	2.458	67	2.467
50	2.450	59	2.459	68	2.468
51	2.451	60	2.460	69	2.469
52	2.452	61	2.461	70	2.470
53	2.453	62	2.462	71	2.471
54	2.454	63	2.463	72	2.472
55	2.455	64	2.464	73	2.473

Table 41—Requirements in France (values specified in GHz)

Channel #	Value	Channel #	Value	Channel #	Value
48	2.448	60	2.460	72	2.472
49	2.449	61	2.461	73	2.473
50	2.450	62	2.462	74	2.474
51	2.451	63	2.463	75	2.475
52	2.452	64	2.464	76	2.476
53	2.453	65	2.465	77	2.477
54	2.454	66	2.466	78	2.478
55	2.455	67	2.467	79	2.479
56	2.456	68	2.468	80	2.480
57	2.457	69	2.469	81	2.481
58	2.458	70	2.470	82	2.482
59	2.459	71	2.471	—	—

14.6.6 Occupied channel bandwidth

Occupied channel bandwidth shall meet all applicable local geographic regulations for 1 MHz channel spacing. The rate at which the PMD entity will hop is governed by the MAC. The hop rate is an attribute with a maximum dwell time subject to local geographic regulations.

14.6.7 Minimum hop rate

The minimum hop rate shall be governed by the regulatory authorities.

14.6.8 Hop sequences

The hopping sequence of an individual PMD entity is used to create a pseudorandom hopping pattern utilizing uniformly the designated frequency band. Sets of hopping sequences are used to co-locate multiple PMD entities in similar networks in the same geographic area and to enhance the overall efficiency and throughput capacity of each individual network.

An FH pattern, F_x, consists of a permutation of all frequency channels defined in Table 38 and Table 39. For a given pattern number, x, the hopping sequence can be written as follows:

$$F_x = \{f_x(1), f_x(2), \ldots f_x(p)\}$$

where

$f_x(i)$ is the channel number (as defined in 14.6.4) for i^{th} frequency in x^{th} hopping pattern;
p is the number of frequency channels in hopping pattern (79 for North America and most of Europe, 23 for Japan, 27 for France, 35 for Spain)

Given the hopping pattern number, x, and the index for the next frequency, i (in the range 1 to p), the channel number shall be defined to be as follows:

$f_x(I) = [b(i) + x] \bmod (79) + 2$ in North America and most of Europe, with $b(i)$ defined in Table 42.
$= [(i - 1) \times x] \bmod (23) + 73$ in Japan.
$= [b(i) + x] \bmod (27) + 47$ in Spain with $b(i)$ defined in Table 43.
$= [b(i) + x] \bmod (35) + 48$ in France with $b(i)$ defined in Table 44.

Table 42—Base-Hopping sequence b(i) for North America and most of Europe

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	0	11	76	21	18	31	34	41	14	51	20	61	48	71	55
2	23	12	29	22	11	32	66	42	57	52	73	62	15	72	35
3	62	13	59	23	36	33	7	43	41	53	64	63	5	73	53
4	8	14	22	24	72	34	68	44	74	54	39	64	17	74	24
5	43	15	52	25	54	35	75	45	32	55	13	65	6	75	44
6	16	16	63	26	69	36	4	46	70	56	33	66	67	76	51
7	71	17	26	27	21	37	60	47	9	57	65	67	49	77	38
8	47	18	77	28	3	38	27	48	58	58	50	68	40	78	30
9	19	19	31	29	37	39	12	49	78	59	56	69	1	79	46
10	61	20	2	30	10	40	25	50	45	60	42	70	28	—	—

Table 43—Base-Hopping sequence b(i) for Spain

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	13	10	19	19	14
2	4	11	8	20	1
3	24	12	23	21	20
4	18	13	15	22	7
5	5	14	22	23	16
6	12	15	9	24	2
7	3	16	21	25	11
8	10	17	0	26	17
9	25	18	6	27	26

Table 44—Base-Hopping sequence b(i) for France

i	*b(i)*	*i*	*b(i)*	*i*	*b(i)*
1	17	13	31	25	15
2	5	14	20	26	3
3	18	15	29	27	11
4	32	16	22	28	30
5	23	17	12	29	24
6	7	18	6	30	9
7	16	19	28	31	27
8	4	20	14	32	19
9	13	21	25	33	2
10	33	22	0	34	21
11	26	23	8	35	34
12	10	24	1	—	—

The sequences are designed to ensure some minimum distance in frequency between contiguous hops. The minimum hop size is 6 MHz for North America and Europe, including Spain and France, and 5 MHz for Japan.

The hopping pattern numbers x are divided into three sets. The sets are designed to avoid prolonged collision periods between different hopping sequences in a set. Hopping sequence sets contain 26 sequences for North America and Europe, and 4 sequences per set for Japan:

For North America and most of Europe:

$x = \{0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,51,54,57,60,63,66,69,72,75\}$ Set 1
$x = \{1,4,7,10,13,16,19,22,25,28,31,34,37,40,43,46,49,52,55,58,61,64,67,70,73,76\}$ Set 2

x = {2,5,8,11,14,17,20,23,26,29,32,35,38,41,44,47,50,53,56,59,62,65,68,72,74,77} Set 3

For Japan:

x = {6,9,12,15} Set 1
x = {7,10,13,16} Set 2
x = {8,11,14,17} Set 3

For Spain:

x = {0,3,6,9,12,15,18,21,24} Set 1
x = {1,4,7,10,13,16,19,22,25} Set 2
x = {2,5,8,11,14,17,20,23,26} Set 3

For France:

x = {0,3,6,9,12,15,18,21,24,27,30} Set 1
x = {1,4,7,10,13,16,19,22,25,28,31} Set 2
x = {2,5,8,11,14,17,20,23,26,29,32} Set 3

The three sets of hopping sequences for North America and most of Europe, of 26 patterns each, are listed Tables B.1, B.2, and B.3 in Annex B. Similarly, there are three sets for Japan of four patterns each. The three sets for Spain have nine patterns each. The three sets for France have 11 patterns each. The channel numbers listed under each pattern refer to the actual frequency values listed in Table 38 and Table 39.

14.6.9 Unwanted emissions

Conformant PMD implementations of this FHSS standard shall limit the emissions that fall outside of the operating frequency range, defined in Table 36 of 14.6.3, to the geographically applicable limits**.**

14.6.10 Modulation

The minimum set of requirements for a PMD to be compliant with the IEEE 802.11 FHSS PHY shall be as follows.

The PMD shall be capable of operating using two-level Gaussian frequency shift key (GFSK) modulation with a nominal bandwidth bit-period (BT)=0.5. The PMD shall accept symbols from the set {{1},{0}}from the PLCP. The symbol {1} shall be encoded with a peak deviation of ($+f_d$), giving a peak transmit frequency of (F_c+f_d), which is greater than the carrier center frequency (F_c). The symbol {0} shall be encoded with a peak frequency deviation of ($-f_d$), giving a peak transmit frequency of (F_c-f_d).

An incoming bit stream at 1 Mbit/s will be converted to symbols at $Fclk$ = 1 Msymbols/s, as shown in Table 45.

Table 45—Symbol encoding into carrier deviation (1 Mbit/s, 2-GFSK)

Symbol	Carrier deviation
1	1/2 × h2 × Fclk
0	–1/2 × h2 × Fclk
NOTE—These deviation values are measured using the center symbol of 7 consecutive symbols of the same value. The instantaneous deviation will vary due to Gaussian pulse shaping.	

The deviation factor h2 for 2GFSK (measured as difference between frequencies measured in the middle of 0000 and 1111 patterns encountered in the SFD, divided by 1 MHz) will nominally be 0.32.

The minimum frequency deviation, as shown in Figure 83, shall be greater than 110 kHz relative to the nominal center frequency F_c. F_d is the average center frequency of the last 8 bits of the Preamble Sync field, measured as the deviation at the midsymbol. Midsymbol is defined as the point that is midway between the zero crossings derived from a best fit to the last 8 bits of the Sync field. Maximum deviation is not specified, but modulation is subject to the occupied bandwidth limits of 14.6.5.

The zero crossing error shall be less than ±1/8 of a symbol period. The zero crossing error is the time difference between the ideal symbol periods and measured crossings of F_c. This is illustrated in Figure 83.

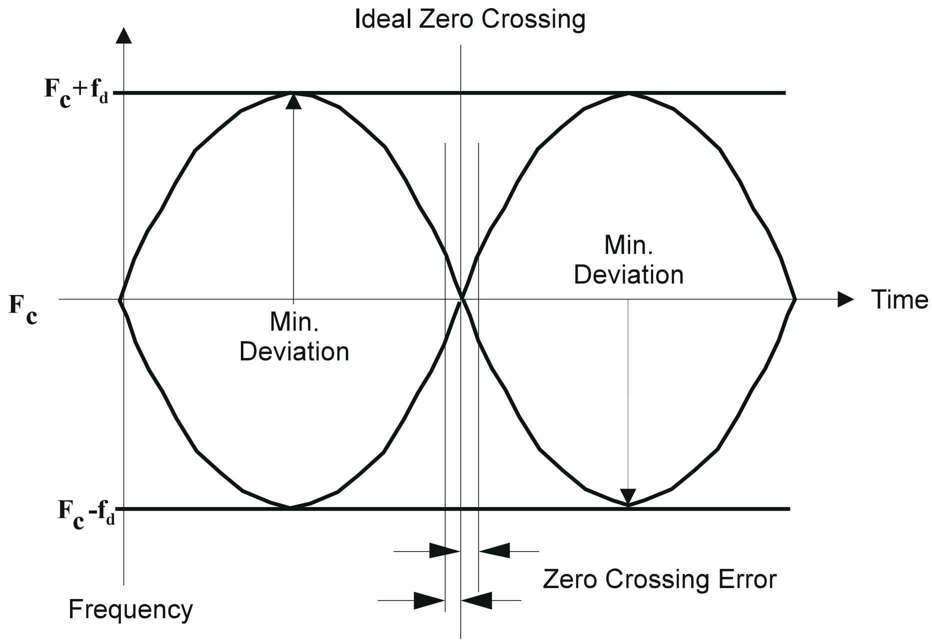


Figure 83—Transmit modulation mask

14.6.11 Channel data rate

A compliant IEEE 802.11 FHSS PMD shall be capable of transmitting and receiving at a nominal data rate of 1.0 Mbit/s ± 50 ppm.

14.6.12 Channel switching/settling time

The time to change from one operating channel frequency, as specified in 14.6.3, is defined as 224 µs. A conformant PMD meets this switching time specification when the operating channel center frequency has settled to within ±60 kHz of the nominal channel center frequency as outlined in 14.6.3.

14.6.13 Receive to transmit switch time

The maximum time for a conformant PMD to switch the radio from the receive state to the transmit state and place the start of the first bit on the air shall be 19 µs. At the end of this 19 µs, the RF carrier shall be within the nominal transmit power level range, and within the described modulation specifications.

14.6.14 PMD transmit specifications

The following portion of this subclause describes the transmit functions and parameters associated with the PMD sublayer. In general, these are specified by primitives from the PLCP, and the transmit PMD entity provides the actual means by which the signals required by the PLCP primitives are imposed onto the medium.

14.6.14.1 Nominal transmit power

The nominal transmit power of a frame is defined as the power averaged between the start of the first symbol in the PLCP Header to the end of the last symbol in the PLCP Header. When in the transmit state, the transmit power shall be within 2 dB of the nominal transmit power from the start of the Preamble SYNC field to the last symbol at the end of the frame.

14.6.14.2 Transmit power levels

Unless governed by more stringent local geographic regulations, the radiated emissions from compliant devices shall meet IEEE Std C95.1-1991 limits for controlled or uncontrolled environments, in accordance with their intended usage. In addition, all conformant PMD implementations shall support at least one power level with a minimum equivalent isotropically radiated power (EIRP) of 10 mW.

14.6.14.3 Transmit power level control

If a conformant PMD implementation has the ability to transmit in a manner that results in the EIRP of the transmit signal exceeding the level of 100 mW, at least one level of transmit power control shall be implemented. This transmit power control shall be such that the level of the emission is reduced to a level at or below 100 mW under the influence of said power control.

14.6.14.4 Transmit spectrum shape

Within the operational frequency band the transmitter shall pass a spectrum mask test. The duty cycle between Tx and Rx is nominally 50% and the transmit frame length is nominally 400 µs. The adjacent channel power is defined as the sum of the power measured in a 1 MHz band. For a pseudorandom data pattern, the adjacent channel power shall be a function of the offset between channel number N and the assigned transmitter channel M, where M is the actual transmitted center frequency and N a channel separated from it by an integer number of MHz**.**

Channel offset:

$|N-M| = 2$ –20 dBm or –40 dBc, whichever is the lower power.

$|N-M| \geq 3$ –40 dBm or –60 dBc, whichever is the lower power.

The levels given in dBc are measured relative to the transmitter power measured in a 1 MHz channel centered on the transmitter center frequency. The adjacent channel power and the transmitter power for this subclause of the specification shall be measured with a resolution bandwidth of 100 kHz, a video bandwidth of 300 kHz, and a peak detector, and with the measurement device set to maximum hold.

For any transmit center frequency M, two exceptions to the spectrum mask requirements are permitted within the operational frequency band, provided the exceptions are less than –50 dBc, where each offset channel exceeded counts as a separate exception. An exception occurs when the total energy within a given 1 MHz channel as defined in 14.6.5 exceeds the levels specified above.

14.6.14.5 Transmit center frequency tolerance

The PMD transmit center frequency shall be within ±60 kHz of the nominal center frequency as specified in 14.6.5.

14.6.14.6 Transmitter ramp periods

The transmitter shall go from off to within 2 dB of the nominal transmit power in 8 μs or less. The transmitter shall go from within 2 dB of the nominal transmit power to off (less than –50 dBm) in 8 μs or less.

14.6.15 PMD receiver specifications

The following portion of this subclause describes the receive functions and parameters associated with the PMD sublayer. In general, these are specified by primitives from the PLCP. The Receive PMD entity provides the actual means by which the signals required by the PLCP primitives are recovered from the medium. The PMD sublayer monitors signals on the medium and will return symbols from the set {{1},{0}} to the PLCP sublayer.

14.6.15.1 Input signal range

The PMD shall be capable of recovering a conformant PMD signal from the medium, as described in related subclauses, with a frame error ratio (FER) ≤3% for PSDUs of 400 octets generated with pseudorandom data, for receiver input signal levels in the range from –20 dBm to the receiver sensitivity (as specified in 14.6.15.4), across the frequency band of operation.

14.6.15.2 Receive center frequency acceptance range

An IEEE 802.11 FHSS compliant PMD shall meet all specifications with an input signal having a center frequency range of ±60 kHz from nominal.

14.6.15.3 CCA power threshold

In the presence of any IEEE 802.11-compliant 1 Mbit/s FH PMD signal above –85 dBm that starts synchronously with respect to slot times as specified in 14.3.3.2.1, the PHY shall signal busy, with a 90% probability of detection, during the preamble within the CCA assessment window. In the presence of any IEEE 802.11-compliant 1 Mbit/s FH PMD signal above –85 dBm that starts asynchronously with respect to slot times as specified in 14.3.3.2.1, the PHY shall signal busy, with a 70% probability of detection, during the preamble within the CCA window. In the presence of any IEEE 802.11 compliant 1 Mbit/s FH PMD signal above –65 dBm, the PHY shall signal busy, with a 70% probability of detection, during random data within the CCA window. This specification applies to a PMD operating with a nominal EIRP of < 100 mW. A compliant PMD operating at a nominal output power greater than 100 mW shall use the following equation to define the CCA threshold, where P_t represents transmit power.

$$\text{CCA threshold (preamble)} = -85\ \text{dBm} - \left[5 \times \log_{10}\left(\frac{P_t}{100\ \text{mW}}\right)\right]\ \text{dBm}$$

$$\text{CCA threshold (random data)} = \text{CCA threshold (preamble)} + 20\ \text{dB}$$

14.6.15.4 Receiver sensitivity

The sensitivity is defined as the minimum signal level required for an FER of 3% for PSDUs of 400 octets generated with pseudorandom data. The sensitivity shall be less than or equal to –80 dBm. The reference sensitivity is defined as –80 dBm for the 1 Mbit/s FH PHY specifications.

14.6.15.5 Intermodulation

Intermodulation protection (IMp) is defined as the ratio of the minimum amplitude of one of two equal interfering signals to the desired signal amplitude, where the interfering signals are spaced 4 and 8 MHz removed from the center frequency of the desired signal, both on the same side of center frequency. The IMp protection ratio is established at the interfering signal level that causes the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –77 dBm. Each interfering signal is modulated with the FH PMD modulation uncorrelated in time to each other or the desired signal. The PMD shall have the IMp for the interfering signal at 4 and 8 MHz be ≥30 dB.

14.6.15.6 Desensitization

Desensitization (Dp) is defined as the ratio to measured sensitivity of the minimum amplitude of an interfering signal that causes the FER at the output of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –77 dBm. The interfering signal shall be modulated with the FHSS PMD modulation uncorrelated in time to the desired signal. The minimum Dp shall be as given in Table 46. The spectral purity of the interferer shall be sufficient to ensure that the measurement is limited by the receiver performance.

Table 46—1 Mbit/s Dp

Interferer frequency[a]	Dp minimum
$M = N \pm 2$	30 dB
$M = N \pm 3$ or more	40 dB

[a]Where *M* is the interferer frequency and *N* is the desired channel frequency.

14.6.15.7 Receiver radiation

The signal leakage when receiving shall not exceed –50 dBm EIRP in the operating frequency range.

14.6.16 Operating temperature range

Two temperature ranges for full operation compliance to the FH PHY are specified. Type 1 is defined as 0 °C to 40 °C and is designated for office environments. Type 2 is defined as –30 °C to +70 °C and is designated for industrial environments.

14.7 FHSS PMD sublayer, 2.0 Mbit/s

14.7.1 Overview

This subclause details the RF specification differences of the optional 2 Mbit/s operation from the baseline 1 Mbit/s PMD as contained in 14.6. Unless otherwise specified in this subclause, the compliant PMD shall also meet all requirements of 14.6 when transmitting at 2 Mbit/s. When implementing the 2 Mbit/s option, the preamble and PHY Header shall be transmitted at 1 Mbit/s. STAs implementing the 2 Mbit/s option shall also be capable of transmitting and receiving PPDUs at 1 Mbit/s.

14.7.2 Four-Level GFSK modulation

For an FHSS 2 Mbit/s PMD, the modulation scheme shall be four-level Gaussian frequency shift keying (4GFSK), with a nominal symbol-period bandwidth product (BT) of 0.5. The four-level deviation factor, defined as the frequency separation of adjacent symbols divided by symbol rate, h4, shall be related to the deviation factor of the 2GFSK modulation, h2, by the following equation:

$$h4/h2 = 0.45 \pm 0.01$$

An incoming bit stream at 2 Mbit/s will be converted to 2-bit words or symbols, with a rate of Fclk = 1 *M*symbol/s. The first received bit will be encoded as the LMB of the symbol in Table 47. The bits will be encoded into symbols as shown in Table 47.

Table 47—Symbol encoding into carrier deviation

1 Mbit/s, 2GFSK	
Symbol	**Carrier deviation**
1	1/2× h2 × Fclk
0	–1/2 × h2 × Fclk
2 Mbit/s, 4GFSK	
Symbol	**Carrier deviation**
10	3/2 × h4 × Fclk
11	1/2 × h4 × Fclk
01	–1/2 × h4 × Fclk
00	–3/2 × h4 × Fclk
NOTE—These deviation values are measured using the center symbol of 7 consecutive symbols of the same value. The instantaneous deviation will vary due to Gaussian pulse shaping.	

The deviation factor h2 for 2GFSK (measured as the difference between frequencies measured in the middle of 0000 and 1111 patterns encountered in the SFD, divided by 1 MHz) will nominally be 0.32. The deviation factor h2 will be no less than 0.30 (with maximum dictated by regulatory bandwidth requirement). Accordingly, h4 (measured as a difference between the outermost frequencies, divided by 3, divided by 1 MHz) is nominally $0.45 \times 0.32 = 0.144$, and it will be no less than $0.45 \times 0.3 = 0.135$.

The modulation error shall be less than ±15 kHz at the midsymbol time for 4GFSK, from the frequency deviations specified above, for a symbol surrounded by identical symbols, and less than ±25 kHz for any symbol. The deviation is relative to the actual center frequency of the RF carrier. For definition purposes, the actual center frequency is the midfrequency between symbols 11 and 01. The actual center frequency shall be within ±60 kHz of the nominal channel center frequency defined in 14.6.5 and shall not vary by more than ±10 kHz/ms, from the start to end of the PPDU. The peak-to-peak variation of the actual center frequency over the PPDU shall not exceed 15 kHz. Symbols and terms used within this subclause are illustrated in Figure 84.

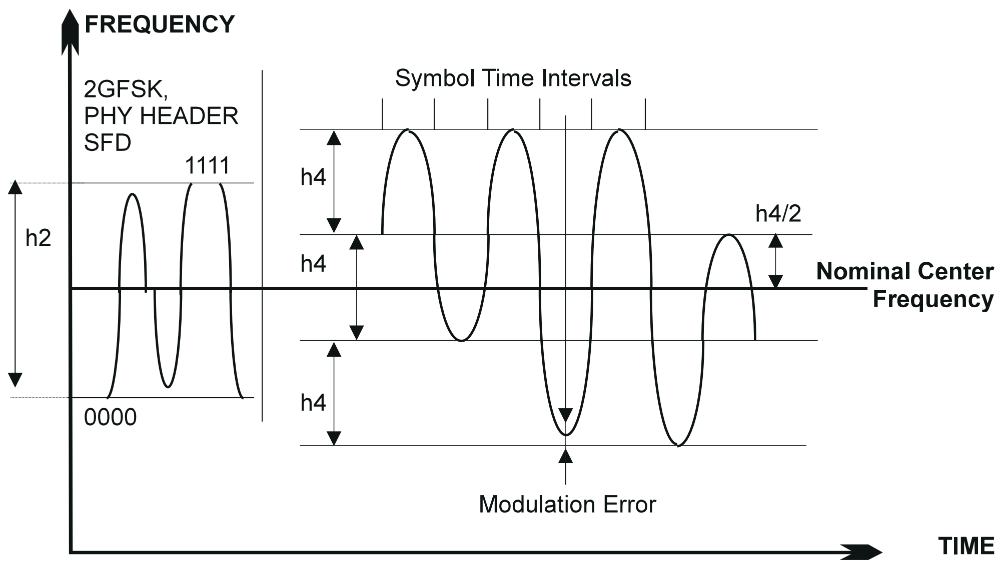


Figure 84—Four-Level GFSK transmit modulation

14.7.2.1 Frame structure for HS FHSS PHY

The high rate FHSS PPDU consists of PLCP Preamble, PLCP Header, and whitened PSDU. The PLCP Preamble and PLCP Header format are identical to 1 Mbit/s PHY, as described in 14.3.2. The whitened PSDU is transmitted in 2GFSK, 4GFSK, or potentially a higher-rate format, according to the rate chosen. The rate is indicated in a 3-bit field in a PLCP Header, having a value of 1 or 2 bits/symbol (or Mbit/s).

The PPDU is transmitted as four-level symbols, with the amount determined by number_of_symbols = (number_of_PSDU_octets × 8)/rate.

The input bits are scrambled according to the method in 14.3.2.3.

The scrambled bit stream is divided into groups of rate (1 or 2) consecutive bits. The bits are mapped into symbols according to Table 47.

A bias suppression algorithm is applied to the resulting symbol stream. The bias suppression algorithm is defined in 14.3.2.3, Figure 71, and Figure 74. A polarity control symbol is inserted prior to each block of 32 symbols (or less for the last block). The polarity control signals are 4GFSK symbols 10 or 00. The algorithm is equivalent to the case of 2GFSK, with the polarity symbol 2GFSK “1” replaced with 4GFSK symbol “10,” and the 2GFSK polarity symbol “0” replaced with a 4GFSK symbol “00.”

14.7.3 Channel data rate

The data rate for the whitened PSDU at the optional rate shall be 2.0 Mbit/s ± 50 ppm.

14.7.3.1 Input dynamic range

The PMD shall be capable of recovering a conformant PMD signal from the medium, as described in related subclauses, with an FER ≤3% for PSDUs of 400 octets generated with pseudorandom data, for receiver input signal levels in the range from –20 dBm to the receiver sensitivity (as specified in 14.7.3.2), across the frequency band of operation.

14.7.3.2 Receiver sensitivity

The sensitivity is defined as the minimum signal level required for an FER of 3% for PSDUs of 400 octets generated with pseudorandom data. The sensitivity shall be less than or equal to –75 dBm. The reference sensitivity is defined as –75 dBm for the 2 Mbit/s FH PHY specifications.

14.7.3.3 IMp

IMp is defined as the ratio to –77 dBm of the minimum amplitude of one of the two equal level interfering signals at 4 and 8 MHz removed from center frequency, both on the same side of center frequency, that cause the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –72 dBm (3 dB above the specified sensitivity specified in 14.7.3.2). Each interfering signal is modulated with the FH 1 Mbit/s PMD modulation uncorrelated in time to each other or the desired signal. The FHSS optional 2 Mbit/s rate IMp shall be ≥25 dB.

14.7.3.4 Dp

Dp is defined as the ratio to measured sensitivity of the minimum amplitude of an interfering signal that causes the FER of the receiver to be increased to 3% for PSDUs of 400 octets generated with pseudorandom data, when the desired signal is –72 dB (3 dB above sensitivity specified in 14.7.3.2). The interfering signal shall be modulated with the FHSS PMD modulation uncorrelated in time to the desired signal. The minimum Dp shall be as given in Table 48.

Table 48—2 Mbit/s Dp

Interferer frequency[a]	DP minimum
$M = N \pm 2$	20 dB
$M = N \pm 3$ or more	30 dB

[a]Where *M* is the interferer frequency and *N* is the desired channel frequency.

14.8 FHSS PHY management information base (MIB)

14.8.1 Overview

The following is the MIB for the FHSS PHY.

14.8.2 FH PHY attributes

This subclause defines the attributes for the FHSS MIB. Table 49 lists these attributes and the default values. Following the table is a description of each attribute.

Table 49—FHSS PHY attributes

Attribute	Default value	Operational semantics	Operational behavior
aPHYType	FHSS = X'01'	Static	Identical for all FH PHYs
aRegDomainsSupported	FCC = X'10' IC = X'20' ETSI = X'30' Spain = X'31' France = X'32' MKK = X'40'	Static	Implementation dependent
aCurrentRegDomain	X'00'	Dynamic LME	Implementation dependent
aSlotTime	50 µs	Static	Identical for all FH PHYs
aCCATime	27 µs	Static	Identical for all FH PHYs
aRxTxTurnaroundTime	20 µs	Static	Identical for all FH PHYs
aTxPLCPDelay	1 µs	Static	Identical for all FH PHYs
aRxTxSwitchTime	10 µs	Static	Identical for all FH PHYs
aTxRampOnTime	8 µs	Static	Identical for all FH PHYs
aTxRFDelay	1 µs	Static	Identical for all FH PHYs
aSIFSTime	28 µs	Static	Identical for all FH PHYs
aRxRFDelay	4 µs	Static	Identical for all FH PHYs
aRxPLCPDelay	2 µs	Static	Identical for all FH PHYs
aMACProcessingDelay	2 µs	Static	Identical for all FH PHYs
aTxRampOffTime	8 µs	Static	Identical for all FH PHYs
aPreambleLength	96 µs	Static	Identical for all FH PHYs
aPLCPHdrLength	32 µs	Static	Identical for all FH PHYs
aMPDUDurationFactor	1.03125	Static	Identical for all FH PHYs
aAirPropagationTime	1 µs	Static	Identical for all FH PHYs
aTempType	Type 1 = X'01' Type 2 = X'02' Type 3 = X'03'	Static	Implementation dependent
aCWmin	15	Static	Identical for all FH PHYs
aCWmax	1023	Static	Identical for all FH PHYs
aSupportedDataRatesTX	1 Mbit/s = X'02' manda- tory 2 Mbit/s = X'04' optional	Static	Identical for all FH PHYs
aSupportedDataRatesRX	1 Mbit/s = X'02' manda- tory 2 Mbit/s = X'04' optional	Static	Identical for all FH PHYs
aMPDUMaxLength	4095 octets	Static	Identical for all FH PHYs
aSupportedTxAntennas	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant n = n	Static	Implementation dependent
aCurrentTxAntenna	Ant 1 = default	Dynamic LME	Implementation dependent

Table 49—FHSS PHY attributes (continued)

Attribute	Default value	Operational semantics	Operational behavior
aSupportedRxAntennas	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant n = n	Static	Implementation dependent
aDiversitySupport	Available = X'01' Not avail. = X'02' Control avail. = X'03'	Static	Implementation dependent
aDiversitySelectionRx	Ant 1 = X'01' Ant 2 = X'02' Ant 3 = X'03' Ant 4 = X'04' Ant 5 = X'05' Ant 6 = X'06' Ant 7 = X'07' Ant 8 = X'08'	Dynamic LME	Implementation dependent
aNumberSupportedPowerLevels	Lvl1 = X'01' Lvl2 = X'02' Lvl3 = X'03' Lvl4 = X'04' Lvl5 = X'05' Lvl6 = X'06' Lvl7 = X'07' Lvl8 = X'08'	Static	Implementation dependent
aTxPowerLevel1	Factory def. default	Static	Implementation dependent
aTxPowerLevel2	Factory def.	Static	Implementation dependent
aTxPowerLevel3	Factory def.	Static	Implementation dependent
aTxPowerLevel4	Factory def.	Static	Implementation dependent
aTxPowerLevel5	Factory def.	Static	Implementation dependent
aTxPowerLevel6	Factory def.	Static	Implementation dependent
aTxPowerLevel7	Factory def.	Static	Implementation dependent
aTxPowerLevel8	Factory def.	Static	Implementation dependent
aCurrentTxPowerLevel	TxPowerLevel1	Dynamic LME	Implementation dependent
aHopTime	224 μs	Static	Identical for all FH PHYs
aCurrentChannelNumber	X'00'	Dynamic PLME	
aMaxDwellTime	390 TU	Static	Regulatory domain dependent
aCurrentSet	X'00'	Dynamic PLME	
aCurrentPattern	X'00'	Dynamic PLME	
aCurrentIndex	X'00'	Dynamic PLME	
aCurrentPowerState	X'01' off X'02' on	Dynamic LME	

NOTE—The column titled “Operational semantics” contains two types: static and dynamic. Static MIB attributes are fixed and cannot be modified for a given PHY implementation. MIB attributes defined as dynamic can be modified by some management entity. Whenever an attribute is defined as dynamic, the column also shows which entity has control over the attribute. LME refers to the MAC sublayer management entity (MLME), while PHY refers to the physical layer management entity (PLME).

14.8.2.1 FH PHY attribute definitions

14.8.2.1.1 aPHYType

The aPHYType is FHSS. The LME uses this attribute to determine what PLCP and PMD is providing services to the MAC. It also is used by the MAC to determine what MAC sublayer management state machines must be invoke to support the PHY. The value of this attribute is defined as the integer 01 to indicate the FHSS PHY.

14.8.2.1.2 aRegDomainsSupported

Operational requirements for FHSS PHY are defined by agencies representing certain geographical regulatory domains. These regulatory agencies may define limits on various parameters that differ from region to region. This parameters may include aTxPowerLevels, and aMaxDwellTime, as well as the total number of frequencies in the hopping pattern. The values shown in Table 50 indicate regulatory agencies supported by this document.

Table 50—Regulatory domain codes

Code point	Regulatory agency	Region
X'10'	FCC	United States
X'20'	IC	Canada
X'30'	ETSI	Most of Europe
X'31'	Spain	Spain
X'32'	France	France
X'40'	MKK	Japan
X'00'	Null terminator	—

Since a PLCP and PMD might be designed to support operation in more than one regulatory domain, this attribute can actually represent a list of agencies. This list can be one or more of the above agencies and must be terminated using the null terminator. Upon activation of the PLCP and PMD, the information in this list must be used to set the value of the aCurrentRegDomain attribute.

14.8.2.1.3 aCurrentRegDomain

The aCurrentRegDomain attribute for the FHSS PHY is defined as the regulatory domain under which the PMD is currently operating. This value must be one of the values listed in the aRegDomainsSupported list. This MIB attribute is managed by the LME.

14.8.2.1.4 aSlotTime

The aSlotTime is a PHY dependent attribute used by the MAC sublayer to determine the PIFS and DIFS periods. It is defined using the following equation:

aCCATime + aRxTxTurnaroundTime + aAirPropagationTime + aMACProcessingDelay

For the FHSS PHY, the aCCATime is 27 μs, and the aRxTxTurnaroundTime is 20 μs. The aAirPropagationTime is fixed at 1 μs. The aMACProcessingDelay is nominally 2 μs. The value of this attribute is 50 μs.

14.8.2.1.5 aCCATime

The aCCATime for the FHSS PHY is defined as the time the receiver must use to evaluate the medium at the antenna to determine the state of the channel. This time period for the FHSS PHY is 27 µs. This period includes the aRxRFDelay and the aRxPLCPDelay.

14.8.2.1.6 aRxTxTurnaround Time

The aRxTxTurnaroundTime for the FHSS PHY is defined as the time it takes a STA to place a valid symbol on the medium after a PHY-TXSTART.request. The aRxTxTurnaroundTime is determined using the following equation.

aTxPLCPDelay + aRxTxSwitchTime + aTxRampOnTime + aTxRFDelay

For the FHSS PHY, the aTxPLCPDelay is 1 µs, the aRxTxSwitchTime is 10 µs, the aTxRampOnTime is 8 µs, and the aTxRFDelay is 1 µs, for a total of 20 µs. This is the maximum time for getting valid data on the medium. STAs can use less time but not more than 20 µs.

14.8.2.1.7 aTxPLCPDelay

The aTxPLCPDelay for the FHSS PHY is defined as the delay the PLCP introduces in getting data onto the air in the transmit direction. This value for the FHSS PHY is nominally 1 µs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.8 aRxTxSwitchTime

The aRxTxSwitchTime for the FHSS PHY is defined as the delay the PMD requires to change from receive to transmit. This value for the FHSS PHY is nominally 10 µs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.9 aTxRampOnTime

The aTxRampOnTime for the FHSS PHY is defined as the delay the PMD requires to turn on the transmit power amplifier. This value for the FHSS PHY is nominally 8 µs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.10 aTxRFDelay

The aTxRFDelay for the FHSS PHY is defined as the nominal time in microseconds between the issuance of a PMDDATA.request to the PMD and the start of the corresponding symbol at the air interface. The start of a symbol is defined to be 1/2 symbol period prior to the center of the symbol. This value for the FHSS PHY is nominally 1 µs. Implementors may choose to increase or decrease this delay as long as the requirements of aRxTxTurnaroundTime are met.

14.8.2.1.11 aSIFSTime

The aSIFSTime for the FHSS PHY is defined as the time the MAC and PHY sublayers will require to receive the last symbol of a frame at the air interface, process the frame, and respond with the first symbol of a preamble on the air interface. The aSIFSTime is determined using the following equation:

aRxRFDelay + aRxPLCPDelay + aMACProcessingDelay + aRxTxTurnaroundTime

For the FHSS PHY, the aRxRFDelay is 4 μs, the aRxPLCPDelay is 2 μs, the aMACProcessingDelay is 2 μs, and the aRxTxTurnaroundTime is 20 μs, for a total of 28 μs. This is the nominal value for aSIFSTime. In order to account for variations between implementations, this value has a tolerance as specified in 9.2.3.1.

14.8.2.1.12 aRXRFDelay

The aRxRFDelay for the FHSS PHY is defined as the nominal time in microseconds between the end of a symbol at the air interface to the issuance of a PMDDATA.indicate to the PLCP. The end of a symbol is defined to be 1/2 symbol period after the center of the symbol. This value for the FHSS PHY is nominally 4 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime and aCCATime are met.

14.8.2.1.13 aRxPLCPDelay

The aRxPLCPDelay for the FHSS PHY is defined as the delay the PLCP introduces in the data path between the PMD and the MAC sublayer. This value for the FHSS PHY is nominally 2 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime and aCCATime are met.

14.8.2.1.14 aMACProcessingDelay

The aMACProcessingDelay for the FHSS PHY is defined as the delay between when a PHY-RXEND.indicate is issued by the PHY till a corresponding PHY-TXSTART.request is issued by the MAC. This value for the FHSS PHY is nominally 2 μs. Implementors may choose to increase or decrease this delay as long as the requirements of aSIFSTime are met.

14.8.2.1.15 aTxRampOffTime

The aTxRampOffTime for the FHSS PHY is defined as the delay the PMD requires to turn off the transmit power amplifier. This value for the FHSS PHY is a maximum of 8 μs.

14.8.2.1.16 aPreambleLength

The parameter aPreambleLength defines the time required by the FHSS PHY to transmit the PLCP Preamble. For both the 1 and 2 Mbit/s FHSS PHYs, this value is 96 μs.

14.8.2.1.17 aPLCPHdrLength

The parameter aPLCPHdrLength defines the time required by the FHSS PHY to transmit the PLCP Header. For both the 1 and 2 Mbit/s FHSS PHYs, this value is 32 μs.

14.8.2.1.18 aMPDUDurationFactor

The parameter aMPDUDurationFactor defines the overhead added by the PHY to the PSDU as it is transmitted over the air. For the FHSS PHY, this factor is 1.03125. This factor is calculated as 33/32 to account for the expansion due to the data whitener encoding algorithm. The total time to transmit a PPDU over the air is the following equation rounded up to the next integer microsecond:

$$\text{aPreambleLength} + \text{aPLCPHdrLength} + \text{aMPDUDurationFactor} \times 8 \times \text{PSDU length (octets)/data rate}$$

The total time in microseconds to the beginning of any octet in an PPDU from the first symbol of the preamble can be calculated using the duration factor in the following equation:

$$\text{Truncate}[\text{aPreambleLength} + \text{aPLCPHdrLength} + \text{aMPDUDurationFactor} \times 8 \times N/\text{data rate}] + 1$$

where *N* is the number of octets prior to the desired octet.

14.8.2.1.19 aAirPropagationTime

The parameter aAirPropagationTime is the time it takes a transmitted signal to go from the transmitting STA to the receiving STA. A nominal value of 1 μs has been allocated for this parameter. Variations in the actual propagation time are accounted for in the allowable range of aSIFSTime.

14.8.2.1.20 aTempType

The parameter aTempType defines the temperature range supported by the PHY. Type 1 equipment (X'01') supports a temperature range of 0 °C to 40 °C. Type 2 equipment (X'02') supports a temperature range of –20 °C to +55 °C. Type 3 equipment (X'03') supports a temperature range of –30 °C to +70 °C.

14.8.2.1.21 aCWmin

The parameter aCWmin defines the minimum size of the contention window, in slots. For the FH PHY, this number is 15 decimal.

14.8.2.1.22 aCWmax

The parameter aCWmin defines the maximum size of the contention window, in slots. For the FH PHY, this number is 1023 decimal.

14.8.2.1.23 aCurrentPowerState

The aCurrentPowerState attribute for the FHSS PHY allows the MAC sublayer management entity to control the power state of the PHY. This attribute can be updated using the PLMESET.request. The permissible values are ON and OFF.

14.8.2.1.24 aSupportedDataRatesTX

The aSupportedDataRatesTX attribute for the FHSS PHY is defined as a null terminated list of supported data rates in the transmit mode for this implementation. Table 51 shows the possible values appearing in the list.

Table 51—Supported data rate codes (aSupportedDataRatesTX)

Code point	Data rate
X'02'	1 Mbit/s
X'04'	2 Mbit/s
X'00'	Null terminator

14.8.2.1.25 aSupportedDataRatesRX

The aSupportedDataRatesRX attribute for the FHSS PHY is defined as a null terminated list of supported data rates in the receive mode for this implementation. Table 52 shows the possible values appearing in the list.

Table 52—Supported data rate codes (aSupportedDataRatesRX)

Code point	Data rate
X'02'	1 Mbit/s
X'04'	2 Mbit/s
X'00'	Null terminator

14.8.2.1.26 aMPDUMaxLength

The aMPDUMaximumLength attribute for the FHSS PHY is defined as the maximum PSDU, in octets, that the PHY shall ever be capable of accepting. This value for the FHSS PHY is set at 4095 octets. The recommended value for maximum PSDU length in an FHSS PHY system is 400 octets at 1 Mbit/s and 800 octets at 2 Mbit/s, which corresponds to a frame duration less than 3.5 ms. These values are optimized to achieve high performance in a variety of RF channel conditions, particularly with respect to indoor multipath, channel stability for moving STAs, and interference in the 2.4 GHz band.

14.8.2.1.27 aSupportedTxAntennas

The aSupportedTxAntennas attribute for the FHSS PHY is defined as a null terminated list of antennas that this implementation can use to transmit data. Table 53 shows the possible values appearing in the list, where $N \leq 255$.

Table 53—Number of transmit antennas

Code point	Antenna number
X'01'	Tx Antenna 1
X'02'	Tx Antenna 2
X'03'	Tx Antenna 3
…	…
N	Tx Antenna *N*
X'00'	Null terminator

14.8.2.1.28 aCurrentTxAntenna

The CurrentTxAntenna attribute for the FHSS PHY is used to describe the current antenna the implementation is using for transmission. This value should represent one of the antennas appearing in the SupportedTx-Antennas list.

14.8.2.1.29 aSupportedRxAntenna

The aSupportedRxAntennas attribute for the FHSS PHY is defined as a null terminated list of antennas that this implementation can use to receive data. In the FHSS PHY primitives, one of these values is passed as part of the PHY-RXSTART.indicate to the MAC sublayer for every received packet. Table 54 shows the possible values appearing in the list, where $N \leq 255$.

Table 54—Number of receive antennas

Code point	Antenna number
X'01'	Rx Antenna 1
X'02'	Rx Antenna 2
X'03'	Rx Antenna 3
…	…
N	Rx Antenna *N*
X'00'	Null terminator

14.8.2.1.30 aDiversitySupport

The aDiversitySupport attribute for the FHSS PHY is used to describe the implementation's diversity support. Table 55 shows the possible values appearing in the list.

Table 55—Diversity support codes

Code point	Diversity support
X'01'	Diversity available
X'02'	No diversity
X'03'	Control available

The value X'01' indicates that this implementation uses two or more antennas for diversity. The value X'02' indicates that the implementation has no diversity support. The value X'03' indicates that the choice of antennas used during diversity is programmable. (See 14.8.2.1.31.)

14.8.2.1.31 aDiversitySelectionRx

The aDiversitySelectionRx attribute for the FHSS PHY is a null terminated list describing the receive antenna or antennas currently in use during diversity and packet reception. Table 56 below shows the possible values appearing in the list, where $N \leq 255$.

Table 56—Diversity select antenna codes

Code point	Antenna number
X'01'	Rx Antenna 1
X'02'	Rx Antenna 2
X'03'	Rx Antenna 3
N	Rx Antenna *N*
X'00'	Null terminator

The null terminated list can consist of one or more of the receive antennas listed in the aSupportedRxAntennas attribute. This attribute can be changed dynamically by the LME.

14.8.2.1.32 aNumberSupportedPowerLevels

The aNumberSupportedPowerLevels attribute for the FHSS PHY describes the number of power levels this implementation supports. This attribute can be an integer of value 1 through 8, inclusive.

14.8.2.1.33 aTxPowerLevel1-8

Some implementations may provide up to eight different transmit power levels. The aTxPowerLevels attribute for the FHSS PHY is a list of up to eight power levels supported. Table 57 describes the list.

Table 57—Transmit power levels

Attribute	Power level
TxPowerLevel1	Default setting
TxPowerLevel2	Level 2
TxPowerLevel3	Level 3
TxPowerLevel4	Level 4
TxPowerLevel5	Level 5
TxPowerLevel6	Level 6
TxPowerLevel7	Level 7
TxPowerLevel8	Level 8

14.8.2.1.34 aCurrentTxPowerLevel

The aCurrentTxPowerLevel attribute for the FHSS PHY is defined as the current transmit output power level. This level shall be one of the levels implemented in the list of attributes called aTxPowerLevel*N* (where *N* is 1–8). This MIB attribute is also used to define the sensitivity of the CCA mechanism when the output power exceeds 100 mW. This MIB attribute is managed by the LME.

14.8.2.1.35 aHopTime

The aHopTime attribute for the FHSS PHY describes the time allocated for the PHY to change to a new frequency. For the FHSS PHY, this time period is 224 μs.

14.8.2.1.36 aCurrentChannelNumber

The aCurrentChannelNumber attribute for the FHSS PHY is defined as the current operating channel number of the PMD. The values of this attribute correspond to the values shown in Table 38. This MIB attribute is managed by the PLME and is updated as the result of a PLMESET.request to aCurrentSet, aCurrentPattern, or aCurrentIndex.

14.8.2.1.37 aMaxDwellTime

The aMaxDwellTime attribute for the FHSS PHY is defined as the maximum time the PMD can dwell on a channel and meet the requirements of the current regulatory domain. For the FCC regulatory domain, this number is 390 TU (FCC = 400 ms). The recommended dwell time for the FHSS PHY is 19 TU.

14.8.2.1.38 aCurrentSet

The FHSS PHY contains three sets of hopping patterns. The aCurrentSet attribute for the FHSS PHY defines what set the STA is using to determine the hopping pattern. Its value can be 1, 2, or 3. This attribute is managed by the PLME.

14.8.2.1.39 aCurrentPattern

There are up to 26 patterns in each hopping set used by the FHSS PHY. The aCurrentPattern attribute for the FHSS PHY defines what pattern the STA is using to determine the hopping sequence. Its value has various ranges, always within the overall range of 1 to 26, depending on the aCurrentRegDomain. This attribute is managed by the PLME.

14.8.2.1.40 aCurrentIndex

The FHSS PHY addresses each channel in the selected hopping pattern through an index. The aCurrentIndex attribute for the FHSS PHY defines what index the STA will use to determine the next hop-channel number. Its value has various ranges, always within the overall range of 1 to 26, depending on the aCurrentRegDomain. This attribute is managed by the PLME.

14.8.2.1.41 aCurrentPowerState

The parameter aCurrentPowerState defines the operational state of the FHSS PHY. When this attribute has a value of X'01', the PHY is “OFF.” When this attribute has a value of X'02', the PHY is “ON.” This attribute is managed by the PLME.

15. Direct sequence spread spectrum (DSSS) PHY specification for the 2.4 GHz band designated for ISM applications

15.1 Overview

The PHY for the direct sequence spread spectrum (DSSS) system is described in this clause. The RF LAN system is initially aimed for the 2.4 GHz band designated for ISM applications as provided in the USA according to FCC 15.247, in Europe by ETS 300–328, and in other countries according to 15.4.6.2.

The DSSS system provides a wireless LAN with both a 1 Mbit/s and a 2 Mbit/s data payload communication capability. According to the FCC regulations, the DSSS system shall provide a processing gain of at least 10 dB. This shall be accomplished by chipping the baseband signal at 11 MHz with an 11-chip PN code. The DSSS system uses baseband modulations of differential binary phase shift keying (DBPSK) and differential quadrature phase shift keying (DQPSK) to provide the 1 and 2 Mbit/s data rates, respectively.

15.1.1 Scope

The PHY services provided to the IEEE 802.11 wireless LAN MAC by the 2.4 GHz DSSS system are described in this clause. The DSSS PHY layer consists of two protocol functions:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function shall be supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDU) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, a wireless medium (WM) between two or more STAs each using the DSSS system.

15.1.2 DSSS PHY functions

The 2.4 GHz DSSS PHY architecture is depicted in the reference model shown in Figure 11. The DSSS PHY contains three functional entities: the PMD function, the physical layer convergence function, and the layer management function. Each of these functions is described in detail in the following subclauses.

The DSSS PHY service shall be provided to the MAC through the PHY service primitives described in Clause 12.

15.1.2.1 PLCP sublayer

To allow the IEEE 802.11 MAC to operate with minimum dependence on the PMD sublayer, a physical layer convergence sublayer is defined. This function simplifies the PHY service interface to the IEEE 802.11 MAC services.

15.1.2.2 PMD sublayer

The PMD sublayer provides a means to send and receive data between two or more STAs. This clause is concerned with the 2.4 GHz ISM bands using direct sequence modulation.

15.1.2.3 Physical layer management entity (PLME)

The PLME performs management of the local PHY functions in conjunction with the MAC management entity.

15.1.3 Service specification method and notation

The models represented by figures and state diagrams are intended to be illustrations of functions provided. It is important to distinguish between a model and a real implementation. The models are optimized for simplicity and clarity of presentation; the actual method of implementation is left to the discretion of the IEEE 802.11 DSSS PHY compliant developer.

The service of a layer or sublayer is a set of capabilities that it offers to a user in the next-higher layer (or sublayer). Abstract services are specified here by describing the service primitives and parameters that characterize each service. This definition is independent of any particular implementation.

15.2 DSSS PLCP sublayer

15.2.1 Overview

This clause provides a convergence procedure in which MPDUs are converted to and from PPDUs. During transmission, the MPDU shall be prepended with a PLCP Preamble and Header to create the PPDU. At the receiver, the PLCP Preamble and header are processed to aid in demodulation and delivery of the MPDU.

15.2.2 PLCP frame format

Figure 85 shows the format for the PPDU including the DSSS PLCP Preamble, the DSSS PLCP Header, and the MPDU. The PLCP Preamble contains the following fields: Synchronization (Sync) and Start Frame Delimiter (SFD). The PLCP Header contains the following fields: IEEE 802.11 Signaling (Signal), IEEE 802.11 Service (Service), LENGTH (Length), and CCITT CRC-16. Each of these fields is described in detail in 15.2.3.

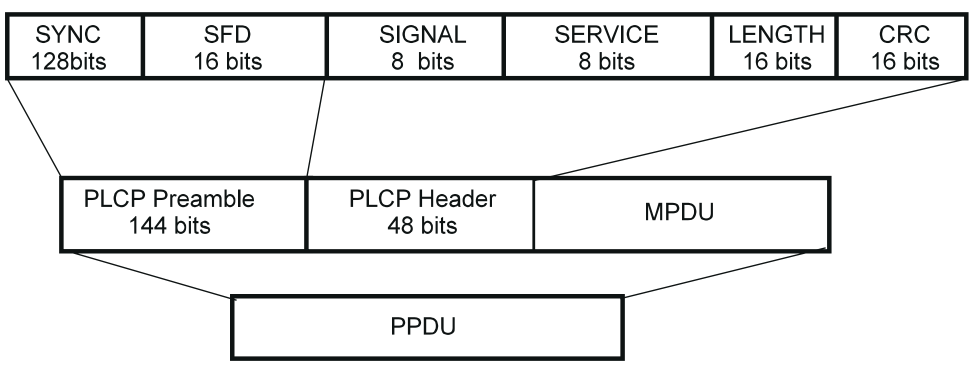


Figure 85—PLCP frame format

15.2.3 PLCP field definitions

The entire PLCP Preamble and Header shall be transmitted using the 1 Mbit/s DBPSK modulation described in 15.4.7. All transmitted bits shall be scrambled using the feedthrough scrambler described in 15.2.4.

15.2.3.1 PLCP Synchronization (SYNC) field

The SYNC field shall consist of 128 bits of scrambled 1 bit. This field shall be provided so that the receiver can perform the necessary operations for synchronization.

15.2.3.2 PLCP Start Frame Delimiter (SFD)

The SFD shall be provided to indicate the start of PHY dependent parameters within the PLCP Preamble. The SFD shall be a 16-bit field, X'F3A0' (msb to lsb). The lsb shall be transmitted first in time.

15.2.3.3 PLCP IEEE 802.11 Signal (SIGNAL) field

The 8-bit IEEE 802.11 signal field indicates to the PHY the modulation that shall be used for transmission (and reception) of the MPDU. The data rate shall be equal to the Signal field value multiplied by 100 kbit/s. The DSSS PHY currently supports two mandatory modulation services given by the following 8-bit words, where the lsb shall be transmitted first in time:

a) X'0A' (msb to lsb) for 1 Mbit/s DBPSK
b) X'14' (msb to lsb) for 2 Mbit/s DQPSK

The DSSS PHY rate change capability is described in 15.2.5. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.4 PLCP IEEE 802.11 Service (SERVICE) field

The 8-bit IEEE 802.11 service field shall be reserved for future use. The value of X'00' signifies IEEE 802.11 device compliance. The lsb shall be transmitted first in time. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.5 PLCP Length (LENGTH) field

The PLCP Length field shall be an unsigned 16-bit integer that indicates the number of microseconds (16 to 2^{16}–1 as defined by aMPDUMaxLength) required to transmit the MPDU. The transmitted value shall be determined from the LENGTH parameter in the TXVECTOR issued with the PHY-TXSTART.request primitive described in 12.3.5.4. The Length field provided in the TXVECTOR is in bytes and is converted to microseconds for inclusion in the PLCP LENGTH field. The lsb shall be transmitted first in time. This field shall be protected by the CCITT CRC-16 frame check sequence described in 15.2.3.6.

15.2.3.6 PLCP CRC (CCITT CRC-16) field

The IEEE 802.11 SIGNAL, IEEE 802.11 SERVICE, and LENGTH fields shall be protected with a CCITT CRC-16 FCS (frame check sequence). The CCITT CRC-16 FCS shall be the one’s complement of the remainder generated by the modulo 2 division of the protected PLCP fields by the polynomial:

$$x^{16} + x^{12} + x^5 + 1$$

The protected bits shall be processed in transmit order. All FCS calculations shall be made prior to data scrambling.

As an example, the SIGNAL, SERVICE, and LENGTH fields for a DBPSK signal with a packet length of 192 µs (24 bytes) would be given by the following:

0101 0000 0000 0000 0000 0011 0000 0000 (leftmost bit transmitted first in time)

The one’s complement FCS for these protected PLCP Preamble bits would be the following:

0101 1011 0101 0111 (leftmost bit transmitted first in time)

Figure 86 depicts this example.

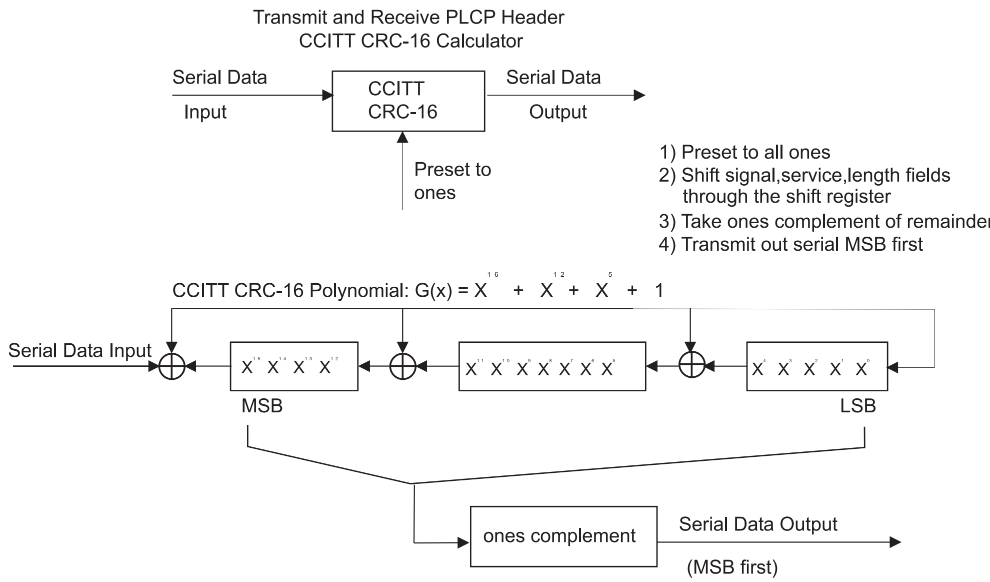


Figure 86—CCITT CRC-16 implementation

An illustrative example of the CCITT CRC-16 FCS using the information from Figure 86 follows in Figure 87.

Data	CRC registers msb lsb	
	1111111111111111	; initialize preset to 1's
0	1110111111011111	
1	1101111110111110	
0	1010111101011101	
1	0101111010111010	
0	1011110101110100	
0	0110101011001001	
0	1101010110010010	
0	1011101100000101	
0	0110011000101011	
0	1100110001010110	
0	1000100010001101	
0	0000000100111011	
0	0000001001110110	
0	0000010011101100	
0	0000100111011000	
0	0001001110110000	
0	0010011101100000	
0	0100111011000000	
0	1001110110000000	
0	0010101100100001	
0	0101011001000010	
0	1010110010000100	
1	0101100100001000	
1	1010001000110001	
0	0101010001000011	
0	1010100010000110	
0	0100000100101101	
0	1000001001011010	
0	0001010010010101	
0	0010100100101010	
0	0101001001010100	
0	1010010010101000	
	0101101101010111	; one's complement, result = CRC FCS parity

Figure 87—Example CRC calculation

15.2.4 PLCP/DSSS PHY data scrambler and descrambler

The polynomial $G(z) = z^{-7} + z^{-4} + 1$ shall be used to scramble *all* bits transmitted by the DSSS PHY. The feedthrough configuration of the scrambler and descrambler is self-synchronizing, which requires no prior knowledge of the transmitter initialization of the scrambler for receive processing. Figure 88 and Figure 89 show typical implementations of the data scrambler and descrambler, but other implementations are possible.

The scrambler should be initialized to any state except all ones when transmitting.

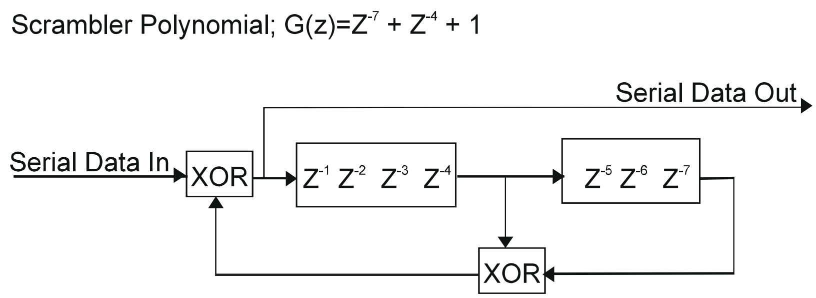


Figure 88—Data scrambler

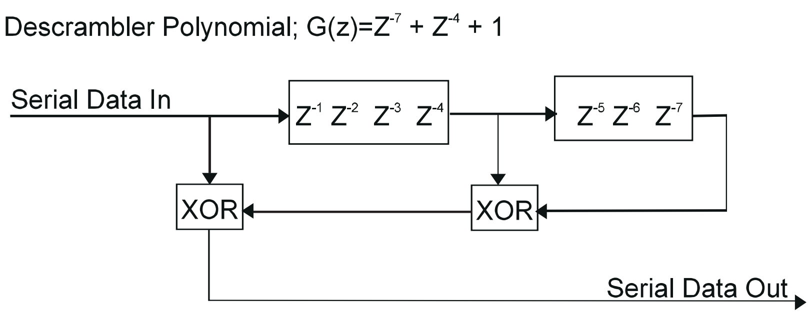


Figure 89—Data descrambler

15.2.5 PLCP data modulation and modulation rate change

The PLCP Preamble shall be transmitted using the 1 Mbit/s DBPSK modulation. The IEEE 802.11 SIGNAL field shall indicate the modulation that shall be used to transmit the MPDU. The transmitter and receiver shall initiate the modulation indicated by the IEEE 802.11 SIGNAL field starting with the first symbol (1 bit for DBPSK or 2 bits for DQPSK) of the MPDU. The MPDU transmission rate shall be set by the DATARATE parameter in the TXVECTOR issued with the PHY-TXSTART.request primitive described in 15.4.4.1.

15.2.6 PLCP transmit procedure

The PLCP transmit procedure is shown in Figure 90.

In order to transmit data, PHY-TXSTART.request shall be enabled so that the PHY entity shall be in the transmit state. Further, the PHY shall be set to operate at the appropriate channel through station management via the PLME. Other transmit parameters such as DATARATE, TX antenna, and TX power are set via the PHY-SAP with the PHY-TXSTART.request(TXVECTOR) as described in 15.4.4.2.

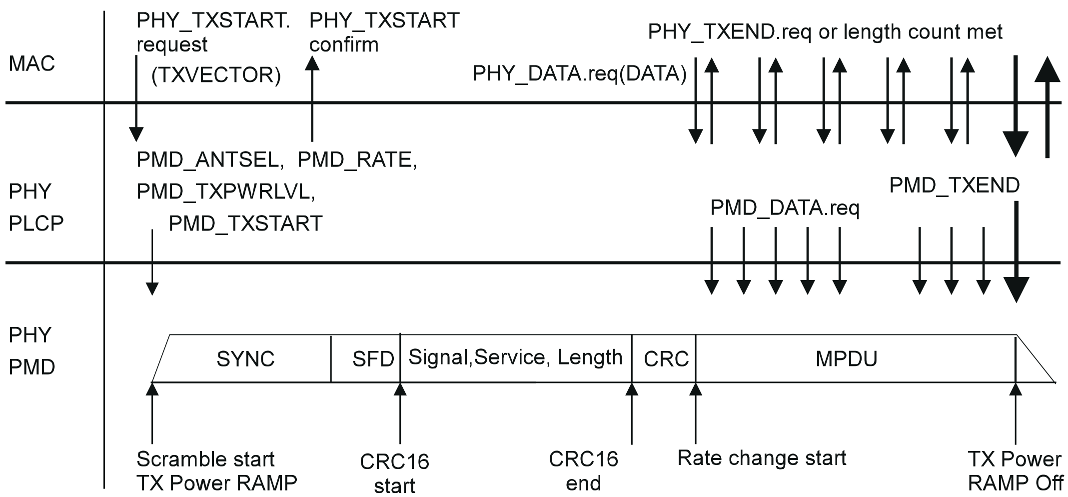


Figure 90—PLCP transmit procedure

Based on the status of clear channel assessment (CCA) indicated by PHY-CCA.indicate, the MAC will assess that the channel is clear. A clear channel shall be indicated by PHY-CCA.indicate(IDLE). If the channel is clear, transmission of the PPDU shall be initiated by issuing the PHY-TXSTART.request (TXVECTOR) primitive. The TXVECTOR elements for the PHY-TXSTART.request are the PLCP Header parameters SIGNAL (DATARATE), SERVICE, and LENGTH, and the PMD parameters of TX_ANTENNA and TXPWR_LEVEL. The PLCP Header parameter LENGTH is calculated from the TXVECTOR element by multiplying by 8 for 1 Mbit/s and by 4 for 2 Mbit/s.

The PLCP shall issue PMD_ANTSEL, PMD_RATE, and PMD_TXPWRLVL primitives to configure the PHY. The PLCP shall then issue a PMD_TXSTART.request and the PHY entity shall immediately initiate data scrambling and transmission of the PLCP Preamble based on the parameters passed in the PHY-TXSTART.request primitive. The time required for TX power on ramp described in 15.4.7.7 shall be included in the PLCP synchronization field. Once the PLCP Preamble transmission is complete, data shall be exchanged between the MAC and the PHY by a series of PHY-DATA.request(DATA) primitives issued by the MAC and PHY-DATA.confirm primitives issued by the PHY. The modulation rate change, if any, shall be initiated with the first data symbol of the MPDU as described in 15.2.5. The PHY proceeds with MPDU transmission through a series of data octet transfers from the MAC. At the PMD layer, the data octets are sent in lsb to msb order and presented to the PHY layer through PMD_DATA.request primitives. Transmission can be prematurely terminated by the MAC through the primitive PHY-TXEND.request. PHY-TXSTART shall be disabled by the issuance of the PHY-TXEND.request. Normal termination occurs after the transmission of the final bit of the last MPDU octet according to the number supplied in the DSSS PHY preamble LENGTH field. The packet transmission shall be completed and the PHY entity shall enter the receive state (i.e., PHY-TXSTART shall be disabled). It is recommended that chipping continue during power-down. Each PHY-TXEND.request is acknowledged with a PHY-TXEND.confirm primitive from the PHY.

A typical state machine implementation of the PLCP transmit procedure is provided in Figure 91.

15.2.7 PLCP receive procedure

The PLCP receive procedure is shown in Figure 92.

In order to receive data, PHY-TXSTART.request shall be disabled so that the PHY entity is in the receive state. Further, through station management via the PLME, the PHY is set to the appropriate channel and the CCA method is chosen. Other receive parameters such as receive signal strength indication (RSSI), signal quality (SQ), and indicated DATARATE may be accessed via the PHY-SAP.

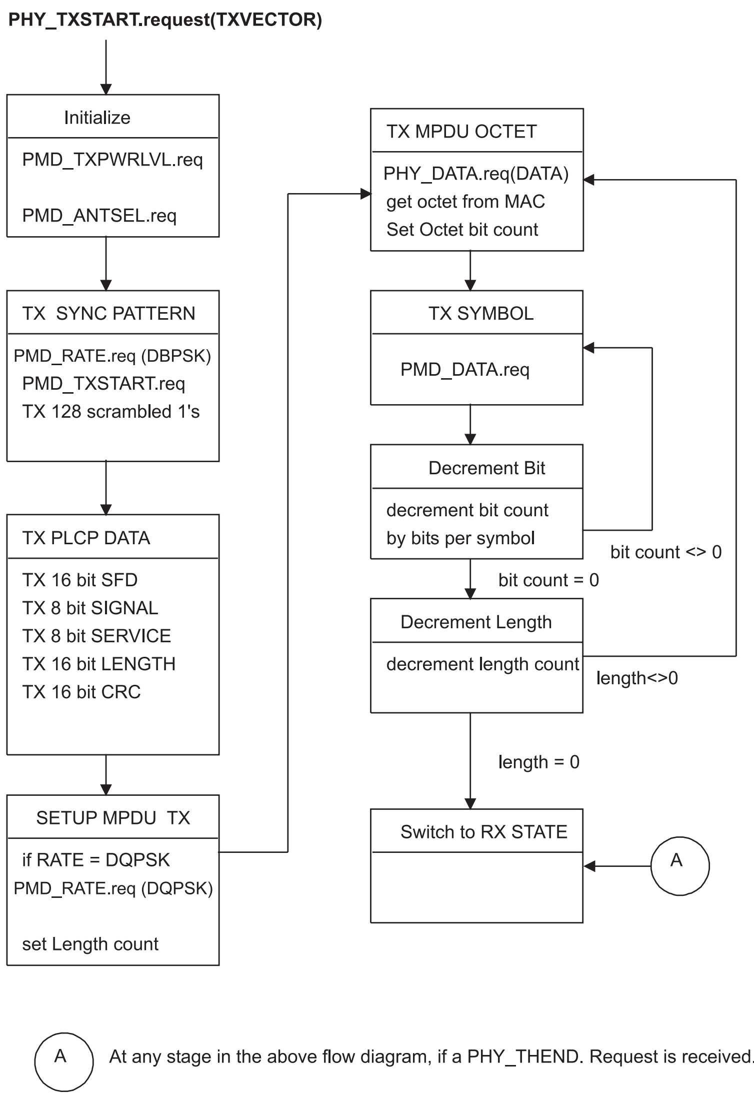


Figure 91—PLCP transmit state machine

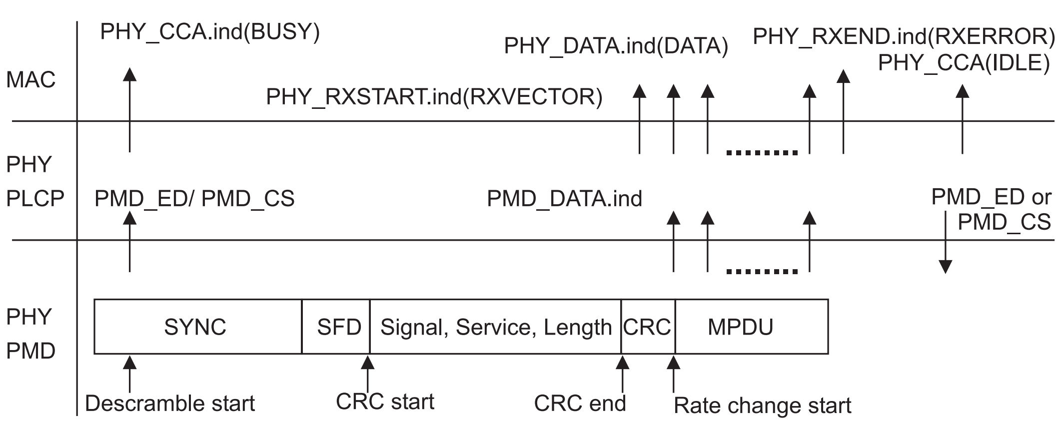


Figure 92—PLCP receive procedure

Upon receiving the transmitted energy, according to the selected CCA mode, the PMD_ED shall be enabled (according to 15.4.8.4) as the RSSI strength reaches the ED_THRESHOLD and/or PMD_CS shall be enabled after code lock is established. These conditions are used to indicate activity to the MAC via PHY-CCA.indicate according to 15.4.8.4. PHY-CCA.indicate(BUSY) shall be issued for energy detection and/or code lock prior to correct reception of the PLCP frame. The PMD primitives PMD_SQ and PMD_RSSI are issued to update the RSSI and SQ parameters reported to the MAC.

After PHY-CCA.indicate is issued, the PHY entity shall begin searching for the SFD field. Once the SFD field is detected, CCITT CRC-16 processing shall be initiated and the PLCP IEEE 802.11 SIGNAL, IEEE 802.11 SERVICE and LENGTH fields are received. The CCITT CRC-16 FCS shall be processed. If the CCITT CRC-16 FCS check fails, the PHY receiver shall return to the RX Idle state as depicted in Figure 93. Should the status of CCA return to the IDLE state during reception prior to completion of the full PLCP processing, the PHY receiver shall return to the RX Idle state.

If the PLCP Header reception is successful (and the SIGNAL field is completely recognizable and supported), a PHY-RXSTART.indicate(RXVECTOR) shall be issued. The RXVECTOR associated with this primitive includes the SIGNAL field, the SERVICE field, the MPDU length in bytes (calculated from the LENGTH field in microseconds), the antenna used for receive (RX_ANTENNA), RSSI, and SQ.

The received MPDU bits are assembled into octets and presented to the MAC using a series of PHY-DATA.indicate(DATA) primitive exchanges. The rate change indicated in the IEEE 802.11 SIGNAL field shall be initiated with the first symbol of the MPDU as described in 15.2.5. The PHY proceeds with MPDU reception. After the reception of the final bit of the last MPDU octet indicated by the PLCP Preamble LENGTH field, the receiver shall be returned to the RX Idle state as shown in Figure 93. A PHY-RXEND.indicate(NoError) primitive shall be issued. A PHY-CCA.indicate(IDLE) primitive shall be issued following a change in PHYCS (PHY carrier sense) and/or PHYED (PHY energy detection) according to the selected CCA method.

In the event that a change in PHYCS or PHYED would cause the status of CCA to return to the IDLE state before the complete reception of the MPDU as indicated by the PLCP LENGTH field, the error condition PHY-RXEND.indicate(CarrierLost) shall be reported to the MAC. The DSSS PHY will ensure that the CCA will indicate a busy medium for the intended duration of the transmitted packet.

If the PLCP Header is successful, but the indicated rate in the SIGNAL field is not receivable, a PHY-RXSTART.indicate will not be issued. The PHY shall issue the error condition PHY-RXEND.indicate(UnsupportedRate). If the PLCP Header is successful, but the SERVICE field is out of IEEE 802.11 DSSS specification, a PHY-RXSTART.indicate will not be issued. The PHY shall issue the error condition PHY-RXEND.indicate(FormatViolation). Also, in both cases, the DSSS PHY will ensure that the CCA shall indicate a busy medium for the intended duration of the transmitted frame as indicated by the Length field. The intended duration is indicated by the Length field (length $\times$ 1 μs).

A typical state machine implementation of the PLCP receive procedure is provided in Figure 93.

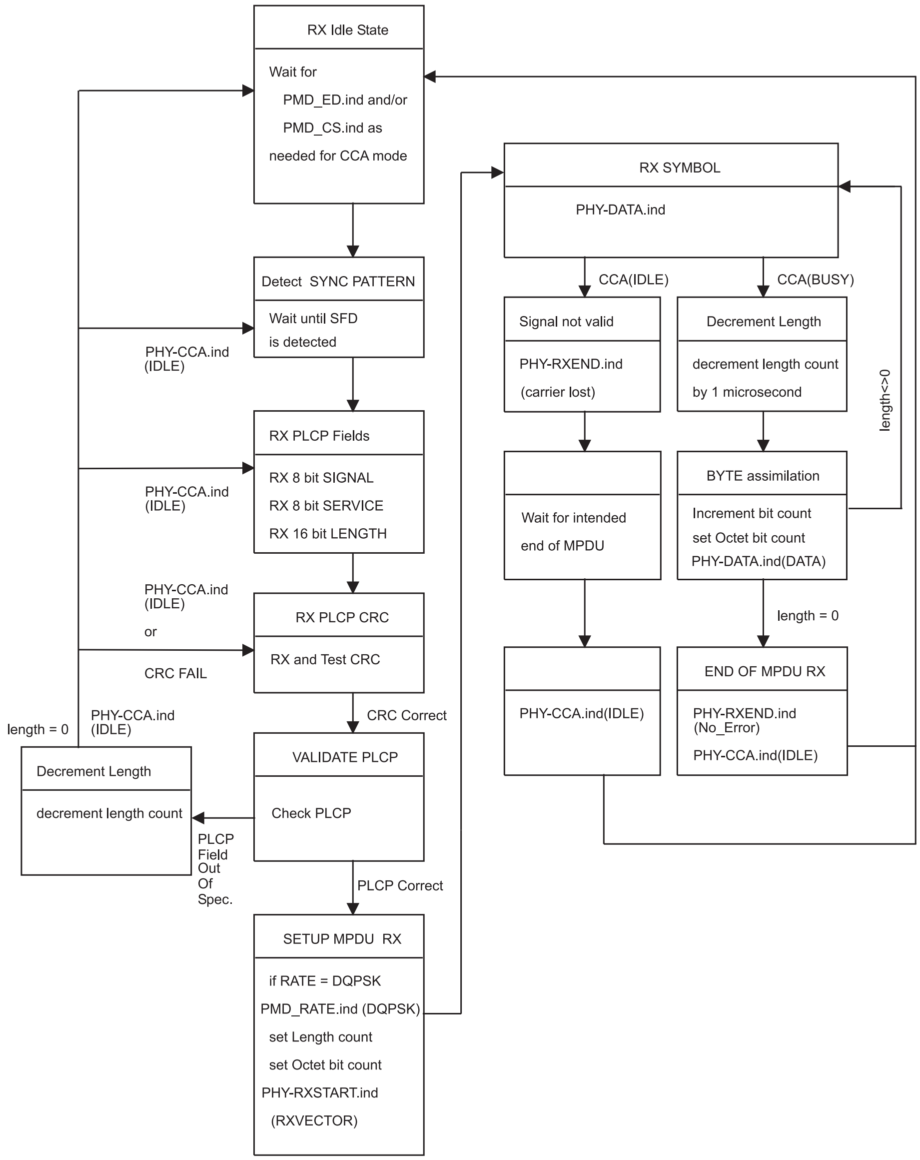


Figure 93—PLCP receive state machine

15.3 DSSS physical layer management entity (PLME)

15.3.1 PLME_SAP sublayer management primitives

Table 58 lists the MIB attributes that may be accessed by the PHY sublayer entities and intralayer of higher layer management entities (LME). These attributes are accessed via the PLME-GET, PLME-SET, and PLME-RESET primitives defined in Clause 10.

15.3.2 DSSS PHY MIB

All DSSS PHY MIB attributes are defined in Clause 12, with specific values defined in Table 58.

Table 58—MIB attribute default values/ranges

Managed object	Default value/range	Operational semantics
agPhyOperationGroup		
aPHYType	DSSS–2.4 (02)	Static
aTempType	Implementation dependent	Static
aCWmin	31	Static
aCWmax	1023	Static
aRegDomainsSupported	Implementation dependent	Static
aCurrentRegDomain	Implementation dependent	Static
aSlotTime	20 μs	Static
aCCATime	≤15 μs	Static
aRxTxTurnaroundTime	≤5 μs	Static
aTxPLCPDelay	Implementation dependent	Static
aRxTxSwitchTime	≤5 μs	Static
aTxRampOnTime	Implementation dependent	Static
aTxRFDelay	Implementation dependent	Static
aSIFSTime	10 μs	Static
aRxRFDelay	Implementation dependent	Static
aRxPLCPDelay	Implementation dependent	Static
aMACProcessingDelay	Not applicable	n/a
aTxRampOffTime	Implementation dependent	Static
aPreambleLength	144 bits	Static
aPLCPHeaderLength	48 bits	Static
agPhyRateGroup		
aSupportedDataRatesTx	X'02', X'04'	Static
aSupportedDataRatesRx	X'02', X'04'	Static
aMPDUMaxLength	$4 \le x \le (2^{13} - 1)$	Static
agPhyAntennaGroup		
aCurrentTxAntenna	Implementation dependent	Dynamic
aDiversitySupport	Implementation dependent	Static
agPhyTxPowerGroup		
aNumberSupportedPowerLevels	Implementation dependent	Static
aTxPowerLevel1	Implementation dependent	Static
aTxPowerLevel2	Implementation dependent	Static
aTxPowerLevel3	Implementation dependent	Static
aTxPowerLevel4	Implementation dependent	Static
aTxPowerLevel5	Implementation dependent	Static
aTxPowerLevel6	Implementation dependent	Static
aTxPowerLevel7	implementation dependent	Static
aTxPowerLevel8	Implementation dependent	Static
aCurrentTxPowerLevel	Implementation dependent	Dynamic

Table 58—MIB attribute default values/ranges (continued)

Managed object	Default value/range	Operational semantics
agPhyStatusGroup		
aSynthesizerLocked	Implementation dependent	Dynamic
agPhyDSSSGroup		
aCurrentChannel	Implementation dependent	Dynamic
aCCAModeSupport	Implementation dependent	Static
aCurrentCCAMode	Implementation dependent	Dynamic
aEDThreshold	Implementation dependent	Dynamic
agPhyPwrSavingGroup		
aDozeTurnonTime	Implementation dependent	Static
aCurrentPowerState	Implementation dependent	Dynamic
agAntennasListGroup		
aSupportTxAntennas	Implementation dependent	Static
aSupportRxAntennas	Implementation dependent	Static
aDiversitySelectRx	Implementation dependent	Dynamic
NOTE—The column titled “Operational semantics” contains two types: static and dynamic. Static MIB attributes are fixed and cannot be modified for a given PHY implementation. MIB attributes defined as dynamic can be modified by some management entities.		

15.4 DSSS PMD sublayer

15.4.1 Scope and field of application

This subclause describes the PMD services provided to the PLCP for the DSSS PHY. Also defined in this subclause are the functional, electrical, and RF characteristics required for interoperability of implementations conforming to this specification. The relationship of this specification to the entire DSSS physical layer is shown in Figure 94.

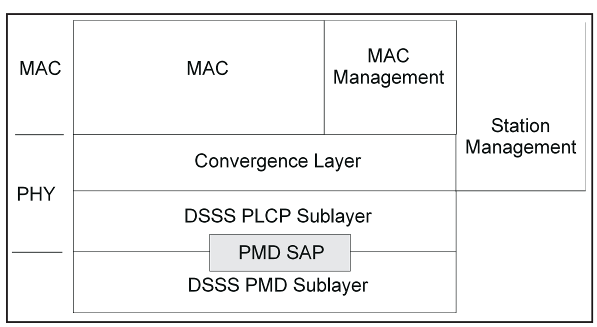


Figure 94—PMD layer reference model

15.4.2 Overview of service

The DSSS PMD sublayer accepts PLCP sublayer service primitives and provides the actual means by which data shall be transmitted or received from the medium. The combined function of DSSS PMD sublayer primitives and parameters for the receive function results in a data stream, timing information, and associated received signal parameters being delivered to the PLCP sublayer. A similar functionality shall be provided for data transmission.

15.4.3 Overview of interactions

The primitives associated with the IEEE 802.11 PLCP sublayer to the DSSS PMD fall into two basic categories:

a) Service primitives that support PLCP peer-to-peer interactions, and
b) Service primitives that have local significance and that support sublayer-to-sublayer interactions.

15.4.4 Basic service and options

All of the service primitives described in this clause are considered mandatory unless otherwise specified.

15.4.4.1 PMD_SAP peer-to-peer service primitives

Table 59 indicates the primitives for peer-to-peer interactions.

Table 59—PMD_SAP peer-to-peer service primitives

Primitive	Request	Indicate	Confirm	Response
PHY-RXSTART		X		
PHY-RXEND		X		
PHY-CCA		X		
PHY-TXSTART	X		X	
PHY-TXEND	X		X	
PHY-DATA	X	X	X	

15.4.4.2 PMD_SAP peer-to-peer service primitive parameters

Several service primitives include a parameter vector. This vector shall be actually a list of parameters that may vary depending on PHY type. Table 60 indicates the parameters required by the MAC or DSSS PHY in each of the parameter vectors used for peer-to-peer interactions.

Table 60—DSSS PMD_SAP peer-to-peer service primitives

Parameter	Associated primitive	Value
LENGTH	RXVECTOR, TXVECTOR	4 to $2^{16} - 1$
DATARATE	RXVECTOR, TXVECTOR	PHY dependent
SERVICE	RXVECTOR, TXVECTOR	PHY dependent
TXPWR_LEVEL	TXVECTOR	PHY dependent
TX_ANTENNA	TXVECTOR	PHY dependent
RSSI	RXVECTOR	PHY dependent
SQ	RXVECTOR	PHY dependent
RX_ANTENNA	RXVECTOR	PHY dependent

15.4.4.3 PMD_SAP sublayer-to-sublayer service primitives

Table 61 indicates the primitives for sublayer-to-sublayer interactions.

Table 61—PMD_SAP sublayer-to-sublayer service primitives

Primitive	Request	Indicate	Confirm	Response
PMD_TXSTART	X			
PMD_TXEND	X			
PMD_ANTSEL	X	X		
PMD_TXPWRLVL	X			
PMD_RATE	X	X		
PMD_RSSI		X		
PMD_SQ		X		
PMD_CS		X		
PMD_ED	X	X		

15.4.4.4 PMD_SAP service primitive parameters

Table 62 indicates the parameters for the PMD primitives.

Table 62—List of parameters for the PMD primitives

Parameter	Associate primitive	Value
DATA	PHY-DATA.request PHY-DATA.indicate	Octet value: X'00'–X'FF'
TXVECTOR	PHY-DATA.request	A set of parameters
RXVECTOR	PHY-DATA.indicate	A set of parameters
TXD_UNIT	PMD_DATA.request	One(1), Zero(0): DBPSK dibit combinations 00,01,11,10: DQPSK
RXD_UNIT	PMD_DATA.indicate	One(1), Zero(0): DBPSK dibit combinations 00,01,11,10: DQPSK
RF_STATE	PMD_TXE.request	Receive, Transmit
ANT_STATE	PMD_ANTSEL.indicate PMD_ANTSEL.request	1 to 256
TXPWR_LEVEL	PHY-TXSTART	0, 1, 2, 3 (max of 4 levels)
RATE	PMD_RATE.indicate PMD_RATE.request	X'0A' for 1 Mbit/s DBPSK X'14' for 2 Mbit/s DQPSK
RSSI	PMD_RSSI.indicate	0–8 bits of RSSI
SQ	PMD_SQ.indicate	0–8 bits of SQ

15.4.5 PMD_SAP detailed service specification

The following subclauses describe the services provided by each PMD primitive.

15.4.5.1 PMD_DATA.request

15.4.5.1.1 Function

This primitive defines the transfer of data from the PLCP sublayer to the PMD entity.

15.4.5.1.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.request(TXD_UNIT)

The TXD_UNIT parameter takes on the value of either one(1) or zero(0) for DBPSK modulation or the dibit combination 00, 01, 11, or 10 for DQPSK modulation. This parameter represents a single block of data, which, in turn, shall be used by the PHY to be differentially encoded into a DBPSK or DQPSK transmitted symbol. The symbol itself shall be spread by the PN code prior to transmission.

15.4.5.1.3 When generated

This primitive shall be generated by the PLCP sublayer to request transmission of a symbol. The data clock for this primitive shall be supplied by PMD layer based on the PN code repetition.

15.4.5.1.4 Effect of receipt

The PMD performs the differential encoding, PN code modulation and transmission of the data.

15.4.5.2 PMD_DATA.indicate

15.4.5.2.1 Function

This primitive defines the transfer of data from the PMD entity to the PLCP sublayer.

15.4.5.2.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_DATA.indicate(RXD_UNIT)

The RXD_UNIT parameter takes on the value of one(1) or zero(0) for DBPSK modulation or as the dibit 00, 01, 11, or 10 for DQPSK modulation. This parameter represents a single symbol that has been demodulated by the PMD entity.

15.4.5.2.3 When generated

This primitive, which is generated by the PMD entity, forwards received data to the PLCP sublayer. The data clock for this primitive shall be supplied by PMD layer based on the PN code repetition.

15.4.5.2.4 Effect of receipt

The PLCP sublayer either interprets the bit or bits that are recovered as part of the PLCP convergence procedure or passes the data to the MAC sublayer as part of the MPDU.

15.4.5.3 PMD_TXSTART.request

15.4.5.3.1 Function

This primitive, which is generated by the PHY PLCP sublayer, initiates PPDU transmission by the PMD layer.

15.4.5.3.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXSTART.request

15.4.5.3.3 When generated

This primitive shall be generated by the PLCP sublayer to initiate the PMD layer transmission of the PPDU. The PHY-DATA.request primitive shall be provided to the PLCP sublayer prior to issuing the PMD_TXSTART command.

15.4.5.3.4 Effect of receipt

PMD_TXSTART initiates transmission of a PPDU by the PMD sublayer.

15.4.5.4 PMD_TXEND.request

15.4.5.4.1 Function

This primitive, which is generated by the PHY PLCP sublayer, ends PPDU transmission by the PMD layer.

15.4.5.4.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXEND.request

15.4.5.4.3 When generated

This primitive shall be generated by the PLCP sublayer to terminate the PMD layer transmission of the PPDU.

15.4.5.4.4 Effect of receipt

PMD_TXEND terminates transmission of a PPDU by the PMD sublayer.

15.4.5.5 PMD_ANTSEL.request

15.4.5.5.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the antenna used by the PHY for transmission or reception (when diversity is disabled).

15.4.5.5.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ANTSEL.request(ANT_STATE)

ANT_STATE selects which of the available antennas should be used for transmit. The number of available antennas shall be determined from the MIB table parameters aSuprtRxAntennas and aSuprtTxAntennas.

15.4.5.5.3 When generated

This primitive shall be generated by the PLCP sublayer to select a specific antenna for transmission (or reception when diversity is disabled).

15.4.5.5.4 Effect of receipt

PMD_ANTSEL immediately selects the antenna specified by ANT_STATE.

15.4.5.6 PMD_ANTSEL.indicate

15.4.5.6.1 Function

This primitive, which is generated by the PHY PLCP sublayer, reports the antenna used by the PHY for reception of the most recent packet.

15.4.5.6.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ANTSEL.indicate(ANT_STATE)

ANT_STATE reports which of the available antennas was used for reception of the most recent packet.

15.4.5.6.3 When generated

This primitive shall be generated by the PLCP sublayer to report the antenna used for the most recent packet reception.

15.4.5.6.4 Effect of receipt

PMD_ANTSEL immediately reports the antenna specified by ANT_STATE.

15.4.5.7 PMD_TXPWRLVL.request

15.4.5.7.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the power level used by the PHY for transmission.

15.4.5.7.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_TXPWRLVL.request(TXPWR_LEVEL)

TXPWR_LEVEL selects which of the optional transmit power levels should be used for the current packet transmission. The number of available power levels shall be determined by the MIB parameter aNumberSupportedPowerLevels. Subclause 15.4.7.3 provides further information on the optional DSSS PHY power level control capabilities.

15.4.5.7.3 When generated

This primitive shall be generated by the PLCP sublayer to select a specific transmit power. This primitive shall be applied prior to setting PMD_TXSTART into the transmit state.

15.4.5.7.4 Effect of receipt

PMD_TXPWRLVL immediately sets the transmit power level given by TXPWR_LEVEL.

15.4.5.8 PMD_RATE.request

15.4.5.8.1 Function

This primitive, which is generated by the PHY PLCP sublayer, selects the modulation rate that shall be used by the DSSS PHY for transmission.

15.4.5.8.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RATE.request(RATE)

RATE selects which of the DSSS PHY data rates shall be used for MPDU transmission. Subclause 15.4.6.4 provides further information on the DSSS PHY modulation rates. The DSSS PHY rate change capability is fully described in 15.2.

15.4.5.8.3 When generated

This primitive shall be generated by the PLCP sublayer to change or set the current DSSS PHY modulation rate used for the MPDU portion of a PPDU.

15.4.5.8.4 Effect of receipt

The receipt of PMD_RATE selects the rate that shall be used for all subsequent MPDU transmissions. This rate shall be used for transmission only. The DSSS PHY shall still be capable of receiving all the required DSSS PHY modulation rates.

15.4.5.9 PMD_RATE.indicate

15.4.5.9.1 Function

This primitive, which is generated by the PMD sublayer, indicates which modulation rate was used to receive the MPDU portion of the PPDU. The modulation shall be indicated in the PLCP Preamble IEEE 802.11 SIGNALING field.

15.4.5.9.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RATE.indicate(RATE)

In receive mode, the RATE parameter informs the PLCP layer which of the DSSS PHY data rates was used to process the MPDU portion of the PPDU. Subclause 15.4.6.4 provides further information on the DSSS PHY modulation rates. The DSSS PHY rate change capability is fully described in 15.2.

15.4.5.9.3 When generated

This primitive shall be generated by the PMD sublayer when the PLCP Preamble IEEE 802.11 SIGNALING field has been properly detected.

15.4.5.9.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only.

15.4.5.10 PMD_RSSI.indicate

15.4.5.10.1 Function

This optional primitive, which is generated by the PMD sublayer, provides to the PLCP and MAC entity the received signal strength.

15.4.5.10.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_RSSI.indicate(RSSI)

The RSSI shall be a measure of the RF energy received by the DSSS PHY. RSSI indications of up to 8 bits (256 levels) are supported.

15.4.5.10.3 When generated

This primitive shall be generated by the PMD when the DSSS PHY is in the receive state. It shall be continuously available to the PLCP, which, in turn, provides the parameter to the MAC entity.

15.4.5.10.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only. The RSSI may be used in conjunction with SQ as part of a CCA scheme.

15.4.5.11 PMD_SQ.indicate

15.4.5.11.1 Function

This optional primitive, which is generated by the PMD sublayer, provides to the PLCP and MAC entity the signal quality (SQ) of the DSSS PHY PN code correlation. The SQ shall be sampled when the DSSS PHY achieves code lock and held until the next code lock acquisition.

15.4.5.11.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_SQ.indicate(SQ)

The SQ shall be a measure of the PN code correlation quality received by the DSSS PHY. SQ indications of up to 8 bits (256 levels) are supported.

15.4.5.11.3 When generated

This primitive shall be generated by the PMD when the DSSS PHY is in the receive state and code lock is achieved. It shall be continuously available to the PLCP, which, in turn, provides the parameter to the MAC entity.

15.4.5.11.4 Effect of receipt

This parameter shall be provided to the PLCP layer for information only. The SQ may be used in conjunction with RSSI as part of a CCA scheme.

15.4.5.12 PMD_CS.indicate

This primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has acquired (locked) the PN code and data is being demodulated.

15.4.5.12.1 Function

This primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has acquired (locked) the PN code and data is being demodulated.

15.4.5.12.2 Semantics of the service primitive

The PMD_CS (carrier sense) primitive in conjunction with PMD_ED provide CCA status through the PLCP layer PHYCCA primitive. PMD_CS indicates a binary status of ENABLED or DISABLED. PMD_CS shall be ENABLED when the correlator SQ indicated in PMD_SQ is greater than the CS_THRESHOLD parameter. PMD_CS shall be DISABLED when the PMD_SQ falls below the correlation threshold.

15.4.5.12.3 When generated

This primitive shall be generated by the PHY sublayer when the DSSS PHY is receiving a PPDU and the PN code has been acquired.

15.4.5.12.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PHYCCA indicator. This parameter shall indicate that the RF medium is busy and occupied by a

DSSS PHY signal. The DSSS PHY should not be placed into the transmit state when PMD_CS is ENABLED.

15.4.5.13 PMD_ED.indicate

15.4.5.13.1 Function

This optional primitive, which is generated by the PMD, shall indicate to the PLCP layer that the receiver has detected RF energy indicated by the PMD_RSSI primitive that is above a predefined threshold.

15.4.5.13.2 Semantics of the service primitive

The PMD_ED (energy detect) primitive, along with the PMD_SQ, provides CCA status at the PLCP layer through the PHYCCA primitive. PMD_ED indicates a binary status of ENABLED or DISABLED. PMD_ED shall be ENABLED when the RSSI indicated in PMD_RSSI is greater than the ED_THRESHOLD parameter. PMD_ED shall be DISABLED when the PMD_RSSI falls below the energy detect threshold.

15.4.5.13.3 When generated

This primitive shall be generated by the PHY sublayer when the PHY is receiving RF energy from any source that exceeds the ED_THRESHOLD parameter.

15.4.5.13.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PMD_ED indicator. This parameter shall indicate that the RF medium may be busy with an RF energy source that is not DSSS PHY compliant. If a DSSS PHY source is being received, the PMD_CS function shall be enabled shortly after the PMD_ED function is enabled.

15.4.5.14 PMD_ED.request

15.4.5.14.1 Function

This optional primitive, which is generated by the PHY PLCP, sets the energy detect ED THRESHOLD value.

15.4.5.14.2 Semantics of the service primitive

The primitive shall provide the following parameters:

PMD_ED.request(ED_THRESHOLD)

ED_THRESHOLD sets the threshold that the RSSI indicated shall be greater than in order for PMD_ED to be enabled.

15.4.5.14.3 When generated

This primitive shall be generated by the PLCP sublayer to change or set the current DSSS PHY energy detect threshold.

15.4.5.14.4 Effect of receipt

The receipt of PMD_ED immediately changes the energy detection threshold as set by the ED_THRESHOLD parameter.

15.4.5.15 PHY-CCA.indicate

15.4.5.15.1 Function

This primitive, which is generated by the PMD, indicates to the PLCP layer that the receiver has detected RF energy that adheres to the CCA algorithm.

15.4.5.15.2 Semantics of the service primitive

The PHY-CCA primitive provides CCA status at the PLCP layer to the MAC.

15.4.5.15.3 When generated

This primitive shall be generated by the PHY sublayer when the PHY is receiving RF energy from any source that exceeds the ED_THRESHOLD parameter (PMD_ED is active), and optionally is a valid correlated DSSS PHY signal whereby PMD_CS would also be active.

15.4.5.15.4 Effect of receipt

This indicator shall be provided to the PLCP for forwarding to the MAC entity for information purposes through the PHY-CCA indicator. This parameter indicates that the RF medium may be busy with an RF energy source that may or may not be DSSS PHY compliant. If a DSSS PHY source is being received, the PMD_CS function shall be enabled shortly after the PMD_ED function is enabled.

15.4.6 PMD operating specifications, general

The following subclauses provide general specifications for the DSSS PMD sublayer. These specifications apply to both the Receive and the Transmit functions and general operation of a DSSS PHY.

15.4.6.1 Operating frequency range

The DSSS PHY shall operate in the frequency range of 2.4 GHz to 2.4835 GHz as allocated by regulatory bodies in the USA and Europe or in the 2.471 GHz to 2.497 GHz frequency band as allocated by regulatory authority in Japan.

15.4.6.2 Number of operating channels

The channel center frequencies and CHNL_ID numbers shall be as shown in Table 63. The FCC (US), IC (Canada), and ETSI (Europe) specify operation from 2.4 GHz to 2.4835 GHz. For Japan, operation is specified as 2.471 GHz to 2.497 GHz. France allows operation from 2.4465 GHz to 2.4835 GHz, and Spain allows operation from 2.445 GHz to 2.475 GHz. For each supported regulatory domain, all channels in Table 63 marked with "X" shall be supported.

Table 63—DSSS PHY frequency channel plan

		Regulatory domains					
CHNL_ID	Frequency	X'10' FCC	X'20' IC	X'30' ETSI	X'31' Spain	X'32' France	X'40' MKK
1	2412 MHz	X	X	X	—	—	—
2	2417 MHz	X	X	X	—	—	—
3	2422 MHz	X	X	X	—	—	—
4	2427 MHz	X	X	X	—	—	—
5	2432 MHz	X	X	X	—	—	—
6	2437 MHz	X	X	X	—	—	—
7	2442 MHz	X	X	X	—	—	—
8	2447 MHz	X	X	X	—	—	—
9	2452 MHz	X	X	X	—	—	—
10	2457 MHz	X	X	X	X	X	—
11	2462 MHz	X	X	X	X	X	—
12	2467 MHz	—	—	X	—	X	—
13	2472 MHz	—	—	X	—	X	—
14	2484 MHz	—	—	—	—	—	X

In a multiple cell network topology, overlapping and/or adjacent cells using different channels can operate simultaneously without interference if the distance between the center frequencies is at least 30 MHz. Channel 14 shall be designated specifically for operation in Japan.

15.4.6.3 Spreading sequence

The following 11-chip Barker sequence shall be used as the PN code sequence:

+1, –1, +1, +1, –1, +1, +1, +1, –1, –1, –1

The leftmost chip shall be output first in time. The first chip shall be aligned at the start of a transmitted symbol. The symbol duration shall be exactly 11 chips long.

15.4.6.4 Modulation and channel data rates

Two modulation formats and data rates are specified for the DSSS PHY: a *basic access rate* and an *enhanced access rate*. The basic access rate shall be based on 1 Mbit/s DBPSK modulation. The DBPSK encoder is specified in Table 64. The enhanced access rate shall be based on 2 Mbit/s DQPSK. The DQPSK encoder is specified in Table 65. (In the tables, +jω shall be defined as counterclockwise rotation.)

Table 64—1 Mbit/s DBPSK encoding table

Bit input	Phase change (+jω)
0	0
1	π

Table 65—2 Mbit/s DQPSK encoding table

Dibit pattern (d0,d1) d0 is first in time	Phase change (+jω)
00	0
01	π/2
11	π
10	3π/2 (–π/2)

15.4.6.5 Transmit and receive in-band and out-of-band spurious emissions

The DSSS PHY shall conform with in-band and out-of-band spurious emissions as set by regulatory bodies. For the USA, refer to FCC 15.247, 15.205, and 15.209. For Europe, refer to ETS 300–328.

15.4.6.6 Transmit-to-receive turnaround time

The TX-to-RX turnaround time shall be less than 10 μs, including the power-down ramp specified in 15.4.7.7.

The TX-to-RX turnaround time shall be measured at the air interface from the trailing edge of the last transmitted symbol to valid CCA detection of the incoming signal. The CCA should occur within 25 μs (10 μs for turnaround time plus 15 μs for energy detect) or by the next slot boundary occurring after the 25 μs has elapsed (refer to 15.4.8.4). A receiver input signal 3 dB above the ED threshold described in 15.4.8.4 shall be present at the receiver.

15.4.6.7 Receive-to-transmit turnaround time

The RX-to-TX turnaround time shall be measured at the MAC/PHY interface, using PHYTXSTART.request and shall be ≤5 μs. This includes the transmit power up ramp described in 15.4.7.7.

15.4.6.8 Slot time

The slot time for the DSSS PHY shall be the sum of the RX-to-TX turnaround time (5 μs) and the energy detect time (15 μs specified in 15.4.8.4). The propagation delay shall be regarded as being included in the energy detect time.

15.4.6.9 Transmit and receive antenna port impedance

The impedance of the transmit and receive antenna port(s) shall be 50 Ω if the port is exposed.

15.4.6.10 Transmit and receive operating temperature range

Three temperature ranges for full operation compliance to the DSSS PHY are specified in Clause 13. Type 1 shall be defined as 0 °C to 40 °C, and is designated for office environments. Type 2 shall be defined as –20 °C to +50 °C, and Type 3 shall be defined as –30 °C to +70 °C. These are designated for industrial environments.

15.4.7 PMD transmit specifications

The following subclauses describe the transmit functions and parameters associated with the PMD sublayer.

15.4.7.1 Transmit power levels

The maximum allowable output power as measured in accordance with practices specified by the regulatory bodies is shown in Table 66. In the USA, the radiated emissions should also conform with the ANSI uncontrolled radiation emission standards (IEEE Std C95.1-1991).

Table 66—Transmit power levels

Maximum output power	Geographic location	Compliance document
1000 mW	USA	FCC 15.247
100 mW (EIRP)	Europe	ETS 300–328
10 mW/MHz	Japan	MPT ordinance for Regulating Radio Equipment, Article 49-20

15.4.7.2 Minimum transmitted power level

The minimum transmitted power shall be no less than 1 mW.

15.4.7.3 Transmit power level control

Power control shall be provided for transmitted power greater than 100 mW. A maximum of four power levels may be provided. At a minimum, a radio capable of transmission greater than 100 mW shall be capable of switching power back to 100 mW or less.

15.4.7.4 Transmit spectrum mask

The transmitted spectral products shall be less than –30 dBr (dB relative to the SINx/x peak) for f_c – 22 MHz $< f < f_c$ –11 MHz, f_c +11 MHz $< f < f_c$ + 22 MHz, –50 dBr for $f < f_c$ –22 MHz, and $f > f_c$ + 22 MHz, where f_c is the channel center frequency. The transmit spectral mask is shown in Figure 95. The measurements shall be made using 100 kHz resolution bandwidth and a 30 kHz video bandwidth.

15.4.7.5 Transmit center frequency tolerance

The transmitted center frequency tolerance shall be ±25 ppm maximum.

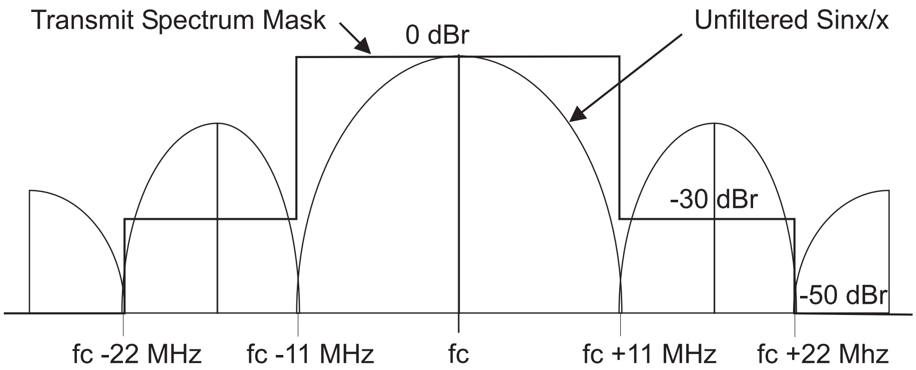


Figure 95—Transmit spectrum mask

15.4.7.6 Chip clock frequency tolerance

The PN code chip clock frequency tolerance shall be better than ±25 ppm maximum.

15.4.7.7 Transmit power-on and power-down ramp

The transmit power-on ramp for 10% to 90% of maximum power shall be no greater than 2 µs. The transmit power-on ramp is shown in Figure 96.

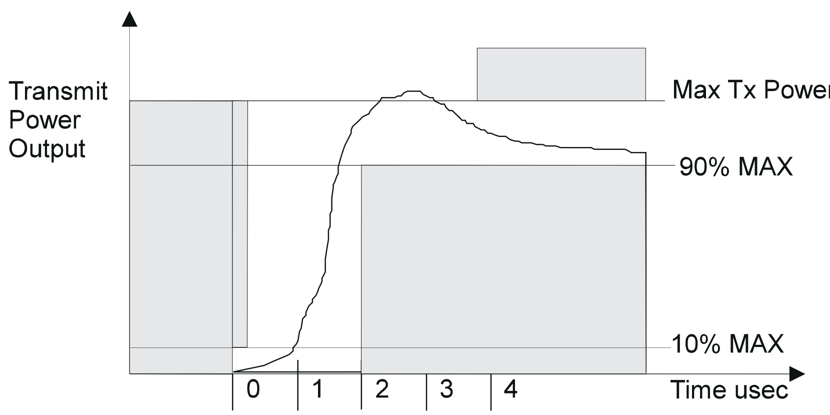


Figure 96—Transmit power-on ramp

The transmit power-down ramp for 90% to 10% maximum power shall be no greater than 2 µs. The transmit power down ramp is shown in Figure 97.

The transmit power ramps shall be constructed such that the DSSS PHY emissions conform with spurious frequency product specification defined in 15.4.6.5.

15.4.7.8 RF carrier suppression

The RF carrier suppression, measured at the channel center frequency, shall be at least 15 dB below the peak SIN(x)/x power spectrum. The RF carrier suppression shall be measured while transmitting a repetitive 01 data sequence with the scrambler disabled using DQPSK modulation. A 100 kHz resolution bandwidth shall be used to perform this measurement.

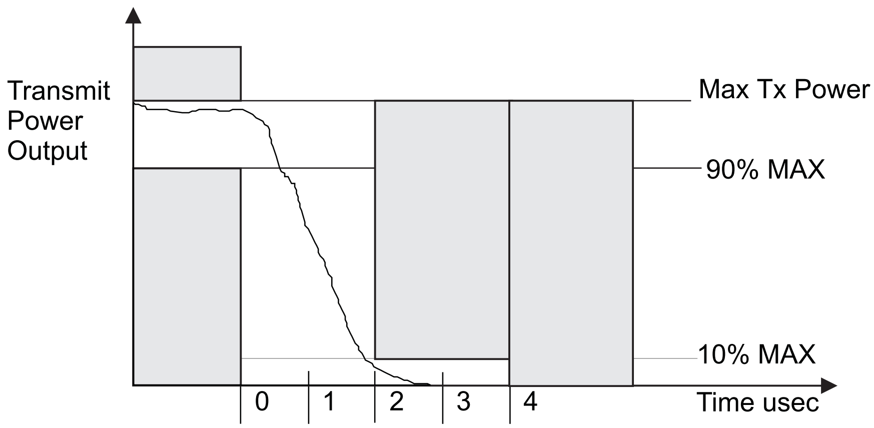


Figure 97—Transmit power-down ramp

15.4.7.9 Transmit modulation accuracy

The transmit modulation accuracy requirement for the DSSS PHY shall be based on the difference between the actual transmitted waveform and the ideal signal waveform. Modulation accuracy shall be determined by measuring the peak vector error magnitude measured during each chip period. Worst-Case vector error magnitude shall not exceeded 0.35 for the normalized sampled chip data. The ideal complex I and Q constellation points associated with DQPSK modulation (0.707,0.707), (0.707, –0.707), (–0.707, 0.707), (–0.707, –0.707) shall be used as the reference. These measurements shall be from baseband I and Q sampled data after recovery through a reference receiver system.

Figure 98 illustrates the ideal DQPSK constellation points and range of worst-case error specified for modulation accuracy.

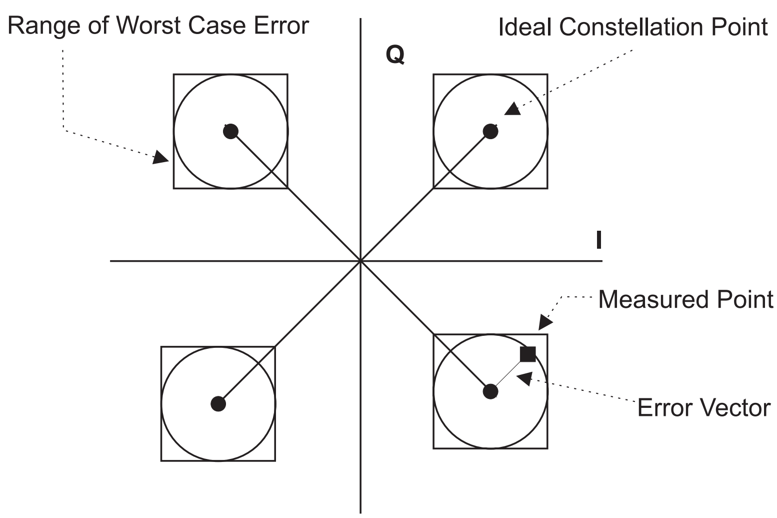


Figure 98—Modulation accuracy measurement example

Error vector measurement requires a reference receiver capable of carrier lock. All measurements shall be made under carrier lock conditions. The distortion induced in the constellation by the reference receiver shall be calibrated and measured. The test data error vectors described below shall be corrected to compensate for the reference receiver distortion.

The IEEE 802.11 vendor compatible radio shall provide an exposed TX chip clock, which shall be used to sample the I and Q outputs of the reference receiver.

The measurement shall be made under the conditions of continuous DQPSK transmission using scrambled all 1's.

The eye pattern of the I channel shall be used to determine the I and Q sampling point. The chip clock provided by the vendor radio shall be time delayed such that the samples fall at a 1/2 chip period offset from the mean of the zero crossing positions of the eye (see Figure 99). This is the ideal center of the eye and may not be the point of maximum eye opening.

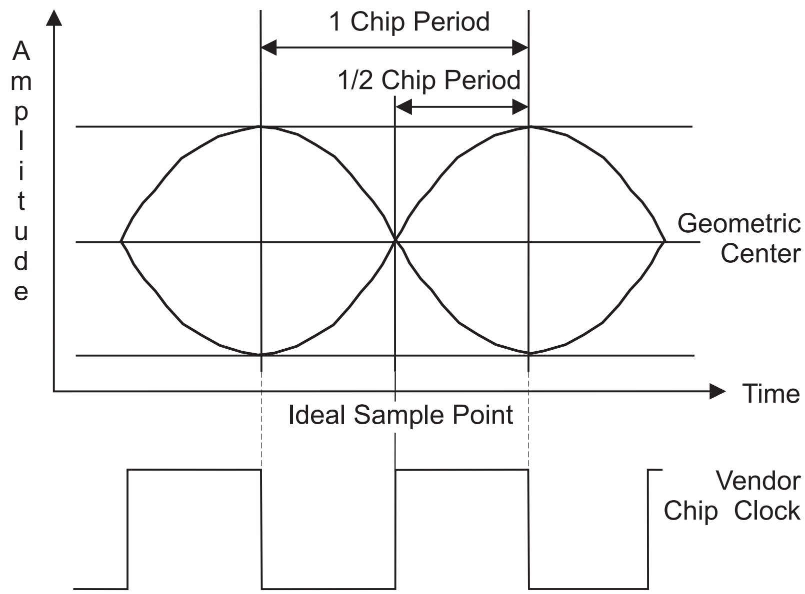


Figure 99—Chip clock alignment with baseband eye pattern

Using the aligned chip clock, 1000 samples of the *I* and *Q* baseband outputs from the reference receiver are captured. The vector error magnitudes shall be calculated as follows:

Calculate the dc offsets for *I* and *Q* samples.

$$I_{mean} = \sum_{n=0}^{1000} I(n)/1000$$

$$Q_{mean} = \sum_{n=0}^{1000} Q(n)/1000$$

Calculate the dc corrected *I* and *Q* samples for all *n* =1000 sample pairs.

$$I_{dc}(n) = I(n) - I_{mean}$$

$$Q_{dc}(n) = Q(n) - Q_{mean}$$

Calculate the average magnitude of *I* and *Q* samples.

$$I_{\text{mag}} = \sum_{n=0}^{1000} |I_{\text{dc}}(n)| / 1000$$

$$Q_{\text{mag}} = \sum_{n=0}^{1000} |Q_{\text{dc}}(n)| / 1000$$

Calculate the normalized error vector magnitude for the $I_{\text{dc}}(n)/Q_{\text{dc}}(n)$ pairs.

$$V_{\text{err}}(n) = \left[\frac{1}{2} \times (\{|I_{\text{dc}}(n)| / I_{\text{mag}}\}^2 + \{|Q_{\text{dc}}(n)| / Q_{\text{mag}}\}^2)\right]^{\frac{1}{2}} - V_{\text{correction}}$$

with $V_{\text{correction}}$ = error induced by the reference receiver system.

A vendor DSSS PHY implementation shall be compliant if for all n =1000 samples the following condition is met:

$V_{\text{err}}(n) < 0.35$

15.4.8 PMD receiver specifications

The following subclauses describe the receive functions and parameters associated with the PMD sublayer.

15.4.8.1 Receiver minimum input level sensitivity

The frame error ratio (FER) shall be less than 8×10^{-2} at an MPDU length of 1024 bytes for an input level of –80 dBm measured at the antenna connector. This FER shall be specified for 2 Mbit/s DQPSK modulation. The test for the minimum input level sensitivity shall be conducted with the energy detection threshold set less than or equal to –80 dBm.

15.4.8.2 Receiver maximum input level

The receiver shall provide a maximum FER of 8×10^{-2} at an MPDU length of 1024 bytes for a maximum input level of –4 dBm measured at the antenna. This FER shall be specified for 2 Mbit/s DQPSK modulation.

15.4.8.3 Receiver adjacent channel rejection

Adjacent channel rejection is defined between any two channels with ≥30 MHz separation in each channel group defined in 15.4.6.2.

The adjacent channel rejection shall be equal to or better than 35 dB with an FER of 8×10^{-2} using 2 Mbit/s DQPSK modulation described in 15.4.6.4 and an MPDU length of 1024 bytes.

The adjacent channel rejection shall be measured using the following method:

Input a 2 Mbit/s DQPSK modulated signal at a level 6 dB greater than specified in 15.4.8.1. In an adjacent channel (≥30 MHz separation as defined by the channel numbering), input a signal modulated in a similar fashion that adheres to the transmit mask specified in 15.4.7.4 to a level 41 dB above the level specified in 15.4.8.1. The adjacent channel signal shall be derived from a separate signal source. It cannot be a frequency shifted version of the reference channel. Under these conditions, the FER shall be no worse than 8×10^{-2}.

15.4.8.4 CCA

The DSSS PHY shall provide the capability to perform CCA according to at least one of the following three methods:

- *CCA Mode 1:* Energy above threshold. CCA shall report a busy medium upon detecting any energy above the ED threshold.
- *CCA Mode 2:* Carrier sense only. CCA shall report a busy medium only upon the detection of a DSSS signal. This signal may be above or below the ED threshold.
- *CCA Mode 3:* Carrier sense with energy above threshold. CCA shall report a busy medium upon the detection of a DSSS signal with energy above the ED threshold.

The energy detection status shall be given by the PMD primitive, PMD_ED. The carrier sense status shall be given by PMD_CS. The status of PMD_ED and PMD_CS is used in the PLCP convergence procedure to indicate activity to the MAC through the PHY interface primitive PHY-CCA.indicate.

A busy channel shall be indicated by PHY-CCA.indicate of class BUSY.

Clear channel shall be indicated by PHY-CCA.indicate of class IDLE.

The PHY MIB attribute aCCAModeSuprt shall indicate the appropriate operation modes. The PHY shall be configured through the PHY MIB attribute aCurrentCCAMode.

The CCA shall be TRUE if there is no energy detect or carrier sense. The CCA parameters are subject to the following criteria:

a) The energy detection threshold shall be less than or equal to –80 dBm for TX power > 100 mW, –76 dBm for 50 mW < TX power ≤ 100 mW, and –70 dBm for TX power ≤ 50 mW.
b) With a valid signal (according to the CCA mode of operation) present at the receiver antenna within 5 μs of the start of a MAC slot boundary, the CCA indicator shall report channel busy before the end of the slot time. This implies that the CCA signal is available as an exposed test point. Refer to Figure 47 for a definition of slot time boundary definition.
c) In the event that a correct PLCP Header is received, the DSSS PHY shall hold the CCA signal inactive (channel busy) for the full duration as indicated by the PLCP LENGTH field. Should a loss of carrier sense occur in the middle of reception, the CCA shall indicate a busy medium for the intended duration of the transmitted packet.

Conformance to DSSS PHY CCA shall be demonstrated by applying a DSSS compliant signal, above the appropriate ED threshold (a), such that all conditions described in b) and c) above are demonstrated.

16. Infrared (IR) PHY specification

16.1 Overview

The physical layer for the infrared system is specified in this clause. The IR PHY uses near-visible light in the 850 nm to 950 nm range for signaling. This is similar to the spectral usage of both common consumer devices such as infrared remote controls, as well as other data communications equipment, such as IrDA (Infrared Data Association) devices.

Unlike many other infrared devices, however, the IR PHY is not directed. That is, the receiver and transmitter do not have to be aimed at each other and do not need a clear line of sight. This permits the construction of a true LAN system, whereas with an aimed system, it would be difficult or impossible to install a LAN because of physical constraints.

A pair of conformant infrared devices would be able to communicate in a typical environment at a range up to about 10 m. The standard allows conformant devices to have more sensitive receivers, and this may increase range up to about 20 m.

The IR PHY relies on both reflected infrared energy as well as line-of-sight infrared energy for communications. Most designs anticipate that *all* of the energy at the receiver is reflected energy. This reliance on reflected infrared energy is called *diffuse infrared* transmission.

The standard specifies the transmitter and receiver in such a way that a conformant design will operate well in most environments where there is no line-of-sight path from the transmitter to the receiver. However, in an environment that has few or no reflecting surfaces, and where there is no line of sight, an IR PHY system may suffer reduced range.

The IR PHY will operate only in indoor environments. Infrared radiation does not pass through walls, and is significantly attenuated passing through most exterior windows. This characteristic can be used to “contain” an IR PHY in a single physical room, like a classroom or conference room. Different LANs using the IR PHY can operate in adjacent rooms separated only by a wall without interference, and without the possibility of eavesdropping.

At the time of this standard’s preparation, the only known regulatory standards that apply to the use of infrared radiation are safety regulations, such as IEC 60825-1 [B2] and ANSI Z136.1 [B1]. While a conformant IR PHY device can be designed to also comply with these safety standards, conformance with this standard does not ensure conformance with other standards.

Worldwide, there are currently no frequency allocation or bandwidth allocation regulatory restrictions on infrared emissions.

Emitter (typically LED) and detector (typically PIN diode) devices for infrared communications are relatively inexpensive at the infrared wavelengths specified in the IR PHY, and at the electrical operating frequencies required by this PHY.

While many other devices in common use also use infrared emissions in the same optical band, these devices usually transmit infrared intermittently and do not interfere with the proper operation of a compliant IR PHY. If such a device does interfere, by transmitting continuously and with a very strong signal, it can be physically isolated (placing it in a different room) from the IEEE 802.11 LAN.

16.1.1 Scope

The PHY services provided to the IEEE 802.11 wireless LAN MAC by the IR system are described in this clause. The IR PHY layer consists of two protocol functions as follows:

a) A physical layer convergence function, which adapts the capabilities of the physical medium dependent (PMD) system to the PHY service. This function is supported by the physical layer convergence procedure (PLCP), which defines a method of mapping the IEEE 802.11 MAC sublayer protocol data units (MPDU) into a framing format suitable for sending and receiving user data and management information between two or more STAs using the associated PMD system.
b) A PMD system, whose function defines the characteristics of, and method of transmitting and receiving data through, the wireless medium (WM) between two or more STAs.

16.1.2 IR PHY functions

The IR PHY contains three functional entities: the PMD function, the physical layer convergence function, and the layer management function. Each of these functions is described in detail below.

The IR PHY service is provided to the MAC entity at the STA through a service access point (SAP) as described in Clause 12. For a visual guide to the relationship of the IR PHY to the remainder of a system, refer to Figure 11.

16.1.2.1 PLCP sublayer

To allow the IEEE 802.11 MAC to operate with minimum dependence on the PMD sublayer, a physical layer convergence sublayer is defined. This function simplifies the PHY service interface to the IEEE 802.11 MAC services. The PHY-specific preamble is normally associated with this convergence layer.

16.1.2.2 PMD sublayer

The PMD sublayer provides a clear channel assessment (CCA) mechanism, transmission mechanism, and reception mechanism that are used by the MAC via the PLCP to send or receive data between two or more STAs.

16.1.2.3 PHY management entity (PLME)

The PLME performs management of the local PHY functions in conjunction with the MAC management entity. Subclause 16.4 lists the MIB variables that may be accessed by the PHY sublayer entities and intralayer of higher layer management entities (LME). These variables are accessed via the PLME-GET, PLME-SET, and PLME-RESET primitives defined in Clause 10.

16.1.3 Service specification method and notation

The models represented by figures and state diagrams are intended as the illustrations of functions provided. It is important to distinguish between a model and a real implementation. The models are optimized for simplicity and clarity of presentation; the actual method of implementation is left to the discretion of the IEEE 802.11 IR PHY compliant developer. Conformance to the standard is not dependent on following the model, and an implementation that follows the model closely may not be conformant.

Abstract services are specified here by describing the service primitives and parameters that characterize each service. This definition is independent of any particular implementation. In particular, the PHY-SAP operations are defined and described as instantaneous; however, this may be difficult to achieve in an implementation.

16.2 IR PLCP sublayer

While the PLCP sublayer and the PMD sublayer are described separately, the separation and distinction between these sublayers is artificial, and is not meant to imply that the implementation must separate these functions. This distinction is made primarily to provide a point of reference from which to describe certain functional components and aspects of the PMD. The functions of the PLCP can be subsumed by a PMD sublayer; in this case, the PMD will incorporate the PHY-SAP as its interface, and will not offer a PMD-SAP.

16.2.1 Overview

A convergence procedure is provided by which MPDUs are converted to and from PLCPDUs. During transmission, the MPDU (PLCSDU) is prepended with a PLCP Preamble and PLCP Header to create the PLCPDU. At the receiver, the PLCP Preamble is processed and the internal data fields are processed to aid in demodulation and delivery of the MPDU (PSDU).

16.2.2 PLCP frame format

Figure 100 shows the format for the PLCPDU including the PLCP Preamble, the PLCP Header, and the PSDU. The PLCP Preamble contains the following fields: Synchronization (SYNC) and Start Frame Delimiter (SFD). The PLCP Header contains the following fields: Data Rate (DR), DC Level Adjustment (DCLA), Length (LENGTH), and Cyclic Redundancy Check (CRC). Each of these fields is described in detail in 16.2.4.

PLCP Preamble		PLCP Header				PSDU
SYNC	SFD	DR	DCLA	LENGTH	CRC	
57-73 slots	4 slots	3 slots	32 slots	16 bits	16 bits	variable number of octets

Figure 100—PLCPDU frame format

16.2.3 PLCP modulation and rate change

The PLCP Preamble shall be transmitted using the basic pulse defined in 16.3.3.2. The PLCSDU, LENGTH, and CRC fields shall be transmitted using pulse position modulation (PPM). PPM maps bits in the octet into symbols: 16-PPM maps four bits into a 16-position symbol, and 4-PPM maps two bits into a 4-position symbol. The basic L-PPM time unit is the slot. A slot corresponds to one of the L positions of a symbol and has a 250 ns duration. The PLCSDU, LENGTH, and CRC fields are transmitted at one of two bit rates: 1 Mbit/s or 2 Mbit/s. The Data Rate field indicates the data rate that will be used to transmit the PLCSDU, LENGTH, and CRC fields. The 1 Mbit/s data rate uses 16-PPM (basic access rate), and the 2 Mbit/s data rate uses 4-PPM (enhanced access rate). The transmitter and receiver will initiate the modulation or demodulation indicated by the DR field starting with the first 4 bits (in 16-PPM) or 2 bits (in 4-PPM) of the LENGTH field. The PSDU transmission rate is set by the DATARATE parameter in the PHY-TXSTART.request primitive. Any conformant IR PHY shall be capable of receiving at 1 Mbit/s and 2 Mbit/s. Transmission at 2 Mbit/s is optional.

A PHY-TXSTART.request that specifies a data rate which is not supported by a PHY instance will cause the PHY to indicate an error to its MAC instance. A PHY is not permitted under any circumstance to transmit at a different rate than the requested rate.

16.2.4 PLCP field definitions

16.2.4.1 PLCP Synchronization (SYNC) field

The SYNC field consists of a sequence of alternated presence and absence of a pulse in consecutive slots. The SYNC field has a minimum length of 57 L-PPM slots and a maximum length of 73 L-PPM slots and shall terminate with the absence of a pulse in the last slot. This field is provided so that the receiver can perform clock recovery (slot synchronization), automatic gain control (optional), signal-to-noise ratio estimation (optional), and diversity selection (optional).

The SYNC field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol. See 16.3.2.1 for legal symbols.

16.2.4.2 PLCP Start Frame Delimiter (SFD) field

The SFD field length is four L-PPM slots and consists of the binary sequence 1001, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot. The leftmost bit shall be transmitted first. The SFD field is provided to indicate the start of the PLCP Preamble and to perform bit and symbol synchronization.

The SFD field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.3 PLCP Data Rate (DR) field

The DR field indicates to the PHY the data rate that shall be used for the transmission or reception of the PLCSDU, LENGTH, and CRC fields. The transmitted value shall be provided by the PHY-TXSTART.request primitive as described in Clause 12. The DR field has a length of three L-PPM slots. The leftmost bit, as shown below, shall be transmitted first. The IR PHY currently supports two data rates defined by the slot pattern shown for the three L-PPM slots following the SFD, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot:

1 Mbit/s: 000
2 Mbit/s: 001

The DR field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.4 PLCP DC Level Adjustment (DCLA) field

The DCLA field is required to allow the receiver to stabilize the dc level after the SYNC, SFD, and DR fields. The leftmost bit, as shown below, shall be transmitted first. The length of the DCLA field is 32 L-PPM slots and consists of the contents shown, where 1 indicates a pulse in the L-PPM slot and 0 indicates no pulse in the L-PPM slot:

1 Mbit/s: 00000000100000000000000010000000
2 Mbit/s: 00100010001000100010001000100010

The DCLA field is not modulated using L-PPM, but instead consists of transitions in L-PPM slots that would otherwise constitute an illegal symbol.

16.2.4.5 PLCP LENGTH field

The LENGTH field is an unsigned 16-bit integer that indicates the number of octets to be transmitted in the PSDU. The transmitted value shall be provided by the PHYTXSTART.request primitive as described in Clause 12. The lsb shall be transmitted first. This field is modulated and sent in L-PPM format. This field is protected by the CRC described in 16.2.4.6.

16.2.4.6 PLCP CRC field

The LENGTH field shall be protected by a 16-bit CRC-CCITT. The CRC-CCITT is the one's complement of the remainder generated by the modulo 2 division of the LENGTH field by the polynomial:

$$x^{16}+x^{12}+x^{5}+1$$

The protected bits will be processed in transmit order. The msb of the 16-bit CRC-CCITT shall be transmitted first. This field shall be modulated and sent in L-PPM format. All CRC-CCITT calculations shall be made prior to L-PPM encoding on transmission and after L-PPM decoding on reception.

16.2.4.7 PSDU field

This field is composed of a variable number of octets. The minimum is 0 (zero) and the maximum is 2500. The lsb of each octet shall be transmitted first. All the octets of this field shall be modulated and sent in L-PPM format.

16.2.5 PLCP procedures

16.2.5.1 PLCP transmit procedure

All commands issued by the MAC require that a confirmation primitive be issued by the PHY. The confirmation primitives provide flow control between the MAC and the PHY.

The steps below are the transmit procedure:

a) Based on the status of CCA, the MAC shall determine whether the channel is clear.
b) If the channel is clear, transmission of the PSDU shall be initiated by a PHY-TXSTART.request with parameters LENGTH and DATARATE.
c) The PHY entity shall immediately initiate transmission of the PLCP Preamble and PLCP Header based on the LENGTH and DATARATE parameters passed in the PHY-TXSTART.request. Once the PLCP Preamble and PLCP Header transmission is completed, the PHY entity shall issue a PHY-TXSTART.confirm.
d) Each octet of the PSDU is passed from the MAC to the PHY by a single PHY-DATA.request primitive. Each PHY-DATA.request shall be confirmed by the PHY with a PHY-DATA.confirm before the next request can be made.
e) At the PHY layer each PSDU octet shall be divided into symbols of 2 or 4 bits each. The symbols shall be modulated using L-PPM and transmitted into the medium.
f) Transmission is terminated by the MAC through the primitive PHY-TXEND.request. The PHY shall confirm the resulting end of transmission with a PHY-TXEND.confirm.

16.2.5.2 PLCP receive procedure

The steps below are the receive procedure:

a) CCA is provided to the MAC via the PHY-CCA.indicate primitive. When the PHY senses activity on the medium, it shall indicate that the medium is busy with a PHY-CCA.indicate with a value of BUSY. This will normally occur during the SYNC field of the PLCP Preamble.
b) The PHY entity shall begin searching for the SFD field. Once the SFD field is detected, the PHY entity shall attempt to receive the PLCP Header. After receiving the DR and DCLA fields, the PHY shall initiate processing of the received CRC and LENGTH fields. The data rate indicated in the DR field applies to all symbols in the latter part of the received PHYSDU, commencing with the first symbol of the LENGTH field. The CRC-CCITT shall be checked for correctness immediately after its reception.
c) If the CRC-CCITT check fails, or the value received in the DR field is not one supported by the PHY, then a PHY-RXSTART.indicate shall not be issued to the MAC. When the medium is again free, the PHY shall issue a PHY-CCA.indicate with a value of IDLE.
d) If the PLCP Preamble and PLCP Header reception is successful, the PHY shall send a PHY-RXSTART.indicate to the MAC; this includes the parameters DATARATE and LENGTH.

   In the absence of errors, the receiving PHY shall report the same length to its local MAC, in the RXVECTOR parameter of the PHY-RXSTART.indicate primitive, that the peer MAC presented to its local PHY entity in the TXVECTOR parameter of its respective PHY-TXSTART.request.
e) The received PLCSDU L-PPM symbols shall be assembled into octets and presented to the MAC using a series of PHY-DATA.indicate primitives, one per octet.
f) Reception shall be terminated after the reception of the final symbol of the last PLCSDU octet indicated by the PLCP Header's LENGTH field. After the PHY-DATA.indicate for that octet is issued, the PHY shall issue a PHY-RXEND.indicate primitive to its MAC.
g) After issuing the PHY-RXEND.indicate primitive, and when the medium is no longer busy, the PHY shall issue a PHY-CCA.indicate primitive with a value of IDLE.

16.2.5.3 CCA procedure

CCA is provided to the MAC via the PHY-CCA.indicate primitive.

The steps below are the CCA procedure:

a) When the PHY senses activity on the medium, a PHY-CCA.indicate primitive with a value of BUSY shall be issued. This will normally occur during reception of the SYNC field of the PLCP Preamble.
b) When the PHY senses that the medium is free, a PHY-CCA.indicate primitive with a value of IDLE shall be issued.
c) At any time, the MAC may issue a PHY-CCARESET.request primitive, which will reset the PHY's internal CCA detection mechanism to the medium not-busy (IDLE) state. This primitive will be acknowledged with a PHY-CCARESET.confirm primitive.

16.2.5.4 PMD_SAP peer-to-peer service primitive parameters

Several service primitives include a parameter vector. This vector shall be actually a list of parameters that may vary depending on PHY type. Table 67 indicates the parameters required by the MAC or IR PHY in each of the parameter vectors used for peer-to-peer interactions.

Table 67—IR PMD_SAP peer-to-peer service primitives

Parameter	Associated primitive	Value
LENGTH	RXVECTOR, TXVECTOR	4 to $2^{16} – 1$
DATARATE	RXVECTOR, TXVECTOR	PHY dependent

16.3 IR PMD sublayer

The IR PMD sublayer does not define PMD SAPs. The mechanism for communications between the PLCP and PMD sublayers, as well as the distinction between these two sublayers, if any, is left to implementors. In particular, it is possible to design and implement in a conformant way a single sublayer that subsumes the functions of both the PLCP and PMD, presenting only the PHY-SAP.

16.3.1 Overview

The PMD functional, electrical, and optical characteristics required for interoperability of implementations conforming to this specification are described in this subclause. The relationship of this specification to the entire IR physical layer is shown in Figure 11.

16.3.2 PMD operating specifications, general

General specifications for the IR PMD sublayer are provided in this subclause. These specifications apply to both the receive and transmit functions and general operation of a compliant IR PHY.

16.3.2.1 Modulation and channel data rates

Two modulation formats and data rates are specified for the IR PHY: a *basic access rate* and an *enhanced access rate*. The basic access rate is based on 1 Mbit/s 16-PPM modulation. The 16-PPM encoding is specified in Table 68. Each group of 4 data bits is mapped to one of the 16-PPM symbols. The enhanced access rate is based on 2 Mbit/s 4-PPM. The 4-PPM encoding is specified in Table 69. Each group of 2 data bits is mapped to one of the 4-PPM symbols. Transmission order of the symbol slots is from left to right, as shown in the table, where a 1 indicates in-band energy in the slot, and a 0 indicates the absence of in-band energy in the slot.

The data in Table 68 and Table 69 have been arranged (gray coded) so that a single out-of-position-by-one error in the medium, caused, for example, by intersymbol interference, results in only a single bit error in the received data, rather than in a multiple bit error.

Table 68—Sixteen-PPM basic rate mapping

Data	16-PPM symbol
0000	0000000000000001
0001	0000000000000010
0011	0000000000000100
0010	0000000000001000
0110	0000000000010000
0111	0000000000100000
0101	0000000001000000
0100	0000000010000000
1100	0000000100000000
1101	0000001000000000
1111	0000010000000000
1110	0000100000000000
1010	0001000000000000
1011	0010000000000000
1001	0100000000000000
1000	1000000000000000

Table 69—Four-PPM enhanced rate mapping

Data	4-PPM symbol
00	0001
01	0010
11	0100
10	1000

16.3.2.2 Octet partition and PPM symbol generation procedure

Since PPM is a block modulation method, with the block size less than a full octet, octets have to be partitioned prior to modulation (mapping into PPM symbols).

Octet partition depends on the PPM order being used.

Assume an octet is formed by eight bits numbered 7 6 5 4 3 2 1 0, where bit 0 is the lsb. Partition the octet as follows:

For 16-PPM, create two PPM symbols:

— The symbol using bits 3 2 1 0 shall be transmitted onto the medium first
— The symbol using bits 7 6 5 4 shall be transmitted onto the medium last

For 4-PPM, create four PPM symbols:

— The symbol using bits 1 0 shall be transmitted onto the medium first
— The symbol using bits 3 2 shall be transmitted onto the medium second
— The symbol using bits 5 4 shall be transmitted onto the medium third
— The symbol using bits 7 6 shall be transmitted onto the medium last

16.3.2.3 Operating environment

The IR PHY will operate only in indoor environments. IR PHY interfaces cannot be exposed to direct sunlight. The IR PHY relies on reflected infrared energy and does not require a line of sight between emitter and receiver in order to work properly. The range and bit-error-rate of the system may vary with the geometry of the environment and with natural and artificial illumination conditions.

16.3.2.4 Operating temperature range

The temperature range for full operation compliance with the IR PHY is specified as 0 °C to 40 °C.

16.3.3 PMD transmit specifications

The following subclauses describe the transmit functions and parameters associated with the PMD sublayer.

16.3.3.1 Transmitted peak optical power

The peak optical power of an emitted pulse shall be as specified in Table 70.

Table 70—Peak optical power as a function of emitter radiation pattern mask

Emitter radiation pattern mask	Peak optical power
Mask 1	2 W ± 20%
Mask 2	0.55 W ± 20%

16.3.3.2 Basic pulse shape and parameters

The basic pulse width, measured between the 50% amplitude points, shall be 250 ± 10 ns. The pulse rise time, measured between the 10% and 90% amplitude points, shall be no more than 40 ns. The pulse fall time, measured between the 10% and 90% amplitude points, shall be no more than 40 ns. The edge jitter, defined as the absolute deviation of the edge from its correct position, shall be no more than 10 ns. The basic pulse shape is shown in Figure 101.

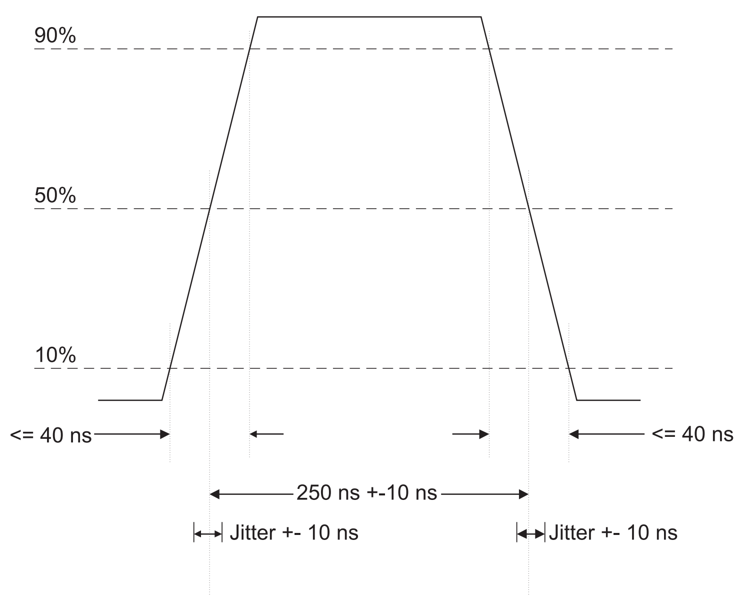


Figure 101—Basic pulse shape

16.3.3.3 Emitter radiation pattern mask

Currently the standard contains two emitter radiation pattern masks. Mask 1 is defined in Table 71 and illustrated in Figure 102. Mask 2 is defined in Table 72 and illustrated in Figure 104.

Table 71—Definition of the emitter radiation pattern mask 1

Declination angle	Normalized irradiance
$\alpha \le 60°$	> 3.5e–6
$\alpha \le 29°$	≤ 2.2e–5
$29° < \alpha \le 43°$	≤ –1.06e–4 + (0.44e–5) α
$43° < \alpha \le 57°$	≤ 1.15e–4 – (7.1e–7) α
$57° < \alpha \le 74°$	≤ 2.98e–4 – (3.9e–6) α
$74° < \alpha \le 90°$	≤ 4.05e–5 – (4.5e–7) α

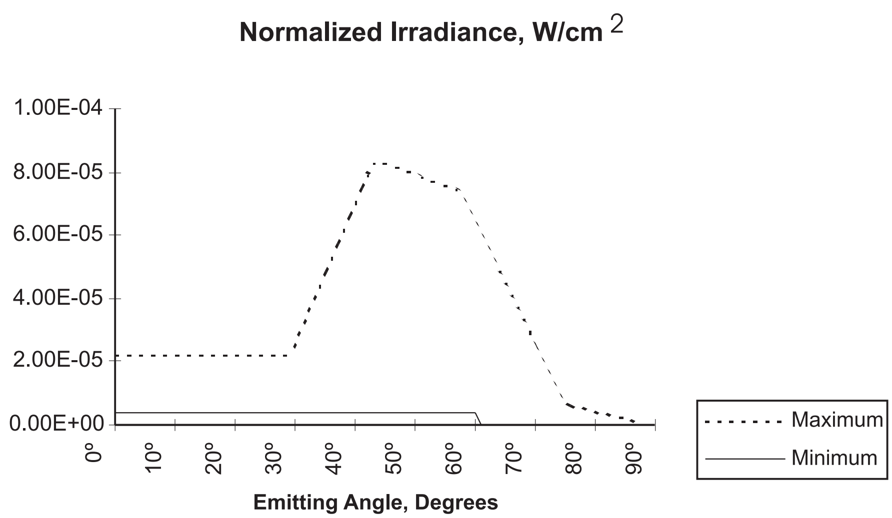


Figure 102—Emitter radiation pattern mask 1

Following is a description of how to interpret the Mask 1 table and figure. Position the conformant Mask 1 device in its recommended attitude. Define the conformant Mask 1 device axis as the axis passing through the emitter center and having the direction of the vertical from the floor. The mask represents the irradiance normalized to the total peak emitted power, as a function of the angle between the conformant Mask 1 device axis and the axis from the emitter center to the test receiver center (declination angle). The distance between emitter and test receiver is 1 m. The test receiver normal is always aimed at the emitter center. The azimuth angle is a rotation angle on the conformant device axis.

A device is conformant if for any azimuth angle its radiation pattern as a function of declination angle falls within the pattern mask.

Figure 103 is a description of how to interpret the Mask 2 table with reference to Figure 104.

Table 72—Definition of emitter radiation pattern mask 2

Declination angle	Pitch angle	Normalized irradiance
$\alpha \le 60$	$\alpha = 0$	0.05 ± 15%
$\alpha \le 90$	$\alpha = 0$	0.025 ± 15%
$\alpha \ge 100$	$\alpha = 0$	≤ 0.015
$0 \le \alpha \le 60$	$0 \le \alpha \le 10$	$0.035 \le I \le 0.055$
$0 \le \alpha \le 60$	$10 \le \alpha \le 20$	$0.0225 \le I \le 0.05$
$0 \le \alpha \le 60$	$\alpha \ge 30$	≤ 0.015

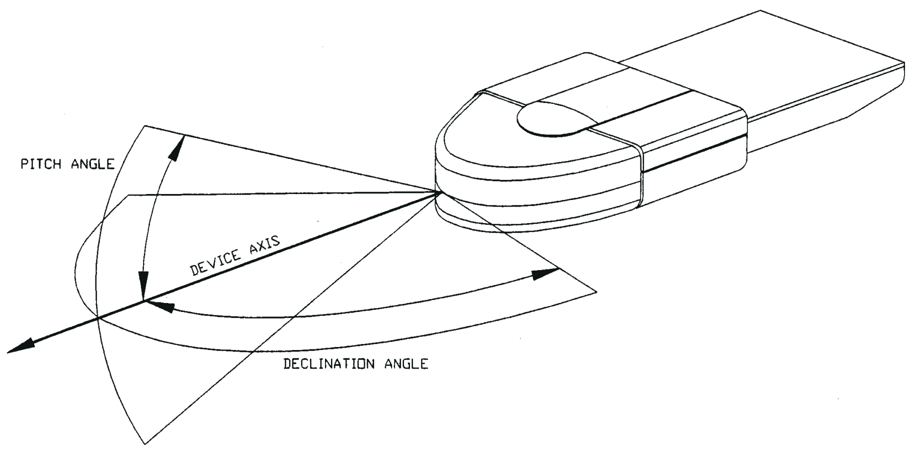


Figure 103—Mask 2 device orientation drawing

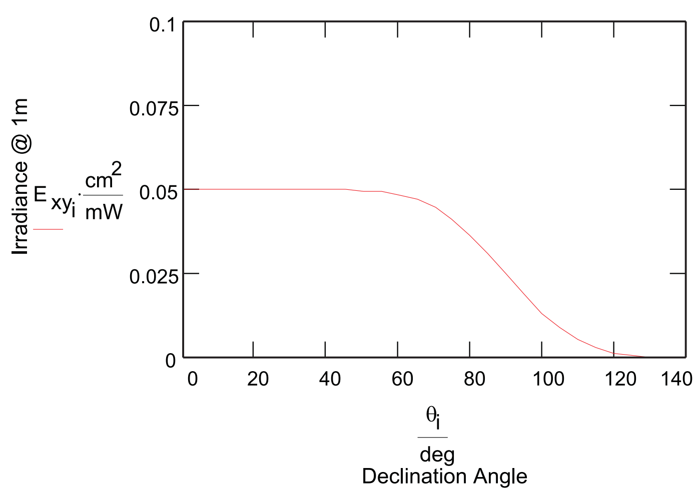


Figure 104—Emitter radiation pattern mask 2

Position the conformant Mask 2 device in its recommended attitude. Define the conformant Mask 2 device axis as passing through the emitter center and having the direction relative to the device as defined by the manufacturer. The declination angle plane is as defined by the manufacturer. The mask represents the irradiance normalized to the peak emitted power on the conformant Mask 2 device axis, as a function of the angle between the conformant device axis and the axis from the emitter center to the test receiver center (declination angle) in the declination plane. The distance between emitter and test receiver is 1 m. The test receiver normal is always aimed at the emitter center. The pitch angle is an angle relative to the conformant device axis which is perpendicular to the declination plane.

The device is conformant if, for a pitch angle of 0 degrees, at any declination angle from 0 to 100 degrees, and if, for any declination angle from 0 to 60 degrees, at any pitch angle from 0 to 20 degrees, its radiation pattern as a function of angle falls within the pattern mask.

Other radiation patterns are for future study.

16.3.3.4 Optical emitter peak wavelength

The optical emitter peak wavelength shall be between 850 and 950 nm.

16.3.3.5 Transmit spectrum mask

Define the transmit spectrum of a transmitter as the Fourier Transform, or equivalent, of a voltage (or current) signal whose amplitude, as a function of time, is proportional to the transmitted optical power.

The transmit spectrum of a conformant transmitter shall be 20 dB below its maximum for all frequencies above 15 MHz. The transmit spectrum mask is shown in Figure 105.

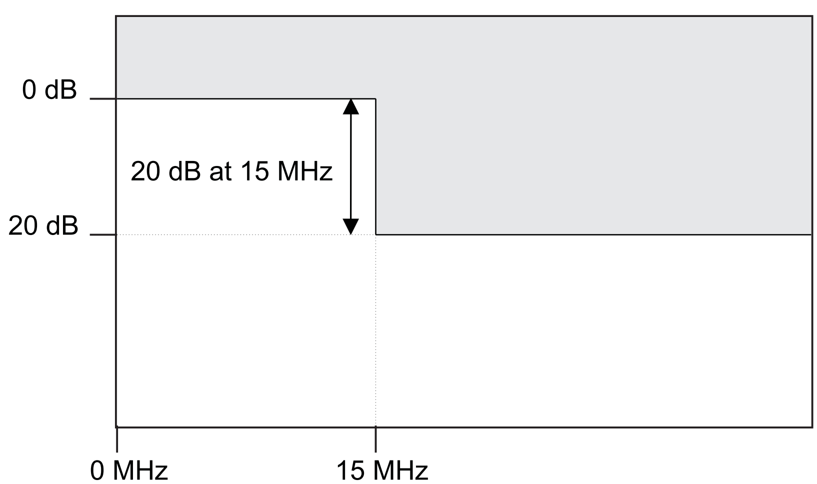


Figure 105—Transmit spectrum mask

16.3.4 PMD receiver specifications

The following subclauses describe the receive functions and parameters associated with the PMD sublayer.

16.3.4.1 Receiver sensitivity

The receiver sensitivity, defined as the minimum irradiance (in mW/cm^2) at the photodetector plane required for a frame error ratio (FER) of 4×10^{-5} with a PLCSDU of 512 octets and with an unmodulated background IR source between 800 nm and 1000 nm with a level of 0.1 mW/cm^2, shall be

1 Mbit/s: 2×10^{-5}mW/cm^2
2 Mbit/s: 8×10^{-5}mW/cm^2

16.3.4.2 Receiver dynamic range

The receiver dynamic range, defined as the ratio between the maximum and minimum irradiance at the plane normal to the receiver axis that assures an FER lower than or equal to 4×10^{-5} with a PLCSDU of 512 octets and with an unmodulated background IR source between 800 nm and 1000 nm with a level of 0.1 mW/cm^2, shall be ≥30 dB.

16.3.4.3 Receiver field-of-view (FOV)

The receiver axis is defined as the direction of incidence of the optical signal at which the received optical power is maximum.

The received optical power shall be greater than the values given in Table 73, at the angles indicated, where "angle of incidence" is the angle of incidence of the optical signal relative to the receiver axis, and "received power" is the received optical power as a percentage of that measured at the receiver axis.

Table 73—Definition of the receiver field of view

Angle of incidence	Received power
$\alpha \le 20°$	≥ 65%
$\alpha \le 40°$	≥ 55%
$\alpha \le 60°$	≥ 35%
$\alpha \le 80°$	≥ 10%

16.3.5 Energy Detect, Carrier Sense, and CCA definitions

16.3.5.1 Energy Detect (ED) signal

The ED signal shall be set true when IR energy variations in the band between 1 MHz and 10 MHz exceed 0.001 mW/cm^2.

The ED shall operate independently of the CS. ED shall not be asserted at the minimum signal level specified in 16.3.4.1, which is below the level specified in this subclause.

This signal is not directly available to the MAC.

16.3.5.2 Carrier Sense (CS) signal

The CS shall be asserted by the PHY when it detects and locks onto an incoming PLCP Preamble signal. Conforming PHYs shall assert this condition within the first 12 µs of signal reception, at the minimum signal level equal to the receiver sensitivity specified in 16.3.4.1, with a background IR level as specified in 16.3.4.1.

The CS shall be deasserted by the PHY when the receiving conformant device loses carrier lock.

NOTE—The 12 μs specification is somewhat less than the minimum length of the PLCP SYNC interval, which is 14.25 μs.

The CS shall operate independently of ED and shall not require a prior ED before the acquisition and assertion of CS. This permits reception of signals at the minimum signal level specified in 16.3.4.1, even though these signals fall below the ED level.

This signal is not directly available to the MAC.

16.3.5.3 CCA

CCA shall be asserted “IDLE” by the PHY when the CS and the ED are both false, or when ED has been continuously asserted for a period of time defined by the product of aCCAWatchdogTimerMax and aCCAWatchdogCountMax without CS becoming active. When either CS or ED go true, CCA is indicated as “BUSY” to the MAC via the primitive PHY-CCA.indicate. CS and DE behavior are defined in 16.3.5.2.

Normally, CCA will be held “BUSY” throughout the period of the PLCP Header. After receiving the last PLCP bit and the first data octet, the PHY shall signal PHY-RXSTART.indicate with the parameters LENGTH and RATE. CCA shall be held “BUSY” until the number of octets specified in the decoded PLCP Header are received. At that time the PHY shall signal PHY-RXEND.indicate. The CCA may remain “BUSY” after the end of data if some form of energy is still being detected. The PHY will signal PHY-CCA.indicate with a value of IDLE only when the CCA goes “CLEAR.”

The transition of CCA from “BUSY” to “IDLE” is indicated to the MAC via the primitive PHY-CCA.indicate.

If CS and ED go false before the PHY signals PHY-RXSTART.indicate, CCA is set to “IDLE” and *immediately* signaled to the MAC via PHY-CCA.indicate with a value of IDLE. If CS and ED go false after the PHY has signaled PHY-RXSTART.indicate, implying that the PLCP Header has been properly decoded, then the PHY shall not signal a change in state of CCA until the proper interval has passed for the number of octets indicated by the received PLCP LENGTH. At that time, the PHY shall signal PHY-RXEND.indicate with an RXERROR parameter of CarrierLost followed by PHY-CCA.indicate with a value of IDLE.

The transition of CCA from “CLEAR” to “BUSY” resets the CCA watchdog timer and CCA watchdog counter. aCCAWatchdogTimerMax and aCCAWatchdogCountMax are parameters available via MIB entries and can be read and set via the LME.

Rise and fall times of CCA relative to the OR’ing of the CS and ED signals shall be less than 30 ns. CS and ED are both internal signals to the PHY and are not available directly to the MAC, nor are they defined at any exposed interface.

16.3.5.4 CHNL_ID

For the IR PHY, CHNL_ID = X'01' is defined as the baseband modulation method. All other values are not defined.

16.4 PHY attributes

PHY attributes have allowed values and default values that are PHY-dependent. Table 74 describes those values, and further specifies whether they are permitted to vary from implementation to implementation.

Table 74 does not provide the definition of the attributes, but only provides the IR PHY-specific values for the attributes whose definitions are in Clause 13 of this standard.

Table 74—IR PHY MIB attributes

PHY MIB object	Default value	Operational semantics	Operational behavior
aCCATime	5 µs	Static	Identical for all conformant PHY
aRxTxTurnaroundTime	0 µs	Static	Identical for all conformant PHY
aSlotTime	8 µs	Static	Identical for all conformant PHY
aRxTxSwitchTime	0	Static	Identical for all conformant PHY
aTxRampOnTime	0	Static	Identical for all conformant PHY
aRxPLCPDelay	1 µs	Static	Identical for all conformant PHY
aTxPLCPDelay	Implementation dependent	Static	Identical for all conformant PHY
aRxRFDelay	Implementation dependent	Static	Identical for all conformant PHY
aTxRFDelay	Implementation dependent	Static	Identical for all conformant PHY
aCCAWatchdogTimerMax	Implementation dependent	Dynamic	A conformant PHY may set this via the LME
aCCAWatchdogCountMax	Implementation dependent	Dynamic	A conformant PHY may set this via the LME
aCCAWatchdogTimerMin	22 µs	Static	Identical for all conformant PHY
aCCAWatchdogCountMin	1	Static	Identical for all conformant PHY
aMACProcessingtDelay	2 µs	Static	Identical for all conformant PHY
aTxRampOffTime	0 µs	Static	Identical for all conformant PHY
aMPDUMaxLength	2500 octets	Static	Identical for all conformant PHY
aSIFSTime	7 µs	Static	Identical for all conformant PHY
aSupportedRatesTx	Implementation dependent	Static	All conformant PHY must include the value X'02' (1 Mbit/s).
aSupportedRatesRx	Implementation dependent	Static	All conformant PHY must include the values X'02' (1 Mbit/s) and X'04' (2 Mbit/s).
aPHYType	03	Static	Identical for all conformant PHY
aCWmin	63	Static	Identical for all conformant PHY
aCWmax	1023	Static	Identical for all conformant PHY
aPLCPHeaderLength	41 µs (1 Mbit/s) 25 µs (2 Mbit/s)	Static	Identical for all conformant PHY
aPreambleLength	16 µs (1 Mbit/s) 20 µs (2 Mbit/s)	Static	Identical for all conformant PHY

Annex A

(normative)

Protocol Implementation Conformance Statement (PICS) proforma

A.1 Introduction

The supplier of a protocol implementation that is claimed to conform to IEEE Std 802.11-1997 shall complete the following PICS proforma.

A completed PICS proforma is the PICS for the implementation in question. The PICS is a statement of which capabilities and options of the protocol have been implemented. The PICS can have a number of uses, including use

a) By the protocol implementor, as a checklist to reduce the risk of failure to conform to the standard through oversight;
b) By the supplier and acquirer, or potential acquirer, of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;
c) By the user, or potential user, of the implementation, as a basis for initially checking the possibility of interworking with another implementation (note that, while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICS proformas);
d) By a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.

A.2 Abbreviations and special symbols

A.2.1 Status symbols

M	mandatory
O	optional
O.<n>	optional, but support of at least one of the group of options labeled by the same numeral <n> is required
pred:	conditional symbol, including predicate identification

A.2.2 General abbreviations

N/A	not applicable
AD	address function capability
CF	implementation under test (IUT) configuration
FR	MAC frame capability
FS	frame sequence capability
PC	protocol capability
PICS	protocol implementation conformance statement

A.3 Instructions for completing the PICS proforma

A.3.1 General structure of the PICS proforma

The first part of the PICS proforma, Implementation Identification and Protocol Summary, is to be completed as indicated with the information necessary to identify fully both the supplier and the implementation.

The main part of the PICS proforma is a fixed questionnaire, divided into subclauses, each containing a number of individual items. Answers to the questionnaire items are to be provided in the rightmost column, either by simply marking an answer to indicate a restricted choice (usually Yes or No) or by entering a value or a set or a range of values. (Note that there are some items where two or more choices from a set of possible answers may apply. All relevant choices are to be marked, in these cases.)

Each item is identified by an item reference in the first column. The second column contains the question to be answered. The third column contains the reference or references to the material that specifies the item in the main body of IEEE Std 802.11-1997. The remaining columns record the status of each item, i.e., whether support is mandatory, optional, or conditional, and provide the space for the answers (see also A.3.4). Marking an item as supported is to be interpreted as a statement that all relevant requirements of the subclauses and normative annexes, cited in the References column for the item, are met by the implementation.

A supplier may also provide, or be required to provide, further information, categorized as either Additional Information or Exception Information. When present, each kind of further information is to be provided in a further subclause of items labeled A<*I*> or X<*I*>, respectively, for cross-referencing purposes, where <*I*> is any unambiguous identification for the item (e.g., simply a numeral). There are no other restrictions on its format or presentation.

The PICS proforma for a station consists of A.4.1, through A.4.4 inclusive, and at least one of A.4.5, A.4.6, or A.4.7 corresponding to the PHY implemented.

A completed PICS proforma, including any Additional Information and Exception Information, is the PICS for the implementation in question.

NOTE—Where an implementation is capable of being configured in more than one way, a single PICS may be able to describe all such configurations. However, the supplier has the choice of providing more than one PICS, each covering some subset of the implementation’s capabilities, if this makes for easier and clearer presentation of the information.

A.3.2 Additional Information

Items of Additional Information allow a supplier to provide further information intended to assist in the interpretation of the PICS. It is not intended or expected that a large quantity of information will be supplied, and a PICS can be considered complete without any such information. Examples of such Additional Information might be a outline of the ways in which an (single) implementation can be set up to operate in a variety of environments and configurations, or information about aspects of the implementation that are outside the scope of this standard but have a bearing upon the answers to some items.

References to items of Additional Information may be entered next to any answer in the questionnaire, and may be included in items of Exception Information.

A.3.3 Exception Information

It may happen occasionally that a supplier will wish to answer an item with mandatory status (after any conditions have been applied) in a way that conflicts with the indicated requirement. No preprinted answer will be found in the Support column for this. Instead, the supplier shall write the missing answer into the Support column, together with an X<*I*> reference to an item of Exception Information, and shall provide the appropriate rationale in the Exception Information item itself.

An implementation for which an Exception Information item is required in this way does not conform to IEEE Std 802.11-1997.

NOTE—A possible reason for the situation described above is that a defect in IEEE Std 802.11-1997 has been reported, a correction for which is expected to change the requirement not met by the implementation.

A.3.4 Conditional status

The PICS proforma contains a number of conditional items. These are items for which both the applicability of the item itself, and its status if it does apply, mandatory or optional, are dependent upon whether or not certain other items are supported.

Where a group of items is subject to the same condition for applicability, a separate preliminary question about the condition appears at the head of the group, with an instruction to skip to a later point in the questionnaire if the "Not Applicable" answer is selected. Otherwise, individual conditional items are indicated by a conditional symbol in the Status column.

A conditional symbol is of the form "<pred>:<S>", where "<pred>" is a predicate as described below, and "<S>" is one of the status symbols M or O.

If the value of the predicate is true, the conditional item is applicable, and its status is given by S: the support column is to be completed in the usual way. Otherwise, the conditional item is not relevant and the Not Applicable (N/A) answer is to be marked.

A predicate is one of the following:

a) An item-reference for an item in the PICS proforma: the value of the predicate is true if the item is marked as supported, and is false otherwise.
b) A boolean expression constructed by combining item-references using the boolean operator OR: the value of the predicate is true if one or more of the items is marked as supported, and is false otherwise.

Each item referenced in a predicate, or in a preliminary question for grouped conditional items, is indicated by an asterisk in the Item column.

A.4 PICS proforma—IEEE Std 802.11-1997[7]

A.4.1 Implementation identification

Supplier	
Contact point for queries about the PICS	
Implementation Name(s) and Version(s)	
Other information necessary for full identification, e.g., name(s) and version(s) of the machines and/or operating systems(s), system names	

NOTES

1—Only the first three items are required for all implementations. Other information may be completed as appropriate in meeting the requirement for full identification.

2—The terms Name and Version should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model).

A.4.2 Protocol summary, IEEE Std 802.11-1997

Identification of protocol standard	IEEE Std 802.11-1997
Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS	Amd. : Corr. : Amd. : Corr. :
Have any exception items been required? (See A.3.3; the answer Yes means that the implementation does not conform to IEEE Std 802.11-1997.)	Yes ❑ No ❑

Date of statement (dd/mm/yy)	

[7]*Copyright release for PICS proforma:* Users of this standard may freely reproduce the PICS proforma in this annex so that it can be used for its intended purpose and may further publish the completed PICS.

A.4.3 IUT configuration

Item	IUT configuration	References	Status	Support
	What is the configuration of the IUT?			
* CF1	Access Point (AP)	5.2	O.1	Yes ❑ No ❑
* CF2	Independent station (*not* an AP)	5.2	O.1	Yes ❑ No ❑
* CF3	Frequency-Hopping spread spectrum (FHSS) PHY for the 2.4 GHz band		O.2	Yes ❑ No ❑
* CF4	Direct Sequence Spread Spectrum (DSSS) PHY for the 2.4 GHz band		O.2	Yes ❑ No ❑
* CF5	Infrared PHY		O.2	Yes ❑ No ❑

A.4.4 MAC protocol

A.4.4.1 MAC protocol capabilities

Item	Protocol capability	References	Status	Support
	Are the following MAC protocol capabilities supported?			
PC1	Authentication service	5.4.3.1, 5.4.3.2, 5.7.6, 5.7.7, 8.1, Annex C	M	Yes ❑ No ❑
PC1.1	Authentication state	5.5	M	Yes ❑ No ❑
PC1.2	Open System authentication	8.1.1	M	Yes ❑ No ❑
PC1.3	Shared Key authentication	8.1.2, 8.3	PC2:M	Yes ❑ No ❑ N/A ❑
* PC2	WEP algorithm	5.4.3.3, 8.2, Annex C	O	Yes ❑ No ❑
PC2.1	WEP Encryption procedure	8.2.3, 8.2.4, 8.2.5	PC2:M	Yes ❑ No ❑ N/A ❑
PC2.2	WEP Decryption procedure	8.2.3, 8.2.4, 8.2.5	PC2:M	Yes ❑ No ❑ N/A ❑
PC2.3	Security services management	8.3	M	Yes ❑ No ❑
PC3	Distributed Coordination function	9.1, 9.2, Annex C	M	Yes ❑ No ❑
PC3.1	Net Allocation Vector (NAV) function	9.2.1, 9.2.5, 9.3.2.2	M	Yes ❑ No ❑
PC3.2	Interframe space usage and timing	9.2.3, 9.2.5, 9.2.10	M	Yes ❑ No ❑
PC3.3	Random Backoff function	9.2.4	M	Yes ❑ No ❑
PC3.4	DCF Access procedure	9.2.5.1, 9.2.5.5	M	Yes ❑ No ❑
PC3.5	Random Backoff procedure	9.2.5.2	M	Yes ❑ No ❑
PC3.6	Recovery procedures and retransmit limits	9.2.5.3	M	Yes ❑ No ❑
PC3.7	RTS/CTS procedure	9.2.5.4, 9.2.5.6, 9.2.5.7	M	Yes ❑ No ❑

A.4.4.1 MAC protocol capabilities (continued)

Item	Protocol capability	References	Status	Support
PC3.8	Directed MPDU transfer	9.2.6	M	Yes ❑ No ❑
PC3.9	Broadcast and multicast MPDU transfer	9.2.7	M	Yes ❑ No ❑
PC3.10	MAC level acknowledgment	9.2.2, 9.2.8	M	Yes ❑ No ❑
PC3.11	Duplicate detection and recovery	9.2.9	M	Yes ❑ No ❑
* PC4	Point coordinator (PC)	9.1, 9.3, Annex C	CF1:O	Yes ❑ No ❑ N/A ❑
PC4.1	Maintenance of CFP structure and timing	9.3.1, 9.3.2	PC4:M	Yes ❑ No ❑ N/A ❑
PC4.2	PCF MPDU transfer from PC	9.3.3	PC4:M	Yes ❑ No ❑ N/A ❑
* PC4.3	PCF MPDU transfer to PC	9.3.3	PC4:O	Yes ❑ No ❑ N/A ❑
PC4.4	Overlapping PC provisions	9.3.3.2	PC4:M	Yes ❑ No ❑ N/A ❑
PC4.5	Polling list maintenance	9.3.4	PC4.3:M	Yes ❑ No ❑ N/A ❑
* PC5	CF-Pollable	9.1, 9.3, Annex C	CF2:O	Yes ❑ No ❑ N/A ❑
PC5.1	Interpretation of CFP structure and timing	9.3.1, 9.3.2	PC5:M	Yes ❑ No ❑ N/A ❑
PC5.2	PCF MPDU transfer to/from and CF-Pollable STA	9.3.3	PC5:M	Yes ❑ No ❑ N/A ❑
PC5.3	Polling list update	9.3.4	PC5:M	Yes ❑ No ❑ N/A ❑
PC6	Fragmentation	9.2, 9.4, Annex C	M	Yes ❑ No ❑
PC7	Defragmentation	9.2, 9.5, Annex C	M	Yes ❑ No ❑
PC8	MAC data service	9.1.5, 9.8, Annex C	M	Yes ❑ No ❑
PC8.1	Reorderable-Multicast service class	9.8	M	Yes ❑ No ❑
PC8.2	StrictlyOrdered service class	9.8	O	Yes ❑ No ❑
PC9	Multirate support	9.6, Annex C	M	Yes ❑ No ❑
* PC10	Multiple outstanding MSDU support	9.8, Annex C	O	Yes ❑ No ❑
PC10.1	Multiple outstanding MSDU transmission restrictions	9.8	PC10:M	Yes ❑ No ❑ N/A ❑
PC11	Timing synchronization	11.1, Annex C	M	Yes ❑ No ❑
PC11.1	Timing in an infrastructure network	11.1.1.1, 11.1.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC11.2	Timing in an Independent BSS (IBSS)	11.1.1.2, 11.1.4	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.3	Beacon Generation function	11.1.2	M	Yes ❑ No ❑ N/A ❑
PC11.5	TSF synchronization and accuracy	11.1.2	M	Yes ❑ No ❑
PC11.5	Infrastructure BSS initialization	11.1.3	CF1:M	Yes ❑ No ❑ N/A ❑
PC11.6	Independent BSS initialization	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.7	Passive scanning	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC11.8	Active scanning	11.1.3	CF2:M	Yes ❑ No ❑ N/A ❑

A.4.4.1 MAC protocol capabilities (continued)

Item	Protocol capability	References	Status	Support
PC11.9	Probe response	11.1.3	M	Yes ❑ No ❑
PC11.10	Hop Synchronization function	11.1.5	CF3:M	Yes ❑ No ❑ N/A ❑
PC12	Infrastructure power management	11.2.1, Annex C	M	Yes ❑ No ❑
PC12.1	Station power management modes	11.2.1.1, 11.2.1.8	CF2:M	Yes ❑ No ❑ N/A ❑
PC12.2	TIM transmission	11.2.1.2, 11.2.1.3	CF1:M	Yes ❑ No ❑ N/A ❑
PC12.3	AP function during CP	11.2.1.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC12.4	AP function during CFP	11.2.1.5	PC4:M	Yes ❑ No ❑ N/A ❑
PC12.5	Receive function during CP	11.2.1.6	CF2:M	Yes ❑ No ❑ N/A ❑
PC12.6	Receive function during CFP	11.2.1.7	PC5:M	Yes ❑ No ❑ N/A ❑
PC12.7	Aging function	11.2.1.9	CF1:M	Yes ❑ No ❑ N/A ❑
PC13	IBSS power management	11.2.2, Annex C	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.1	Initialization of power management	11.2.2.2	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.2	STA power state transitions	11.2.2.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC13.3	ATIM and frame transmission	11.2.2.4	CF2:M	Yes ❑ No ❑ N/A ❑
PC14	Association and reassociation	5.4, 5.7, 11.3, Annex C	M	Yes ❑ No ❑
PC14.1	Association state	5.5	M	Yes ❑ No ❑
PC14.2	STA association procedure	11.3.1	CF2:M	Yes ❑ No ❑ N/A ❑
PC14.3	AP association procedure	11.3.2	CF1:M	Yes ❑ No ❑ N/A ❑
PC14.4	STA reassociation procedure	11.3.3	CF2:M	Yes ❑ No ❑ N/A ❑
PC14.5	AP reassociation procedure	11.3.4	CF1:M	Yes ❑ No ❑ N/A ❑
PC15	Management information base (MIB)	11.4, Annex C	M	Yes ❑ No ❑
PC15.1	SMT object class	11.4.2.1	M	Yes ❑ No ❑
* PC15.2	Privacy package	11.4.2.1	PC2:M	Yes ❑ No ❑ N/A ❑
PC15.3	MAC object class	11.4.2.2	M	Yes ❑ No ❑
* PC15.4	MAC statistics package	11.4.2.2	O	Yes ❑ No ❑
PC15.3	Resource type object class	11.4.2.3	M	Yes ❑ No ❑

A.4.4.2 MAC frames

Item	MAC frame	References	Status	Support
	Is transmission of the following MAC frames supported?	7, Annex C		
FT1	Association request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT2	Association response	7	CF1:M	Yes ❑ No ❑ N/A ❑
FT3	Reassociation request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT4	Reassociation response	7	CF1:M	Yes ❑ No ❑ N/A ❑
FT5	Probe request	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT6	Probe response	7	M	Yes ❑ No ❑
FT7	Beacon	7	M	Yes ❑ No ❑
FT8	ATIM	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT9	Disassociation	7	M	Yes ❑ No ❑
FT10	Authentication	7	M	Yes ❑ No ❑
FT11	Deauthentication	7	M	Yes ❑ No ❑
FT12	PS-Poll	7	CF2:M	Yes ❑ No ❑ N/A ❑
FT13	RTS	7	M	Yes ❑ No ❑
FT14	CTS	7	M	Yes ❑ No ❑
FT15	ACK	7	M	Yes ❑ No ❑
FT16	CF-End	7	PC4:M	Yes ❑ No ❑ N/A ❑
FT17	CF End+CF-Ack	7	PC4:M	Yes ❑ No ❑ N/A ❑
FT18	Data	7	M	Yes ❑ No ❑
FT19	Data + CF-Ack	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FT20	Data + CF-Poll	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT21	Data + CF-Ack+CF-Poll	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT22	Null	7	M	Yes ❑ No ❑
FT23	CF-Ack (no data)	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FT24	CF-Poll (no data)	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
FT25	CF-Ack+CF-Poll (no data)	7	PC4.3:M	Yes ❑ No ❑ N/A ❑
	Is reception of the following MAC frames supported?	7, Annex C		
FR1	Association request	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR2	Association response	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR3	Reassociation request	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR4	Reassociation response	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR5	Probe request	7	M	Yes ❑ No ❑
FR6	Probe response	7	M	Yes ❑ No ❑
FR7	Beacon	7	M	Yes ❑ No ❑
FR8	ATIM	7	CF2:M	Yes ❑ No ❑ N/A ❑
FR9	Disassociation	7	M	Yes ❑ No ❑
FR10	Authentication	7	M	Yes ❑ No ❑

A.4.4.2 MAC frames (continued)

Item	MAC frame	References	Status	Support
FR11	Deauthentication	7	M	Yes ❑ No ❑
FR12	PS-Poll	7	CF1:M	Yes ❑ No ❑ N/A ❑
FR13	RTS	7	M	Yes ❑ No ❑
FR14	CTS	7	M	Yes ❑ No ❑
FR15	ACK	7	M	Yes ❑ No ❑
FR16	CF-End	7	M	Yes ❑ No ❑
FR17	CF End+CF-Ack	7	M	Yes ❑ No ❑
FR18	Data	7	M	Yes ❑ No ❑
FR19	Data + CF-Ack	7	M	Yes ❑ No ❑
FR20	Data + CF-Poll	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR21	Data + CF-Ack+CF-Poll	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR22	Null	7	M	Yes ❑ No ❑
FR23	CF-Ack (no data)	7	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑
FR24	CF-Poll (no data)	7	PC5:M	Yes ❑ No ❑ N/A ❑
FR25	CF-Ack+CF-Poll (no data)	7	PC5:M	Yes ❑ No ❑ N/A ❑

A.4.4.3 Frame exchange sequences

Item	Frame exchange sequence	References	Status	Support
	Are the following frame sequences supported?			
FS1	Basic frame sequences	9.7, Annex C	M	Yes ❑ No ❑
FS2	CF-Frame sequences	9.7, Annex C	(PC4 OR PC5):M	Yes ❑ No ❑ N/A ❑

A.4.4.4 MAC addressing functions

Item	MAC Address function	References	Status	Support
	Are the following MAC Addressing functions supported?			
AD1	STA universal individual IEEE802 address	5.3.3, 7.1.3.3	M	Yes ❑ No ❑
AD2	BSS identifier generation	7.1.3.3, 11.1.3, Annex C	M	Yes ❑ No ❑
AD3	Receive address matching	7.1.3.3, 7.2.2, Annex C	M	Yes ❑ No ❑ N/A ❑

A.4.5 Frequency-Hopping PHY functions

Item	Protocol feature	References	Status	Support
	Which requirements and options does the PHY support?			
FH1	PHY service primitive parameters			
FH1.1	TXVECTOR parameter: LENGTH	14.2.2.1	M	Yes ❑ No ❑
FH1.2	TXVECTOR parameter: PLCPBITRATE	14.2.2.2	M	Yes ❑ No ❑
FH1.2.1	PLCPBITRATE = X'00' (1.0 Mbit/s)	14.2.2.2	M	Yes ❑ No ❑
* FH1.2.2	PLCPBITRATE = X'02' (2.0 Mbit/s)	14.2.2.2	O	Yes ❑ No ❑
FH1.3	RXVECTOR parameter: LENGTH	14.2.3.1	M	Yes ❑ No ❑
FH1.4	RXVECTOR parameter: RSSI	14.2.3.2	O	Yes ❑ No ❑
FH2	PLCP frame format			
FH2.1	PLCP Preamble: Sync	14.3.2.1.1	M	Yes ❑ No ❑
FH2.2	PLCP Preamble: Start Frame Delimiter	14.3.2.1.2	M	Yes ❑ No ❑
FH2.3	PLCP Header: Length Word	14.3.2.2.1	M	Yes ❑ No ❑
FH2.4	PLCP Header: Signaling field	14.3.2.2.2	M	Yes ❑ No ❑
FH2.5	PLCP Header: Header Error Check	14.3.2.2.3	M	Yes ❑ No ❑
FH2.6	PLCP Data Whitener: Scrambling and bias suppression encoding	14.3.2.3, 14.3.3.1.1	M	Yes ❑ No ❑
FH3	PLCP Transmit procedure			
FH3.1	Transmit: transmit on MAC request	14.3.3.1.1	M	Yes ❑ No ❑
FH3.2	Transmit: format and whiten frame	14.3.3.1.1	M	Yes ❑ No ❑
FH3.3	Transmit: Timing	14.3.3.1.1	M	Yes ❑ No ❑
FH4	PLCP CS/CCA procedure			
FH4.1	CS/CCA: perform on a minimum of one antenna	14.3.3.2.1	M	Yes ❑ No ❑
FH4.2.	CS/CCA: Detect preamble starting up to 20 µs after start of slot time	14.3.3.2.1	M	Yes ❑ No ❑
FH4.3	CS/CCA: Detect preamble starting at least 16 µs prior to end of slot time	14.3.3.2.1	M	Yes ❑ No ❑
FH4.4	CS/CCA: Detect random data	14.3.3.2.1	M	Yes ❑ No ❑
FH4.5	CS/CCA: Perform on antenna with essentially same gain and pattern as transmit antenna	14.3.3.2.1	M	Yes ❑ No ❑
FH4.6	CS/CCA: Detect valid SFD and PLCP header	14.3.3.2.1	M	Yes ❑ No ❑
FH4.7	CS/CCA: Maintain BUSY indication until end of length contained in valid PLCP header	14.3.3.2.1	M	Yes ❑ No ❑
FH5	PLCP Receive procedure			
FH5.1	Receive: Receive and dewhiten frame	14.3.3.3.1	M	Yes ❑ No ❑
FH6	PHY LME			
FH6.1	PLME: Support FH sync	14.4.2.2	M	Yes ❑ No ❑
FH6.2	PLME: Support PLME primitives	14.4.3.2	O	Yes ❑ No ❑

A.4.5 Frequency-Hopping PHY functions (continued)

Item	Protocol feature	References	Status	Support
FH7	Geographic area specific requirements			
* FH7.1	Geographic areas			
FH7.1.1	North America	14.6.2	O.1	Yes ❑ No ❑
FH7.1.2	Most of Europe	14.6.2	O.1	Yes ❑ No ❑
FH7.1.3	Japan	14.6.2	O.1	Yes ❑ No ❑
FH7.1.4	Spain	14.6.2	O.1	Yes ❑ No ❑
FH7.1.5	France	14.6.2	O.1	Yes ❑ No ❑
FH7.2	Operating frequency range	14.6.3	FH7.1:M	Yes ❑ No ❑
FH7.3	Number of operating channels	14.6.4	FH7.1:M	Yes ❑ No ❑
FH7.4	Operating channel frequencies	14.6.5	FH7.1:M	Yes ❑ No ❑
FH7.5	Occupied channel bandwidth	14.6.6	FH7.1:M	Yes ❑ No ❑
FH7.6	Minimum hop rate	14.6.7	FH7.1:M	Yes ❑ No ❑
FH7.7	Hop sequences	14.6.8	FH7.1:M	Yes ❑ No ❑
FH7.8	Unwanted emissions	14.6.9	FH7.1:M	Yes ❑ No ❑
FH8	1 Mbit/s PMD			
FH8.1	Modulation 2GFSK, BT=0.5, 1=positive freq. dev, 0=negative freq. dev.	14.6.10	M	Yes ❑ No ❑
FH8.2	Peak frequency deviation	14.6.10	M	Yes ❑ No ❑
FH8.3	Zero-Crossing error	14.6.10	M	Yes ❑ No ❑
FH8.4	Nominal channel data rate	14.6.11	M	Yes ❑ No ❑
FH8.5	Channel switching/settling time	14.6.12	M	Yes ❑ No ❑
FH8.6	Receive to transmit switch time	14.6.13	M	Yes ❑ No ❑
FH8.7	Nominal transmit power	14.6.14.1	M	Yes ❑ No ❑
FH8.8	Transmit power levels	14.6.14.2	M	Yes ❑ No ❑
FH8.9	Transmit power level control to <100 mW	14.6.14.3	M	Yes ❑ No ❑
FH8.10	Transmit spectrum shape	14.6.14.4	M	Yes ❑ No ❑
FH8.11	Transmit center frequency tolerance	14.6.14.5	M	Yes ❑ No ❑
FH8.12	Transmitter ramp periods	14.6.14.6	M	Yes ❑ No ❑
FH8.13	Receiver input dynamic range	14.6.15.1	M	Yes ❑ No ❑
FH8.14	Receiver center frequency acceptance range	14.6.15.2	M	Yes ❑ No ❑
FH8.15	Clear channel assessment power threshold for Pdet 90% (preamble)/70% (random data) for 100 mW units	14.6.15.3	M	Yes ❑ No ❑
FH8.16	Clear channel assessment power threshold for units >100 mW; sensitivity threshold is 1/2 dB lower for every dB above 20 dBm	14.6.15.3	M	Yes ❑ No ❑
FH8.17	Minimum receiver sensitivity at FER=3% with 400 octet frames	14.6.15.4	M	Yes ❑ No ❑
FH8.18	Intermodulation protection	14.6.15.5	M	Yes ❑ No ❑
FH8.19	Desensitization	14.6.15.6	M	Yes ❑ No ❑

A.4.5 Frequency-Hopping PHY functions (continued)

Item	Protocol feature	References	Status	Support
FH8.20	Operating temperature range	14.6.16	M	Yes ❑ No ❑
FH8.20.1	Temperature type 1	14.6.16	O	Yes ❑ No ❑
FH8.20.2	Temperature type 2	14.6.16	O	Yes ❑ No ❑
FH8.20.3	Temperature type 3	14.6.16	O	Yes ❑ No ❑
FH9	2 Mbit/s PMD			
FH9.1	All 1M PMD requirements	14.7.1	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.2	Modulation 4GFSK, BT=0.5	14.7.2	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.3	Frame structure for 2M PHY	14.7.2.1	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.4	Nominal channel data rate	14.7.3	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.5	Input dynamic range	14.7.4	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.6	Minimum receiver sensitivity at FER=3% with 400 octet frames	14.7.5	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.7	Intermodulation protection	14.7.6	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH9.8	Desensitization	14.7.7	FH1.2.2:M	Yes ❑ No ❑ N/A ❑
FH10	MIB	13.1, 14.8, Annex C	M	Yes ❑ No ❑
FH10.1	PHY object class	13.1,14.8	M	Yes ❑ No ❑

A.4.6 Direct sequence PHY functions

Item	PHY feature	References	Status	Support
	PLCP sublayer procedures	15.2		
DS1	Preamble prepend on TX	15.2.1	M	Yes ❑ No ❑
DS1.1	PLCP frame format	15.2.2, 15.2.3	M	Yes ❑ No ❑
DS1.2	PLCP integrity check generation	15.2.3, 15.2.3.6	M	Yes ❑ No ❑
DS1.3	TX rate change capability	15.2.3.3, 15.2.5	M	Yes ❑ No ❑
DS1.4	Supported data rates	15.1, 15.2.3.3	M	Yes ❑ No ❑
DS1.5	Data whitener scrambler	15.2.4	M	Yes ❑ No ❑
DS1.6	Scrambler initialization	15.2.4	M	Yes ❑ No ❑
DS2	Preamble process on RX	15.2.1		
DS2.1	PLCP frame format	15.2.2, 15.2.3	M	Yes ❑ No ❑
DS2.2	PLCP integrity check verify	15.2.3, 15.2.3.6	M	Yes ❑ No ❑
DS2.3	RX Rate change capability	15.2.3.3, 15.2.5	M	Yes ❑ No ❑
DS2.4	Data whitener descrambler	15.2.4	M	Yes ❑ No ❑
DS3	PN code sequence	15.4.6.3	M	Yes ❑ No ❑
DS4	Chipping continue on power down	15.2.6	O	Yes ❑ No ❑
*DS5	Operating channel capability	15.2.6, 15.4.6.2		
* DS5.1	North America (FCC)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.1.1	channel 1	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS5.1.2	channel 2	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.3	channel 3	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.4	channel 4	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.5	channel 5	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.6	channel 6	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.7	channel 7	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.8	channel 8	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.9	channel 9	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.10	channel 10	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
DS5.1.11	channel 11	15.2.6, 15.4.6.2	DS5.1:M	Yes ❑ No ❑ N/A ❑
* DS5.2	Canada (IC)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.2.1	channel 1	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.2	channel 2	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.3	channel 3	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.4	channel 4	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.5	channel 5	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.6	channel 6	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.7	channel 7	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.8	channel 8	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.9	channel 9	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.10	channel 10	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
DS5.2.11	channel 11	15.2.6, 15.4.6.2	DS5.2:M	Yes ❑ No ❑ N/A ❑
* DS5.3	Europe (ETSI)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.3.1	channel 1	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.2	channel 2	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.3	channel 3	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.4	channel 4	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.5	channel 5	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.6	channel 6	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.7	channel 7	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.8	channel 8	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.9	channel 9	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.10	channel 10	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.11	channel 11	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.12	channel 12	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
DS5.3.13	channel 13	15.2.6, 15.4.6.2	DS5.3:M	Yes ❑ No ❑ N/A ❑
* DS5.4	France	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.4.1	channel 10	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
DS5.4.2	channel 11	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
DS5.4.3	channel 12	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS5.4.4	channel 13	15.2.6, 15.4.6.2	DS5.4:M	Yes ❑ No ❑ N/A ❑
* DS5.5	Spain	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS5.5.1	channel 10	15.2.6, 15.4.6.2	DS5.5:M	Yes ❑ No ❑ N/A ❑
DS5.5.2	channel 11	15.2.6, 15.4.6.2	DS5.5:M	Yes ❑ No ❑ N/A ❑
* DS5.6	Japan (RCR)	15.2.6, 15.4.6.2	DS5:O.1	Yes ❑ No ❑ N/A ❑
DS6	Bits to symbol mapping	15.4.6.4		
DS6.1	1 Mbit/s	15.4.6.4	M	Yes ❑ No ❑
DS6.2	2 Mbit/s	15.4.6.4	M	Yes ❑ No ❑
*DS7	CCA functionality	15.4.8.4		
DS7.1	Energy Only (RSSI above threshold)	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.2	IEEE 802.11 DSSS correlation	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.3	Both methods	15.4.8.4	DS7:O.2	Yes ❑ No ❑
DS7.4	Hold CCA busy for packet duration of a correctly received PLCP but carrier lost during reception of MPDU	15.2.7	M	Yes ❑ No ❑
DS7.5	Hold CCA busy for packet duration of a correctly received but out of spec PLCP	15.2.7	M	Yes ❑ No ❑
DS8	Transmit antenna selection	15.4.5.5, 15.4.5.6	O	Yes ❑ No ❑
DS9	Receive antenna diversity	15.4.5.5, 15.4.5.6, 15.4.5.7	O	Yes ❑ No ❑
*DS10	Antenna port(s) availability	15.4.6.9	O	Yes ❑ No ❑
DS10.1	50 Ω impedance	15.4.6.9	DS10:M	Yes ❑ No ❑ N/A ❑
*DS11	Transmit power level support	15.4.5.8, 15.4.7.3	O	Yes ❑ No ❑
DS11.1	If greater than 100 mW capability	15.4.7.3	DS11:M	Yes ❑ No ❑ N/A ❑
*DS12	Radio type (temperature range)	15.4.6.10		
DS12.1	Type 1	15.4.6.10	DS12:O.3	Yes ❑ No ❑ N/A ❑
DS12.2	Type 2	15.4.6.10	DS12:O.3	Yes ❑ No ❑ N/A ❑
DS13	Spurious emissions conformance	15.4.6.5	M	Yes ❑ No ❑
DS14	TX-RX turnaround time	15.4.6.6	M	Yes ❑ No ❑
DS15	RX-TX turnaround time	15.4.6.7	M	Yes ❑ No ❑
DS16	Slot time	15.4.6.8	M	Yes ❑ No ❑
DS17	ED reporting time	15.4.6.8, 15.4.8.4	M	Yes ❑ No ❑
DS18	Minimum transmit power level	15.4.7.2	M	Yes ❑ No ❑
DS19	Transmit spectral mask conformance	15.4.7.4	M	Yes ❑ No ❑
DS20	Transmitted center frequency tolerance	15.4.7.5	M	Yes ❑ No ❑

A.4.6 Direct sequence PHY functions *(continued)*

Item	PHY feature	References	Status	Support
DS21	Chip clock frequency tolerance	15.4.7.6	M	Yes ❑ No ❑
DS22	Transmit power on ramp	15.4.7.7	M	Yes ❑ No ❑
DS23	Transmit power down ramp	15.4.7.7	M	Yes ❑ No ❑
DS24	RF carrier suppression	15.4.7.8	M	Yes ❑ No ❑
DS25	Transmit modulation accuracy	15.4.7.9	M	Yes ❑ No ❑
DS26	Receiver minimum input level sensitivity	15.4.8.1	M	Yes ❑ No ❑
DS27	Receiver maximum input level	15.4.8.2	M	Yes ❑ No ❑
DS28	Receiver adjacent channel rejection	15.4.8.3	M	Yes ❑ No ❑
DS29	MIB	13.1, 15.3.4, Annex C	M	Yes ❑ No ❑
DS29.1	PHY object class	13.1, 15.3.4	M	Yes ❑ No ❑

A.4.7 Infrared baseband PHY functions

Item	Feature	References	Status	Support
IR1	Is the transmitted SYNC field length in the range of required number of PPM slots, with the absence of a pulse in the last slot of the field?	16.2.4.1	M	Yes ❑
IR2	Is the transmitted SYNC field entirely populated by alternating presence and absence of pulses in consecutive PPM slots, with the absence of a pulse in the last slot of the field?	16.2.4.1	M	Yes ❑
IR3	Is the transmitted SFD field the binary sequence 1001, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot?	16.2.4.2	M	Yes ❑
IR4	Is the transmitted DR field pulse sequence equal to the correct value for the data rate provided by the TXVECTOR parameter PLCP BITRATE, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot?	16.2.4.3	M	Yes ❑
IR5	Is the transmitted DCLA field 32 PPM slots long with the specified sequence for 1 Mbit/s, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot? 1 Mbit/s: 00000000100000000000000010000000	16.2.4.4	M	Yes ❑
* IR5a	Does the unit support 2 Mbit/s transmission?	16.2.4.4	O	Yes ❑ No ❑
IR5b	If the unit supports 2 Mbit/s transmission, is the transmitted DCLA field 32 PPM slots long with the specified sequence for 2 Mbit/s, where 1 indicates a pulse in the PPM slot and 0 indicates no pulse in the PPM slot? 2 Mbit/s: 00100010001000100010001000100010	16.2.4.4	IR5a:M	Yes ❑ No ❑ N/A ❑

A.4.7 Infrared baseband PHY functions (continued)

Item	Feature	References	Status	Support
IR6	Is the transmitted LENGTH field the correct PPM representation of the unsigned 16-bit binary integer, lsb transmitted first, equal to the correct value provided by the TXVECTOR parameter LENGTH?	16.2.4.5	M	Yes ❑
IR7	Is the transmitted CRC field the correct PPM representation of the CRC value calculated as per reference subclause, transmitted lsb first?	16.2.4.6	M	Yes ❑
IR8	Is the transmitted PSDU field the correct PPM representation of the PSDU, transmitted lsb first?	16.2.4.7	M	Yes ❑
IR9	When the CCA is false does transmission begin based on PHYTXSTART.request?	16.2.5.1	M	Yes ❑
IR10	Does the PHY issue a PHYTXSTART.confirm after the transmission of the PLCP header?	16.2.5.1	M	Yes ❑
IR11	Does the PHY accept each octet of the PSDU in a PHYDATA.request and answer with a PHYDATA.confirm?	16.2.5.1	M	Yes ❑
IR12	Does the PHY cease transmission in response to a PHYTXEND.request and answer with a PHYTXEND.confirm?	16.2.5.1	M	Yes ❑
IR13	Does the PHY of a receiving STA send a PHYCCA.indicate during reception of the SYNC field?	16.2.5.2	M	Yes ❑
IR14	Does the PHY of a receiving STA properly receive a transmission that changes data rate according to the DR field?	16.2.5.2	M	Yes ❑
IR15	Does the PHY of a receiving STA properly reject an incorrect CRC?	16.2.5.2	M	Yes ❑
IR16	Does the PHY of a receiving STA properly reject a DR field other than those specified in reference subclause?	16.2.5.2, 16.2.4.3	M	Yes ❑
IR17	Does the PHY of a receiving STA send PHYRXSTART.indicate with correct RATE and LENGTH parameters after proper reception of PLCP preamble and PLCP header?	16.2.5.2	M	Yes ❑
IR18	Does the PHY of a receiving STA forward receive octets in PHYDATA.indicate primitives?	16.2.5.2	M	Yes ❑
IR19	Does the PHY of a receiving STA send a PHYRXEND.indicate after the final octet indicated by the LENGTH field?	16.2.5.2	M	Yes ❑
IR20	Does the PHY of a receiving STA send a PHYCCA.indicate with a state value of IDLE after the PHYRXEND.indicate?	16.2.5.2	M	Yes ❑
IR21	Does the PHY reset its CCA detection mechanism upon receiving a PHYCCARST.request, and respond with a PHYCCARST.indicate?	16.2.5.3	M	Yes ❑

A.4.7 Infrared baseband PHY functions (continued)

Item	Feature	References	Status	Support
IR22	When transmitting at 1 Mbit/s does the PHY transmit PPM symbols according the 16-PPM Basic Rate Mapping table, transmitting from left to right?	16.3.2.1, 16.3.2.2	M	Yes ❑
IR23	When transmitting at 2 Mbit/s does the PHY transmit PPM symbols according to the 4-PPM Enhanced Rate Mapping table, transmitting from left to right?	16.3.2.1, 16.3.2.2	IR5a:M	Yes ❑
IR24	Does the PHY operate over a temperature range of 0 to 40 °C?	16.3.2.4	M	Yes ❑
* IR25	If the unit is conformant to emitter radiation mask 1, is the peak optical power of an emitted pulse within the specification range averaged over the pulse width?	16.3.3.1	O.1	Yes ❑ No ❑ N/A ❑
* IR26	If the unit is conformant to emitter radiation mask 2, is the peak optical power of an emitted pulse within the specification range averaged over the pulse width?	16.3.3.1	O.1	Yes ❑ No ❑ N/A ❑
IR27	Does the transmitted pulse shape conform to the description of the reference subclause?	16.3.3.2	M	Yes ❑
IR28	Does the emitter radiation pattern as a function of angle conform to the requirements of the reference subclause as applicable based on conformance to emitter radiation mask 1?	16.3.3.3	IR25:M	Yes ❑ No ❑ N/A ❑
IR28a	Does the emitter radiation pattern as a function of angle conform to the requirements of the reference subclause as applicable based on conformance to emitter radiation mask 2?	16.3.3.3	IR26:M	Yes ❑ No ❑ N/A ❑
IR29	Is the peak emitter optical output as a function of wavelength in the range specified?	16.3.3.4	M	Yes ❑
IR30	Does the spectrum of the transmit signal amplitude as a voltage or current meet the requirements of the reference subclause?	16.3.3.5	M	Yes ❑
IR31	Does the receiver sensitivity meet the requirements of the reference subclause for receive signals of both 1 and 2 Mbit/s?	16.3.4.1	M	Yes ❑
IR32	Does the receiver exhibit a dynamic range as specified in reference subclause?	16.3.4.2	M	Yes ❑
IR33	Does the receiver field of view conform to the requirements of the reference subclause?	16.3.4.3	M	Yes ❑
IR34	When it is known that the conditions are such that the Carrier Detect Signal and the Energy Detect Signal are false is the CCA asserted IDLE?	16.3.5.1	M	Yes ❑

A.4.7 Infrared baseband PHY functions (continued)

Item	Feature	References	Status	Support
IR35	When the conditions are such that Energy Detect is true for greater then the time defined in reference subclause, does CCA become IDLE?	16.3.5.1	M	Yes ❑
IR36	When conditions are such that either Carrier Detect or Energy Detect go true, does CCA go BUSY?	16.3.5.1	M	Yes ❑
IR37	Is the MIB completely supported?	16.4	M	Yes ❑

Annex B

(informative)

Hopping sequences

The following tables pertain to the hopping sequences for North America and ETSI.

Table B.1—Hopping sequence set 1

index	0	3	6	9	12	15	18	21	24	27	30	33	36
1	2	5	8	11	14	17	20	23	26	29	32	35	38
2	25	28	31	34	37	40	43	46	49	52	55	58	61
3	64	67	70	73	76	79	3	6	9	12	15	18	21
4	10	13	16	19	22	25	28	31	34	37	40	43	46
5	45	48	51	54	57	60	63	66	69	72	75	78	2
6	18	21	24	27	30	33	36	39	42	45	48	51	54
7	73	76	79	3	6	9	12	15	18	21	24	27	30
8	49	52	55	58	61	64	67	70	73	76	79	3	6
9	21	24	27	30	33	36	39	42	45	48	51	54	57
10	63	66	69	72	75	78	2	5	8	11	14	17	20
11	78	2	5	8	11	14	17	20	23	26	29	32	35
12	31	34	37	40	43	46	49	52	55	58	61	64	67
13	61	64	67	70	73	76	79	3	6	9	12	15	18
14	24	27	30	33	36	39	42	45	48	51	54	57	60
15	54	57	60	63	66	69	72	75	78	2	5	8	11
16	65	68	71	74	77	80	4	7	10	13	16	19	22
17	28	31	34	37	40	43	46	49	52	55	58	61	64
18	79	3	6	9	12	15	18	21	24	27	30	33	36
19	33	36	39	42	45	48	51	54	57	60	63	66	69
20	4	7	10	13	16	19	22	25	28	31	34	37	40
21	20	23	26	29	32	35	38	41	44	47	50	53	56
22	13	16	19	22	25	28	31	34	37	40	43	46	49
23	38	41	44	47	50	53	56	59	62	65	68	71	74
24	74	77	80	4	7	10	13	16	19	22	25	28	31
25	56	59	62	65	68	71	74	77	80	4	7	10	13
26	71	74	77	80	4	7	10	13	16	19	22	25	28
27	23	26	29	32	35	38	41	44	47	50	53	56	59
28	5	8	11	14	17	20	23	26	29	32	35	38	41
29	39	42	45	48	51	54	57	60	63	66	69	72	75
30	12	15	18	21	24	27	30	33	36	39	42	45	48
31	36	39	42	45	48	51	54	57	60	63	66	69	72
32	68	71	74	77	80	4	7	10	13	16	19	22	25
33	9	12	15	18	21	24	27	30	33	36	39	42	45
34	70	73	76	79	3	6	9	12	15	18	21	24	27
35	77	80	4	7	10	13	16	19	22	25	28	31	34
36	6	9	12	15	18	21	24	27	30	33	36	39	42
37	62	65	68	71	74	77	80	4	7	10	13	16	19
38	29	32	35	38	41	44	47	50	53	56	59	62	65
39	14	17	20	23	26	29	32	35	38	41	44	47	50

Table B.1—Hopping sequence set 1 (continued)

index	0	3	6	9	12	15	18	21	24	27	30	33	36
40	27	30	33	36	39	42	45	48	51	54	57	60	63
41	16	19	22	25	28	31	34	37	40	43	46	49	52
42	59	62	65	68	71	74	77	80	4	7	10	13	16
43	43	46	49	52	55	58	61	64	67	70	73	76	79
44	76	79	3	6	9	12	15	18	21	24	27	30	33
45	34	37	40	43	46	49	52	55	58	61	64	67	70
46	72	75	78	2	5	8	11	14	17	20	23	26	29
47	11	14	17	20	23	26	29	32	35	38	41	44	47
48	60	63	66	69	72	75	78	2	5	8	11	14	17
49	80	4	7	10	13	16	19	22	25	28	31	34	37
50	47	50	53	56	59	62	65	68	71	74	77	80	4
51	22	25	28	31	34	37	40	43	46	49	52	55	58
52	75	78	2	5	8	11	14	17	20	23	26	29	32
53	66	69	72	75	78	2	5	8	11	14	17	20	23
54	41	44	47	50	53	56	59	62	65	68	71	74	77
55	15	18	21	24	27	30	33	36	39	42	45	48	51
56	35	38	41	44	47	50	53	56	59	62	65	68	71
57	67	70	73	76	79	3	6	9	12	15	18	21	24
58	52	55	58	61	64	67	70	73	76	79	3	6	9
59	58	61	64	67	70	73	76	79	3	6	9	12	15
60	44	47	50	53	56	59	62	65	68	71	74	77	80
61	50	53	56	59	62	65	68	71	74	77	80	4	7
62	17	20	23	26	29	32	35	38	41	44	47	50	53
63	7	10	13	16	19	22	25	28	31	34	37	40	43
64	19	22	25	28	31	34	37	40	43	46	49	52	55
65	8	11	14	17	20	23	26	29	32	35	38	41	44
66	69	72	75	78	2	5	8	11	14	17	20	23	26
67	51	54	57	60	63	66	69	72	75	78	2	5	8
68	42	45	48	51	54	57	60	63	66	69	72	75	78
69	3	6	9	12	15	18	21	24	27	30	33	36	39
70	30	33	36	39	42	45	48	51	54	57	60	63	66
71	57	60	63	66	69	72	75	78	2	5	8	11	14
72	37	40	43	46	49	52	55	58	61	64	67	70	73
73	55	58	61	64	67	70	73	76	79	3	6	9	12
74	26	29	32	35	38	41	44	47	50	53	56	59	62
75	46	49	52	55	58	61	64	67	70	73	76	79	3
76	53	56	59	62	65	68	71	74	77	80	4	7	10
77	40	43	46	49	52	55	58	61	64	67	70	73	76
78	32	35	38	41	44	47	50	53	56	59	62	65	68
79	48	51	54	57	60	63	66	69	72	75	78	2	5

Table B.1—Hopping sequence set 1 (continued)

index	39	42	45	48	51	54	57	60	63	66	69	72	75
1	41	44	47	50	53	56	59	62	65	68	71	74	77
2	64	67	70	73	76	79	3	6	9	12	15	18	21
3	24	27	30	33	36	39	42	45	48	51	54	57	60
4	49	52	55	58	61	64	67	70	73	76	79	3	6
5	5	8	11	14	17	20	23	26	29	32	35	38	41
6	57	60	63	66	69	72	75	78	2	5	8	11	14
7	33	36	39	42	45	48	51	54	57	60	63	66	69
8	9	12	15	18	21	24	27	30	33	36	39	42	45
9	60	63	66	69	72	75	78	2	5	8	11	14	17
10	23	26	29	32	35	38	41	44	47	50	53	56	59
11	38	41	44	47	50	53	56	59	62	65	68	71	74
12	70	73	76	79	3	6	9	12	15	18	21	24	27
13	21	24	27	30	33	36	39	42	45	48	51	54	57
14	63	66	69	72	75	78	2	5	8	11	14	17	20
15	14	17	20	23	26	29	32	35	38	41	44	47	50
16	25	28	31	34	37	40	43	46	49	52	55	58	61
17	67	70	73	76	79	3	6	9	12	15	18	21	24
18	39	42	45	48	51	54	57	60	63	66	69	72	75
19	72	75	78	2	5	8	11	14	17	20	23	26	29
20	43	46	49	52	55	58	61	64	67	70	73	76	79
21	59	62	65	68	71	74	77	80	4	7	10	13	16
22	52	55	58	61	64	67	70	73	76	79	3	6	9
23	77	80	4	7	10	13	16	19	22	25	28	31	34
24	34	37	40	43	46	49	52	55	58	61	64	67	70
25	16	19	22	25	28	31	34	37	40	43	46	49	52
26	31	34	37	40	43	46	49	52	55	58	61	64	67
27	62	65	68	71	74	77	80	4	7	10	13	16	19
28	44	47	50	53	56	59	62	65	68	71	74	77	80
29	78	2	5	8	11	14	17	20	23	26	29	32	35
30	51	54	57	60	63	66	69	72	75	78	2	5	8
31	75	78	2	5	8	11	14	17	20	23	26	29	32
32	28	31	34	37	40	43	46	49	52	55	58	61	64
33	48	51	54	57	60	63	66	69	72	75	78	2	5
34	30	33	36	39	42	45	48	51	54	57	60	63	66
35	37	40	43	46	49	52	55	58	61	64	67	70	73
36	45	48	51	54	57	60	63	66	69	72	75	78	2
37	22	25	28	31	34	37	40	43	46	49	52	55	58
38	68	71	74	77	80	4	7	10	13	16	19	22	25
39	53	56	59	62	65	68	71	74	77	80	4	7	10

Table B.1—Hopping sequence set 1 (continued)

index	39	42	45	48	51	54	57	60	63	66	69	72	75
40	66	69	72	75	78	2	5	8	11	14	17	20	23
41	55	58	61	64	67	70	73	76	79	3	6	9	12
42	19	22	25	28	31	34	37	40	43	46	49	52	55
43	3	6	9	12	15	18	21	24	27	30	33	36	39
44	36	39	42	45	48	51	54	57	60	63	66	69	72
45	73	76	79	3	6	9	12	15	18	21	24	27	30
46	32	35	38	41	44	47	50	53	56	59	62	65	68
47	50	53	56	59	62	65	68	71	74	77	80	4	7
48	20	23	26	29	32	35	38	41	44	47	50	53	56
49	40	43	46	49	52	55	58	61	64	67	70	73	76
50	7	10	13	16	19	22	25	28	31	34	37	40	43
51	61	64	67	70	73	76	79	3	6	9	12	15	18
52	35	38	41	44	47	50	53	56	59	62	65	68	71
53	26	29	32	35	38	41	44	47	50	53	56	59	62
54	80	4	7	10	13	16	19	22	25	28	31	34	37
55	54	57	60	63	66	69	72	75	78	2	5	8	11
56	74	77	80	4	7	10	13	16	19	22	25	28	31
57	27	30	33	36	39	42	45	48	51	54	57	60	63
58	12	15	18	21	24	27	30	33	36	39	42	45	48
59	18	21	24	27	30	33	36	39	42	45	48	51	54
60	4	7	10	13	16	19	22	25	28	31	34	37	40
61	10	13	16	19	22	25	28	31	34	37	40	43	46
62	56	59	62	65	68	71	74	77	80	4	7	10	13
63	46	49	52	55	58	61	64	67	70	73	76	79	3
64	58	61	64	67	70	73	76	79	3	6	9	12	15
65	47	50	53	56	59	62	65	68	71	74	77	80	4
66	29	32	35	38	41	44	47	50	53	56	59	62	65
67	11	14	17	20	23	26	29	32	35	38	41	44	47
68	2	5	8	11	14	17	20	23	26	29	32	35	38
69	42	45	48	51	54	57	60	63	66	69	72	75	78
70	69	72	75	78	2	5	8	11	14	17	20	23	26
71	17	20	23	26	29	32	35	38	41	44	47	50	53
72	76	79	3	6	9	12	15	18	21	24	27	30	33
73	15	18	21	24	27	30	33	36	39	42	45	48	51
74	65	68	71	74	77	80	4	7	10	13	16	19	22
75	6	9	12	15	18	21	24	27	30	33	36	39	42
76	13	16	19	22	25	28	31	34	37	40	43	46	49
77	79	3	6	9	12	15	18	21	24	27	30	33	36
78	71	74	77	80	4	7	10	13	16	19	22	25	28
79	8	11	14	17	20	23	26	29	32	35	38	41	44

Table B.2—Hopping sequence set 2

index	1	4	7	10	13	16	19	22	25	28	31	34	37
1	3	6	9	12	15	18	21	24	27	30	33	36	39
2	26	29	32	35	38	41	44	47	50	53	56	59	62
3	65	68	71	74	77	80	4	7	10	13	16	19	22
4	11	14	17	20	23	26	29	32	35	38	41	44	47
5	46	49	52	55	58	61	64	67	70	73	76	79	3
6	19	22	25	28	31	34	37	40	43	46	49	52	55
7	74	77	80	4	7	10	13	16	19	22	25	28	31
8	50	53	56	59	62	65	68	71	74	77	80	4	7
9	22	25	28	31	34	37	40	43	46	49	52	55	58
10	64	67	70	73	76	79	3	6	9	12	15	18	21
11	79	3	6	9	12	15	18	21	24	27	30	33	36
12	32	35	38	41	44	47	50	53	56	59	62	65	68
13	62	65	68	71	74	77	80	4	7	10	13	16	19
14	25	28	31	34	37	40	43	46	49	52	55	58	61
15	55	58	61	64	67	70	73	76	79	3	6	9	12
16	66	69	72	75	78	2	5	8	11	14	17	20	23
17	29	32	35	38	41	44	47	50	53	56	59	62	65
18	80	4	7	10	13	16	19	22	25	28	31	34	37
19	34	37	40	43	46	49	52	55	58	61	64	67	70
20	5	8	11	14	17	20	23	26	29	32	35	38	41
21	21	24	27	30	33	36	39	42	45	48	51	54	57
22	14	17	20	23	26	29	32	35	38	41	44	47	50
23	39	42	45	48	51	54	57	60	63	66	69	72	75
24	75	78	2	5	8	11	14	17	20	23	26	29	32
25	57	60	63	66	69	72	75	78	2	5	8	11	14
26	72	75	78	2	5	8	11	14	17	20	23	26	29
27	24	27	30	33	36	39	42	45	48	51	54	57	60
28	6	9	12	15	18	21	24	27	30	33	36	39	42
29	40	43	46	49	52	55	58	61	64	67	70	73	76
30	13	16	19	22	25	28	31	34	37	40	43	46	49
31	37	40	43	46	49	52	55	58	61	64	67	70	73
32	69	72	75	78	2	5	8	11	14	17	20	23	26
33	10	13	16	19	22	25	28	31	34	37	40	43	46
34	71	74	77	80	4	7	10	13	16	19	22	25	28
35	78	2	5	8	11	14	17	20	23	26	29	32	35
36	7	10	13	16	19	22	25	28	31	34	37	40	43
37	63	66	69	72	75	78	2	5	8	11	14	17	20
38	30	33	36	39	42	45	48	51	54	57	60	63	66
39	15	18	21	24	27	30	33	36	39	42	45	48	51

Table B.2—Hopping sequence set 2 (continued)

index	1	4	7	10	13	16	19	22	25	28	31	34	37
40	28	31	34	37	40	43	46	49	52	55	58	61	64
41	17	20	23	26	29	32	35	38	41	44	47	50	53
42	60	63	66	69	72	75	78	2	5	8	11	14	17
43	44	47	50	53	56	59	62	65	68	71	74	77	80
44	77	80	4	7	10	13	16	19	22	25	28	31	34
45	35	38	41	44	47	50	53	56	59	62	65	68	71
46	73	76	79	3	6	9	12	15	18	21	24	27	30
47	12	15	18	21	24	27	30	33	36	39	42	45	48
48	61	64	67	70	73	76	79	3	6	9	12	15	18
49	2	5	8	11	14	17	20	23	26	29	32	35	38
50	48	51	54	57	60	63	66	69	72	75	78	2	5
51	23	26	29	32	35	38	41	44	47	50	53	56	59
52	76	79	3	6	9	12	15	18	21	24	27	30	33
53	67	70	73	76	79	3	6	9	12	15	18	21	24
54	42	45	48	51	54	57	60	63	66	69	72	75	78
55	16	19	22	25	28	31	34	37	40	43	46	49	52
56	36	39	42	45	48	51	54	57	60	63	66	69	72
57	68	71	74	77	80	4	7	10	13	16	19	22	25
58	53	56	59	62	65	68	71	74	77	80	4	7	10
59	59	62	65	68	71	74	77	80	4	7	10	13	16
60	45	48	51	54	57	60	63	66	69	72	75	78	2
61	51	54	57	60	63	66	69	72	75	78	2	5	8
62	18	21	24	27	30	33	36	39	42	45	48	51	54
63	8	11	14	17	20	23	26	29	32	35	38	41	44
64	20	23	26	29	32	35	38	41	44	47	50	53	56
65	9	12	15	18	21	24	27	30	33	36	39	42	45
66	70	73	76	79	3	6	9	12	15	18	21	24	27
67	52	55	58	61	64	67	70	73	76	79	3	6	9
68	43	46	49	52	55	58	61	64	67	70	73	76	79
69	4	7	10	13	16	19	22	25	28	31	34	37	40
70	31	34	37	40	43	46	49	52	55	58	61	64	67
71	58	61	64	67	70	73	76	79	3	6	9	12	15
72	38	41	44	47	50	53	56	59	62	65	68	71	74
73	56	59	62	65	68	71	74	77	80	4	7	10	13
74	27	30	33	36	39	42	45	48	51	54	57	60	63
75	47	50	53	56	59	62	65	68	71	74	77	80	4
76	54	57	60	63	66	69	72	75	78	2	5	8	11
77	41	44	47	50	53	56	59	62	65	68	71	74	77
78	33	36	39	42	45	48	51	54	57	60	63	66	69
79	49	52	55	58	61	64	67	70	73	76	79	3	6

Table B.2—Hopping sequence set 2 (continued)

index	40	43	46	49	52	55	58	61	64	67	70	73	76
1	42	45	48	51	54	57	60	63	66	69	72	75	78
2	65	68	71	74	77	80	4	7	10	13	16	19	22
3	25	28	31	34	37	40	43	46	49	52	55	58	61
4	50	53	56	59	62	65	68	71	74	77	80	4	7
5	6	9	12	15	18	21	24	27	30	33	36	39	42
6	58	61	64	67	70	73	76	79	3	6	9	12	15
7	34	37	40	43	46	49	52	55	58	61	64	67	70
8	10	13	16	19	22	25	28	31	34	37	40	43	46
9	61	64	67	70	73	76	79	3	6	9	12	15	18
10	24	27	30	33	36	39	42	45	48	51	54	57	60
11	39	42	45	48	51	54	57	60	63	66	69	72	75
12	71	74	77	80	4	7	10	13	16	19	22	25	28
13	22	25	28	31	34	37	40	43	46	49	52	55	58
14	64	67	70	73	76	79	3	6	9	12	15	18	21
15	15	18	21	24	27	30	33	36	39	42	45	48	51
16	26	29	32	35	38	41	44	47	50	53	56	59	62
17	68	71	74	77	80	4	7	10	13	16	19	22	25
18	40	43	46	49	52	55	58	61	64	67	70	73	76
19	73	76	79	3	6	9	12	15	18	21	24	27	30
20	44	47	50	53	56	59	62	65	68	71	74	77	80
21	60	63	66	69	72	75	78	2	5	8	11	14	17
22	53	56	59	62	65	68	71	74	77	80	4	7	10
23	78	2	5	8	11	14	17	20	23	26	29	32	35
24	35	38	41	44	47	50	53	56	59	62	65	68	71
25	17	20	23	26	29	32	35	38	41	44	47	50	53
26	32	35	38	41	44	47	50	53	56	59	62	65	68
27	63	66	69	72	75	78	2	5	8	11	14	17	20
28	45	48	51	54	57	60	63	66	69	72	75	78	2
29	79	3	6	9	12	15	18	21	24	27	30	33	36
30	52	55	58	61	64	67	70	73	76	79	3	6	9
31	76	79	3	6	9	12	15	18	21	24	27	30	33
32	29	32	35	38	41	44	47	50	53	56	59	62	65
33	49	52	55	58	61	64	67	70	73	76	79	3	6
34	31	34	37	40	43	46	49	52	55	58	61	64	67
35	38	41	44	47	50	53	56	59	62	65	68	71	74
36	46	49	52	55	58	61	64	67	70	73	76	79	3
37	23	26	29	32	35	38	41	44	47	50	53	56	59
38	69	72	75	78	2	5	8	11	14	17	20	23	26
39	54	57	60	63	66	69	72	75	78	2	5	8	11

Table B.2—Hopping sequence set 2 (continued)

index	40	43	46	49	52	55	58	61	64	67	70	73	76
40	67	70	73	76	79	3	6	9	12	15	18	21	24
41	56	59	62	65	68	71	74	77	80	4	7	10	13
42	20	23	26	29	32	35	38	41	44	47	50	53	56
43	4	7	10	13	16	19	22	25	28	31	34	37	40
44	37	40	43	46	49	52	55	58	61	64	67	70	73
45	74	77	80	4	7	10	13	16	19	22	25	28	31
46	33	36	39	42	45	48	51	54	57	60	63	66	69
47	51	54	57	60	63	66	69	72	75	78	2	5	8
48	21	24	27	30	33	36	39	42	45	48	51	54	57
49	41	44	47	50	53	56	59	62	65	68	71	74	77
50	8	11	14	17	20	23	26	29	32	35	38	41	44
51	62	65	68	71	74	77	80	4	7	10	13	16	19
52	36	39	42	45	48	51	54	57	60	63	66	69	72
53	27	30	33	36	39	42	45	48	51	54	57	60	63
54	2	5	8	11	14	17	20	23	26	29	32	35	38
55	55	58	61	64	67	70	73	76	79	3	6	9	12
56	75	78	2	5	8	11	14	17	20	23	26	29	32
57	28	31	34	37	40	43	46	49	52	55	58	61	64
58	13	16	19	22	25	28	31	34	37	40	43	46	49
59	19	22	25	28	31	34	37	40	43	46	49	52	55
60	5	8	11	14	17	20	23	26	29	32	35	38	41
61	11	14	17	20	23	26	29	32	35	38	41	44	47
62	57	60	63	66	69	72	75	78	2	5	8	11	14
63	47	50	53	56	59	62	65	68	71	74	77	80	4
64	59	62	65	68	71	74	77	80	4	7	10	13	16
65	48	51	54	57	60	63	66	69	72	75	78	2	5
66	30	33	36	39	42	45	48	51	54	57	60	63	66
67	12	15	18	21	24	27	30	33	36	39	42	45	48
68	3	6	9	12	15	18	21	24	27	30	33	36	39
69	43	46	49	52	55	58	61	64	67	70	73	76	79
70	70	73	76	79	3	6	9	12	15	18	21	24	27
71	18	21	24	27	30	33	36	39	42	45	48	51	54
72	77	80	4	7	10	13	16	19	22	25	28	31	34
73	16	19	22	25	28	31	34	37	40	43	46	49	52
74	66	69	72	75	78	2	5	8	11	14	17	20	23
75	7	10	13	16	19	22	25	28	31	34	37	40	43
76	14	17	20	23	26	29	32	35	38	41	44	47	50
77	80	4	7	10	13	16	19	22	25	28	31	34	37
78	72	75	78	2	5	8	11	14	17	20	23	26	29
79	9	12	15	18	21	24	27	30	33	36	39	42	45

Table B.3—Hopping sequence set 3

index	2	5	8	11	14	17	20	23	26	29	32	35	38
1	4	7	10	13	16	19	22	25	28	31	34	37	40
2	27	30	33	36	39	42	45	48	51	54	57	60	63
3	66	69	72	75	78	2	5	8	11	14	17	20	23
4	12	15	18	21	24	27	30	33	36	39	42	45	48
5	47	50	53	56	59	62	65	68	71	74	77	80	4
6	20	23	26	29	32	35	38	41	44	47	50	53	56
7	75	78	2	5	8	11	14	17	20	23	26	29	32
8	51	54	57	60	63	66	69	72	75	78	2	5	8
9	23	26	29	32	35	38	41	44	47	50	53	56	59
10	65	68	71	74	77	80	4	7	10	13	16	19	22
11	80	4	7	10	13	16	19	22	25	28	31	34	37
12	33	36	39	42	45	48	51	54	57	60	63	66	69
13	63	66	69	72	75	78	2	5	8	11	14	17	20
14	26	29	32	35	38	41	44	47	50	53	56	59	62
15	56	59	62	65	68	71	74	77	80	4	7	10	13
16	67	70	73	76	79	3	6	9	12	15	18	21	24
17	30	33	36	39	42	45	48	51	54	57	60	63	66
18	2	5	8	11	14	17	20	23	26	29	32	35	38
19	35	38	41	44	47	50	53	56	59	62	65	68	71
20	6	9	12	15	18	21	24	27	30	33	36	39	42
21	22	25	28	31	34	37	40	43	46	49	52	55	58
22	15	18	21	24	27	30	33	36	39	42	45	48	51
23	40	43	46	49	52	55	58	61	64	67	70	73	76
24	76	79	3	6	9	12	15	18	21	24	27	30	33
25	58	61	64	67	70	73	76	79	3	6	9	12	15
26	73	76	79	3	6	9	12	15	18	21	24	27	30
27	25	28	31	34	37	40	43	46	49	52	55	58	61
28	7	10	13	16	19	22	25	28	31	34	37	40	43
29	41	44	47	50	53	56	59	62	65	68	71	74	77
30	14	17	20	23	26	29	32	35	38	41	44	47	50
31	38	41	44	47	50	53	56	59	62	65	68	71	74
32	70	73	76	79	3	6	9	12	15	18	21	24	27
33	11	14	17	20	23	26	29	32	35	38	41	44	47
34	72	75	78	2	5	8	11	14	17	20	23	26	29
35	79	3	6	9	12	15	18	21	24	27	30	33	36
36	8	11	14	17	20	23	26	29	32	35	38	41	44
37	64	67	70	73	76	79	3	6	9	12	15	18	21
38	31	34	37	40	43	46	49	52	55	58	61	64	67
39	16	19	22	25	28	31	34	37	40	43	46	49	52

Table B.3—Hopping sequence set 3 (continued)

index	2	5	8	11	14	17	20	23	26	29	32	35	38
40	29	32	35	38	41	44	47	50	53	56	59	62	65
41	18	21	24	27	30	33	36	39	42	45	48	51	54
42	61	64	67	70	73	76	79	3	6	9	12	15	18
43	45	48	51	54	57	60	63	66	69	72	75	78	2
44	78	2	5	8	11	14	17	20	23	26	29	32	35
45	36	39	42	45	48	51	54	57	60	63	66	69	72
46	74	77	80	4	7	10	13	16	19	22	25	28	31
47	13	16	19	22	25	28	31	34	37	40	43	46	49
48	62	65	68	71	74	77	80	4	7	10	13	16	19
49	3	6	9	12	15	18	21	24	27	30	33	36	39
50	49	52	55	58	61	64	67	70	73	76	79	3	6
51	24	27	30	33	36	39	42	45	48	51	54	57	60
52	77	80	4	7	10	13	16	19	22	25	28	31	34
53	68	71	74	77	80	4	7	10	13	16	19	22	25
54	43	46	49	52	55	58	61	64	67	70	73	76	79
55	17	20	23	26	29	32	35	38	41	44	47	50	53
56	37	40	43	46	49	52	55	58	61	64	67	70	73
57	69	72	75	78	2	5	8	11	14	17	20	23	26
58	54	57	60	63	66	69	72	75	78	2	5	8	11
59	60	63	66	69	72	75	78	2	5	8	11	14	17
60	46	49	52	55	58	61	64	67	70	73	76	79	3
61	52	55	58	61	64	67	70	73	76	79	3	6	9
62	19	22	25	28	31	34	37	40	43	46	49	52	55
63	9	12	15	18	21	24	27	30	33	36	39	42	45
64	21	24	27	30	33	36	39	42	45	48	51	54	57
65	10	13	16	19	22	25	28	31	34	37	40	43	46
66	71	74	77	80	4	7	10	13	16	19	22	25	28
67	53	56	59	62	65	68	71	74	77	80	4	7	10
68	44	47	50	53	56	59	62	65	68	71	74	77	80
69	5	8	11	14	17	20	23	26	29	32	35	38	41
70	32	35	38	41	44	47	50	53	56	59	62	65	68
71	59	62	65	68	71	74	77	80	4	7	10	13	16
72	39	42	45	48	51	54	57	60	63	66	69	72	75
73	57	60	63	66	69	72	75	78	2	5	8	11	14
74	28	31	34	37	40	43	46	49	52	55	58	61	64
75	48	51	54	57	60	63	66	69	72	75	78	2	5
76	55	58	61	64	67	70	73	76	79	3	6	9	12
77	42	45	48	51	54	57	60	63	66	69	72	75	78
78	34	37	40	43	46	49	52	55	58	61	64	67	70
79	50	53	56	59	62	65	68	71	74	77	80	4	7

Table B.3—Hopping sequence set 3 (continued)

index	41	44	47	50	53	56	59	62	65	68	71	74	77
1	43	46	49	52	55	58	61	64	67	70	73	76	79
2	66	69	72	75	78	2	5	8	11	14	17	20	23
3	26	29	32	35	38	41	44	47	50	53	56	59	62
4	51	54	57	60	63	66	69	72	75	78	2	5	8
5	7	10	13	16	19	22	25	28	31	34	37	40	43
6	59	62	65	68	71	74	77	80	4	7	10	13	16
7	35	38	41	44	47	50	53	56	59	62	65	68	71
8	11	14	17	20	23	26	29	32	35	38	41	44	47
9	62	65	68	71	74	77	80	4	7	10	13	16	19
10	25	28	31	34	37	40	43	46	49	52	55	58	61
11	40	43	46	49	52	55	58	61	64	67	70	73	76
12	72	75	78	2	5	8	11	14	17	20	23	26	29
13	23	26	29	32	35	38	41	44	47	50	53	56	59
14	65	68	71	74	77	80	4	7	10	13	16	19	22
15	16	19	22	25	28	31	34	37	40	43	46	49	52
16	27	30	33	36	39	42	45	48	51	54	57	60	63
17	69	72	75	78	2	5	8	11	14	17	20	23	26
18	41	44	47	50	53	56	59	62	65	68	71	74	77
19	74	77	80	4	7	10	13	16	19	22	25	28	31
20	45	48	51	54	57	60	63	66	69	72	75	78	2
21	61	64	67	70	73	76	79	3	6	9	12	15	18
22	54	57	60	63	66	69	72	75	78	2	5	8	11
23	79	3	6	9	12	15	18	21	24	27	30	33	36
24	36	39	42	45	48	51	54	57	60	63	66	69	72
25	18	21	24	27	30	33	36	39	42	45	48	51	54
26	33	36	39	42	45	48	51	54	57	60	63	66	69
27	64	67	70	73	76	79	3	6	9	12	15	18	21
28	46	49	52	55	58	61	64	67	70	73	76	79	3
29	80	4	7	10	13	16	19	22	25	28	31	34	37
30	53	56	59	62	65	68	71	74	77	80	4	7	10
31	77	80	4	7	10	13	16	19	22	25	28	31	34
32	30	33	36	39	42	45	48	51	54	57	60	63	66
33	50	53	56	59	62	65	68	71	74	77	80	4	7
34	32	35	38	41	44	47	50	53	56	59	62	65	68
35	39	42	45	48	51	54	57	60	63	66	69	72	75
36	47	50	53	56	59	62	65	68	71	74	77	80	4
37	24	27	30	33	36	39	42	45	48	51	54	57	60
38	70	73	76	79	3	6	9	12	15	18	21	24	27
39	55	58	61	64	67	70	73	76	79	3	6	9	12

Table B.3—Hopping sequence set 3 (continued)

index	41	44	47	50	53	56	59	62	65	68	71	74	77
40	68	71	74	77	80	4	7	10	13	16	19	22	25
41	57	60	63	66	69	72	75	78	2	5	8	11	14
42	21	24	27	30	33	36	39	42	45	48	51	54	57
43	5	8	11	14	17	20	23	26	29	32	35	38	41
44	38	41	44	47	50	53	56	59	62	65	68	71	74
45	75	78	2	5	8	11	14	17	20	23	26	29	32
46	34	37	40	43	46	49	52	55	58	61	64	67	70
47	52	55	58	61	64	67	70	73	76	79	3	6	9
48	22	25	28	31	34	37	40	43	46	49	52	55	58
49	42	45	48	51	54	57	60	63	66	69	72	75	78
50	9	12	15	18	21	24	27	30	33	36	39	42	45
51	63	66	69	72	75	78	2	5	8	11	14	17	20
52	37	40	43	46	49	52	55	58	61	64	67	70	73
53	28	31	34	37	40	43	46	49	52	55	58	61	64
54	3	6	9	12	15	18	21	24	27	30	33	36	39
55	56	59	62	65	68	71	74	77	80	4	7	10	13
56	76	79	3	6	9	12	15	18	21	24	27	30	33
57	29	32	35	38	41	44	47	50	53	56	59	62	65
58	14	17	20	23	26	29	32	35	38	41	44	47	50
59	20	23	26	29	32	35	38	41	44	47	50	53	56
60	6	9	12	15	18	21	24	27	30	33	36	39	42
61	12	15	18	21	24	27	30	33	36	39	42	45	48
62	58	61	64	67	70	73	76	79	3	6	9	12	15
63	48	51	54	57	60	63	66	69	72	75	78	2	5
64	60	63	66	69	72	75	78	2	5	8	11	14	17
65	49	52	55	58	61	64	67	70	73	76	79	3	6
66	31	34	37	40	43	46	49	52	55	58	61	64	67
67	13	16	19	22	25	28	31	34	37	40	43	46	49
68	4	7	10	13	16	19	22	25	28	31	34	37	40
69	44	47	50	53	56	59	62	65	68	71	74	77	80
70	71	74	77	80	4	7	10	13	16	19	22	25	28
71	19	22	25	28	31	34	37	40	43	46	49	52	55
72	78	2	5	8	11	14	17	20	23	26	29	32	35
73	17	20	23	26	29	32	35	38	41	44	47	50	53
74	67	70	73	76	79	3	6	9	12	15	18	21	24
75	8	11	14	17	20	23	26	29	32	35	38	41	44
76	15	18	21	24	27	30	33	36	39	42	45	48	51
77	2	5	8	11	14	17	20	23	26	29	32	35	38
78	73	76	79	3	6	9	12	15	18	21	24	27	30
79	10	13	16	19	22	25	28	31	34	37	40	43	46

Annex C

(normative)

Formal description of MAC operation

This annex contains formal descriptions of the behavior of MAC station (STA) and access point (AP) entities. These descriptions also describe the frame formats and the generation and interpretation of information encoded in MAC frames, in the parameters of service primitives supported by the MAC, and in MIB attributes used or generated by the MAC. The MAC is described using the 1992 version of the ITU Specification and Description Language (SDL-92). SDL-92 is defined in ITU-T Recommendation Z.100 (03/93). An update to Z.100 was approved in 1996 (SDL-96), but none of the SDL facilities used in this annex were modified. An introduction to the MAC formal description is provided in Clause C.1. Definitions of the data types and operators used by the MAC state machines are provided in Clause C.2. An SDL system describing MAC operation at an IEEE 802.11 station is contained in Clause C.3. Finally, a subset of an SDL system describing the aspects of MAC operation at an IEEE 802.11 AP that differ from operation at a non-AP station is provided in Clause C.4.

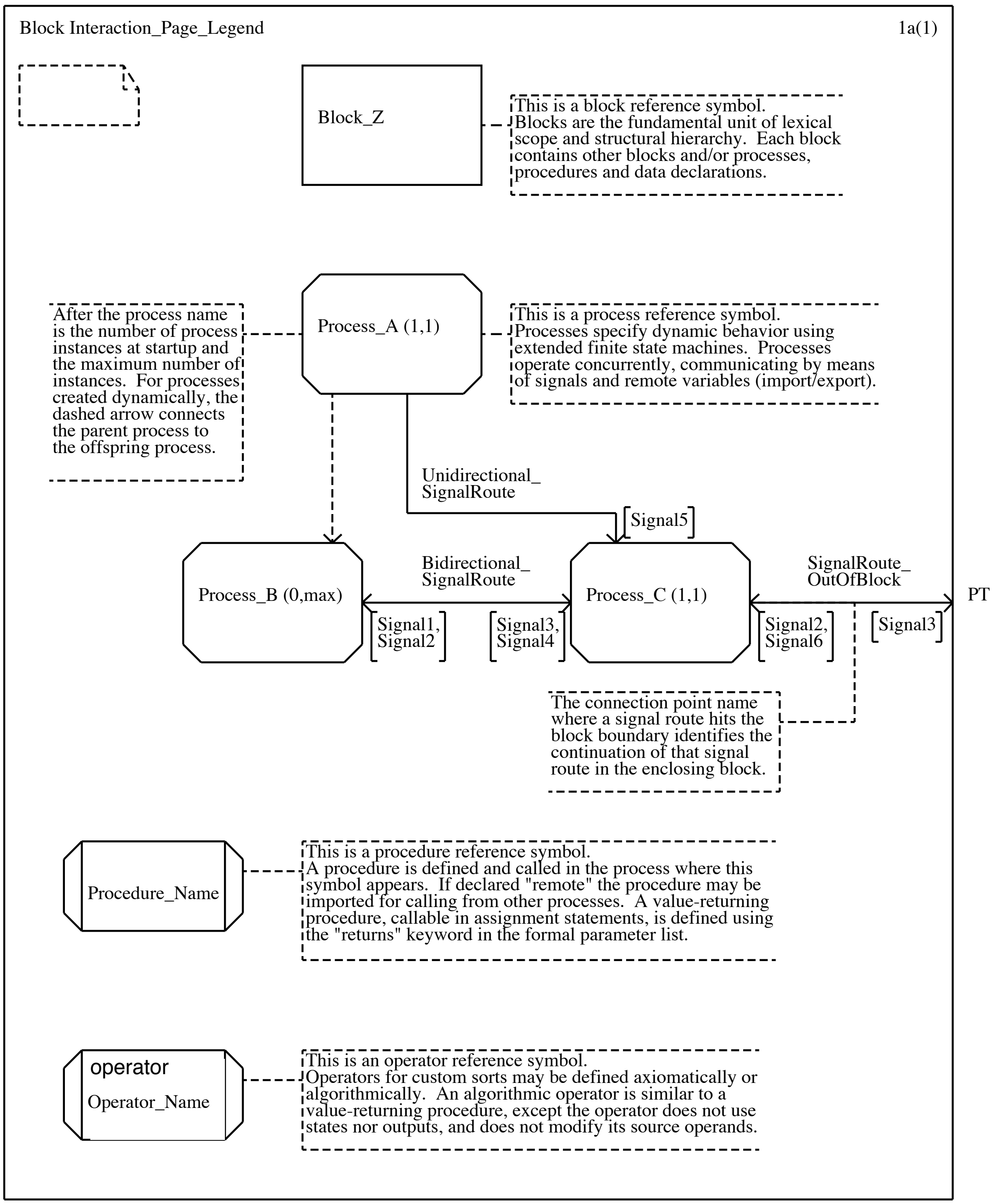
In Annex D, the MAC and PHY management information bases are described in Abstract Syntax Notation One (ASN.1), defined in ISO/IEC 8824 and ISO/IEC 8825. ITU-T Recommendation Z.105 (03/95) defines the use of SDL in conjunction with ASN.1, allowing system behavior to be defined using SDL and data types to be defined using ASN.1. Incomplete tool support precluded the use of Z.105 in this annex. However, within the limits of Z.100, the data types in Clause C.2 are defined in a similar manner to Z.105. Annex E contains a listing of available documentation.

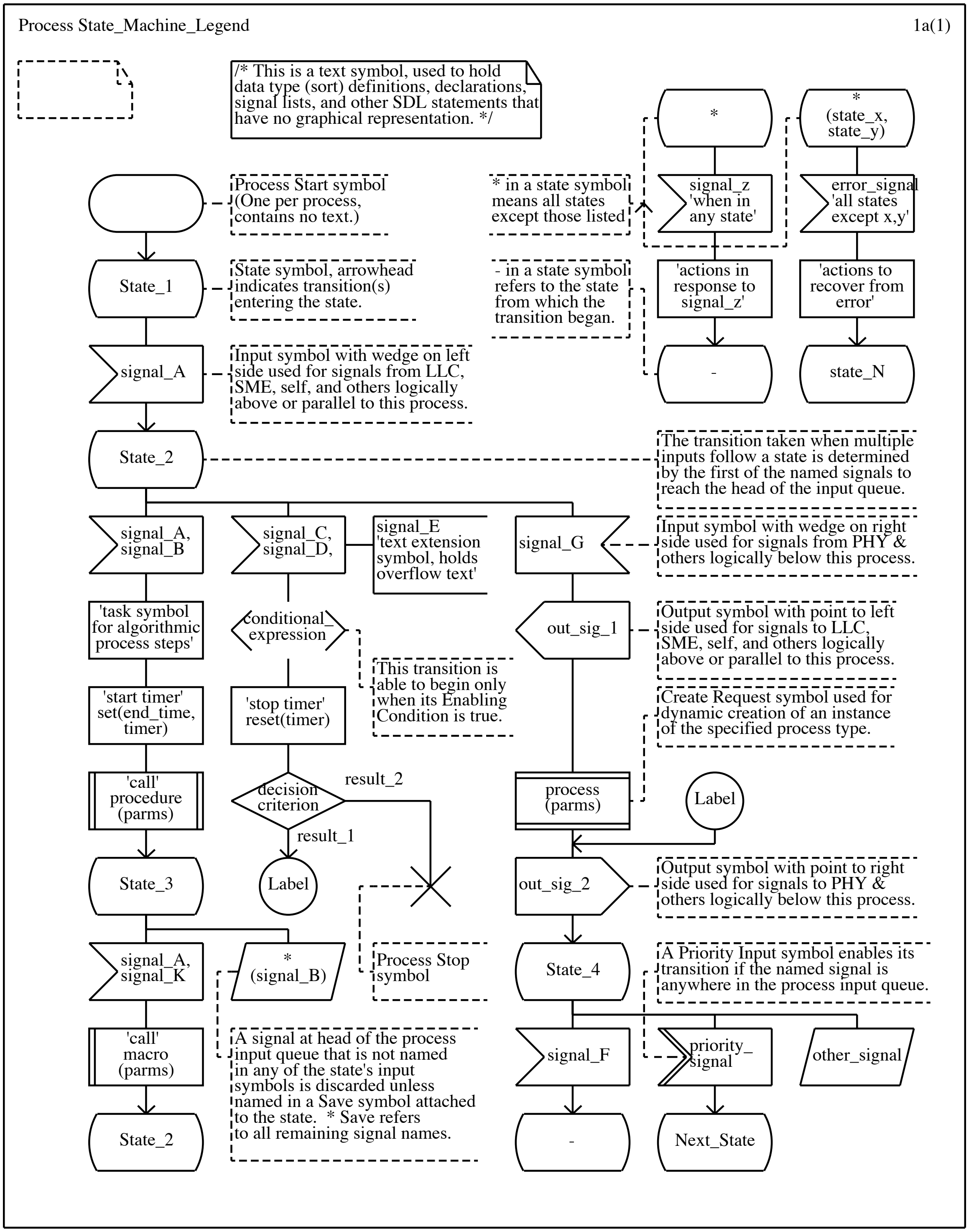
NOTES

1—Software for generating, analyzing, verifying, and simulating SDL system descriptions is available from several sources. The SDL code in this annex was generated using *SDT/PC version 3.02*; from Telelogic AB, Malmo, Sweden (+46-40-174700; internet: telelogic.se); USA office in Princeton, NJ (+1-609-520-1935; internet: telelogic.com). Telelogic offers SDT for several workstation platforms in addition to SDT/PC.

2—The SDL definitions in this annex should be usable with any SDL tool that supports the 1993 version or 1996 update of Recommendation Z.100. The use of Telelogic's product to prepare this annex does not constitute an endorsement of SDT by the IEEE LAN/MAN Standards Committee or by the IEEE.

2—The diagrams on the next two pages show most of the symbols of SDL graphical syntax (SDL-GR) used in the MAC formal description. The symbols in these diagrams have labels and comments that explain their meanings. These diagrams are intended to serve as a legend for the SDL-GR symbols that comprise most of the process interaction and state transition diagrams. These diagrams are neither a complete SDL system, nor a complete presentation of SDL-GR symbology. Also, this state machine fragment exists to illustrate the SDL graphical syntax, and does not describe any useful behavior.


Block Interaction_Page_Legend
1a(1)
Block_Z
This is a block reference symbol.
Blocks are the fundamental unit of lexical scope and structural hierarchy. Each block contains other blocks and/or processes, procedures and data declarations.
After the process name is the number of process instances at startup and the maximum number of instances. For processes created dynamically, the dashed arrow connects the parent process to the offspring process.
Process_A (1,1)
This is a process reference symbol.
Processes specify dynamic behavior using extended finite state machines. Processes operate concurrently, communicating by means of signals and remote variables (import/export).
Unidirectional_
SignalRoute
Signal5
Bidirectional_
SignalRoute
SignalRoute_
OutOfBlock
Process_B (0,max)
Process_C (1,1)
PT
Signal1,
Signal2
Signal3,
Signal4
Signal2,
Signal6
Signal3
The connection point name where a signal route hits the block boundary identifies the continuation of that signal route in the enclosing block.
Procedure_Name
This is a procedure reference symbol.
A procedure is defined and called in the process where this symbol appears. If declared "remote" the procedure may be imported for calling from other processes. A value-returning procedure, callable in assignment statements, is defined using the "returns" keyword in the formal parameter list.
operator
Operator_Name
This is an operator reference symbol.
Operators for custom sorts may be defined axiomatically or algorithmically. An algorithmic operator is similar to a value-returning procedure, except the operator does not use states nor outputs, and does not modify its source operands.


Process State_Machine_Legend
1a(1)
/* 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, SME, self, and others logically above or parallel to this process.
State_2
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_A, signal_B
signal_C, signal_D,
signal_E 'text extension symbol, holds overflow text'
signal_G
Input symbol with wedge on right side used for signals from PHY & others logically below this process.
'task symbol for algorithmic process steps'
conditional_ expression
This transition is able to begin only when its Enabling Condition is true.
out_sig_1
Output symbol with point to left side used for signals to LLC, SME, self, and others logically above or parallel to this process.
'start timer' set(end_time, timer)
'stop timer' reset(timer)
Create Request symbol used for dynamic creation of an instance of the specified process type.
'call' procedure (parms)
decision criterion
result_2
result_1
process (parms)
Label
State_3
Label
out_sig_2
Output symbol with point to right side used for signals to PHY & others logically below this process.
signal_A, signal_K
* (signal_B)
Process Stop symbol
State_4
A Priority Input symbol enables its transition if the named signal is anywhere in the process input queue.
'call' macro (parms)
A signal at head of the process input queue that is not named in any of the state's input symbols is discarded unless named in a Save symbol attached to the state. * Save refers to all remaining signal names.
signal_F
priority_ signal
other_signal
State_2
-
Next_State
*
* (state_x, state_y)
* in a state symbol means all states except those listed
signal_z 'when in any state'
error_signal 'all states except x,y'
- in a state symbol refers to the state from which the transition began.
'actions in response to signal_z'
'actions to recover from error'
-
state_N

C.1 Introduction to the MAC formal description

This formal description defines the behavior of IEEE 802.11 MAC entities. The MAC protocol functional decomposition used herein facilitates explicit description of the reference points and durations of the various timed intervals; the bases for generation and/or validation of header fields, service parameters, and MIB attributes; and the interpretation of each value in cases where enumerated data types are used in service parameters.

C.1.1 Fundamental assumptions

The MAC protocol is described as an SDL system, which is a set of extended finite state machines. Each state machine is a set of independent processes, all of which operate concurrently. All variable data-holding entities and procedures exist solely within the context of a single process. In SDL all interprocess communication is done with signals (there are no global variables). Signals may be sent and received explicitly, using SDL's output and input symbols, or implicitly, using SDL's export/import mechanism (only if the variables or procedures are declared "remote"). By default, signals incur delays when traversing channels between blocks; however, only nondelaying channels and signal routes are used in the MAC state machines, and all remote variables and procedures are declared with the "nodelay" property.

State transitions, procedure calls, and tasks (assignment statements and other algorithmic processing steps) are assumed to require zero time. This permits the time intervals that are part of the normative MAC behavior to be defined explicitly, using SDL timers. One unit of system time (a 1.0 change in the value of "now") is assumed to represent one microsecond of real time. Usec (microsecond) and TU (time unit) data types are defined, with operators to convert Usec and TU values to SDL time or duration when necessary.

The SDL system boundary encloses the MAC entities. The LLC, SME, PHY, and distribution system are part of the environment. SDL generally assumes that entities in the environment operate as specified; however, the MAC state machines that communicate with the various SAPs attempt to validate inputs from the environment, and to handle cases where a pair of communicating entities, one within the system and the other outside the system boundary, have different local views of the medium, station, or service state. All stations in an IEEE 802.11 service set are assumed to exhibit the behaviors described herein. Nevertheless, because of the open nature of the wireless medium, the MAC state machines check for error cases that can arise only when an entity on the wireless medium is transmitting IEEE 802.11 PDUs, but is not obeying the communication protocols specified by this standard.

C.1.2 Notation conventions

When practical, names used in the clauses of this standard are spelled identically in this annex. The principal exceptions are those names that conflict with one of SDL's reserved words (such as power management mode "active" is renamed "sta_active" in SDL). To help fit the SDL text into the graphic symbols, acronyms with multiple, sequential capital letters are written with only the first letter capitalized (e.g., "MSDU" is written "Msdu" and "MLMEJoin.request" is written "MlmeJoin.request").

SDL reserved words and the names of variables and synonyms (named constants) begin with lowercase letters. The names of sorts (data types), signals, signal routes, channels, blocks, and processes begin with uppercase letters. The names of certain groups of variables and/or synonyms begin with a particular lowercase letter, followed by the remainder of the name, beginning with an uppercase letter. These groups are

"aNameOfAttribute"	MIB attributes
"cNameOfCapability"	Capability bits, also used for internal values exported as MIB counters
"dNameOfDuration"	Duration (relative time) values, declared as Usec, TU, or Duration
"eNameOfElement"	Element ID values

“mNameOfVariable”	Remote variables used for intra-MAC communication, but not part of the MIB. Most of these variables are exported from the MLME block.
“sNameOfStaticValue”	Synonyms for static data values used within the MAC
“tNameOfTime”	Time (absolute time) values, declared as Usec, TU, or Time. The names of timers begin with “T.”

C.1.3 Modeling techniques

State machines are grouped according to defined function sets that are visible, directly or indirectly, at an exposed interface. The emphasis in the organization of the state machines is explicitly to show initiation of and response to events at the exposed interfaces, and time-related actions, including those dependent on the absence of external events (e.g., response timeouts) and intervals measured in derived units (e.g., backoff “time” in units of slots during which the wireless medium is idle). The operations associated with the various state transitions emphasize communication functions. Most of the details regarding insertion, extraction, and encoding of information in fields of the protocol data units is encapsulated with the definitions of those fields. This approach, which relies heavily on SDL’s abstract data type and inheritance mechanisms, permits the behavior of the data-holding entities to be precisely defined, without obscuring process flow by adding in-line complexity to the individual state transitions.

The modeling of protocol and service data units requires sorts such as octet strings, and operators such as bitwise boolean functions, which are not predefined in SDL. These sorts and operators are defined in Package macsorts, which appears in Clause C.2.

Protocol and service data unit sorts are based on the **Bit** sort. Bit is a subtype of SDL’s predefined Boolean sort. As a result, Bit literals “0” and “1”are alternative names for “false” and “true,” and have no numeric significance. To use “0” or “1” as integer values requires a conversion operation. Items of the **Bitstring** sort are 0-origin, variable-length strings of Bits. With Bitstring operands, operators “and,” “or,” “xor,” and “not”operate bitwise, with the length of the result equal to the length of the longest (or only) source string. The **Octet** sort is a subtype of Bitstring that adds conversion operators to and from Integer. Each item of the Octet sort has length=8 {by usage convention in Z.100, enforced in Z.105}. Items of the **Octetstring** sort are 0-origin, variable-length strings of Octets. The **Frame** sort is a subtype of Octetstring that adds operators to extract and to modify all MAC header fields and most other MAC frame fields and elements. Most MAC fields and elements that contain named values with specific value assignments or enumerations are defined as subtypes of Frame, Octetstring, or Bitstring with the names added as literals or synonyms, so the state machines can refer to the names without introducing ambiguity about the value encodings.

Where communication at a SAP or between processes is strictly FIFO, the (implicit) input queue of the SDL processes is used. When more sophisticated queue management is needed, a queue whose entries are instances of one, specified sort is created using the **Queue** generator. Entries on Queue sorts may be added and removed at either the tail or the head, and the number of queue entries may be determined. The contents of a Queue may also be searched to locate entries with particular parameter values.

Clause C.2 contains an SDL-92 Package (a named collection of SDL definitions that can be included by reference into SDL System specification), which is a formal description of the formats and data encodings used in IEEE 802.11 service data units, protocol data units, and the parameters of the service primitives used at each of the service access points supported by the IEEE 802.11 MAC. This package also contains definitions for some data structures and operators used internally by one or more of the MAC state machines.

The behaviors of many intra-MAC operators are part of the normative description of the MAC protocol because results of the specified operations are visible, directly or indirectly, at exposed interfaces. For example, custom operators are used to define the generation of the CRC-32 value used in the FCS field (operator crc32, page 330), the calculation of frame transmission time used as part of the value in the Duration/ID field in certain types of frames (operator calcDur, page 343), the comparison of the values of particular fields of a received MAC header with cached data values as part of the procedure for detecting duplicate frames (operator searchTupleCache, page 320), and numerous other aspects of frame formats and information encoding. On the other hand, data structures used solely for intra-MAC storage or for transferring of information between different state machines of a single station or access point, are only normative to the extent that they define items of internal state and the temporal sequence necessary for proper operation of the MAC protocol. The specific structures and encodings used for internal data storage and communication functions in this formal description do *not* constrain MAC implementations, provided those implementations exhibit the specified behaviors at the defined service access points and, in conjunction with an appropriate PHY, on the wireless medium.

C.2 Data type and operator definitions for the MAC state machines

This clause is in SDL/PR (phrase notation), with the exception of procedural operators, which are defined in SDL/GR (graphic notation). Package macsorts contains the definitions of the sorts (data types with associated operators and literals) and synonyms (named constants) used by the MAC state machines. Package macmib defines data types for attributes in the MAC MIB, and portions of the PHY MIB, accessed by the MAC state machines. Package macmib exists solely to satisfy SDL's strong type checking in the absence of an SDL tool that fully supports Z.105 (the combined use of SDL with ASN.1).

```
/********************************************************************/
Package macsorts;


/********************************************************************
 *     Enumerated types used within the MAC state machines
 ********************************************************************/
newtype ChangeType                 /* type of change due at the next boundary */
  literals          dwell,         /* dwell (only with FH PHY) */
                    mocp;          /* medium occupancy (only with PCF) */
endnewtype ChangeType;

newtype Imed       /* priority for queuing MMPDUs, relative to MSDUs */
  literals          head,          /* place MMPDU at head of transmit queue */
                    norm;          /* place MMPDU at tail of transmit queue */
endnewtype Imed;

newtype NavSrc     /* source of duration in SetNav & ClearNav signals */
  literals    rts,                        /* RTS frame */
              cfpBss,  cfendBss,          /* start/end of CFP in own BSS */
              cfpOther,  cfendOther,      /* start/end of CFP in other BSS */
              cswitch,                    /* channel switch */
              misc,                       /* durId from other frame types */
              nosrc;                      /* non-reception events */
endnewtype NavSrc;

newtype PsMode         /* power-save mode of a station (PsResponse signal) */
  literals  sta_active,  power_save,  unknown;   endnewtype PsMode;

newtype PsState        /* power-save state of this station */
  literals  awake,  doze;   endnewtype PsState;

newtype StateErr       /* requests disasoc or deauth (MmIndicate signal) */
  literals  noerr,  class2,  class3;   endnewtype StateErr;

newtype StationState   /* asoc/auth state of sta (SsResponse signal) */
  literals  not_auth,  auth_open,  auth_key,  asoc,  dis_asoc;
endnewtype StationState;

newtype TxResult       /* transmission attempt status (PduConfirm signal) */
```



```
  literals  successful,  partial,  retryLimit,  txLifetime
  atimAck,  atimNak;   endnewtype TxResult;

/*******************************************************************
 *     Enumerated types used in PHY service primitives
 *******************************************************************/
newtype CcaStatus      /* <state> parameter of PhyCca.indication */
  literals  busy,  idle;   endnewtype CcaStatus;

newtype PhyRxStat      /* <rxerror> parameter of PhyRxEnd.indication */
  literals  no_error,  fmt_violation,  carrier_lost,  unsupt_rate;
endnewtype PhyRxStat;

/*******************************************************************
 *     PLACEHOLDERS FOR MLME/PLME GET/SET PARAMETER VALUES
 *******************************************************************/
       /* MibAtrib (placeholder in MlmeGet/Set definitions) */
syntype MibAtrib = Charstring   endsyntype MibAtrib;

       /* MibValue (placeholder in MlmeGet/Set definitions) */
syntype MibValue = Integer   endsyntype MibValue;

/*******************************************************************
 *     Enumerated types used in MAC and MLME service primitives
 *******************************************************************/
newtype AuthTyp    /* <authentication type> parm in Mlme primitives */
  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;

newtype BssType     /* <BSS type> parameter & BSS description element */
  literals  infrastructure,  independent,  any_bss;   endnewtype BssType;

newtype BssTypeSet  powerset( BssType);   endnewtype BssTypeSet;

newtype CfPriority  /* <priority> parameter of various requests */
  literals  contention,  contentionFree;   endnewtype CfPriority;

newtype MibStatus   /* <status> parm of Mlme/Plme Get/Set.confirm */
  literals  success,   invalid,   write_only,   read_only;
endnewtype MibStatus;

newtype MlmeStatus  /* <status> parm of Mlme operation confirm */
  literals  success,  invalid,  timeout,  refused,
              tomany_req,  already_bss;   endnewtype MlmeStatus;

newtype PwrSave     /* <power save mode> parameter of MlmePowerMgt */
  literals  sta_active,  power_save;   endnewtype PwrSave;

newtype Routing     /* <routing info> parameter for MAC data service */
  literals  null_rt;   endnewtype Routing;

newtype RxStatus    /* <reception status> parm of MaUnitdata indication */
  literals  rx_success,  rx_failure;   endnewtype RxStatus;

newtype ScanType    /* <scan type> parameter of MlmeScan.request */
  literals  active_scan,  passive_scan;   endnewtype ScanType;

newtype ServiceClass  /* <service class> parameter for MaUnitdata */
  literals  reorderable,  strictlyOrdered;   endnewtype ServiceClass;
```

```
newtype TxStatus  /* <transmission status> parm of MaUnitdataStatus */
  literals  successful,   retryLimit,   txLifetime,   noBss,
              excessiveDataLength,       nonNullSourceRouting,
              unsupportedPriority,       unavailablePriority,
              unsupportedServiceClass,   unavailableServiceClass,
              unavailableKeyMapping;     endnewtype TxStatus;

/*******************************************************************
 *     Intra-MAC remote variables (names of form mXYZ)
 *******************************************************************/
remote mActingAsAp Boolean nodelay;       /* =true if STA started BSS */
remote mAId  AsocId nodelay;              /* AID assigned to STA by AP */
remote mAssoc  Boolean nodelay;           /* =true if STA associated w/BSS */
remote mAtimW  Boolean nodelay;           /* =true if ATIM window in prog */
remote mBkIP  Boolean nodelay;            /* =true if backoff in prog */
remote mBrates Ratestring nodelay;        /* basic rate set for this sta */
remote mBssId  MacAddr nodelay;           /* identifier of current (I)BSS */
remote mCap  Octetstring nodelay;         /* capability info from MlmeJoin */
remote mCfp  Boolean nodelay;             /* =true if CF period in progress */
remote mDisable  Boolean nodelay;         /* =true if not in any BSS; then */
                                          /* TX only sends probe_req; RX */
                                          /* only accepts beacon, probe_rsp */
remote mDtimCount  Integer nodelay;       /* =0 at Tbtt of Beacon with DTIM */
remote mDtimPeriod Integer nodelay;       /* beacon periods per DTIM period */
remote mFxIP  Boolean nodelay;            /* =true during frame exchange seq */
remote mIbss  Boolean nodelay;            /* =true if STA is member of IBSS */
remote mListenInt  Integer nodelay;       /* beacons between wake up @TBTT */
remote mNavEnd  Time nodelay;             /* NAV end Time, <=now when idle */
remote mNextBdry  Time nodelay;           /* next boundary Time; =0 if none */
remote mNextTbtt  Time nodelay;           /* Time next beacon due to occur */
remote mOrates Ratestring nodelay;        /* operational rate set in use */
remote mPcAvail  Boolean nodelay;         /* =true if point coord in BSS */
remote mPcDlvr  Boolean nodelay;          /* =true if CF delivery only */
remote mPcPoll  Boolean nodelay;          /* =true if CF delivery & polling */
remote mPdly  Usec nodelay;               /* probe delay from start or join */
remote mPsm  PwrSave nodelay;             /* power save mode of STA */
remote mPss  PsState nodelay;             /* power save state of STA */
remote mRxA  Boolean nodelay;             /* =true if RX indicated by PHY */
remote mSsId  Octetstring nodelay;        /* name of the current (I)BSS */
remote procedure TSF nodelay;             /* read & update 64-bit TSF timer */
    fpar Integer, Boolean;   returns Integer;

/*******************************************************************
 *     Named static data values (names of form sXYZ)
 *******************************************************************/
synonym sMaxMsduLng Integer = 2304;       /* max octets in an MSDU */
synonym sMacHdrLng  Integer = 24;         /* octets in data header, no WEP */
synonym sWepHdrLng  Integer = 28;         /* octets in data header with WEP */
synonym sWepAddLng  Integer = 8;          /* octets added for WEP */
synonym sWdsAddLng  Integer = 6;          /* octets added for WDS (addr4) */
synonym sCrcLng  Integer = 4;             /* octets for crc32 (FCS, ICV) */
synonym sMaxMpduLng  Integer =            /* max octets in an MPDU */
      (sMaxMsduLng + sMacHdrLng + sWdsAddLng + sWepAddLng + sCrcLng);
syntype FrameIndexRange = Integer         /* index range for octets in MPDU */
      constants 0 : sMaxMpduLng   endsyntype FrameIndexRange;
synonym sTsOctet  Integer = 24;           /* first octet of Timestamp field */
synonym sMinFragLng Integer = 256;        /* min value for aMpduMaxLength */
synonym sMaxFragNum Integer =             /* maximum fragment number */
      (sMaxMsduLng / (sMinFragLng - sMacHdrLng - sCrcLng));
synonym sAckCtsLng Integer = 112;         /* bits in ACK and CTS frames */
```



```
/*******************************************************************
 *     Station configuration flags (static, supplementary to MIB)
 *******************************************************************/
synonym sVersion  Integer = 0;        /* supported Protocol Version */
synonym sCanBeAp  Boolean = false;    /* =true if STA can operate as AP */
synonym sCanBePc  Boolean = false;    /* =true if AP can be Point Coord */
synonym sCfPollable Boolean =true;    /* =true if responds to CF-polls */

/*******************************************************************
 *     Discrete microsecond and Time Unit 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, with the semantics
       /* of Natural and a step size (resolution) of 1 micosecond. */

       /* Most MAC times are defined using the subtypes of Integer
       /* Usec and TU.  These have operators for explicit conversion
       /* to SDL Time (tUsec, tTU), SDL Duration (dUsec, dTU), and
       /* from SDL Time (uTime, tuTime) as needed to comply with SDL's
       /* strong type checking.  Where the MAC state machines need to
       /* access the contents of the TSF timer, SDL's "now" (current
       /* time) is used.  This yields readable time-dependent code,
       /* but the value of "now" cannot be modified by an SDL program,
       /* so adopting the TSF time from timestamps in received Beacons
       /* or Probe Responses is shown as a call to remote procedure TSF. */

       /* Microsecond sort -- also has operators tmin and tmax */
newtype Usec   inherits Integer   operators all;
  adding  operators
    dUsec : Usec -> Duration;
    tUsec : Usec -> Time;
    uTime : Time -> Usec;
    tmax  : Usec, Usec -> Usec;
    tmin  : Usec, Usec -> Usec;
  axioms
    for all u, w in Usec(
      u >= w ==> tmax(u, w) == u;
      u < w ==> tmax(u, w) == w;
      u >= w ==> tmin(u, w) == w;
      u < w ==> tmin(u, w) == u;
      for all t in Time(     for all r in Real(
          r = float(u) ==> tUsec(u) == Time!(Duration!(r));
          t = Time!(Duration!(r)) and u = fix(r) ==> u == uTime(t);));
      for all d in Duration(     for all r in Real(
          r = float(u) ==> dUsec(u) == Duration!(r); )));
  constants >= 0 /* constrain value range to be non-negative */
endnewtype Usec;

       /* Time Unit sort -- (1 * TU) = (1024 * Usec) */
newtype TU  inherits Integer  operators all;
  adding  operators
    dTU    : TU -> Duration;
    tTU    : TU -> Time;
    tuTime : Time -> TU;
    u2TU   : Usec -> TU;
    tu2U   : TU -> Usec;
  axioms
    for all k in TU(     for all t in Time(     for all r in Real(
          r = float(k) ==> tTU(k) == Time!(Duration!(1024 * r));
          t = Time!(Duration!(r)) and k = (fix(r) / 1024) ==>
              k == tuTime(t);));
```

```
      for all d in Duration(     for all r in Real(
          r = float(k) ==> dTU(k) == Duration!(1024 * r);));
      for all u in Usec(
        u2TU(u) == u / 1024;        k2U(k) == k * 1024; ));
  constants >= 0 /* constrain value range to be non-negative */
endnewtype TU;

/*******************************************************************
 *     Generator for 0-origin String sorts (adapted from Z.105, Annex A)
 *******************************************************************/
       /* String0(sort, nullSymbol) can define strings of any sort.
       /* These strings are indexed starting from 0 rather than 1.
       /* Sorts defined by String0 have the normal String operators, plus
       /* Tail (all but first item), Head (all but last item), and
       /* aggregators S2, S3, S4, S6, S8 (make fixed length strings). */
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 last item in string */
    "//" : String0, String0 -> String0;           /* concatenation */
    Extract! : String0, Integer -> Item;          /* get item from string */
    Modify! : String0, Integer, Item -> String0;  /* modify string */
    SubStr : String0, Integer, Integer -> String0;
       /* SubStr(s,i,j) is string0 of length j starting at string0(i) */
    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, */
          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, */
          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, 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 (from Z.105, Annex A)
 ******************************************************************/
       /* Bit is a subtype of Boolean -- bit values 0 and 1 are
       /* not numerals and cannot be used with Integer operators */
newtype Bit   inherits Boolean
    literals  0 = false,  1 = true;   operators all;   endnewtype Bit;

/******************************************************************
 *     ASN.1-style BIT STRING sort (adapted from Z.105, Annex A)
 ******************************************************************/
       /* Bitstrings are 0-origin strings of Bit.  Z.105 uses ASN.1-style
       /* literals in binary ('1011'B) or hexadecimal ('D3'H), but this
       /* syntax is not accepted for Z.100 string literals.  Therefore,
       /* this version provides only hexadecimal literals 0x00-0xFF.
       /* Bitstring operators '=>', 'not', 'and', 'or', and 'xor' act
       /* bitwise, with the length of the result string equal to the
       /* length of the longest (or only) source string. */
newtype Bitstring String0(Bit, '')
  adding  literals     macro Hex_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;
    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 sort           (influenced by Z.105, Annex A)
 *******************************************************************/
       /* Octet is a subtype of Bitstring where length always =8.
       /* Z.105 adds a "size "keyword to SDL and defines Octet with
       /* "... constants size (8) ... " to impose this length constraint.
       /* Here Octet relies on proper use maintain lengths as multiples
       /* of 8.  Proper length strings are created by the hexadecimal
       /* Bitstring literals (e.g. 0xD5) and operator mkOctet:
       /*     o:= mkOctet(i)       converts a non-negative Integer (mod 256)
       /*                          to an Octet (exactly 8 bits)
       /*     i:= octetVal(o)      converts an Octet to an Integer (0:255)
       /*     o:= flip(o)          reverses bit order of 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(1) ==> 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;

/*******************************************************************
 *     OCTET STRING sort    (somewhat influenced by Z.105, Annex A)
 *******************************************************************/
       /* Octetstrings are 0-ORIGIN strings of Octet, NOT 1-ORIGIN
       /* strings like Octet_String in Z.105 (hence the name change).
       /* Octetstring has conversion operators to and from Bitstring,
       /* and integer to Octetstring.  Octetstring literals are "null"
       /* and 1-4, 6, 8 item 0x00 strings O1, O2, O3, O4, O6, O8. */
newtype Octetstring String0(Octet, null)
  adding  literals O1, O2, O3, O4, O6, O8;
  operators
    B_S  : Octetstring -> Bitstring;              /* name changed from Z.105 */
    O_S  : Bitstring -> Octetstring;              /* name changed from Z.105 */
    mkOS : Integer,Integer -> Octetstring;        /* mkOS(i1,i2) returns */
             /* mkstring(mkOctet(i1)) padded (0x00) to length i2 */
    mk2octets : Integer -> Octetstring; /* 16-bit integer to 2-octets */
  axioms
    for all b, b1, b2 in Bitstring(
      for all s in Octetstring(     for all o in Octet(
          B_S(null) == '';
```

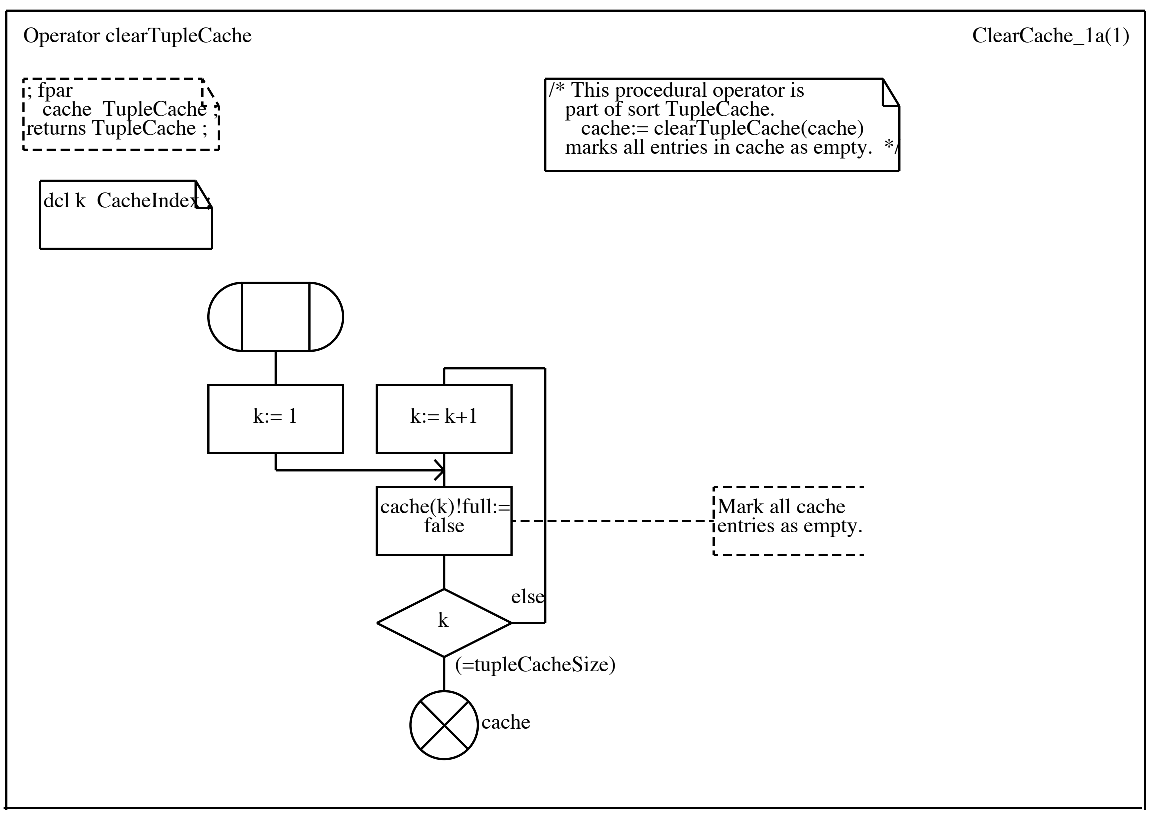


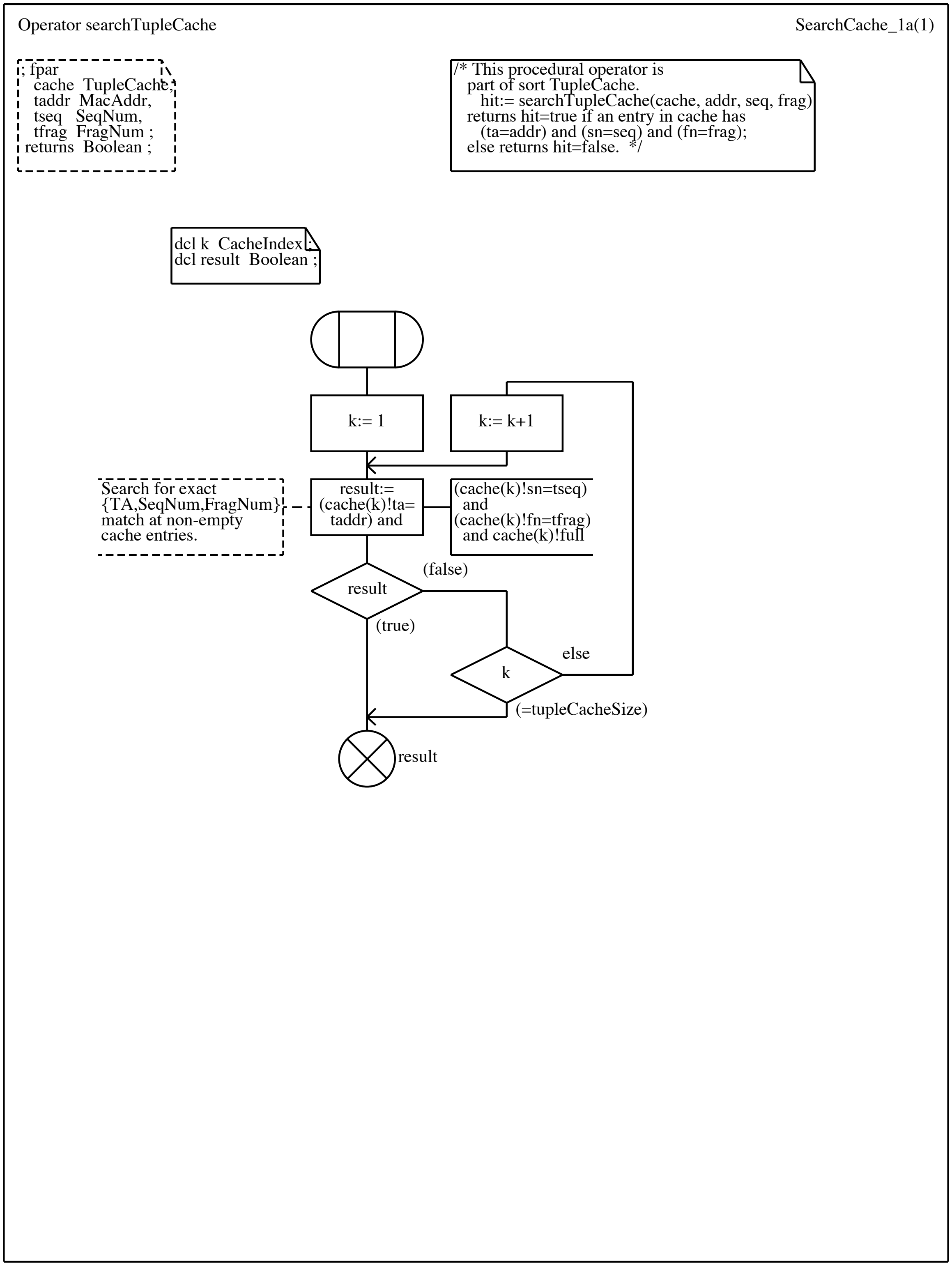
```
          B_S(MkString(o) // s) == o // B_S(s);
          O_S('') == null;
          Length(b1) > 0, Length(b1) < 8 ==>
            O_S(b1) == MkString(b1 or 0x00);  /* expand b1 to 8 bits */
          b == b1 // b2, Length(b1) = 8 ==>
            O_S(b) == MkString(b1) // O_S(b2);
          for 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!;
            mk2octets(i) == MkString(mkOctet(i mod 256)) //
              MkString(mkOctet(i / 256)); );
          O1 == MkString(0x00);     O2 == O1 // O1;
          O3 == O2 // O1;           O4 == O2 // O2;
          O6 == O4 // O2;           O8 == O4 // O4; )));
  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) =true if given a group address
       /*     isBcst(m)  =true if given the broadcast address
       /*     isLocal(m) =true if given a locally-administered address
       /*     adrOs(m)   converts MacAddr to Octetstring
       /* MAC addresses must be defined to be exactly 6 octets long,
       /* typically using the S6 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;

newtype MacAddrSet   powerset( MacAddr)   endnewtype MacAddrSet;

synonym bcstAddr  MacAddr =         /* Broadcast Address */
              <<type MacAddr>>  S6(0xFF,0xFF,0xFF,0xFF,0xFF,0xFF);

synonym nullAddr  MacAddr =         /* Null Address */
              << type MacAddr>>  S6(0x00,0x00,0x00,0x00,0x00,0x00);
```

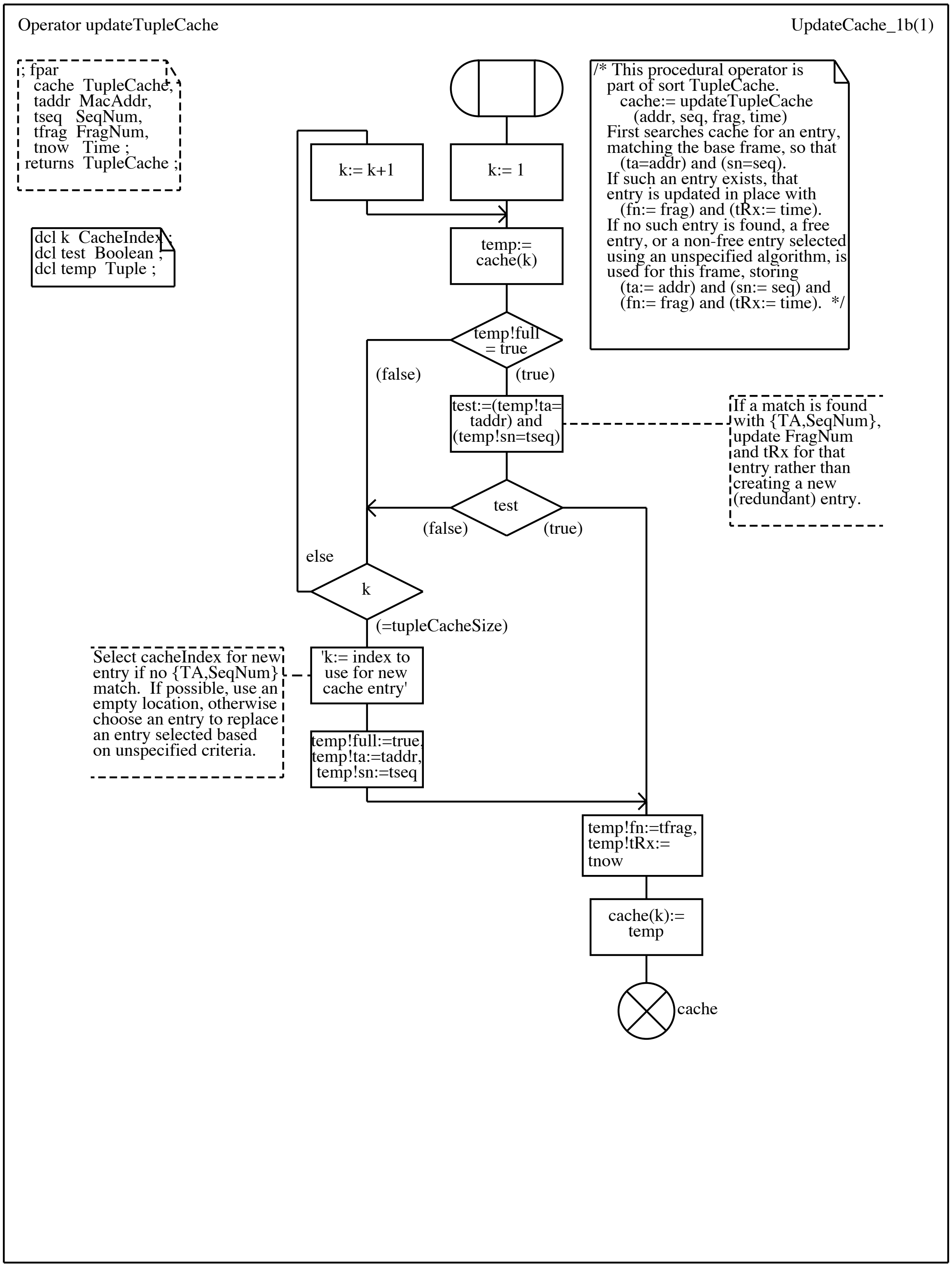
```
/*******************************************************************
 *     BSS description sorts
 *******************************************************************/
       /* BssDscr is used with MlmeScan.confirm and MlmeJoin.request */
newtype BssDscr  struct
    bdBssId        MacAddr;
    bdSsId         Octetstring;     /* 1 <= length <= 32 */
    bdType         BssType;
    bdBcnPer       TU;              /* beacon period in Time Units */
    bdDtimPer      Integer;         /* DTIM period in beacon periods */
    bdTstamp       Octetstring;     /* 8 Octets from ProbeRsp/Beacon */
    bdStartTs      Octetstring;     /* 8 Octets TSF when rx Tstamp */
    bdPhyParms     PhyParms;        /* empty if not needed by PHY */
    bdCfParms      CfParms;         /* empty if not CfPollable/no PCF */
    bdIbssParms    IbssParms;       /* empty if infrastructure BSS */
    bdCap          Capability;      /* capability information */
    bdBrates       Ratestring;      /* BSS basic rate set */
endnewtype BssDscr;

newtype BssDscrSet   powerset( BssDscr)   endnewtype BssDscrSet;

/*******************************************************************
 *     Duplicate filtering support sorts
 *******************************************************************/
syntype FragNum = Integer     /* Range of possible fragment numbers */
      constants 0:sMaxFragNum  endsyntype FragNum;
syntype SeqNum = Integer      /* Range of possible sequence numbers */
      constants 0:4095  endsyntype SeqNum;
newtype Tuple  struct /* for duplicate filtering & defragmentation */
    full      Boolean;        /* =true if Tuple contains valid info */
    ta        MacAddr;        /* transmitting station address (Addr2) */
    sn        SeqNum;         /* Msdu/Mmpdu sequence number */
    fn        FragNum;        /* most recent Mpdu fragment number */
    tRx       Time;           /* reception time (endRx of fragment) */
  default (. false, nullAddr, 0, 0, 0 .);
endnewtype Tuple;

/*******************************************************************
 *     TupleCache support sorts
 *******************************************************************/
       /* Number of TupleCache entries and associated index range */
synonym tupleCacheSize Integer = 32;         /* this value is an example,
                             TupleCache size is implementation dependent */
syntype CacheIndex = Integer  constants 1:tupleCacheSize
  endsyntype CacheIndex;

       /* TupleCache array
       /*     cache:= ClearTupleCache(cache)  to initialize cache
       /*     cache:= UpdateTupleCache(cache, addr, seq, frag, endRx)
       /*             if <addr,seq> is already cached, updates frag
       /*             if <addr,seq> not cached, fills an empty entry
       /*               or replaces an entry using an unspecified algorithm
       /*     SearchTupleCache(cache, addr, seq, frag)
       /*             returns true if specified <addr,seq,frag> in cache */
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 Counter sort and Integer string sort
 ******************************************************************/
       /* This sort used for MIB counters, needed because SDL Integers
       /* have no specified maximum value.  inc(counter) increments the
       /* counter 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 (1-origin) of Integer */
newtype Intstring   String( Integer, noInt);   endnewtype Intstring;
```



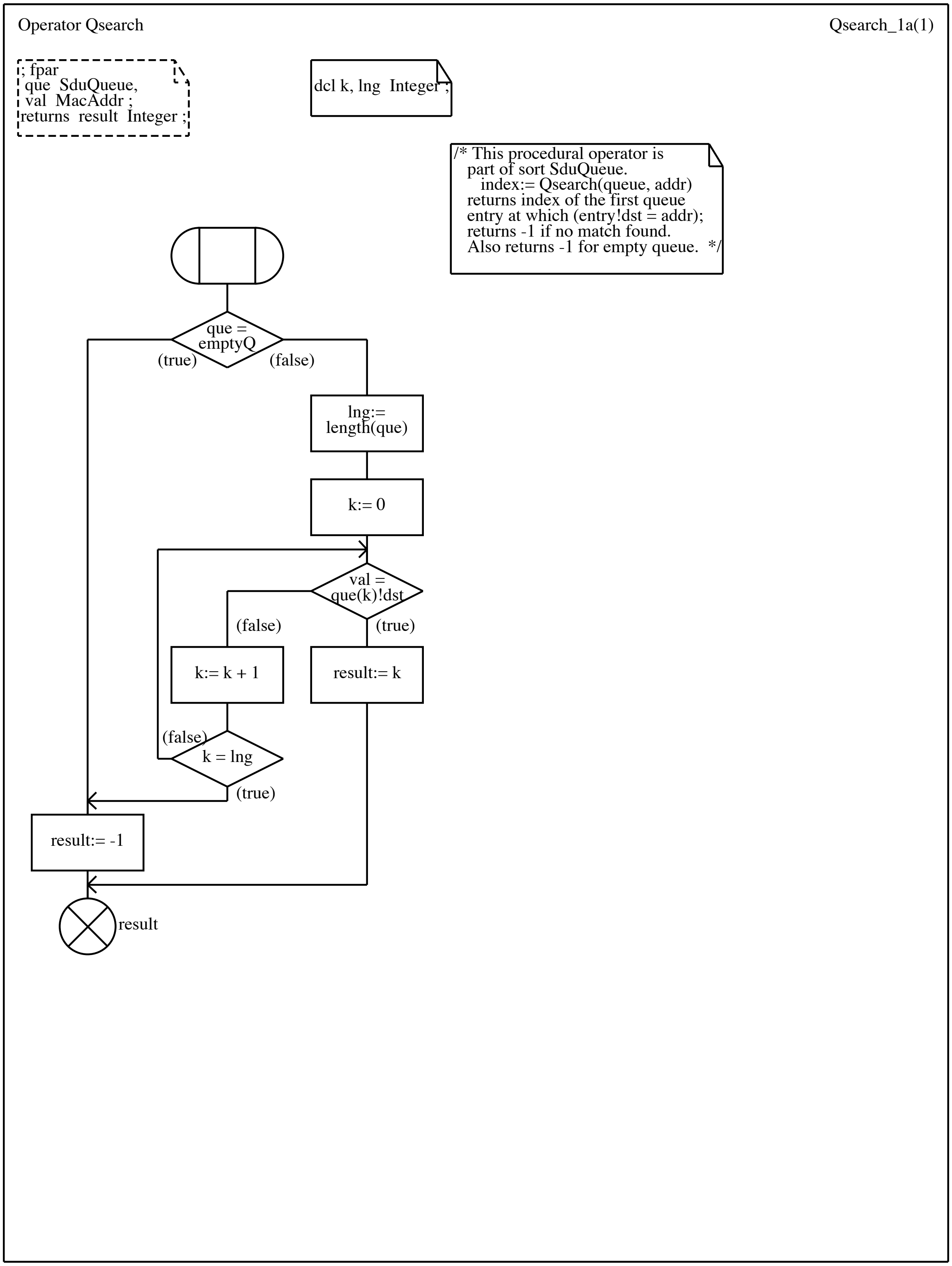

Operator searchTupleCache
SearchCache_1a(1)
; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum ;
returns Boolean ;
/* This procedural operator is
part of sort TupleCache.
hit:= searchTupleCache(cache, addr, seq, frag)
returns hit=true if an entry in cache has
(ta=addr) and (sn=seq) and (fn=frag);
else returns hit=false. */
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


Operator updateTupleCache
UpdateCache_1b(1)
; fpar
cache TupleCache,
taddr MacAddr,
tseq SeqNum,
tfrag FragNum,
tnow Time ;
returns TupleCache ;
dcl k CacheIndex ;
dcl test Boolean ;
dcl temp Tuple ;
/* This procedural operator is part of sort TupleCache.
cache:= updateTupleCache (addr, seq, frag, time)
First searches cache for an entry, matching the base frame, so that (ta=addr) and (sn=seq).
If such an entry exists, that entry is updated in place with (fn:= frag) and (tRx:= time).
If no such entry is found, a free entry, or a non-free entry selected using an unspecified algorithm, is used for this frame, storing (ta:= addr) and (sn:= seq) and (fn:= frag) and (tRx:= time). */
k:= k+1
k:= 1
temp:= cache(k)
temp!full = true
(false)
(true)
test:=(temp!ta= taddr) and (temp!sn=tseq)
If a match is found with {TA,SeqNum}, update FragNum and tRx for that entry rather than creating a new (redundant) entry.
test
(false)
(true)
else
k
(=tupleCacheSize)
Select cacheIndex for new entry if no {TA,SeqNum} match. If possible, use an empty location, otherwise choose an entry to replace an entry selected based on unspecified criteria.
'k:= index to use for new cache entry'
temp!full:=true, temp!ta:=taddr, temp!sn:=tseq
temp!fn:=tfrag, temp!tRx:= tnow
cache(k):= temp
cache

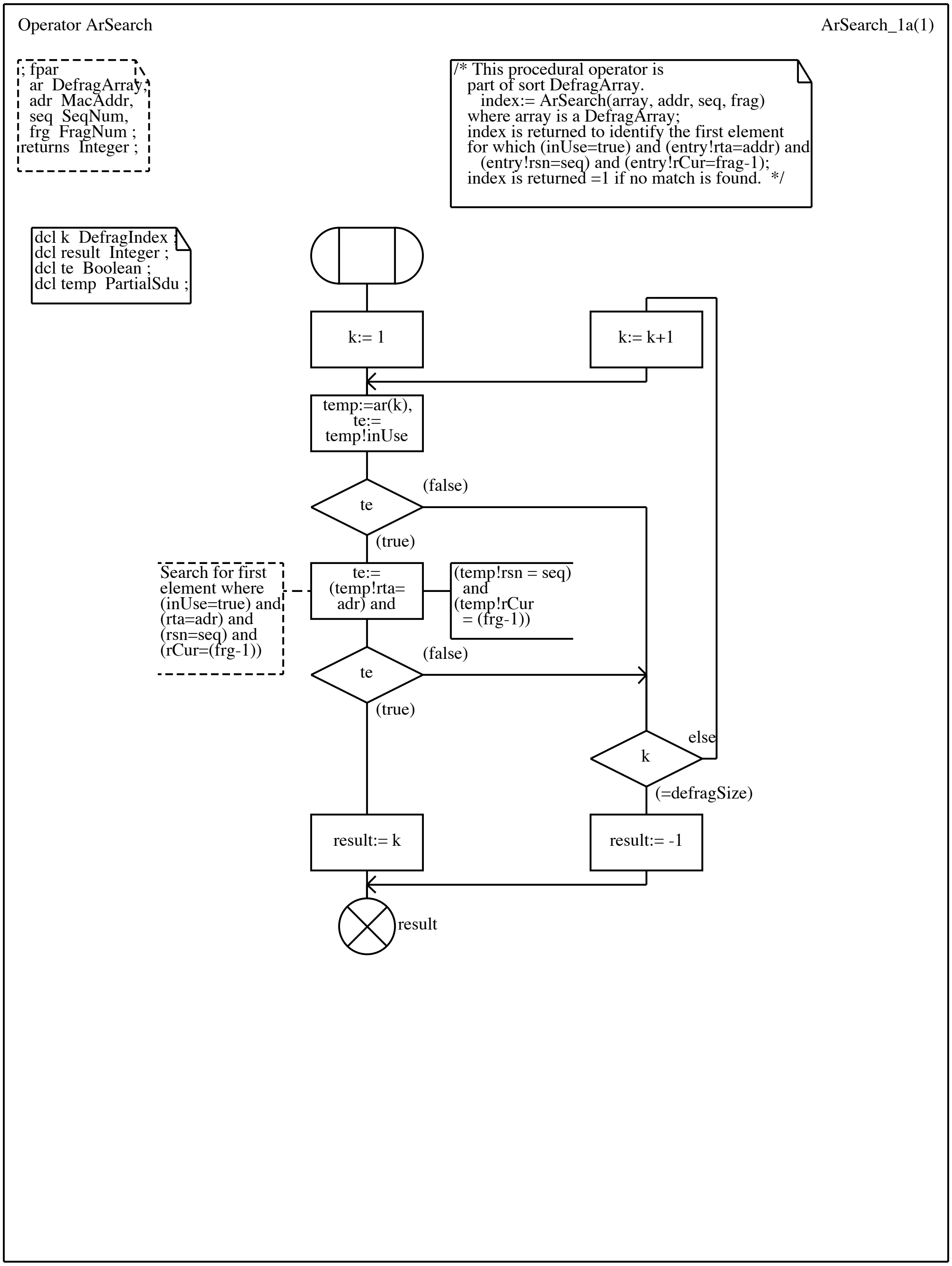

```
/******************************************************************
 *     Generator for Queue sorts
 ******************************************************************/
       /* The Queue generator is derived from the String0 generator
       /* to create Queues of any sort.  Queues operators are:
       /*    Qfirst(queue,item)  adds item as the first queue element
       /*    Qlast(queue,item)   adds item as the last queue element
       /* and the String0 operators Length, //, First, Last, Head, Tail
       /* Because operators can only return a single value, removing an
       /* element from a queue is a 2-step process:
       /*    dequeue first:   item:=First(queue);   queue:=Tail(queue);
       /*    dequeue last:    item:=Last(queue);    queue:=Head(queue); */
generator Queue(type Item, literal Emptyqueue)
  literals Emptyqueue;
  operators
    MkQ       : Item -> Queue;              /* make a queue from an item */
    Lengt     : Queue -> Integer;           /* number of items on queue */
    First     : Queue -> Item;              /* first item on queue */
    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 last item on queue */
    "//"      : Queue,Queue -> Queue;       /* concatenation */
    Extract!  : Queue,Integer -> Item;      /* copy item from queue */
    Modify!   : Queue,Integer,Item -> Queue; /* modify item in queue */
    SubQ      : Queue,Integer,Integer -> Queue;
       /* SubQ(q,i,j) queue of length j starting from queue(i) */
  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, */
        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, */
          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!;
      /* define Modify!, Head, Tail, Qfirst, Qlast by other ops */
          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;
```

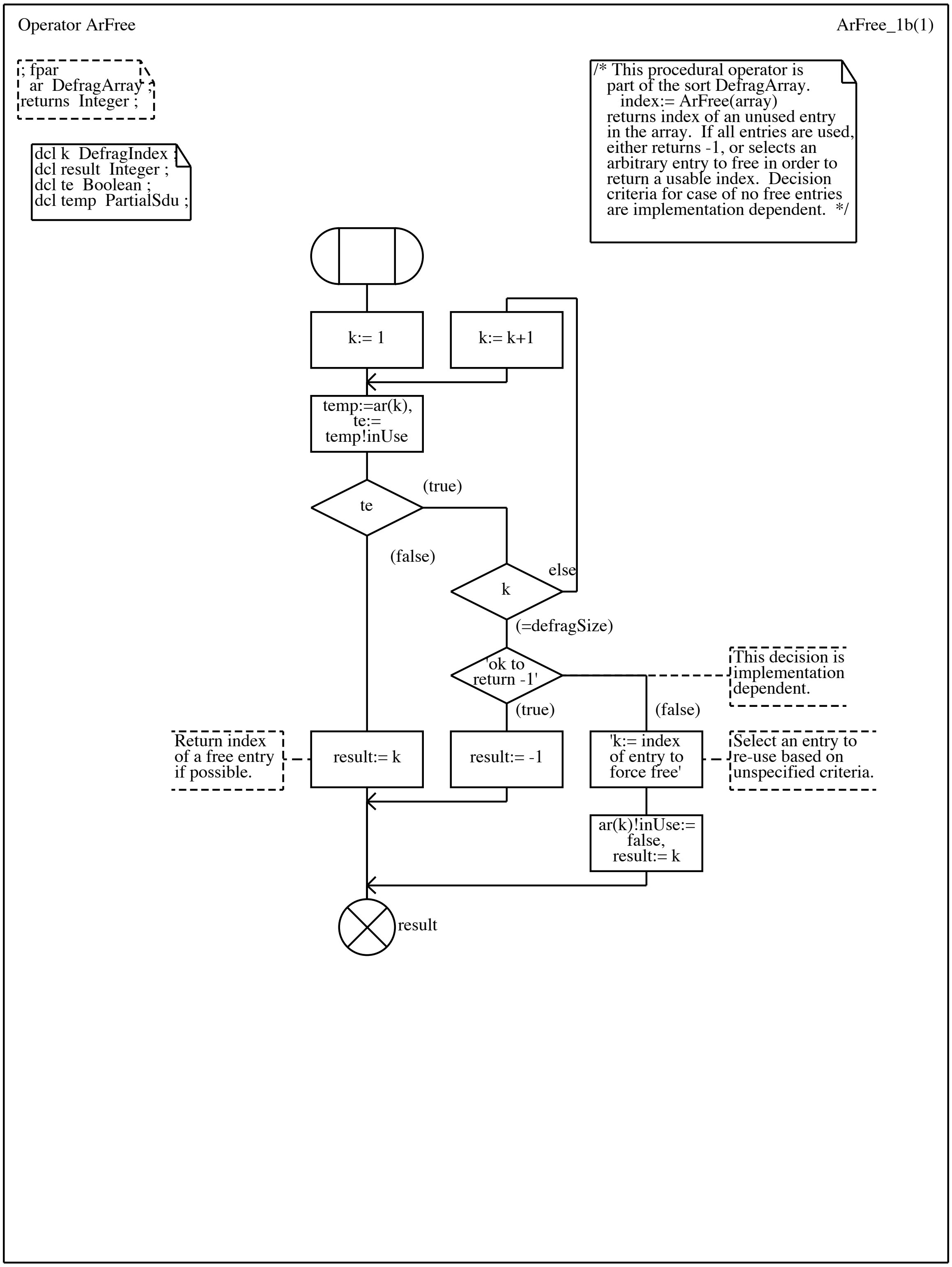


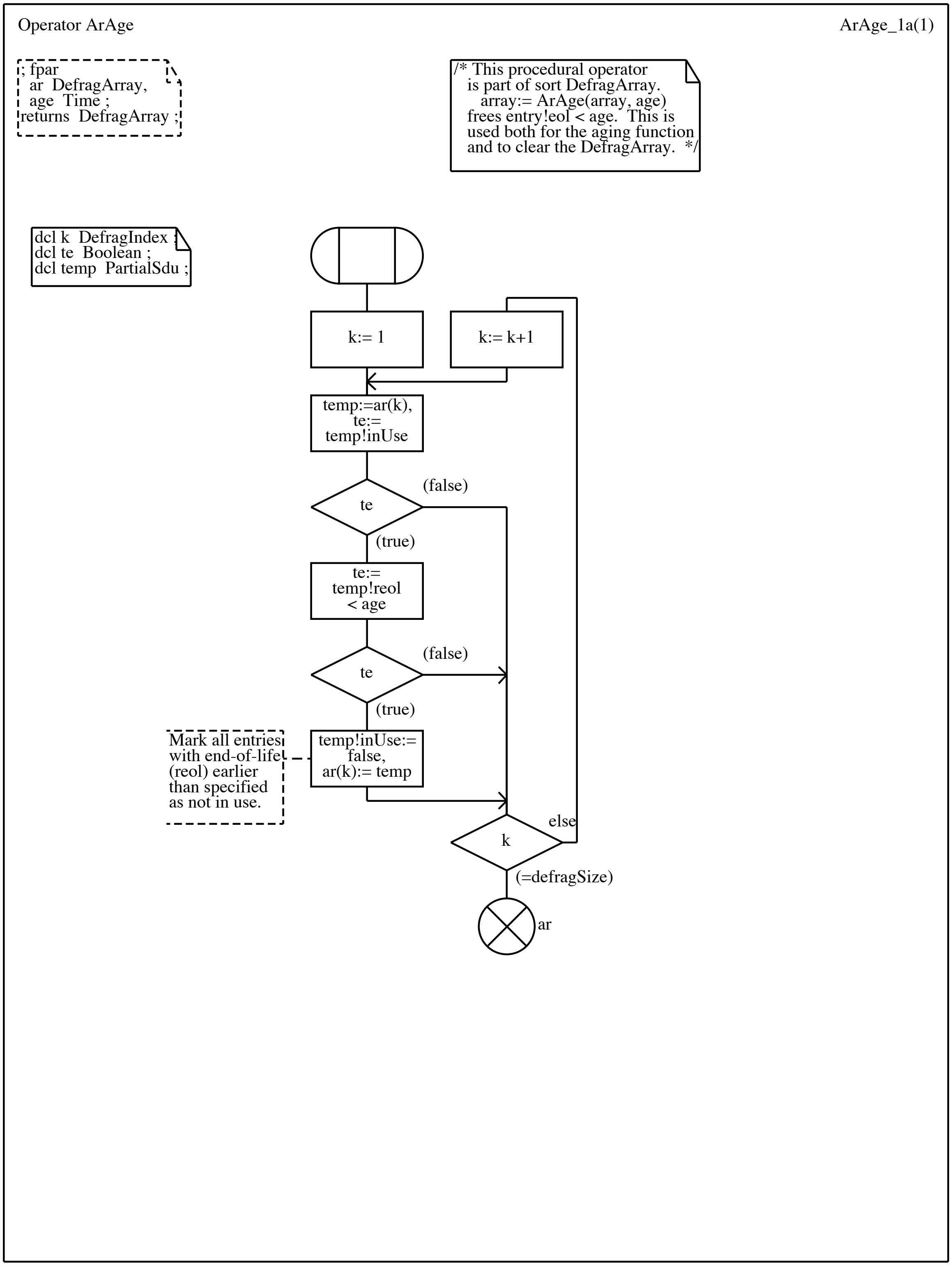
```
/******************************************************************
 *     Fragmentation support sorts
 ******************************************************************/
       /* Array to hold up to FragNum fragments of an Msdu/Mmpdu */
newtype FragArray  Array(FragNum, Frame);   endnewtype FragArray;

       /* FragSdu structure is for OUTGOING MSDUs/MMPDUs (called 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. */
newtype FragSdu struct
    fTot      FragNum;        /* number of fragments in pdus FragArray */
    fCur      FragNum;        /* next fragment number to announce send */
    fAnc      FragNum;        /* next fragment to announce in ATIM or TIM
                                 when fAnc > fCur, pdus(fCur)+ may be sent */
    eol       Time;           /* set to (now + dUsec(aMaxTxMsduLifetime))
                                 when the entry is created */
    sqf       SeqNum;         /* SDU sequence number, set at 1st Tx attempt */
    src       Integer;        /* short retry counter for this SDU */
    lrc       Integer;        /* long retry counter for this SDU */
    dst       MacAddr;        /* destinaton address */
    grpa      Boolean;        /* =true if RA (not DA) is a group address */
    psm       Boolean;        /* =true if RA (not DA) may be in pwr_save */
    resume    Boolean;        /* =true if fragment burst being resumed */
    cnfTo     PId;            /* address to which confirmation is sent */
    txrate    Rate;           /* data rate used for initial fragment */
    cf        CfPriority;     /* requested priority (from LLC) */
    pdus      FragArray;      /* array of Frame to hold fragments */
endnewtype FragSdu;

       /* Queue of FragSdu
       /* for power save buffers, etc., searchable with Qsearch operator:
       /*     index:= Qsearch(queue, addr)     where queue is an SduQueue,
       /* index identifies the first queue entry at which
       /* entry!dst = addr; or as -1 if no match (or queue empty). */
newtype SduQueue Queue(FragSdu, emptyQ);
  adding  operators
    qSearch : SduQueue, MacAddr -> Integer;
  operator qSearch;
    fpar que SduQueue,  val MacAddr;  returns Integer; referenced;
endnewtype SduQueue;
```


Operator Qsearch
Qsearch_1a(1)
; fpar
que SduQueue,
val MacAddr ;
returns result Integer ;
dcl k, lng Integer ;
/* This procedural operator is
part of sort SduQueue.
index:= Qsearch(queue, addr)
returns index of the first queue
entry at which (entry!dst = addr);
returns -1 if no match found.
Also returns -1 for empty queue. */
que =
emptyQ
(true)
(false)
lng:=
length(que)
k:= 0
val =
que(k)!dst
(false)
(true)
k:= k + 1
result:= k
(false)
k = lng
(true)
result:= -1
result

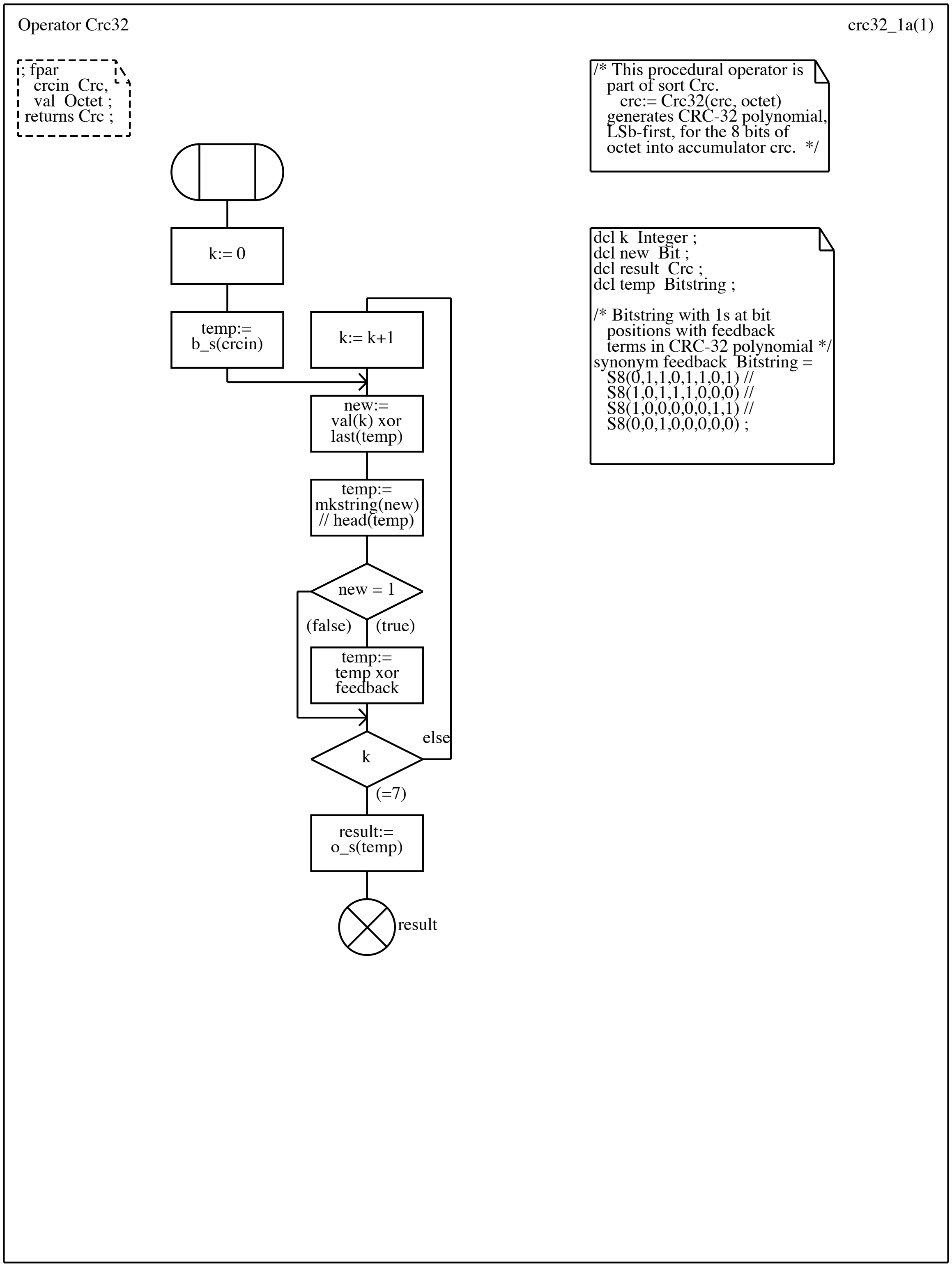

```
/******************************************************************
 *     Defragmentation support sorts
 ******************************************************************/

       /* The PartialSdu structure is for INCOMPLETE MSDUs/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 this instance holds any fragments */
    rta       MacAddr;      /* transmitting station (Addr2) */
    rsn       SeqNum;       /* SDU sequence number */
    rCur      FragNum;      /* fragment number of most recent Mpdu */
    reol      Time;         /* (now+dUsec(aMaxReceiveLifetime) @ 1st Mpdu */
    rsdu      Frame;        /* buffer where Mpdus are concatenated */
  default (. false, nullAddr, 0, 0, 0, null .);
endnewtype PartialSdu;

newtype PartialSduKeys struct        /* if aPrivacyOptionImplemented=true *
    wDefKeys       KeyVector;        /* default keys when 1st frag received */
    wKeyMap        KeyMapArray;      /* key mappings when 1st frag received */
    wExclude       Boolean;          /* aExcludeUnencrypted @ 1st frag rx */
endnewtype PartialSduKeys;


       /* Number of entries in defragmentation array at this station.
       /* The value is implementation dependent (min=3, see 9.5). */
synonym defragSize Integer = 6;
syntype defragIndex = Integer   constants 1:defragSize
endsyntype defragIndex;


       /* Array of PartialSdu for use defragmenting Msdus and Mmpdus.
       /* Searchable using the ArSearch operator
       /*     index:= ArSearch(array, addr, seq, frag)
       /* where 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 the index of a free entry,
       /* or -1 if no entries free.  May free an entry, selected using
       /* an unspecified algorithm, to avoid returning -1.
       /*     array:= ArAge(array, age)
       /* frees where (entry!eol < age), also used to clear array. */
newtype DefragArray  Array( defragIndex, PartialSdu);
  adding  operators
    ArSearch : DefragArray, MacAddr, SeqNum, FragNum -> Integer;
    ArFree   : DefragArray -> Integer;
    ArAge    : DefragArray, Time -> DefragArray;
  operator ArSearch;
    fpar  ar DefragArray,   adr MacAddr,   seq SeqNum,   frg FragNum;
    returns Integer; referenced;
  operator ArFree;  fpar  ar DefragArray;   returns Integer; referenced;
  operator ArAge;  fpar  ar DefragArray,   age Time;
    returns DefragArray; referenced;
endnewtype DefragArray;

newtype DefragKeysArray  Array( defragIndex, PartialSduKeys);
endnewtype DefragKeysArray;
```

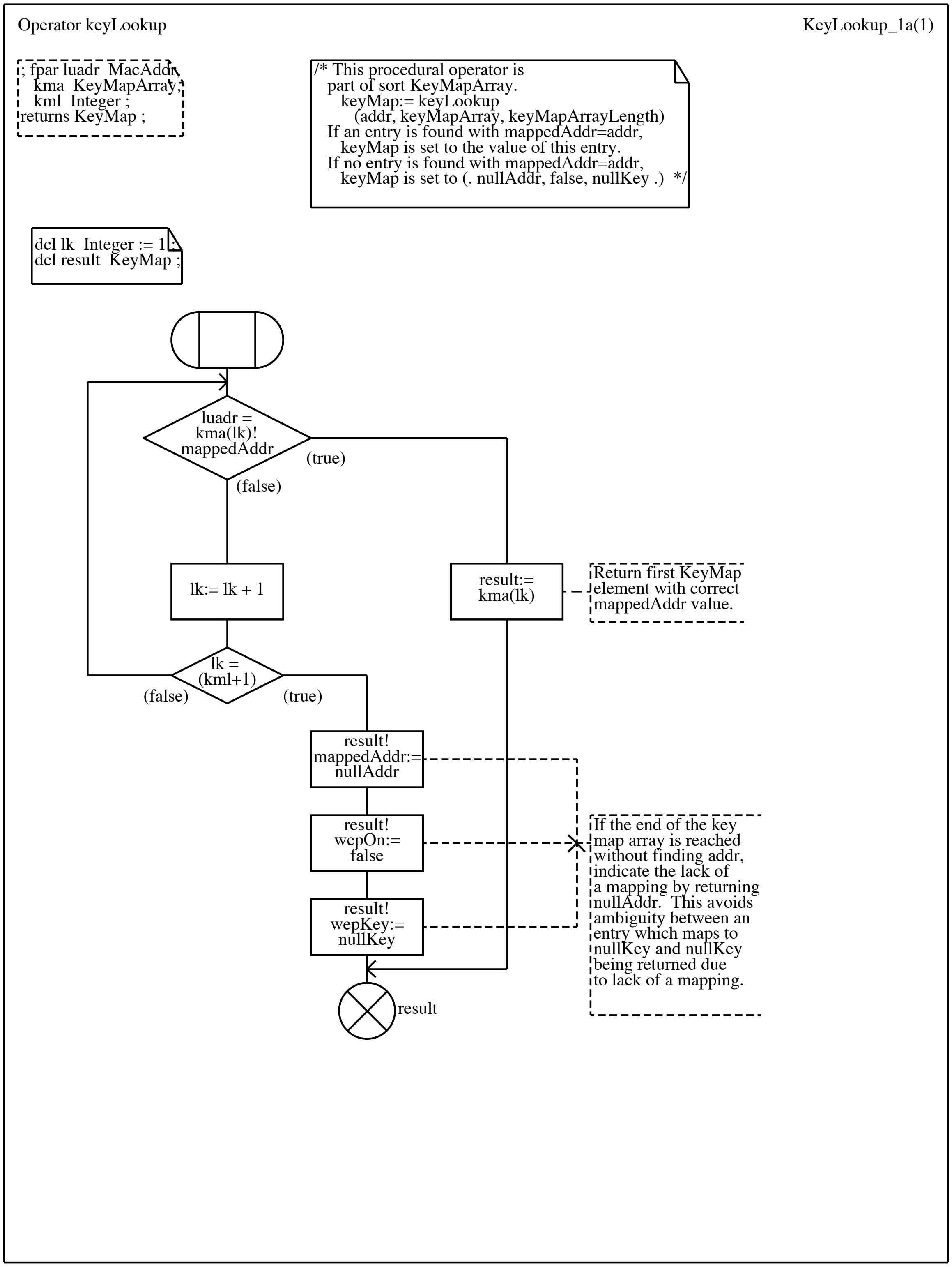

Operator ArSearch
ArSearch_1a(1)
; fpar
ar DefragArray,
adr MacAddr,
seq SeqNum,
frg FragNum ;
returns Integer ;
/* This procedural operator is
part of sort DefragArray.
index:= ArSearch(array, addr, seq, frag)
where array is a DefragArray;
index is returned to identify the first element
for which (inUse=true) and (entry!rta=addr) and
(entry!rsn=seq) and (entry!rCur=frag-1);
index is returned =1 if no match is found. */
dcl k DefragIndex ;
dcl result Integer ;
dcl te Boolean ;
dcl temp PartialSdu ;
k:= 1
k:= k+1
temp:=ar(k),
te:=
temp!inUse
te
(false)
(true)
Search for first
element where
(inUse=true) and
(rta=adr) and
(rsn=seq) and
(rCur=(frg-1))
te:=
(temp!rta=
adr) and
(temp!rsn = seq)
and
(temp!rCur
= (frg-1))
te
(false)
(true)
k
else
(=defragSize)
result:= k
result:= -1
result


Operator ArFree
ArFree_1b(1)
; fpar
ar DefragArray ;
returns Integer ;
/* This procedural operator is
part of the sort DefragArray.
index:= ArFree(array)
returns index of an unused entry
in the array. If all entries are used,
either returns -1, or selects an
arbitrary entry to free in order to
return a usable index. Decision
criteria for case of no free entries
are implementation dependent. */
dcl k DefragIndex ;
dcl result Integer ;
dcl te Boolean ;
dcl temp PartialSdu ;
k:= 1
k:= k+1
temp:=ar(k),
te:=
temp!inUse
te
(true)
(false)
k
else
(=defragSize)
'ok to
return -1'
This decision is
implementation
dependent.
(true)
(false)
Return index
of a free entry
if possible.
result:= k
result:= -1
'k:= index
of entry to
force free'
Select an entry to
re-use based on
unspecified criteria.
ar(k)!inUse:=
false,
result:= k
result


Operator ArAge
ArAge_1a(1)
; fpar
ar DefragArray,
age Time ;
returns DefragArray ;
/* This procedural operator
is part of sort DefragArray.
array:= ArAge(array, age)
frees entry!eol < age. This is
used both for the aging function
and to clear the DefragArray. */
dcl k DefragIndex ;
dcl te Boolean ;
dcl temp PartialSdu ;
k:= 1
k:= k+1
temp:=ar(k),
te:=
temp!inUse
te
(false)
(true)
te:=
temp!reol
< age
te
(false)
(true)
Mark all entries
with end-of-life
(reol) earlier
than specified
as not in use.
temp!inUse:=
false,
ar(k):= temp
else
k
(=defragSize)
ar


```
/*******************************************************************
 *     CRC-32 sorts (for FCS and ICV)
 *******************************************************************/

       /* Crc is a subtype of Octetstring with added operators:
       /*     crc:= Crc32(crc,octet)
       /* updates the crc value to include 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 for
       /* MSb-first transmission (see 7.1.1).  Crc variables must have
       /* exactly 4 octets, which is done using initCrc or S4. */
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;

synonym initCrc Crc =    /* Initial Crc value (all 1s) */
       << type Crc>> S4(0xFF,0xFF,0xFF,0xFF);
synonym goodCrc Crc =    /* Unique remainder for valid CRC-32 */
       << type Crc>> S4(0x7B,0xDD,0x04,0xC7);


/*******************************************************************
 *     WEP support sorts
 *******************************************************************/

syntype KeyIndex = Integer   constants 0:3   endsyntype KeyIndex;
newtype PrngKey inherits Octetstring   operators all;
  adding literals nullKey;  /* nullKey is not any of 2^40 key values */
  axioms nullKey == null;   default nullKey;   endnewtype PrngKey;
newtype KeyVector       /* vector of default WEP keys */
  Array( KeyIndex, PrngKey);   endnewtype KeyVector;

       /* Number of entries in aWepKeyMappings array at this station.
       /* implementation dependent value, minimum=10 (see 8.3.2). */
synonym sWepKeyMappingLength Integer = 10;
syntype KeyMappingRange = Integer
       constants 1:sWepKeyMappingLength   endsyntype KeyMappingRange;

newtype KeyMap  struct  /* structure used for entries in KeyMapArray */
    mappedAddr    MacAddr;
    wepOn         Boolean;
    wepKey        PrngKey;
endnewtype KeyMap;

       /* KeyMapArray -- used for aWepKeyMapping table;
       /* an array of KeyMap indexed by KeyMappingRange, with operator
       /*     KeyMap := keyLookup(addr, keyMapArray, keyMapArrayLength)
       /* returns the KeyMap entry for the specified addr, or
       /* (. nullAddr, false, nullKey .) if no mapping for addr. */
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;
```


Operator Crc32
crc32_1a(1)
; fpar
crcin Crc,
val Octet ;
returns Crc ;
/* This procedural operator is
part of sort Crc.
crc:= Crc32(crc, octet)
generates CRC-32 polynomial,
LSb-first, for the 8 bits of
octet into accumulator crc. */
dcl k Integer ;
dcl new Bit ;
dcl result Crc ;
dcl temp Bitstring ;
/* Bitstring with 1s at bit
positions with feedback
terms in CRC-32 polynomial */
synonym feedback Bitstring =
S8(0,1,1,0,1,1,0,1) //
S8(1,0,1,1,1,0,0,0) //
S8(1,0,0,0,0,0,1,1) //
S8(0,0,1,0,0,0,0,0) ;
k:= 0
temp:=
b_s(crcin)
k:= k+1
new:=
val(k) xor
last(temp)
temp:=
mkstring(new)
// head(temp)
new = 1
(false)
(true)
temp:=
temp xor
feedback
else
k
(=7)
result:=
o_s(temp)
result


Operator keyLookup
KeyLookup_1a(1)
; fpar luadr MacAddr,
kma KeyMapArray,
kml Integer ;
returns KeyMap ;
/* This procedural operator is
part of sort KeyMapArray.
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 (. nullAddr, false, nullKey .) */
dcl lk Integer := 1;
dcl result KeyMap ;
luadr =
kma(lk)!
mappedAddr
(true)
(false)
lk:= lk + 1
result:=
kma(lk)
Return first KeyMap
element with correct
mappedAddr value.
lk =
(kml+1)
(false)
(true)
result!
mappedAddr:=
nullAddr
result!
wepOn:=
false
result!
wepKey:=
nullKey
If the end of the key
map array is reached
without finding addr,
indicate the lack of
a mapping by returning
nullAddr. This avoids
ambiguity between an
entry which maps to
nullKey and nullKey
being returned due
to lack of a mapping.
result

```
/******************************************************************
 *     FRAME sort (the basic definition of fields in MAC frames)
 ******************************************************************/

      /* Frame is a subtype of Octetstring with operators for creating
      /* MAC headers, extracting each of the header fields and some
      /* management frame fields, and modifying most of these fields.
      /* There are operators to create and extract management frame
      /* elements, but no operators for the frame body, IV, ICV, and FCS
      /* fields, which are handled directly as Octetstrings. */

newtype Frame   inherits Octetstring   operators all;
  adding  operators
    mkFrame          : TypeSubtype, MacAddr, MacAddr, Octetstring -> Frame;
    mkCtl            : TypeSubtype, Octetstring, MacAddr -> Frame;
    protocolVer      : Frame -> Integer;   /* Protocol version (2 bits) */
    basetype         : Frame -> BasicType; /* Type field (2 bits) */
    ftype            : Frame -> TypeSubtype; /* Type & Subtype (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;   /* {strictly}Order{ed} (1 bit) */
    setOrderBit      : Frame, Bit -> Frame;
    durId            : Frame -> Integer; /* Duration/ID field (2) */
    setDurId         : Frame, Integer -> Frame;
    addr1            : Frame -> MacAddr; /* Address 1 [DA/RA] field (6) */
    setAddr1         : Frame, MacAddr -> Frame;
    addr2            : Frame -> MacAddr; /* Address 2 [SA/TA] field (6) */
    setAddr2         : Frame, MacAddr -> Frame;
    addr3            : Frame -> MacAddr; /* Address 3 [Bss/DA/SA] field */
    setAddr3         : Frame, MacAddr -> Frame;
    addr4            : Frame -> MacAddr; /* Address 4 [WDS-SA] field (6) */
    insAddr4         : Frame, MacAddr -> Frame;
    seq              : Frame -> SeqNum; /* Sequence Number (12 bits) */
    setSeq           : Frame, SeqNum -> Frame;
    frag             : Frame -> FragNum; /* Fragment Number (4 bits) */
    setFrag          : Frame, FragNum -> Frame;
    ts               : Frame -> Time; /* Timestamp field (8) */
    setTs            : Frame, Time -> Frame;
    mkElem           : ElementID, Octetstring -> Frame; /* make element */
    GetElem          : Frame, ElementID -> Frame; /* get element if aval */
    status           : Frame -> StatusCode; /* Status Code field (2) */
    setStatus        : Frame, StatusCode -> Frame;
    authStat         : Frame -> StatusCode; /* Status Code in Auth frame */
    reason           : Frame -> ReasonCode; /* Reason Code field (2) */
    authSeqNum       : Frame -> Integer; /* Auth Sequence Number (2) */
    authAlg          : Frame -> AuthType; /* Auth Algorithm field (2) */
    beaconInt        : Frame -> TU;    /* Beacon Interval field (2) */
    listenInt        : Frame -> TU;    /* Listen Interval field (2) */
    AId              : Frame -> AsocId; /* Association ID field (2) */
    setAId           : Frame, AsocId -> Frame;
    curApAddr        : Frame -> MacAddr; /* Current AP Addr field (6) */
    capA             : Frame, Capability -> Bit; /* Capability (Re)Asoc */
    setCapA          : Frame, Capability, Bit -> Frame;
```

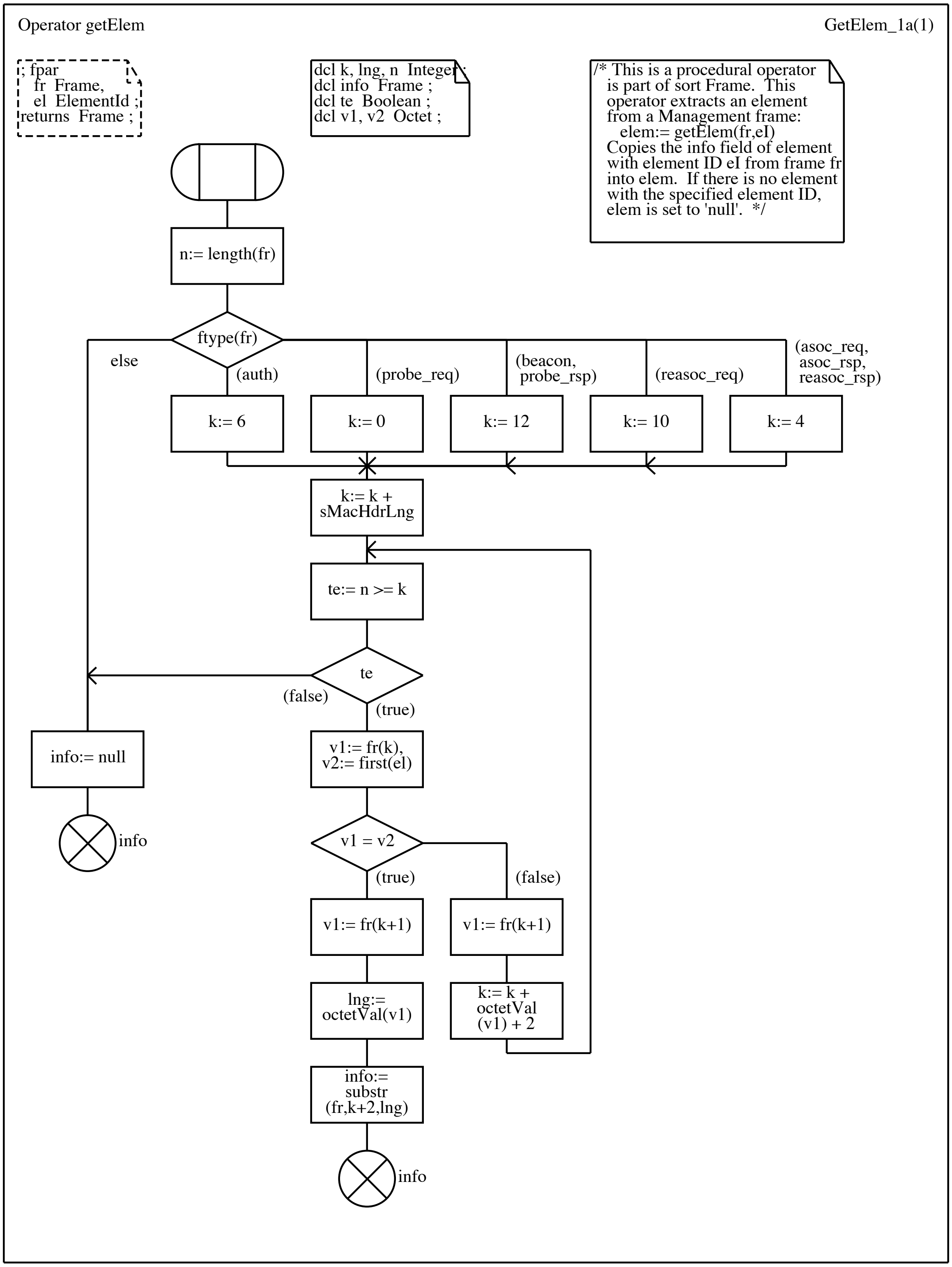


```
    capB              : Frame, Capability -> Bit; /* Capability Bcn/Probe */
    setCapB           : Frame, Capability, Bit -> Frame;
    keyId             : Frame -> KeyIndex; /* Key ID subfield (2 bits) */
    setKeyId: Frame, KeyIndex -> Frame;
  operator GetElem;
    fpar   fr Frame,   el ElementID;   returns Frame;   referenced;
  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 // aStationID;
          (ft = ps_poll) ==> mkCtl(ft, sid, bssa) ==
              ft // O1 // sid // bssa // aStationID;
          (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);
          authSeqNum(f) == octetVal(f(26)) + (octetVal(f(27)) * 256);
          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); );
        for all e in ElementID(
          mkElem(e, info) == e // mkOS(Length(info) + 2, 1) // 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;
          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 (B_S(SubStr(f,24,2)) and c) then 1 else 0 fi;
       setCapA(f,c,b) == SubStr(f,0,24) // (B_S(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 (B_S(SubStr(f,34,2)) and c) then 1 else 0 fi;
       setCapB(f,c,b) == SubStr(f,0,34) // (B_S(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) == tUsec( Usec!(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 TU(
      beaconInt(f) == octetVal(f(32)) + (octetVal(f(33)) * 256);
      listenInt(f) == octetVal(f(26)) + (octetVal(f(27)) * 256); );
    for all sta in AssocId(
      AId(f) == octetVal(f(28)) + (octetVal(f(29)) * 256);
      setAId(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;
```



```
/*****************************************************************
 *     ReasonCode sort
 *****************************************************************/
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;

/*****************************************************************
 *     StatusCode sort
 *****************************************************************/
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;
```


Operator getElem
GetElem_1a(1)
; fpar
fr Frame,
el ElementId ;
returns Frame ;
dcl k, lng, n Integer ;
dcl info Frame ;
dcl te Boolean ;
dcl v1, v2 Octet ;
/* This is a procedural operator
is part of sort Frame. This
operator extracts an element
from a Management frame:
elem:= getElem(fr,eI)
Copies the info field of element
with element ID eI from frame fr
into elem. If there is no element
with the specified element ID,
elem is set to 'null'. */
n:= length(fr)
ftype(fr)
else
(auth)
(probe_req)
(beacon,
probe_rsp)
(reasoc_req)
(asoc_req,
asoc_rsp,
reasoc_rsp)
k:= 6
k:= 0
k:= 12
k:= 10
k:= 4
k:= k +
sMacHdrLng
te:= n >= k
te
(false)
(true)
info:= null
info
v1:= fr(k),
v2:= first(el)
v1 = v2
(true)
(false)
v1:= fr(k+1)
v1:= fr(k+1)
lng:=
octetVal(v1)
k:= k +
octetVal
(v1) + 2
info:=
substr
(fr,k+2,lng)
info


```
/*******************************************************************
 *     Frame Type sorts
 *******************************************************************/
       /* TypeSubtype defines the full, 6-bit frame type identifiers.
       /* These values are useful with ftype operator of 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;
```

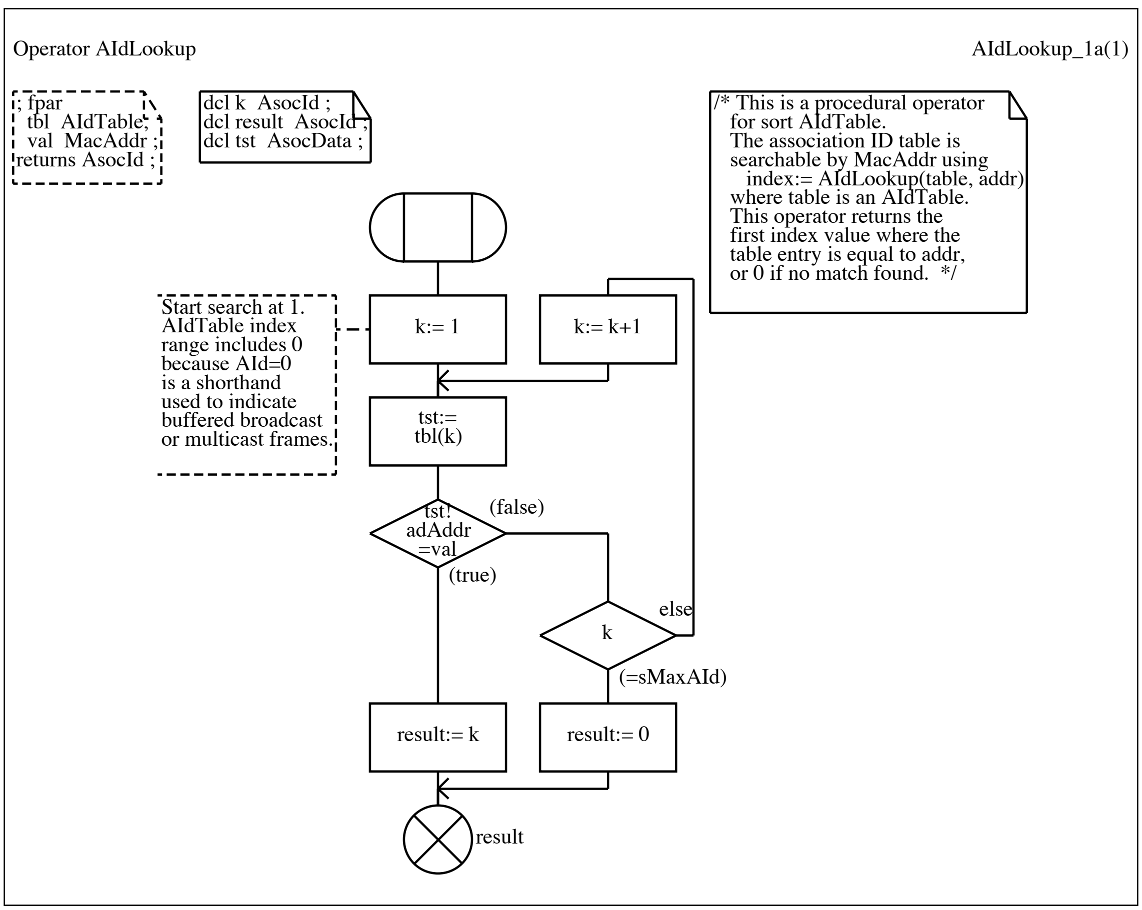


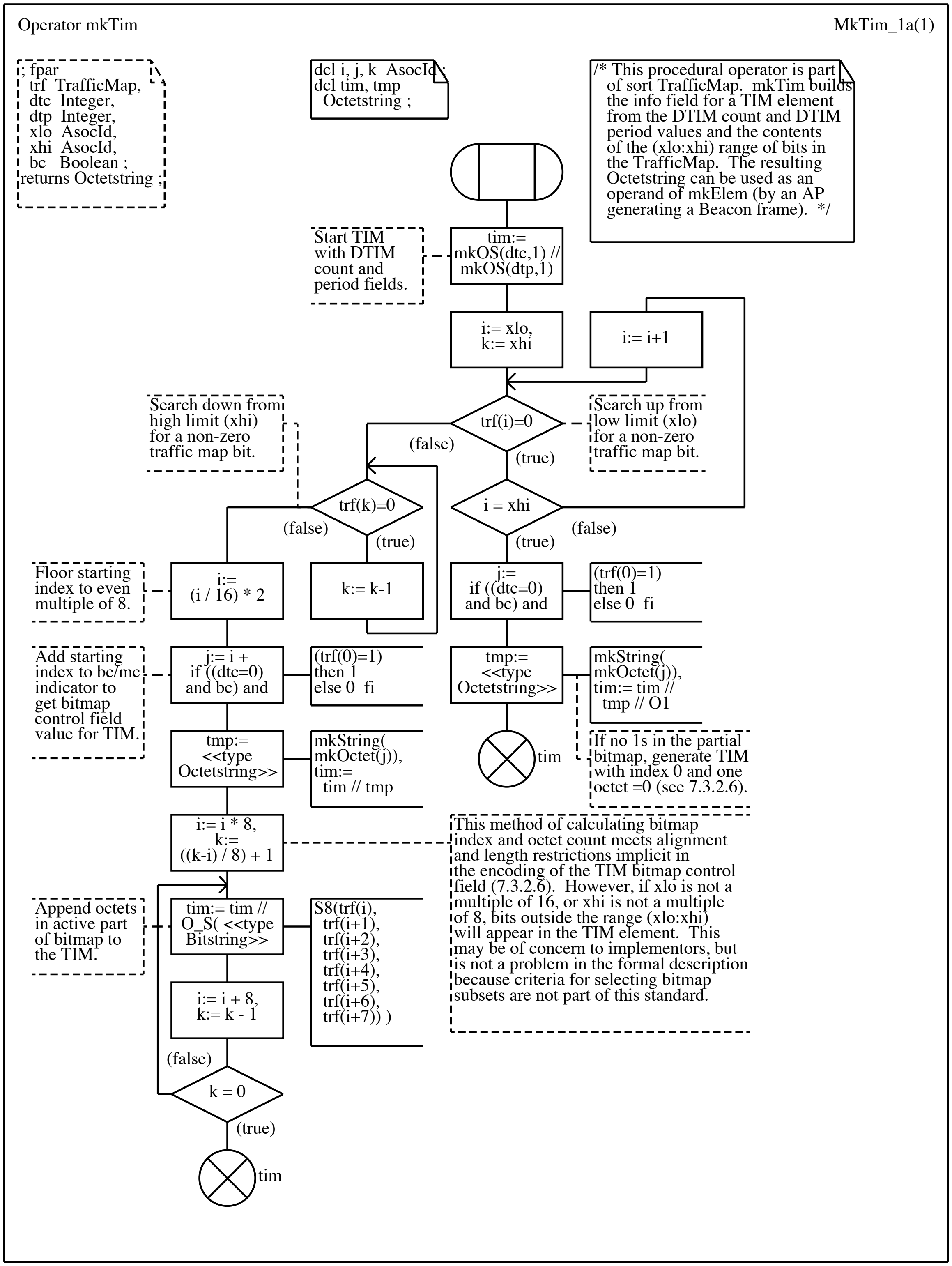
```
/*******************************************************************
 *     ElementID sort
 *******************************************************************/
newtype ElementID   inherits Octetstring   operators all;
  adding  literals eSsId,   eSupRates,   eFhParms,   eDsParms,
    eCfParms,   eTim,   eIbParms,   eCtext;
  axioms
    eSsId     == mkOS(0, 1);  /* service set identifier (0:32) */
    eSupRates == mkOS(1, 1);  /* supported rates (1:8) */
    eFhParms  == mkOS(2, 1);  /* FH parameter set (5) */
    eDsParms  == mkOS(3, 1);  /* DS parameter set (1) */
    eCfParms  == mkOS(4, 1);  /* CF parameter set (6) */
    eTim      == mkOS(5, 1);  /* Traffic Information Map (4:254) */
    eIbParms  == mkOS(6, 1);  /* IBSS parameter set (2) */
    eCtext    == mkOS(16, 1); /* challenge text (128, see 8.1.2.2) */
endnewtype ElementID;

/*******************************************************************
 *     Capability field bit assignments sort
 *******************************************************************/
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 */
endnewtype Capability;

/*******************************************************************
 *     IBSS parameter set sort
 *******************************************************************/
newtype IbssParms   inherits Octetstring   operators all;
  adding  operators
    atimWin : IbssParms -> TU;
    setAtimWin : IbssParms, TU -> IbssParms;
  axioms
    for all ib in IbssParms(     for all u in TU(
        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 sort
 *******************************************************************/
newtype CfParms   inherits Octetstring   operators all;
  adding  operators
    cfpCount          : CfParms -> Integer; /* CfpCount field (1) */
    setCfpCount       : CfParms, Integer -> CfParms;
    cfpPeriod         : CfParms -> Integer; /* CfpPeriod field (1) */
    setCfpPeriod      : CfParms, Integer -> CfParms;
    cfpMaxDur         : CfParms -> TU;      /* CfpMaxDuration field (2) */
    setCfpMaxDur      : CfParms, TU -> CfParms;
    cfpDurRem         : CfParms -> TU;      /* CfpDurRemaining field (2) */
    setCfpDurRem      : CfParms, TU -> CfParms;
  axioms
    for all cf in CfParms(   for all i in Integer(   for all u in TU(
          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);
```

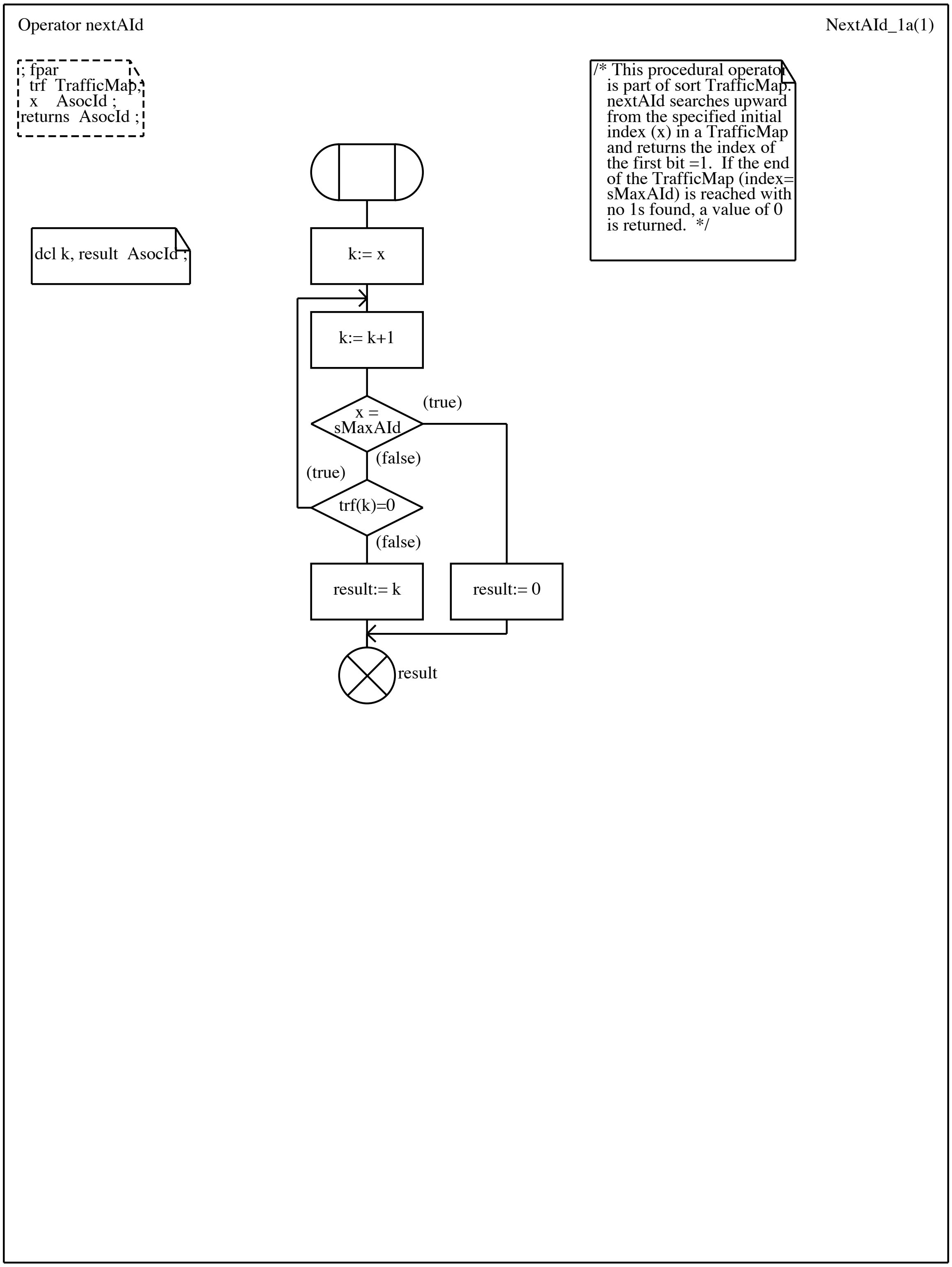


```
          cfpMaxDur(cf) == octetVal(cf(2)) + (octetVal(cf(3)) * 256);
          setCfpMaxDur(cf, u) == SubStr(cf, 0, 2) // mkOS(u mod 256, 1)
              // mkOS(u / 256, 1) // SubStr(cf, 4, 2);
          cfpDurRem(cf) == octetVal(cf(4)) + (octetVal(cf(5)) * 256);
          setCfpDurRem(cf, u) == SubStr(cf, 0, 4) // mkOS(u mod 256, 1)
              // mkOS(u / 256, 1); )));
endnewtype CfParms;

/********************************************************************
 *     Sorts for association management at AP
 ********************************************************************/
synonym sMaxAId Integer = 2007;     /* 2007 is largest allowable value */
                                    /* implementation limit may be lower */
syntype AsocId = Integer   constants 0:sMaxAId   endsyntype AsocId;
      /* Station Association Record -- only used at APs */
newtype AsocData struct
    adAddr    MacAddr;       /* address of associated station */
    adPsm     PwrSave;       /* power save mode of the station */
    adCfPoll  Boolean;       /* true if station is CfPollable */
    adPollRq  Boolean;       /* true if station requested polling */
    adNoPoll  Boolean;       /* true if station requested no polling */
    adMsduIP  Boolean;       /* true if partial Msdu outstanding to sta */
    adAuth    AuthType;      /* authentication type used by station */
    adRates   RateSet;       /* supported rates from association request */
    adAge     Time;          /* time of association */
endnewtype AsocData;
       /* Association table -- array of AsocData, only used at APs
       /*     index:= AIdLookup(table, addr)
       /* returns the index of location where table(x)!adAddr=addr
       /* or 0 if no such location found. */
newtype AIdTable   Array(AsocId, AsocData);
  adding  operators
    AIdLookup : AIdTable, MacAddr -> AsocId;
  operator AIdLookup;
    fpar tbl AIdTable,   val MacAddr;   returns AsocId;   referenced;
endnewtype AIdTable;

/********************************************************************
 *     Traffic Information Map (TIM) support sorts
 ********************************************************************/
       /* TrafficMap is an Array of Bit indexed by AId.
       /* Bits =1 in TrafficMap denote the presence of buffered frame(s)
       /* for the station assigned that AId.  TrafficMap operators are:
       /*     mkTim(trafficMap, dtimCnt, dtimPer, lowAId, highAId, bcst)
       /* returns Octetstring to use as the info field of a TIM element
       /* The TIM will contain bits =1 for TrafficMap locations in the
       /* range (lowAId):(highAId).  Buffered broadcasts and multicasts
       /* (AId 0) are indicated if dtimCnt=0 and if bcst=true.
       /*     nextAId(trafficMap, currentAId)
       /* returns index greater than currentAId at which TrafficMap=1.
       /* If no locations before sMaxAId are =1, returns 0. */
newtype TrafficMap  Array( AsocId, Bit);
  adding  operators
    mkTim : TrafficMap, Integer, Integer, AsocId, AsocId, Boolean ->
        Octetstring;
    nextAId : TrafficMap, AsocId -> AsocId;
  operator mkTim;
    fpar trf TrafficMap,  dtc Integer,  dtp Integer,  xlo AsocId,
    xhi AsocId,  bc Boolean;  returns Octetstring;  referenced;
  operator nextAId;
    fpar trf TrafficMap,  x AsocId;  returns AsocId;  referenced;
endnewtype TrafficMap;
       /* TIM is a subtype of Octetstring with operators:
       /*     bufFrame(tim,AId)   returns true if the TIM info field
```

```
       /*             (obtained using getElem) is =1 at tim(AId).
       /*      bufBcst(tim)   returns true if the TIM info field
       /*             indicates buffered broadcast/multicast traffic
       /*      dtCount(tim)   returns DTIM count value from TIM
       /*      dtPeriod(tim)  returns DTIM period value from TIM */
newtype TIM   inherits Octetstring   operators all;
  adding  operators
    bufFrame : TIM, AsocId -> Boolean;
    bufBcst  : TIM -> Boolean;
    dtCount  : TIM -> Integer;
    dtPeriod : TIM -> Integer;
  axioms
    for all el in TIM(     for all a in AsocId(
      bufFrame(el, a) ==
        if a < (octetVal(el(2) and 0xFE) * 8) then false
          else
          if a >= ((octetVal(el(2) and 0xFE)*8) + ((Length(el)-3)*8))
            then false
            else
              Extract!(B_S(el), (a-(octetVal(el(2) and 0xFE)*8)+24)) = 1
          fi fi;
      bufBcst(el) == (el(2) and 0x01) = 0x01;
      dtCount(el) == octetVal(el(0));
      dtPeriod(el) == octetVal(el(1)); ));
endnewtype TIM;
```




Operator mkTim
MkTim_1a(1)
; fpar
trf TrafficMap,
dtc Integer,
dtp Integer,
xlo AsocId,
xhi AsocId,
bc Boolean ;
returns Octetstring ;
dcl i, j, k AsocId ;
dcl tim, tmp
Octetstring ;
/* This procedural operator is part of sort TrafficMap. mkTim builds the info field for a TIM element from the DTIM count and DTIM period values and the contents of the (xlo:xhi) range of bits in the TrafficMap. The resulting Octetstring can be used as an operand of mkElem (by an AP generating a Beacon frame). */
Start TIM with DTIM count and period fields.
tim:= mkOS(dtc,1) // mkOS(dtp,1)
i:= xlo,
k:= xhi
i:= i+1
Search down from high limit (xhi) for a non-zero traffic map bit.
trf(i)=0
(false)
(true)
Search up from low limit (xlo) for a non-zero traffic map bit.
trf(k)=0
(false)
(true)
i = xhi
(true)
(false)
Floor starting index to even multiple of 8.
i:= (i / 16) * 2
k:= k-1
j:= if ((dtc=0) and bc) and
(trf(0)=1) then 1 else 0 fi
Add starting index to bc/mc indicator to get bitmap control field value for TIM.
j:= i + if ((dtc=0) and bc) and
(trf(0)=1) then 1 else 0 fi
tmp:= <<type Octetstring>>
mkString(mkOctet(j)), tim:= tim // tmp // O1
tim
If no 1s in the partial bitmap, generate TIM with index 0 and one octet =0 (see 7.3.2.6).
tmp:= <<type Octetstring>>
mkString(mkOctet(j)), tim:= tim // tmp
i:= i * 8,
k:= ((k-i) / 8) + 1
This method of calculating bitmap index and octet count meets alignment and length restrictions implicit in the encoding of the TIM bitmap control field (7.3.2.6). However, if xlo is not a multiple of 16, or xhi is not a multiple of 8, bits outside the range (xlo:xhi) will appear in the TIM element. This may be of concern to implementors, but is not a problem in the formal description because criteria for selecting bitmap subsets are not part of this standard.
Append octets in active part of bitmap to the TIM.
tim:= tim // O_S(<<type Bitstring>>
S8(trf(i), trf(i+1), trf(i+2), trf(i+3), trf(i+4), trf(i+5), trf(i+6), trf(i+7)))
i:= i + 8,
k:= k - 1
(false)
k = 0
(true)
tim


Operator nextAId
NextAId_1a(1)
; fpar
trf TrafficMap;
x AsocId ;
returns AsocId ;
/* This procedural operator is part of sort TrafficMap. nextAId searches upward from the specified initial index (x) in a TrafficMap and returns the index of the first bit =1. If the end of the TrafficMap (index= sMaxAId) is reached with no 1s found, a value of 0 is returned. */
dcl k, result AsocId ;
k:= x
k:= k+1
x = sMaxAId
(true)
(false)
(true)
trf(k)=0
(false)
result:= k
result:= 0
result


```
/********************************************************************
 *     Multi-rate support sorts
 ********************************************************************/
newtype Rate   inherits Octet   operators all;
  adding  operators
    calcDur : Rate, Integer -> Integer;
                      /* converts (rate,bitCount) to integer microseconds */
    rateVal : Rate -> Rate; /* clears high-order bit */
    basicRate : Rate -> Rate; /* sets high-order bit */
    isBasic : Rate -> Boolean; /* true if high-order bit set */
  axioms
    for all r in Rate(    for all i in Integer(    for all b in Boolean(
          calcDur(r, i) == ((((10000000 + (octetVal(r and 0x7F) - 1)) /
              (500 * octetVal(r and 0x7F))) * i) + 9999) / 10000;
          rateVal(r) == r and 0x7F;
          basicRate(r) == r or 0x80;
          isBasic(r) == (r and 0x80) = 0x80; )));
endnewtype Rate;

syntype RateString = Octetstring   endsyntype RateString;

/********************************************************************
 *     MPDU duration factor support sort
 ********************************************************************/
       /* These operators support the encoding used to allow
       /* an Integer to represent the value of aMpduDurationFactor.
       /*     calcDF(PlcpBits, MpduBits)  returns an Integer which is
       /* the fractional part of ((PlcpBits/MpduBits)-1)*(1e9).
       /*     stuff(durFactor, MpduBits)  returns the number of PlcpBits
       /* which result from MpduBits at the specified durFactor. */
newtype DurFactor   inherits Integer   operators all;
  adding  operators
    calcDF : Integer, Integer -> DurFactor;
    stuff  : DurFactor, Integer -> Integer;
  axioms
    for all df in DurFactor(     for all mb, pb in Integer(
        calcDF(pb, mb) == ((pb * 1000000000) / mb) - 1000000000;
        stuff(df, mb) == ((mb * df) + (mb - 1)) / 1000000000; ));
endnewtype DurFactor;

/********************************************************************
 *     Generic PHY parameter set sort
 ********************************************************************/
       /* Generic PHY parameter element for signals related to Beacons
       /* and Probe Responses that are PHY-type independent. */
syntype PhyParms = Octetstring   endsyntype PhyParms;

/********************************************************************
 *     FH parameter set sort
 ********************************************************************/
newtype FhParms   inherits Octetstring   operators all;
  adding  operators
    dwellTime          : FhParms -> TU;      /* Dwell Time field (2) */
    setDwellTime       : FhParms, TU -> FhParms;
    hopSet             : FhParms -> Integer; /* Hop Set field (1) */
    setHopSet          : FhParms, Integer -> FhParms;
    hopPattern         : FhParms -> Integer; /* Hop Pattern field (1) */
    setHopPattern      : FhParms, Integer -> FhParms;
    hopIndex           : FhParms -> Integer; /* Hop Index field (1) */
    setHopIndex        : FhParms, Integer -> FhParms;
  axioms
    for all fh in FhParms(   for all i in Integer(   for all u in TU(
      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;

/*****************************************************************
 *     DS parameter set sort
 *****************************************************************/
newtype DsParms   inherits Octetstring   operators all;
  adding  operators
    curChannel: DsParms -> Integer; /* Current Channel (1) */
    setCurChannel: DsParms, Integer -> DsParms;
  axioms
    for all ds in DsParms(     for all i in Integer(
        curChannel(ds) == octetVal(ds(0));
        setCurChannel(ds, i) == mkOS(i); ));
endnewtype DsParms;

endpackage;
/******************************************************************/


/******************************************************************/
use macsorts;
Package macmib;

       /* This Package contains definitions of the MAC MIB attributes
       /* and the subset of the PHY MIB attributes used by the MAC state
       /* machines.  These are needed under Z.100 to permit analysis of
       /* the state machine definitions.  In future revisions these may
       /* be replaced with the ASN.1 MIB definition, from Annex D, if
       /* a Z.105-compliant SDL tool is available. */

/*****************************************************************
 *     StationConfig Group
 *****************************************************************/
remote aMediumOccupancyLimit  TU nodelay;
remote aReceiveDTIMs  Boolean nodelay;
synonym aCfPollable  Boolean = <<package macsorts>> sCfPollable;
remote aCfpPeriod  Integer nodelay;
remote aCfpMaxDuration  TU nodelay;
remote aAuthenticationResponseTimeout  TU nodelay;
remote aAuthenticationType  AuthTypeSet nodelay;
remote aWepUndecryptableCount  Counter32 nodelay;

/*****************************************************************
 *     AuthenticationAlgorithms Table
 *****************************************************************/
synonym aAuthenticationAlgorithms  AuthTypeSet =
    incl(open_system, incl(shared_key));
              /* NOTE:  Only  include shared_key in this set
                 if aPrivacyOptionImplemented=true. */

/*****************************************************************
 *     WepDefaultKeys Table       (only if aPrivacyOptionImplemented=true)
 *****************************************************************/
remote aWepDefaultKeys  KeyVector nodelay;

/*****************************************************************
 *     WepKeyMappings Table       (only if aPrivacyOptionImplemented=true)
```



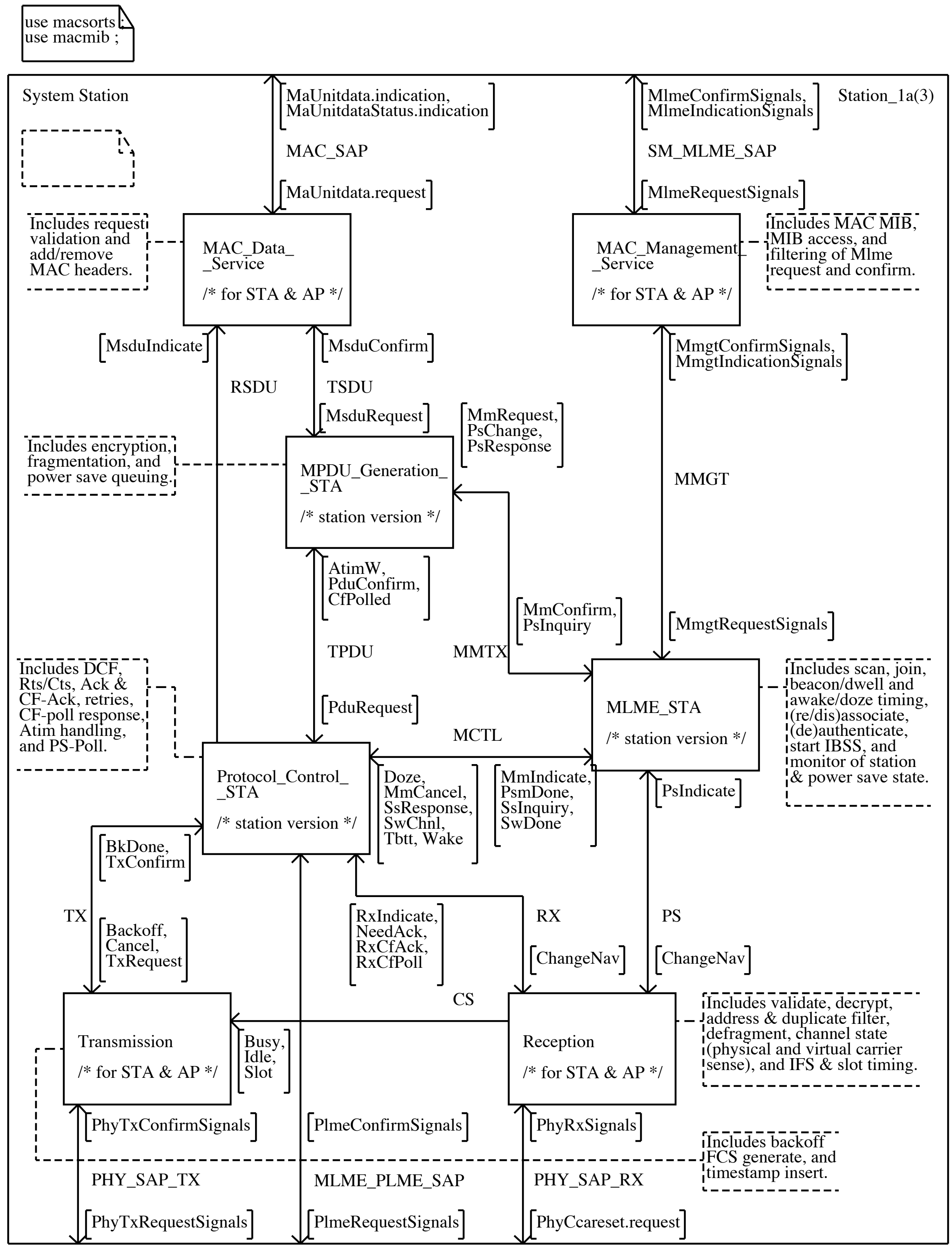
```
 *******************************************************************/
remote aWepKeyMappings  KeyMapArray nodelay;
synonym aWepKeyMappingLength  Integer =
    <<package macsorts>> sWepKeyMappingLength;

/*******************************************************************
 *     Privacy Group     (only 1 item if aPrivacyOptionImplemented=false)
 *******************************************************************/
synonym aPrivacyOptionImplemented  Boolean = true;
remote aPrivacyInvoked  Boolean nodelay;
remote aWepDefaultKeyId  KeyIndex nodelay;
remote aExcludeUnencrypted  Boolean nodelay;
remote aWepIcvErrorCount  Counter32 nodelay;
remote aWepExcludedCount  Counter32 nodelay;

/*******************************************************************
 *     Operation Group
 *******************************************************************/
synonym aMacAddress MacAddr =
  <<type MacAddr>> S6(0x00, 0x11, 0x22, 0x33, 0x44, 0x55);
    /* each station has a unique globally administered address */
    /* Value may be overwritten with locally administered address at */
    /* MlmeReset, but is always a static value during MAC operation */
remote aRtsThreshold  Integer nodelay;
remote aShortRetryLimit  Integer nodelay;
remote aLongRetryLimit  Integer nodelay;
remote aFragmentationThreshold  Integer nodelay;
remote aMaxTransmitMsduLifetime  TU nodelay;
remote aMaxReceiveLifetime  TU nodelay;
synonym aManufacturerId  Charstring = 'name of manufacturer';
synonym aProductId  Charstring = 'identifier unique to manufacturer';

/*******************************************************************
 *     GroupAddresses Table
 *******************************************************************/
remote aGroupAddresses  MacAddrSet nodelay;

/*******************************************************************
 *     Counters Group
 *******************************************************************/
remote aTransmittedFragmentCount  Counter32 nodelay;
remote aMulticastTransmittedFrameCount  Counter32 nodelay;
remote aFailedCount  Counter32 nodelay;
remote aRetryCountCounter32 nodelay;
remote aMultipleRetryCount  Counter32 nodelay;
remote aRtsSuccessCount  Counter32 nodelay;
remote aRtsFailureCount  Counter32 nodelay;
remote aAckFailureCount  Counter32 nodelay;
remote aReceivedFragmentCount  Counter32 nodelay;
remote aMulticastReceivedFrameCount  Counter32 nodelay;
remote aFcsErrorCount  Counter32 nodelay;
remote aFrameDuplicateCount  Counter32 nodelay;

/*******************************************************************
 *     PhyOperation Group     (values shown are mostly for FH PHY)
 *******************************************************************/
synonym aPHYType  Integer = 01;
synonym RegDomainSupported Octetstring = S4(0x10, 0x20, 0x30, 0x00);
remote aCurrentRegDomain  Integer nodelay;
synonym aSlotTime  Usec = (aCcaTime + aRxTxTurnaroundTime +
    aAirPropagationTime + aMacProcessingTime);
synonym aCcaTime  Usec = 27;
synonym aRxTxTurnaroundTime Usec = (aTxPlcpDelay + aRxTxSwitchTime +
    aTxRampOnTime + aTxRfDelay);
```

```
synonym aTxPlcpDelay  Usec = 1;
synonym aRxTxSwitchTime  Usec = 10;
synonym aTxRampOnTime  Usec = 8;
synonym aTxRfDelay  Usec = 1;
synonym aSifsTime Usec = (aRxRfDelay + aRxPlcpDelay +
    aMacProcessingTime + aRxTxTurnaroundTime);
synonym aRxRfDelay  Usec = 4;
synonym aRxPlcpDelay  Usec = 2;
synonym aMacProcessingTime  Usec = 2;
synonym aTxRampOffTime  Usec = 8;
synonym aPreambleLength  Usec = 96;
synonym aPlcpHeaderLength  Usec = 32;
synonym aMpduDurationFactor  <<package macsorts>> DurFactor = 31250000;
synonym aAirPropagationTime  Usec = 1;
synonym aTempType  Integer = 01;
synonym aCWmax  Integer = 1023;
synonym aCWmin  Integer = 15;

/*******************************************************************
 *     PhyRate Group     (values shown are mostly for FH PHY)
 *******************************************************************/
synonym aSupportedRatesTx  Octetstring = S3(0x82, 0x04, 0x00);
synonym aSupportedRatesRx  Octetstring = S3(0x82, 0x04, 0x00);
synonym aMpduMaxLength  Integer = 4095;

/*******************************************************************
 *     PhyFHSS Group     (only used with FH PHY)
 *******************************************************************/
synonym aHopTime  Usec = 224;
remote aCurrentChannelNumber  Integer nodelay;
synonym aMaxDwellTime  TU = 390;
remote aCurrentSet  Integer nodelay;
remote aCurrentPattern  Integer nodelay;
remote aCurrentIndex  Integer nodelay;

/*******************************************************************/
       /*     The MAC state machines do not reference any attributes in
       /*            PhyAntennaGroup,   PhyTxPowerGroup,   PhyDsssGroup,
       /*            PhyStatusGroup,    PhyPowerSavingGroup,
       /*            PhyIR Group,       AntennasList  */

endpackage;
/*******************************************************************/
```


C.3 State machines for MAC stations

The following SDL-92 system specification defines operation of the MAC protocol at an IEEE 802.11 STA. Many aspects of STA operation also apply to AP operation. These are defined in blocks and processes referenced from both the STA and AP system specifications. Blocks and processes used in both STA and AP are identifiable by the SDL comment /* for STA & AP */ below the block or process name. Blocks and processes specific to STA operation are identifiable by the SDL comment /* station version */ below the block or process name. The definitions of all blocks and processes referenced in the station system specification appear in Clause C.3.

The remainder of Clause C.3 is the formal description, in SDL/GR, of an IEEE 802.11 STA.


use macsorts ;
use macmib ;
System Station
Station_1a(3)
MaUnitdata.indication,
MaUnitdataStatus.indication
MAC_SAP
MaUnitdata.request
MlmeConfirmSignals,
MlmeIndicationSignals
SM_MLME_SAP
MlmeRequestSignals
Includes request validation and add/remove MAC headers.
MAC_Data_
_Service
/* for STA & AP */
MAC_Management_
_Service
/* for STA & AP */
Includes MAC MIB, MIB access, and filtering of Mlme request and confirm.
MsduIndicate
MsduConfirm
RSDU
TSDU
MsduRequest
MmRequest,
PsChange,
PsResponse
MmgtConfirmSignals,
MmgtIndicationSignals
MMGT
Includes encryption, fragmentation, and power save queuing.
MPDU_Generation_
_STA
/* station version */
AtimW,
PduConfirm,
CfPolled
MmConfirm,
PsInquiry
MmgtRequestSignals
TPDU
MMTX
Includes DCF, Rts/Cts, Ack & CF-Ack, retries, CF-poll response, Atim handling, and PS-Poll.
PduRequest
MCTL
MLME_STA
/* station version */
Includes scan, join, beacon/dwell and awake/doze timing, (re/dis)associate, (de)authenticate, start IBSS, and monitor of station & power save state.
Protocol_Control_
_STA
/* station version */
Doze,
MmCancel,
SsResponse,
SwChnl,
Tbtt, Wake
MmIndicate,
PsmDone,
SsInquiry,
SwDone
PsIndicate
BkDone,
TxConfirm
TX
Backoff,
Cancel,
TxRequest
RxIndicate,
NeedAck,
RxCfAck,
RxCfPoll
RX
PS
ChangeNav
ChangeNav
CS
Transmission
/* for STA & AP */
Busy,
Idle,
Slot
Reception
/* for STA & AP */
Includes validate, decrypt, address & duplicate filter, defragment, channel state (physical and virtual carrier sense), and IFS & slot timing.
PhyTxConfirmSignals
PlmeConfirmSignals
PhyRxSignals
Includes backoff FCS generate, and timestamp insert.
PHY_SAP_TX
MLME_PLME_SAP
PHY_SAP_RX
PhyTxRequestSignals
PlmeRequestSignals
PhyCcareset.request


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

System Station   Sta_signals_2b(3)

```
signal
  AtimW,
  Backoff(Integer,Integer),
  BkDone(Integer),
  Busy,
  Cancel,
  CfPolled,
  ChangeNav(Time,Duration,NavSrc),
  Doze,
  Idle,
  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(MlmeStatus),
  MlmeAssociate.indication(MacAddr),
  MlmeAssociate.request(MacAddr,TU,Capability,Integer),
  MlmeAuthenticate.confirm
    (MacAddr,AuthType,MlmeStatus),
  MlmeAuthenticate.indication(MacAddr,AuthType),
  MlmeAuthenticate.request(MacAddr,AuthType,TU),
  MlmeDeauthenticate.confirm(MacAddr,MlmeStatus),
  MlmeDeauthenticate.indication(MacAddr,ReasonCode),
  MlmeDeauthenticate.request(MacAddr,ReasonCode),
  MlmeDisassociate.confirm(MlmeStatus),
  MlmeDisassociate.indication(MacAddr,ReasonCode),
  MlmeDisassociate.request(MacAddr,ReasonCode),
  MlmeGet.confirm(MibStatus,MibAtrib,MibValue),
  MlmeGet.request(MibAtrib),
  MlmeJoin.confirm(MlmeStatus),
  MlmeJoin.request(BssDscr,Integer,Usec,Ratestring),
  MlmePowermgt.confirm(MlmeStatus),
  MlmePowermgt.request(PwrSave,Boolean,Boolean),
  MlmeReassociate.confirm(MlmeStatus),
  MlmeReassociate.indication(MacAddr),
  MlmeReassociate.request(MacAddr,TU,Capability,Integer),
  MlmeReset.confirm(MlmeStatus),
  MlmeReset.request(MacAddr,Boolean),
  MlmeScan.confirm(BssDscrSet,MlmeStatus),
  MlmeScan.request(BssTypeSet,MacAddr,Octetstring,
    ScanType,Usec,Intstring,TU,TU),
  MlmeSet.confirm(MibStatus,MibAtrib),
  MlmeSet.request(MibAtrib,MibValue),
  MlmeStart.confirm(MlmeStatus),
  MlmeStart.request(Octetstring,BssType,TU,
    Integer,CfParms,PhyParms,IbssParms,Usec,
    Capability,Ratestring,Ratestring) ;
```

```
signal
  MmCancel,
  MmConfirm(Frame,TxStatus),
  MmIndicate(Frame,Time,Time,StateErr),
  MmRequest(Frame,Imed,Rate),
  MsduConfirm(Frame,CfPriority,TxStatus),
  MsduIndicate(Frame,CfPriority),
  MsduRequest(Frame,CfPriority),
  NeedAck(MacAddr,Time,Duration,Rate),
  PduConfirm(FragSdu,TxResult),
  PduRequest(FragSdu),
  PhyCca.indication(Ccastatus),
  PhyCcarst.confirm,
  PhyCcarst.request,
  PhyData.confirm,
  PhyData.indication(Octet),
  PhyData.request(Octet),
  PhyRxEnd.indication(PhyRxStat),
  PhyRxStart.indication(Integer,Rate),
  PhyTxEnd.confirm,
  PhyTxEnd.request,
  PhyTxStart.confirm,
  PhyTxStart.request(Integer,Rate),
  PlmeGet.confirm(MibStatus,
    MibAtrib,MibValue),
  PlmeGet.request(MibAtrib),
  PlmeReset.confirm(Boolean),
  PlmeReset.request,
  PlmeSet.confirm(MibStatus,MibAtrib),
  PlmeSet.request(MibAtrib,MibValue),
  PsmDone,
  PsChange(MacAddr,PsMode),
  PsIndicate(MacAddr,PsMode),
  PsInquiry(MacAddr),
  PsResponse(MacAddr,PsMode),
  ResetMAC,
  RxCfAck(MacAddr),
  RxIndicate(Frame,Time,Time,Rate),
  Slot,
  SsInquiry(MacAddr),
  SsResponse(MacAddr,
    StationState,StationState),
  SwChnl(Integer,Boolean),
  SwDone,
  TBTT,
  TxConfirm,
  TxRequest(Frame,Rate),
  Wake ;
```

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

System Station Sta_signallists_3a(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
MlmeIndicationSignals=
   MlmeAuthenticate.indication,
   MlmeDeauthenticate.indication,
   MlmeDisassociate.indication,
   MlmeAssociate.indication,
   MlmeReassociate.indication ;
```

```
signallist
MmgtRequestSignals=
   MlmeAssociate.request,
   MlmeAuthenticate.request,
   MlmeDeauthenticate.request,
   MlmeDisassociate.request,
   MlmeJoin.request,
   MlmePowermgt.request,
   MlmeReassociate.request,
   MlmeScan.request,
   MlmeStart.request ;
```

```
signallist
MmgtConfirmSignals=
   MlmeAssociate.confirm,
   MlmeAuthenticate.confirm,
   MlmeDeauthenticate.confirm,
   MlmeDisassociate.confirm,
   MlmeJoin.confirm,
   MlmePowermgt.confirm,
   MlmeReassociate.confirm,
   MlmeScan.confirm,
   MlmeStart.confirm ;
```

```
signallist
MmgtIndicationSignals=
   MlmeAuthenticate.indication,
   MlmeDeauthenticate.indication,
   MlmeDisassociate.indication,
   MlmeAssociate.indication,
   MlmeReassociate.indication ;
```

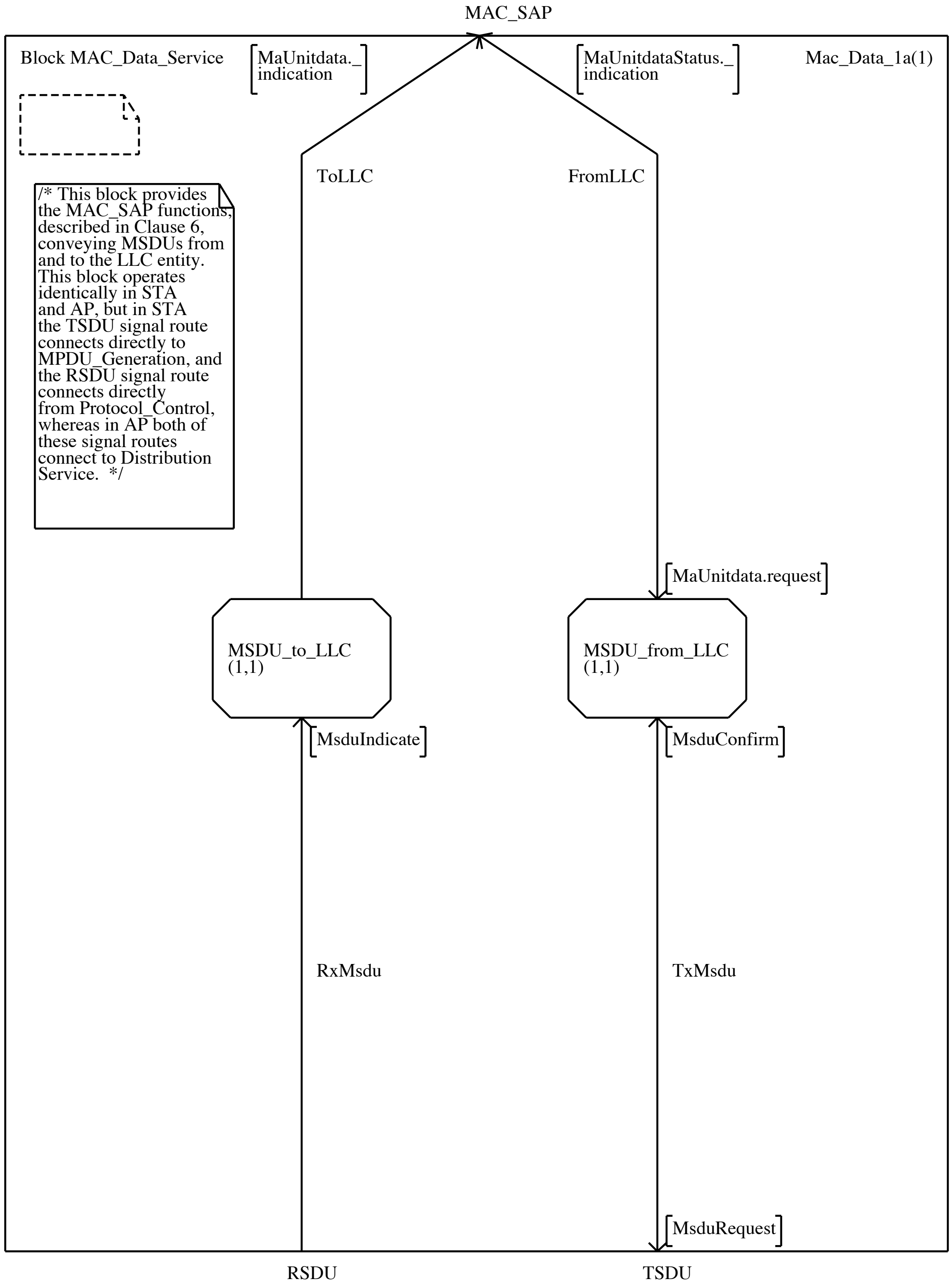
```
signallist
PhyTxRequestSignals=
   PhyTxStart.request,
   PhyTxEnd.request,
   PhyData.request ;
```

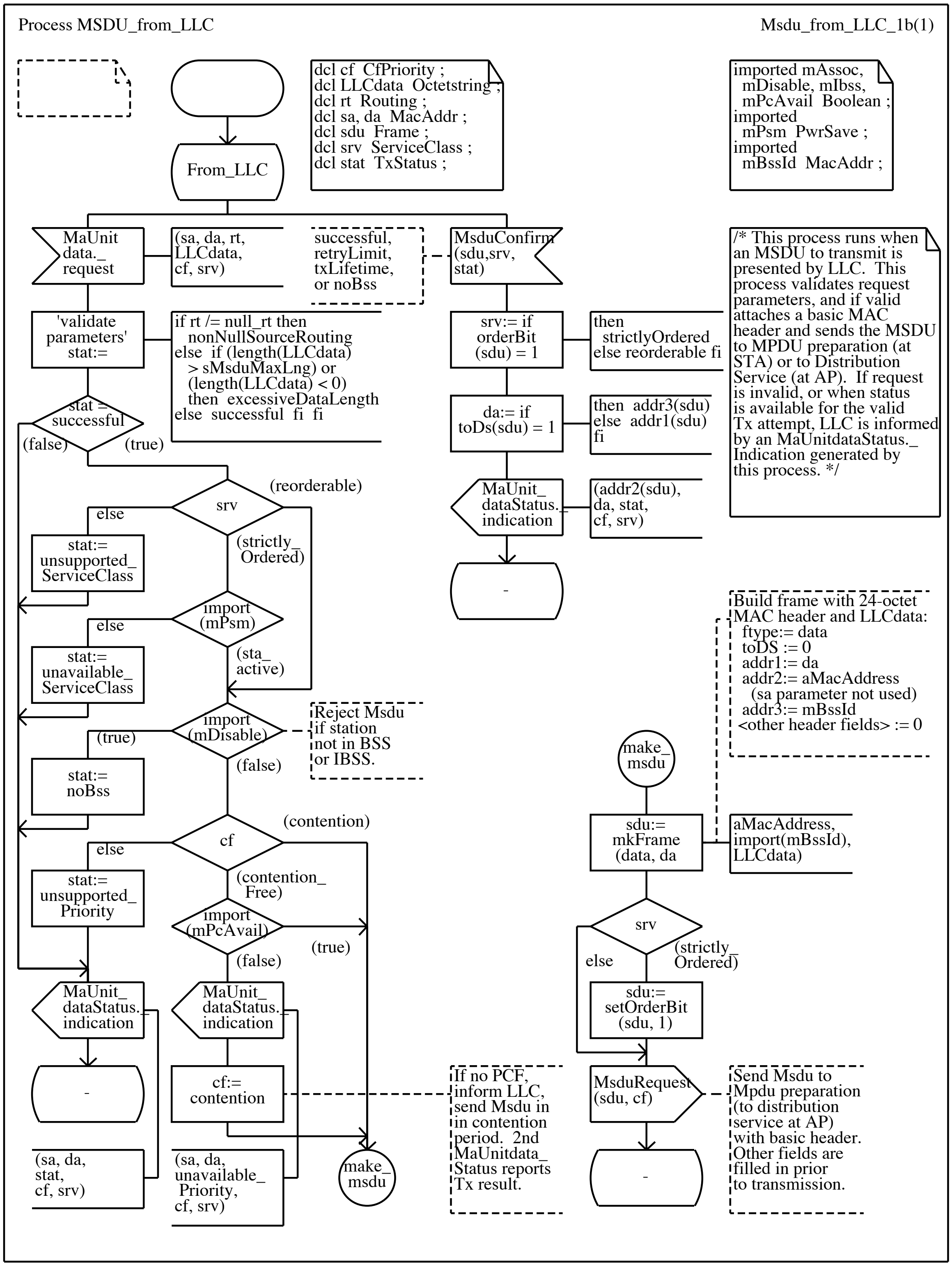
```
signallist
PhyTxConfirmSignals=
   PhyTxStart.confirm,
   PhyTxEnd.confirm,
   PhyData.confirm ;
```

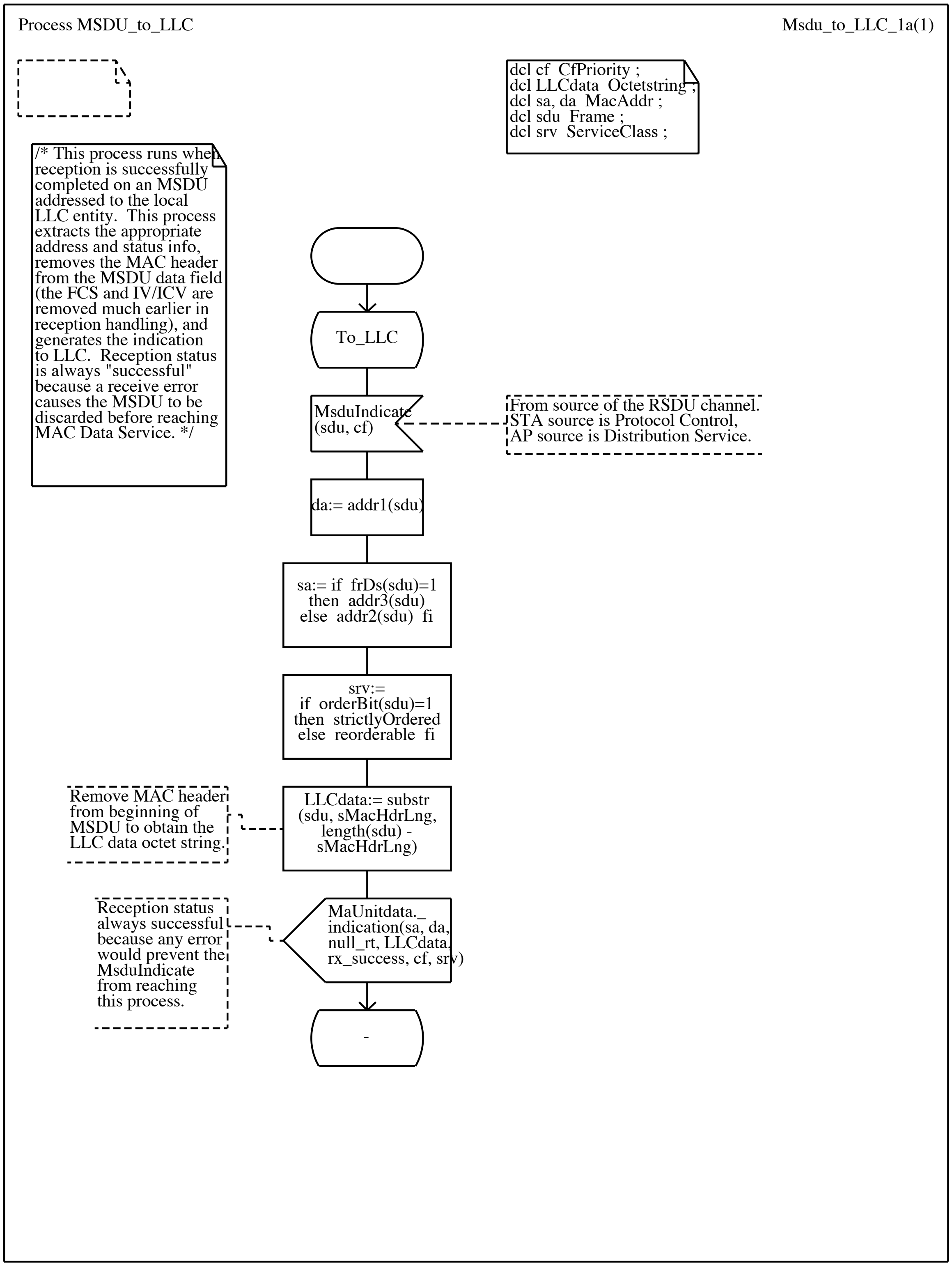
```
signallist
PhyRxSignals=
   PhyRxStart.indication,
   PhyRxEnd.indication,
   PhyData.indication,
   PhyCca.indication,
   PhyCcareset.confirm ;
```

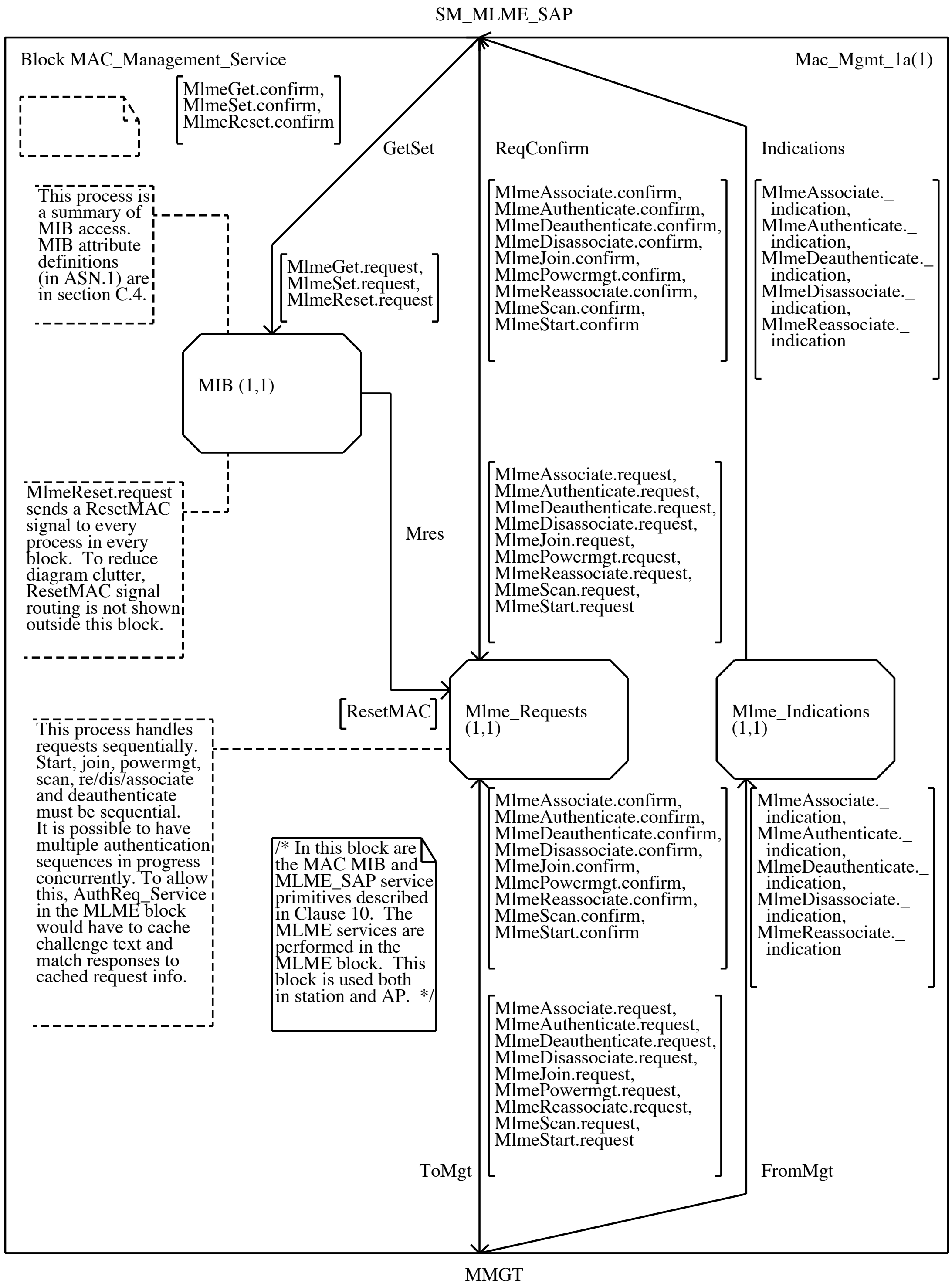
```
signallist
PlmeRequestSignals=
   PlmeGet.request,
   PlmeSet.request,
   PlmeReset.request ;
```

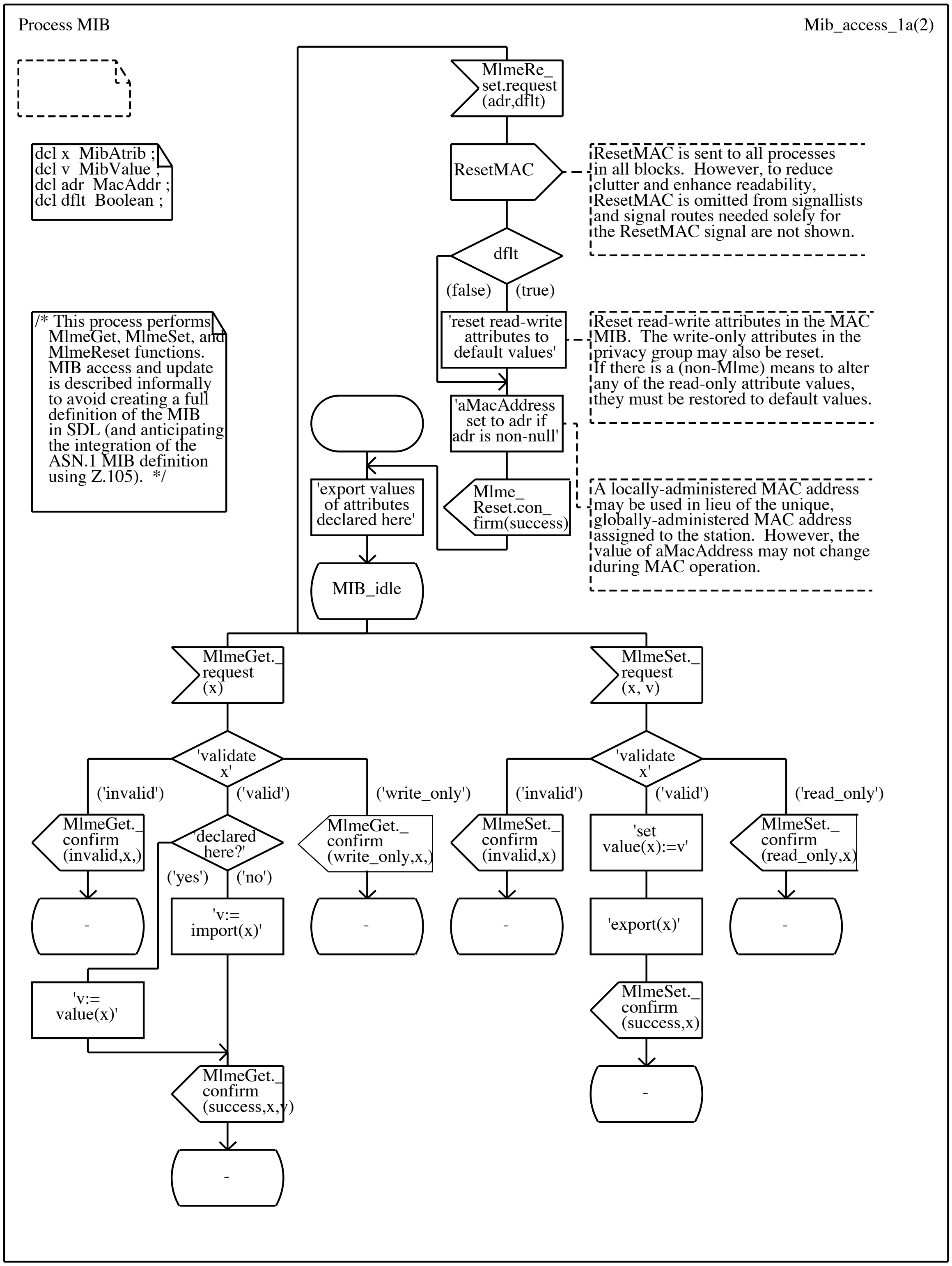
```
signallist
PlmeConfirmSignals=
   PlmeGet.confirm,
   PlmeReset.confirm,
   PlmeSet.confirm ;
```



MAC_SAP
Block MAC_Data_Service
MaUnitdata._ indication
MaUnitdataStatus._ indication
Mac_Data_1a(1)
/* This block provides the MAC_SAP functions, described in Clause 6, conveying MSDUs from and to the LLC entity. This block operates identically in STA and AP, but in STA the TSDU signal route connects directly to MPDU_Generation, and the RSDU signal route connects directly from Protocol_Control, whereas in AP both of these signal routes connect to Distribution Service. */
ToLLC
FromLLC
MaUnitdata.request
MSDU_to_LLC (1,1)
MSDU_from_LLC (1,1)
MsduIndicate
MsduConfirm
RxMsdu
TxMsdu
MsduRequest
RSDU
TSDU


Process MSDU_from_LLC
Msdu_from_LLC_1b(1)
dcl cf CfPriority ;
dcl LLCdata Octetstring ;
dcl rt Routing ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl srv ServiceClass ;
dcl stat TxStatus ;
imported mAssoc, mDisable, mIbss, mPcAvail Boolean ;
imported mPsm PwrSave ;
imported mBssId MacAddr ;
From_LLC
MaUnit data._ request
(sa, da, rt, LLCdata, cf, srv)
successful, retryLimit, txLifetime, or noBss
MsduConfirm (sdu,srv, stat)
/* This process runs when an MSDU to transmit is presented by LLC. This process validates request parameters, and if valid attaches a basic MAC header and sends the MSDU to MPDU preparation (at STA) or to Distribution Service (at AP). If request is invalid, or when status is available for the valid Tx attempt, LLC is informed by an MaUnitdataStatus._ Indication generated by this process. */
'validate parameters' stat:=
if rt /= null_rt then nonNullSourceRouting else if (length(LLCdata) > sMsduMaxLng) or (length(LLCdata) < 0) then excessiveDataLength else successful fi fi
srv:= if orderBit (sdu) = 1
then strictlyOrdered else reorderable fi
da:= if toDs(sdu) = 1
then addr3(sdu) else addr1(sdu) fi
MaUnit_ dataStatus._ indication
(addr2(sdu), da, stat, cf, srv)
-
stat = successful
(false)
(true)
srv
(reorderable)
else
(strictly_ Ordered)
stat:= unsupported_ ServiceClass
import (mPsm)
else
(sta_ active)
stat:= unavailable_ ServiceClass
import (mDisable)
(true)
(false)
Reject Msdu if station not in BSS or IBSS.
stat:= noBss
cf
(contention)
else
(contention_ Free)
stat:= unsupported_ Priority
import (mPcAvail)
(true)
(false)
MaUnit_ dataStatus._ indication
MaUnit_ dataStatus._ indication
-
cf:= contention
If no PCF, inform LLC, send Msdu in in contention period. 2nd MaUnitdata_ Status reports Tx result.
(sa, da, stat, cf, srv)
(sa, da, unavailable_ Priority, cf, srv)
make_ msdu
Build frame with 24-octet MAC header and LLCdata:
ftype:= data
toDS := 0
addr1:= da
addr2:= aMacAddress
(sa parameter not used)
addr3:= mBssId
<other header fields> := 0
make_ msdu
sdu:= mkFrame (data, da
aMacAddress, import(mBssId), LLCdata)
srv
else
(strictly_ Ordered)
sdu:= setOrderBit (sdu, 1)
MsduRequest (sdu, cf)
Send Msdu to Mpdu preparation (to distribution service at AP) with basic header. Other fields are filled in prior to transmission.
-


Process MSDU_to_LLC
Msdu_to_LLC_1a(1)
dcl cf CfPriority ;
dcl LLCdata Octetstring ;
dcl sa, da MacAddr ;
dcl sdu Frame ;
dcl srv ServiceClass ;
/* This process runs when reception is successfully completed on an MSDU addressed to the local LLC entity. This process extracts the appropriate address and status info, removes the MAC header from the MSDU data field (the FCS and IV/ICV are removed much earlier in reception handling), and generates the indication to LLC. Reception status is always "successful" because a receive error causes the MSDU to be discarded before reaching MAC Data Service. */
To_LLC
MsduIndicate (sdu, cf)
From source of the RSDU channel. STA source is Protocol Control, AP source is Distribution Service.
da:= addr1(sdu)
sa:= if frDs(sdu)=1 then addr3(sdu) else addr2(sdu) fi
srv:= 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, rx_success, cf, srv)
-


SM_MLME_SAP
Block MAC_Management_Service
Mac_Mgmt_1a(1)
MlmeGet.confirm, MlmeSet.confirm, MlmeReset.confirm
GetSet
ReqConfirm
Indications
This process is a summary of MIB access. MIB attribute definitions (in ASN.1) are in section C.4.
MlmeGet.request, MlmeSet.request, MlmeReset.request
MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm
MlmeAssociate._ indication, MlmeAuthenticate._ indication, MlmeDeauthenticate._ indication, MlmeDisassociate._ indication, MlmeReassociate._ indication
MIB (1,1)
MlmeReset.request sends a ResetMAC signal to every process in every block. To reduce diagram clutter, ResetMAC signal routing is not shown outside this block.
Mres
MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request, MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request
ResetMAC
Mlme_Requests (1,1)
Mlme_Indications (1,1)
This process handles requests sequentially. Start, join, powermgt, scan, re/dis/associate and deauthenticate must be sequential. It is possible to have multiple authentication sequences in progress concurrently. To allow this, AuthReq_Service in the MLME block would have to cache challenge text and match responses to cached request info.
/* In this block are the MAC MIB and MLME_SAP service primitives described in Clause 10. The MLME services are performed in the MLME block. This block is used both in station and AP. */
MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm
MlmeAssociate._ indication, MlmeAuthenticate._ indication, MlmeDeauthenticate._ indication, MlmeDisassociate._ indication, MlmeReassociate._ indication
MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request, MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request
ToMgt
FromMgt
MMGT


Process MIB
Mib_access_1a(2)
dcl x MibAtrib ;
dcl v MibValue ;
dcl adr MacAddr ;
dcl dflt Boolean ;
/* This process performs MlmeGet, MlmeSet, and MlmeReset functions. MIB access and update is described informally to avoid creating a full definition of the MIB in SDL (and anticipating the integration of the ASN.1 MIB definition using Z.105). */
MlmeRe_ set.request (adr,dflt)
ResetMAC
ResetMAC is sent to all processes in all blocks. However, to reduce clutter and enhance readability, ResetMAC is omitted from signallists and signal routes needed solely for the ResetMAC signal are not shown.
dflt
(false)
(true)
'reset read-write attributes to default values'
Reset read-write attributes in the MAC MIB. The write-only attributes in the privacy group may also be reset. If there is a (non-Mlme) means to alter any of the read-only attribute values, they must be restored to default values.
'aMacAddress set to adr if adr is non-null'
A locally-administered MAC address may be used in lieu of the unique, globally-administered MAC address assigned to the station. However, the value of aMacAddress may not change during MAC operation.
Mlme_ Reset.con_ firm(success)
'export values of attributes declared here'
MIB_idle
MlmeGet._ request (x)
'validate x'
('invalid')
('valid')
('write_only')
MlmeGet._ confirm (invalid,x,)
'declared here?'
('yes')
('no')
MlmeGet._ confirm (write_only,x,)
-
'v:= import(x)'
'v:= value(x)'
MlmeGet._ confirm (success,x,v)
MlmeSet._ request (x, v)
('read_only')
MlmeSet._ confirm (invalid,x)
'set value(x):=v'
MlmeSet._ confirm (read_only,x)
'export(x)'
MlmeSet._ confirm (success,x)

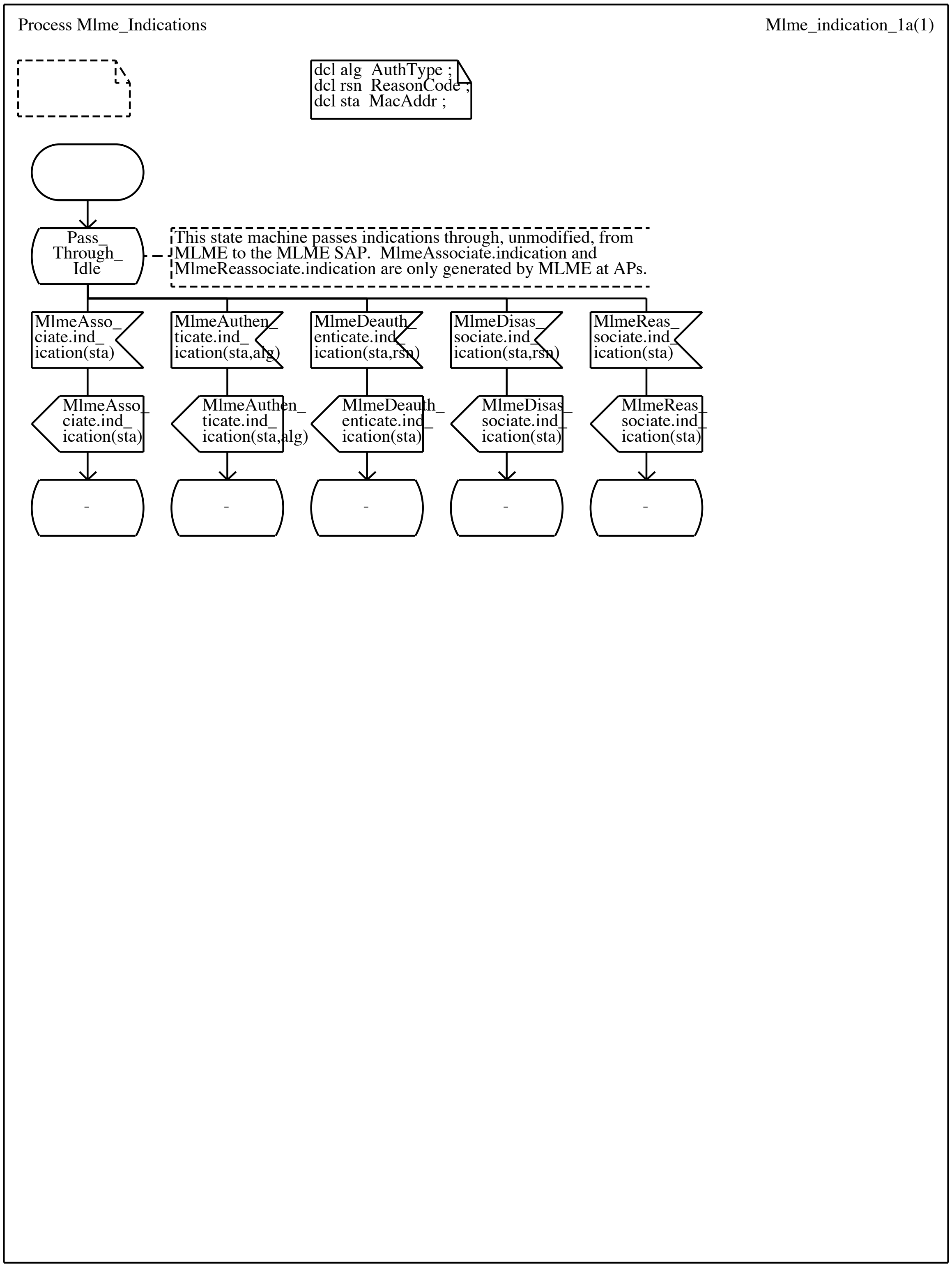
Process MIB Mib_import_export_2a(2)

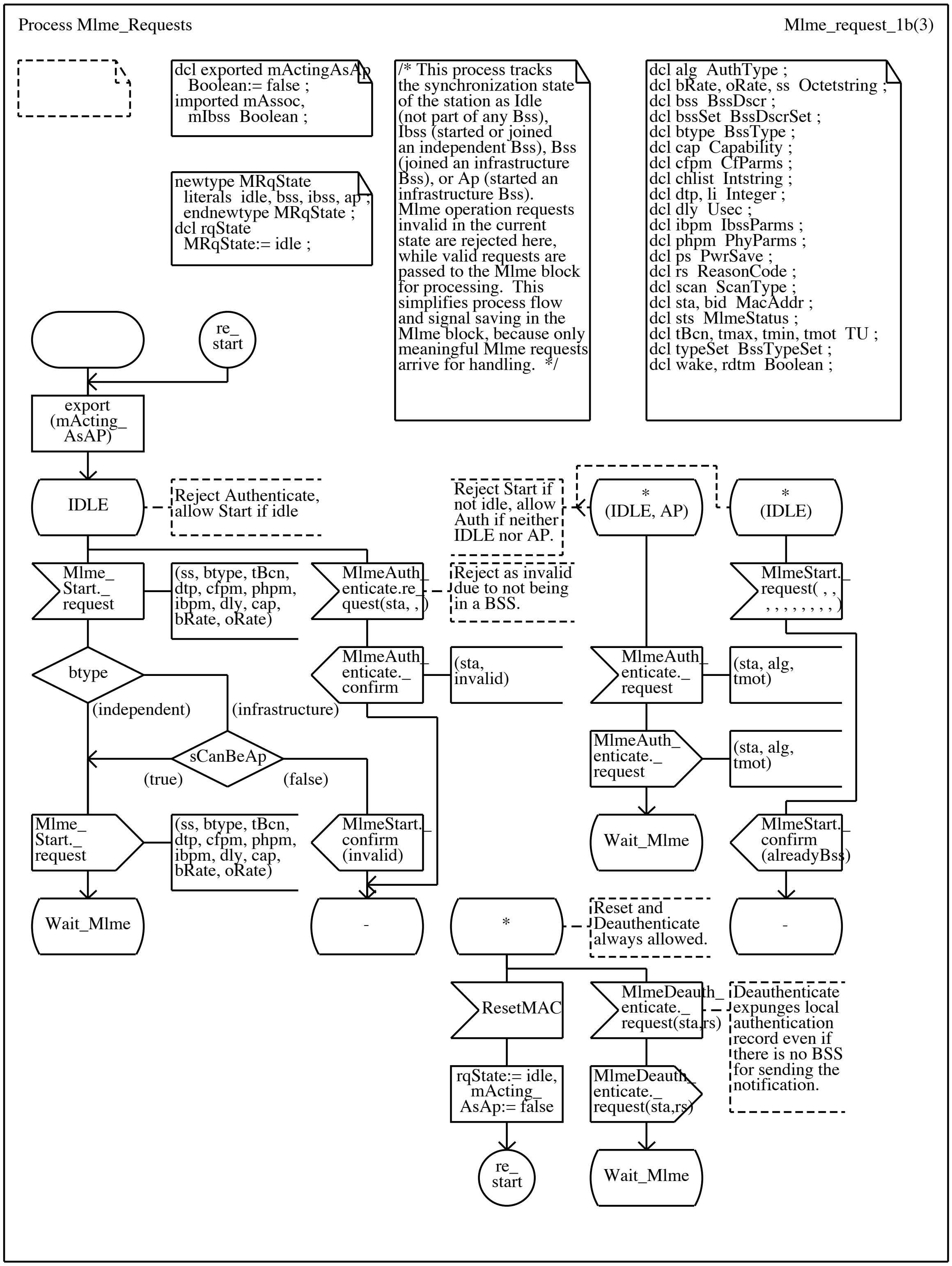
```
/* Import of {Read-Only} MIB counter
   values exported from other processes */
imported
  aAckFailureCount,
  aFailedCount,
  aFcsErrorCount,
  aFrameDuplicateCount,
  aMulticastReceivedFrameCount,
  aMulticastTransmittedFrameCount,
  aMultipleRetryCount,
  aReceivedFragmentCount,
  aRetryCount,
  aRtsFailureCount,
  aRtsSuccessCount,
  aTransmittedFragmentCount,
  aWepExcludedCount,
  aWepIcvErrorCount,
  aWepUndecryptableCount  Counter32 ;
```

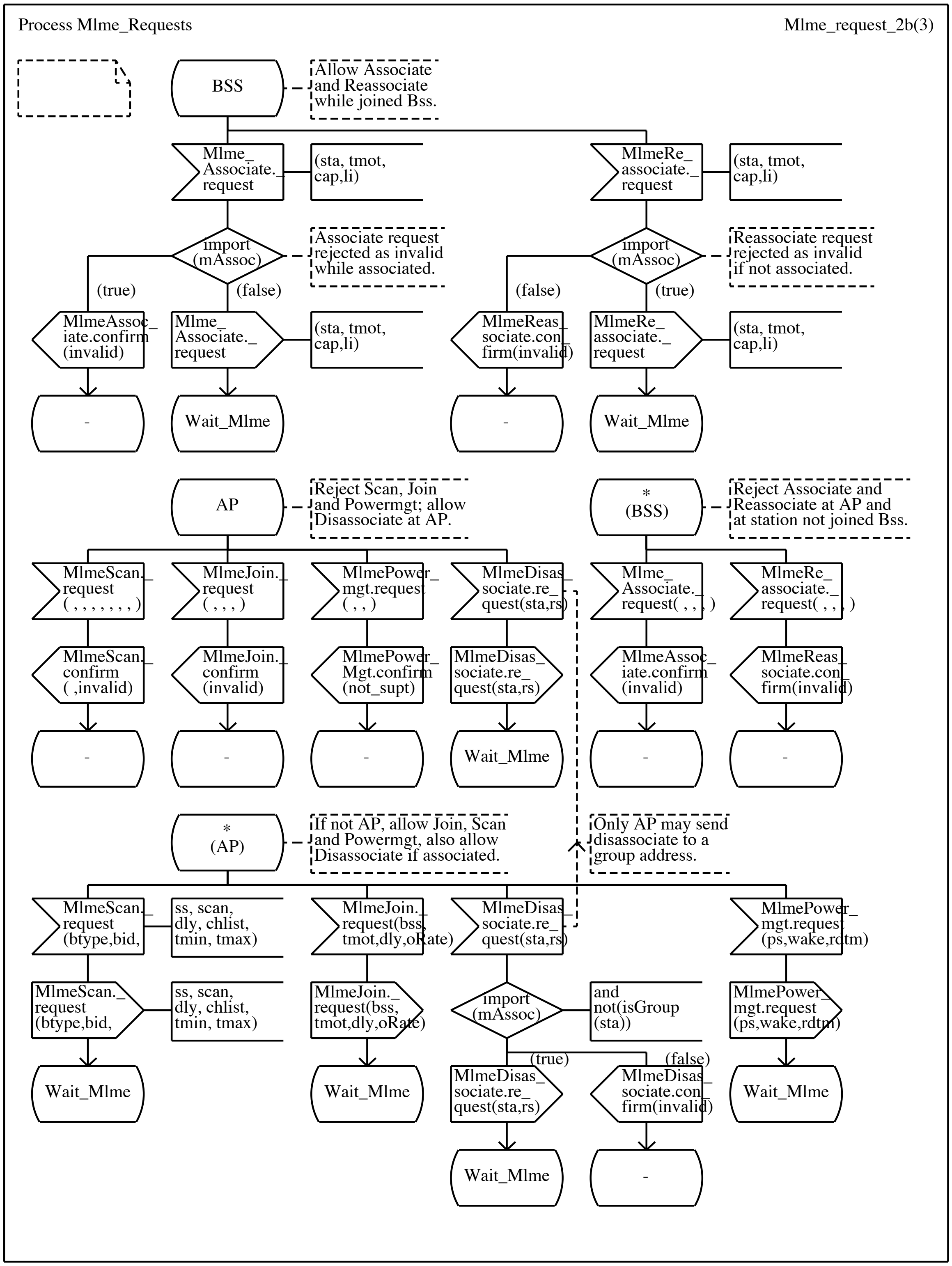
```
/* Declarations of MIB attributes exported from this process */

      /* Read-Write attributes */
dcl exported
  aAuthenticationType  AuthTypeSet:=
    incl(open_system, shared_key),
  aExcludeUnencrypted  Boolean:= false,
  aFragmentationThreshold  Integer:= 2346,
  aGroupAddresses  MacAddrSet:= empty,
  aLongRetryLimit  Integer:= 4,
  aMaxReceiveLifetime  TU:= 512,
  aMaxTransmitMsduLifetime  TU:= 512,
  aMediumOccupancyLimit  TU:= 100,
  aPrivacyInvoked  Boolean:= false,
  aReceiveDTIMs  Boolean:= true,
  aCfpPeriod  Integer:= 1,
  aCfpMaxDuration  TU:= 200,
  aAuthenticationResponseTimeout  TU:= 512,
  aRtsThreshold  Integer:= 3000,
  aShortRetryLimit  Integer:= 7,
  aWepDefaultKeyId  KeyIndex:= 0,
  aCurrentChannelNumber  Integer:= 0,
  aCurrentSet  Integer:= 0,
  aCurrentPattern  Integer:= 0,
  aCurrentIndex  Integer:= 0 ;

      /* Write-Only attributes */
dcl exported
  aWepDefaultKeys  KeyVector:= nullKey,
  aWepKeyMappings
    KeyMapArray:= (. nullAddr, false, nullKey .) ;
```

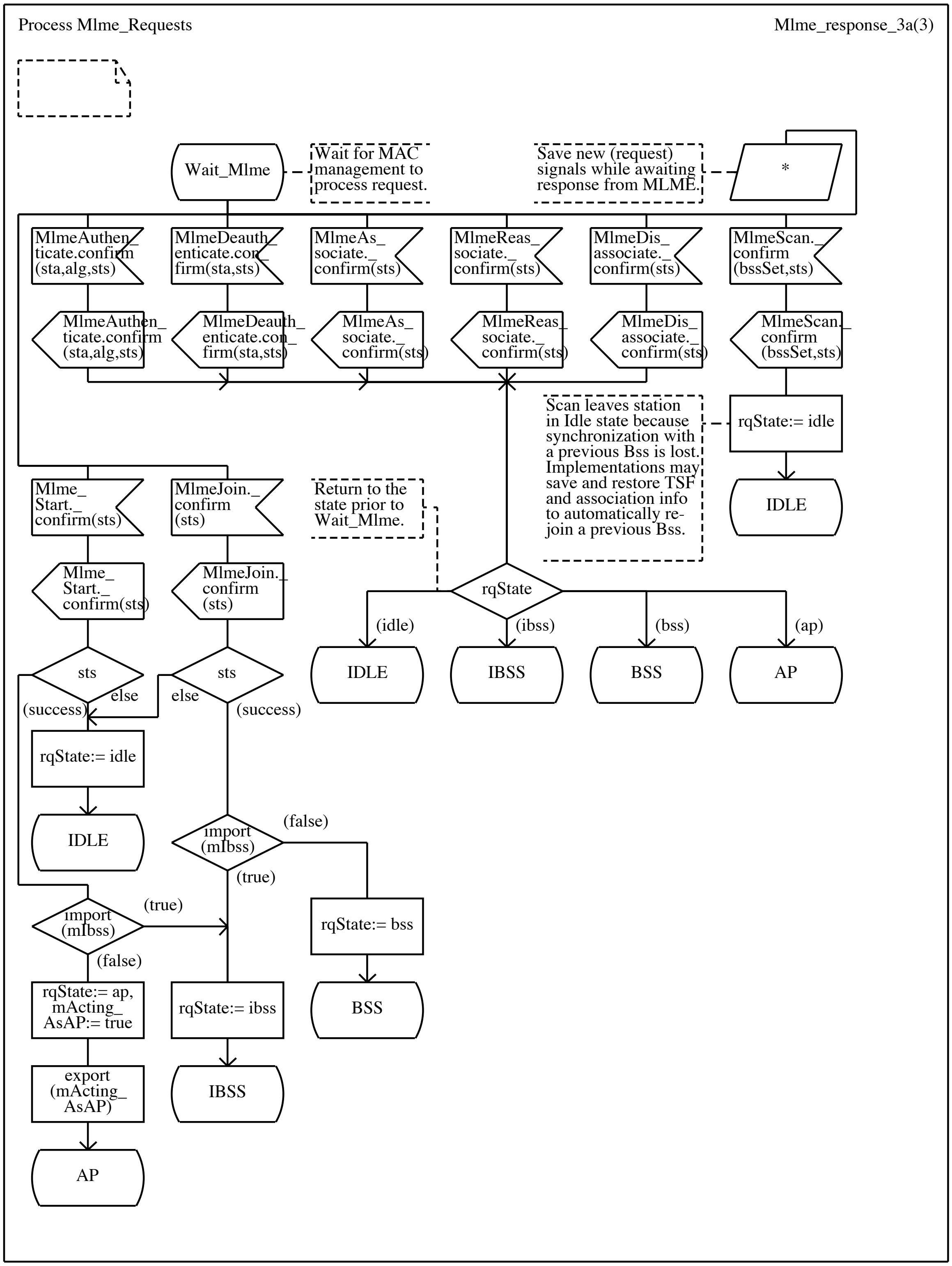
```
/* The following Read-Only attributes in the
   MAC MIB are defined as synonyms (named
   constants) rather than remote variables
   because they describe properties of the
   station which are static, at least during
   any single instance of MAC operation:
     aAuthenticationAlgorithms  AuthTypeSet,
     aCfPollable  Boolean,
     aMacAddress  MacAddr,
     aManufacturerID  Octetstring,
     aPrivacyOptionImplemented  Boolean,
     aProductID  Octetstring,
     aStationID  MacAddr,
     aWepKeyMappingLength  Integer ;

   In addition, all Read-Only attributes in the
   PHY MIB which are accessed by the MAC
   are defined as synonyms.
*/
```

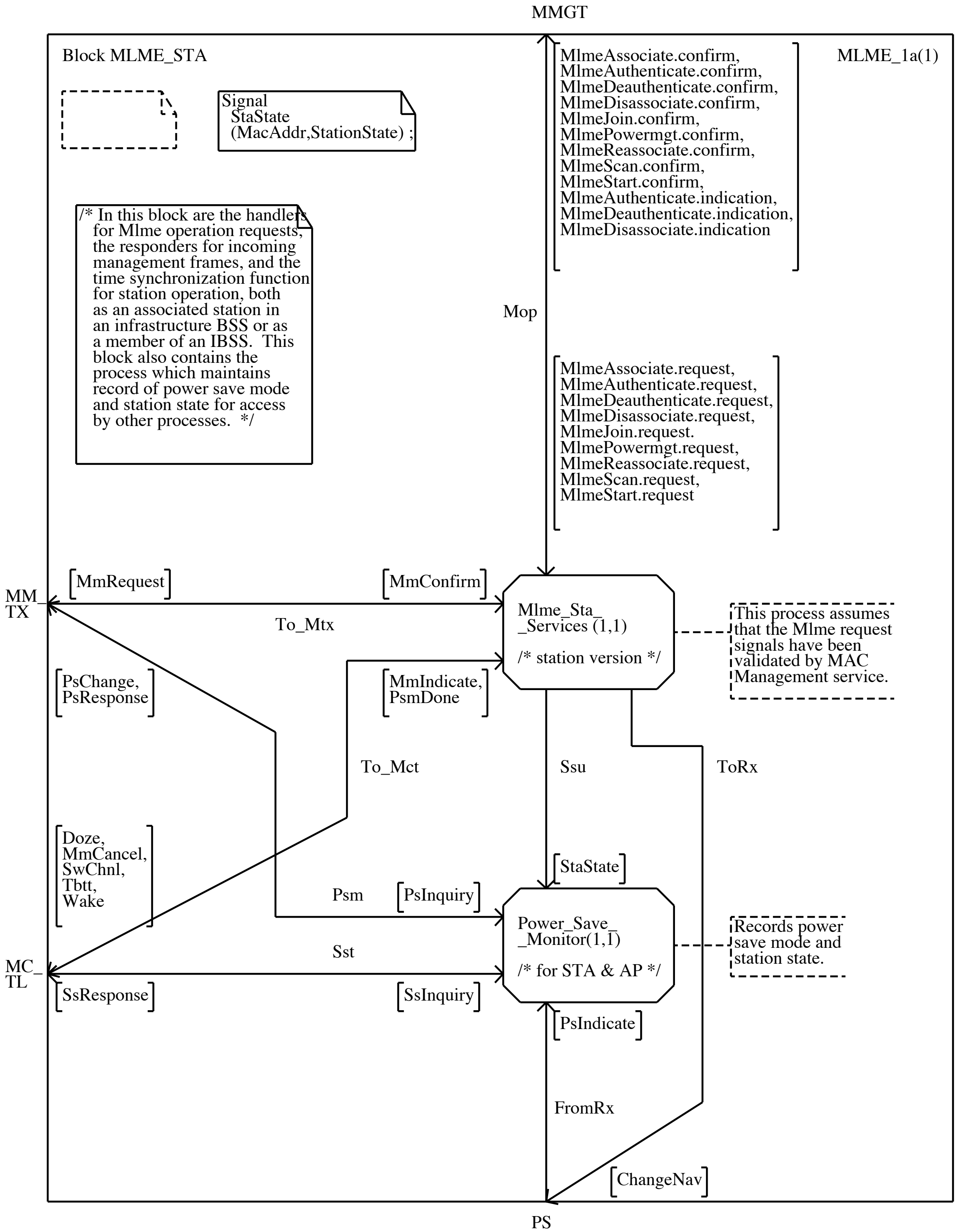
```
/* NOTE:
   The values listed for MAC MIB attributes are the
   specified default values for those attributes.
   The values listed for PHY MIB attributes are either
   the default values for the FH PHY, or arbitrary
   values within the specified range.  The specific
   values for PHY attributes in this SDL description
   of the MAC do not have normative significance.
*/
```

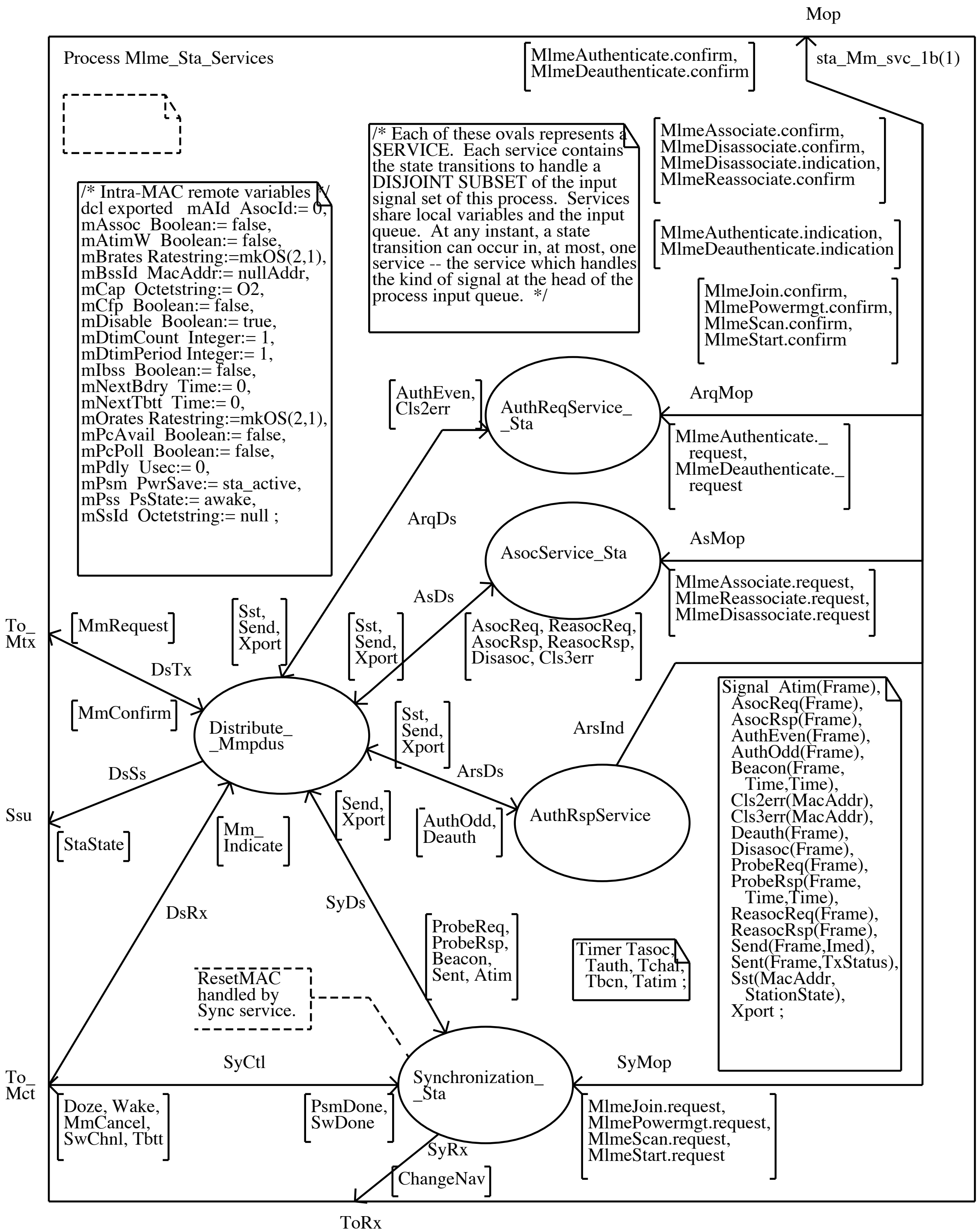

Process Mlme_Indications
Mlme_indication_1a(1)
dcl alg AuthType ;
dcl rsn ReasonCode ;
dcl sta MacAddr ;
Pass_ Through_ Idle
This state machine passes indications through, unmodified, from MLME to the MLME SAP. MlmeAssociate.indication and MlmeReassociate.indication are only generated by MLME at APs.
MlmeAsso_ ciate.ind_ ication(sta)
MlmeAuthen_ ticate.ind_ ication(sta,alg)
MlmeDeauth_ enticate.ind_ ication(sta,rsn)
MlmeDisas_ sociate.ind_ ication(sta,rsn)
MlmeReas_ sociate.ind_ ication(sta)
MlmeAsso_ ciate.ind_ ication(sta)
MlmeAuthen_ ticate.ind_ ication(sta,alg)
MlmeDeauth_ enticate.ind_ ication(sta)
MlmeDisas_ sociate.ind_ ication(sta)
MlmeReas_ sociate.ind_ ication(sta)
-
-
-
-
-

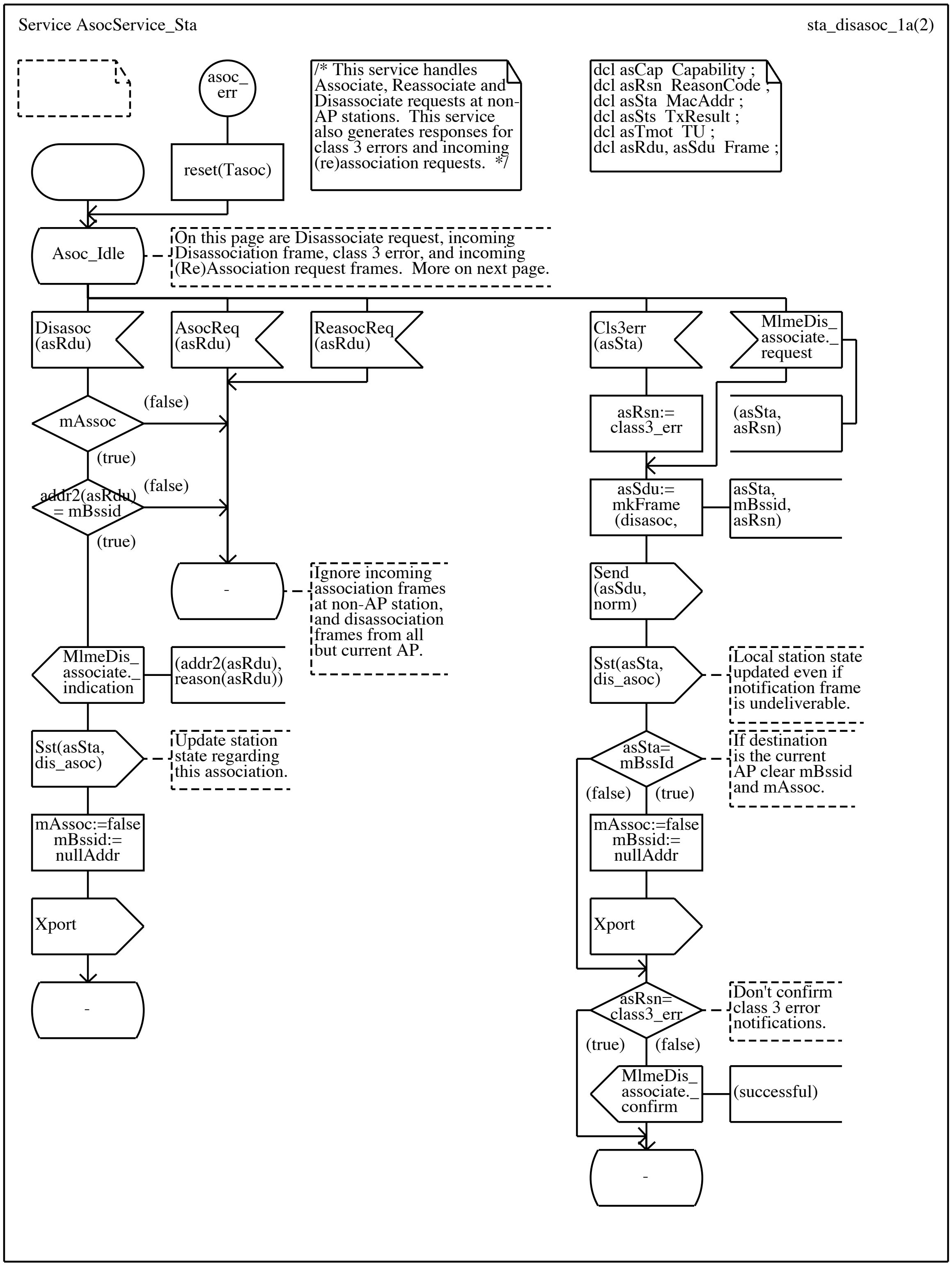

Process Mlme_Requests
Mlme_request_1b(3)
dcl exported mActingAsAp Boolean:= false ; imported mAssoc, mIbss Boolean ;
newtype MRqState literals idle, bss, ibss, ap ; endnewtype MRqState ; dcl rqState MRqState:= idle ;
/* This process tracks the synchronization state of the station as Idle (not part of any Bss), Ibss (started or joined an independent Bss), Bss (joined an infrastructure Bss), or Ap (started an infrastructure Bss). Mlme operation requests invalid in the current state are rejected here, while valid requests are passed to the Mlme block for processing. This simplifies process flow and signal saving in the Mlme block, because only meaningful Mlme requests arrive for handling. */
dcl alg AuthType ;
dcl bRate, oRate, ss Octetstring ;
dcl bss BssDscr ;
dcl bssSet BssDscrSet ;
dcl btype BssType ;
dcl cap Capability ;
dcl cfpm CfParms ;
dcl chlist Intstring ;
dcl dtp, li Integer ;
dcl dly Usec ;
dcl ibpm IbssParms ;
dcl phpm PhyParms ;
dcl ps PwrSave ;
dcl rs ReasonCode ;
dcl scan ScanType ;
dcl sta, bid MacAddr ;
dcl sts MlmeStatus ;
dcl tBcn, tmax, tmin, tmot TU ;
dcl typeSet BssTypeSet ;
dcl wake, rdtm Boolean ;
re_ start
export (mActing_ AsAP)
IDLE
Reject Authenticate, allow Start if idle
Mlme_ Start._ request
(ss, btype, tBcn, dtp, cfpm, phpm, ibpm, dly, cap, bRate, oRate)
MlmeAuth_ enticate.re_ quest(sta, ,)
Reject as invalid due to not being in a BSS.
Reject Start if not idle, allow Auth if neither IDLE nor AP.
* (IDLE, AP)
* (IDLE)
MlmeStart._ request(, , , , , , , , , ,)
btype
(independent)
(infrastructure)
MlmeAuth_ enticate._ confirm
(sta, invalid)
MlmeAuth_ enticate._ request
(sta, alg, tmot)
sCanBeAp
(true)
(false)
MlmeAuth_ enticate._ request
(sta, alg, tmot)
Mlme_ Start._ request
(ss, btype, tBcn, dtp, cfpm, phpm, ibpm, dly, cap, bRate, oRate)
MlmeStart._ confirm (invalid)
Wait_Mlme
MlmeStart._ confirm (alreadyBss)
Wait_Mlme
-
*
Reset and Deauthenticate always allowed.
-
ResetMAC
MlmeDeauth_ enticate._ request(sta,rs)
Deauthenticate expunges local authentication record even if there is no BSS for sending the notification.
rqState:= idle, mActing_ AsAp:= false
MlmeDeauth_ enticate._ request(sta,rs)
re_ start
Wait_Mlme

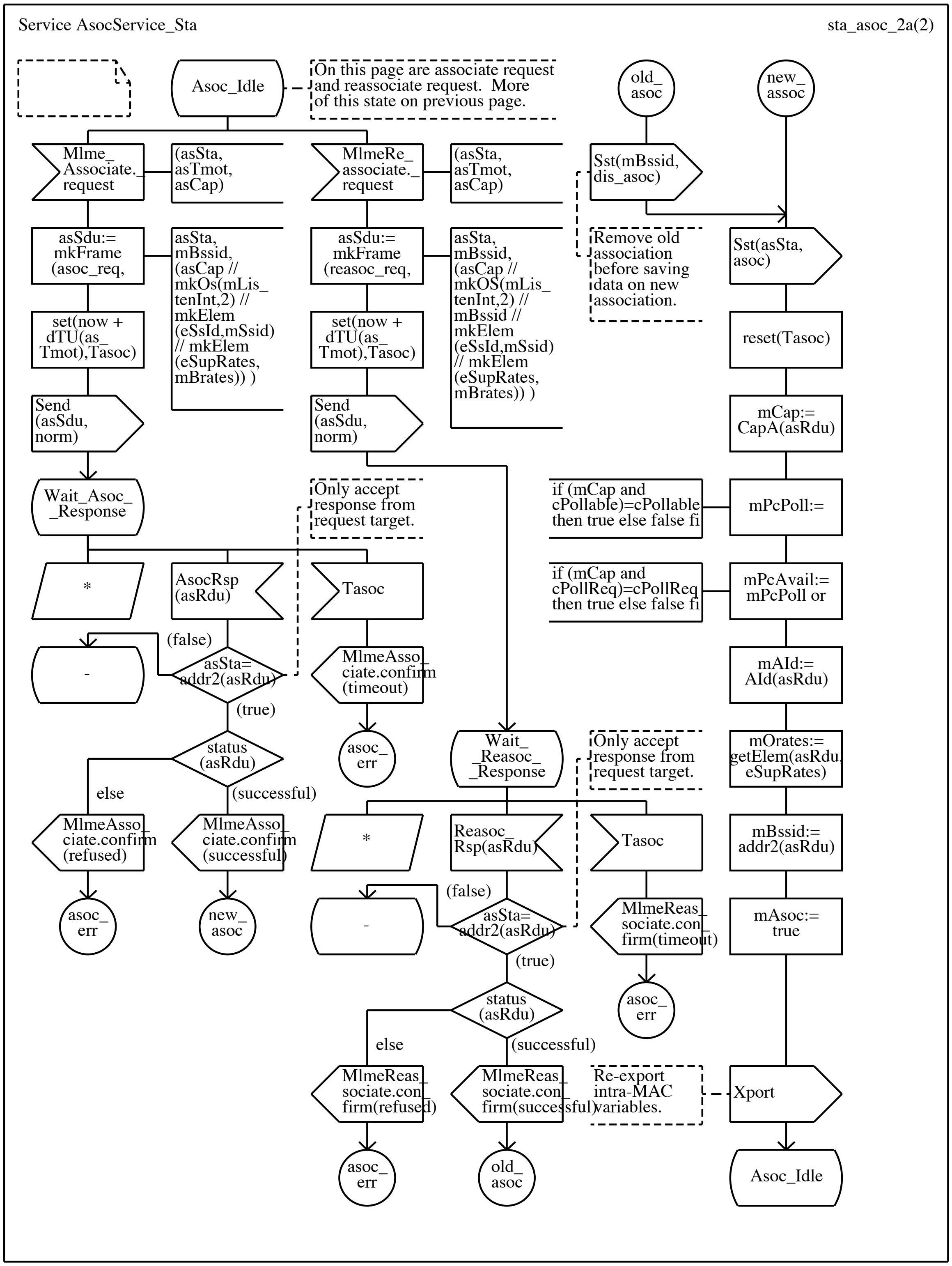

Process Mlme_Requests
Mlme_request_2b(3)
BSS
Allow Associate and Reassociate while joined Bss.
Mlme_ Associate._ request
(sta, tmot, cap,li)
MlmeRe_ associate._ request
(sta, tmot, cap,li)
import (mAssoc)
Associate request rejected as invalid while associated.
(true)
(false)
MlmeAssoc_ iate.confirm (invalid)
Mlme_ Associate._ request
(sta, tmot, cap,li)
-
Wait_Mlme
import (mAssoc)
Reassociate request rejected as invalid if not associated.
(false)
(true)
MlmeReas_ sociate.con_ firm(invalid)
MlmeRe_ associate._ request
(sta, tmot, cap,li)
-
Wait_Mlme
AP
Reject Scan, Join and Powermgt; allow Disassociate at AP.
MlmeScan._ request (, , , , , , ,)
MlmeJoin._ request (, , ,)
MlmePower_ mgt.request (, ,)
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmeScan._ confirm (,invalid)
MlmeJoin._ confirm (invalid)
MlmePower_ Mgt.confirm (not_supt)
MlmeDisas_ sociate.re_ quest(sta,rs)
-
-
-
Wait_Mlme
* (BSS)
Reject Associate and Reassociate at AP and at station not joined Bss.
Mlme_ Associate._ request(, , ,)
MlmeRe_ associate._ request(, , ,)
MlmeAssoc_ iate.confirm (invalid)
MlmeReas_ sociate.con_ firm(invalid)
-
-
* (AP)
If not AP, allow Join, Scan and Powermgt, also allow Disassociate if associated.
Only AP may send disassociate to a group address.
MlmeScan._ request (btype,bid,
ss, scan, dly, chlist, tmin, tmax)
MlmeJoin._ request(bss, tmot,dly,oRate)
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmePower_ mgt.request (ps,wake,rdtm)
MlmeScan._ request (btype,bid,
ss, scan, dly, chlist, tmin, tmax)
MlmeJoin._ request(bss, tmot,dly,oRate)
import (mAssoc)
and not(isGroup (sta))
MlmePower_ mgt.request (ps,wake,rdtm)
Wait_Mlme
Wait_Mlme
(true)
(false)
MlmeDisas_ sociate.re_ quest(sta,rs)
MlmeDisas_ sociate.con_ firm(invalid)
Wait_Mlme
Wait_Mlme
-

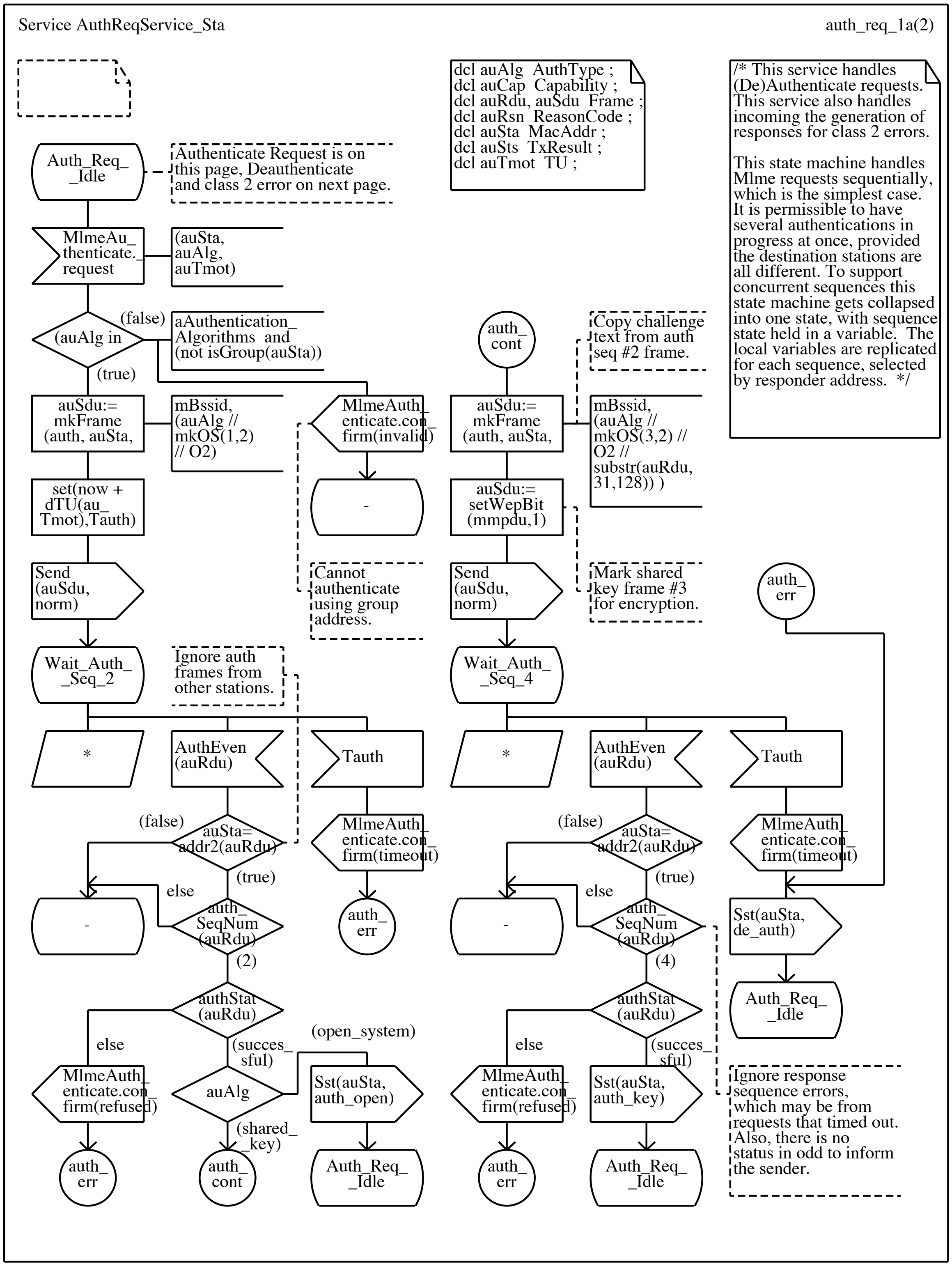

Process Mlme_Requests
Mlme_response_3a(3)
Wait_Mlme
Wait for MAC management to process request.
Save new (request) signals while awaiting response from MLME.
*
MlmeAuthen_ticate.confirm (sta,alg,sts)
MlmeDeauth_enticate.con_firm(sta,sts)
MlmeAs_sociate._confirm(sts)
MlmeReas_sociate._confirm(sts)
MlmeDis_associate._confirm(sts)
MlmeScan._confirm (bssSet,sts)
Scan leaves station in Idle state because synchronization with a previous Bss is lost. Implementations may save and restore TSF and association info to automatically re-join a previous Bss.
rqState:= idle
IDLE
Mlme_Start._confirm(sts)
MlmeJoin._confirm (sts)
Return to the state prior to Wait_Mlme.
rqState
(idle)
(ibss)
(bss)
(ap)
IDLE
IBSS
BSS
AP
sts
else
(success)
rqState:= idle
IDLE
import (mIbss)
(false)
(true)
rqState:= bss
BSS
rqState:= ap, mActing_AsAP:= true
rqState:= ibss
export (mActing_AsAP)
IBSS
AP

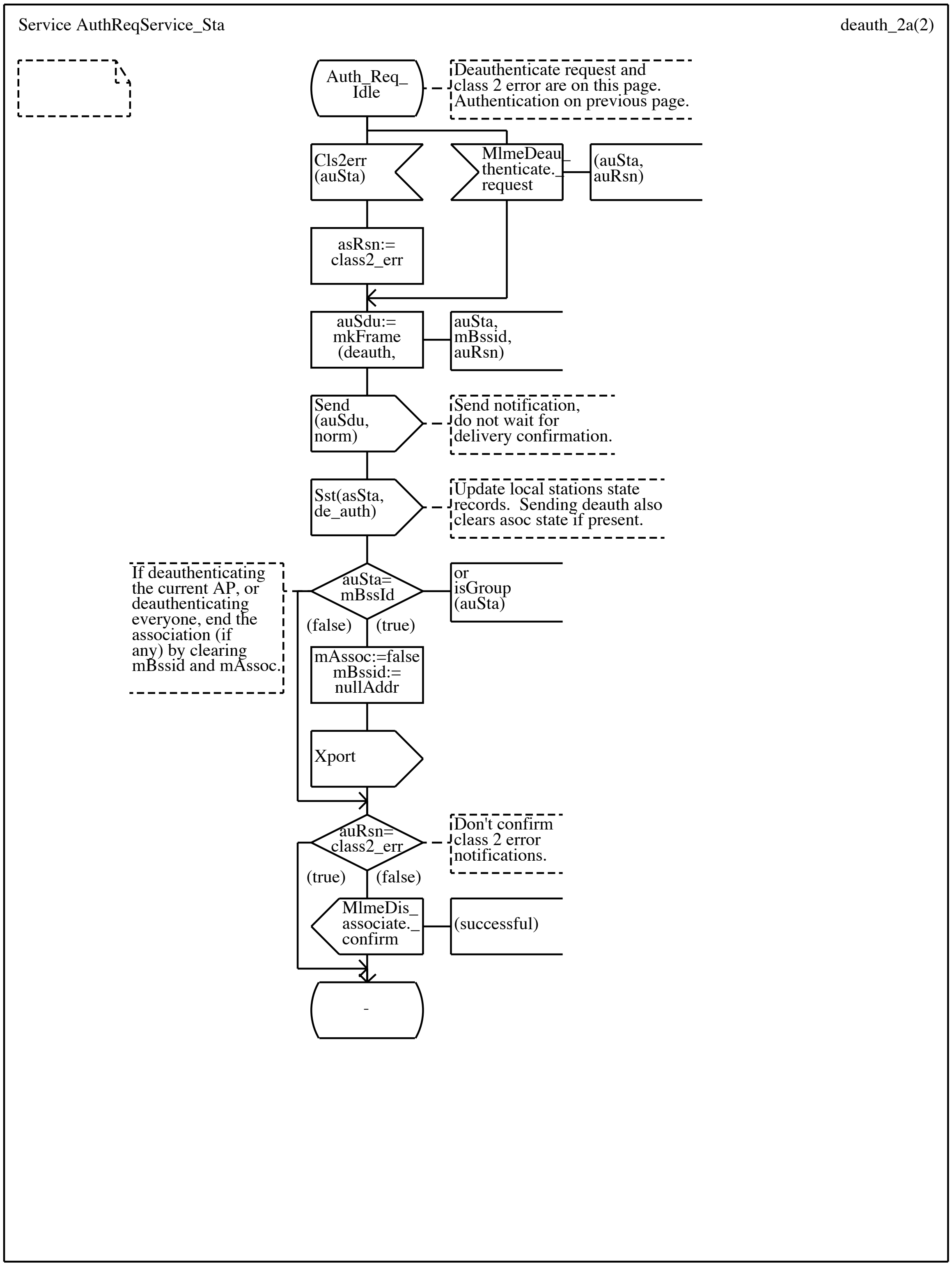

MMGT
Block MLME_STA
MLME_1a(1)
Signal
StaState
(MacAddr,StationState) ;
/* In this block are the handlers for Mlme operation requests, the responders for incoming management frames, and the time synchronization function for station operation, both as an associated station in an infrastructure BSS or as a member of an IBSS. This block also contains the process which maintains record of power save mode and station state for access by other processes. */
MlmeAssociate.confirm, MlmeAuthenticate.confirm, MlmeDeauthenticate.confirm, MlmeDisassociate.confirm, MlmeJoin.confirm, MlmePowermgt.confirm, MlmeReassociate.confirm, MlmeScan.confirm, MlmeStart.confirm, MlmeAuthenticate.indication, MlmeDeauthenticate.indication, MlmeDisassociate.indication
Mop
MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeJoin.request. MlmePowermgt.request, MlmeReassociate.request, MlmeScan.request, MlmeStart.request
MM_
TX
MmRequest
MmConfirm
To_Mtx
Mlme_Sta_
_Services (1,1)
/* station version */
This process assumes that the Mlme request signals have been validated by MAC Management service.
PsChange, PsResponse
MmIndicate, PsmDone
To_Mct
Ssu
ToRx
Doze, MmCancel, SwChnl, Tbtt, Wake
StaState
Psm
PsInquiry
Power_Save_
_Monitor(1,1)
/* for STA & AP */
Records power save mode and station state.
Sst
MC_
TL
SsResponse
SsInquiry
PsIndicate
FromRx
ChangeNav
PS

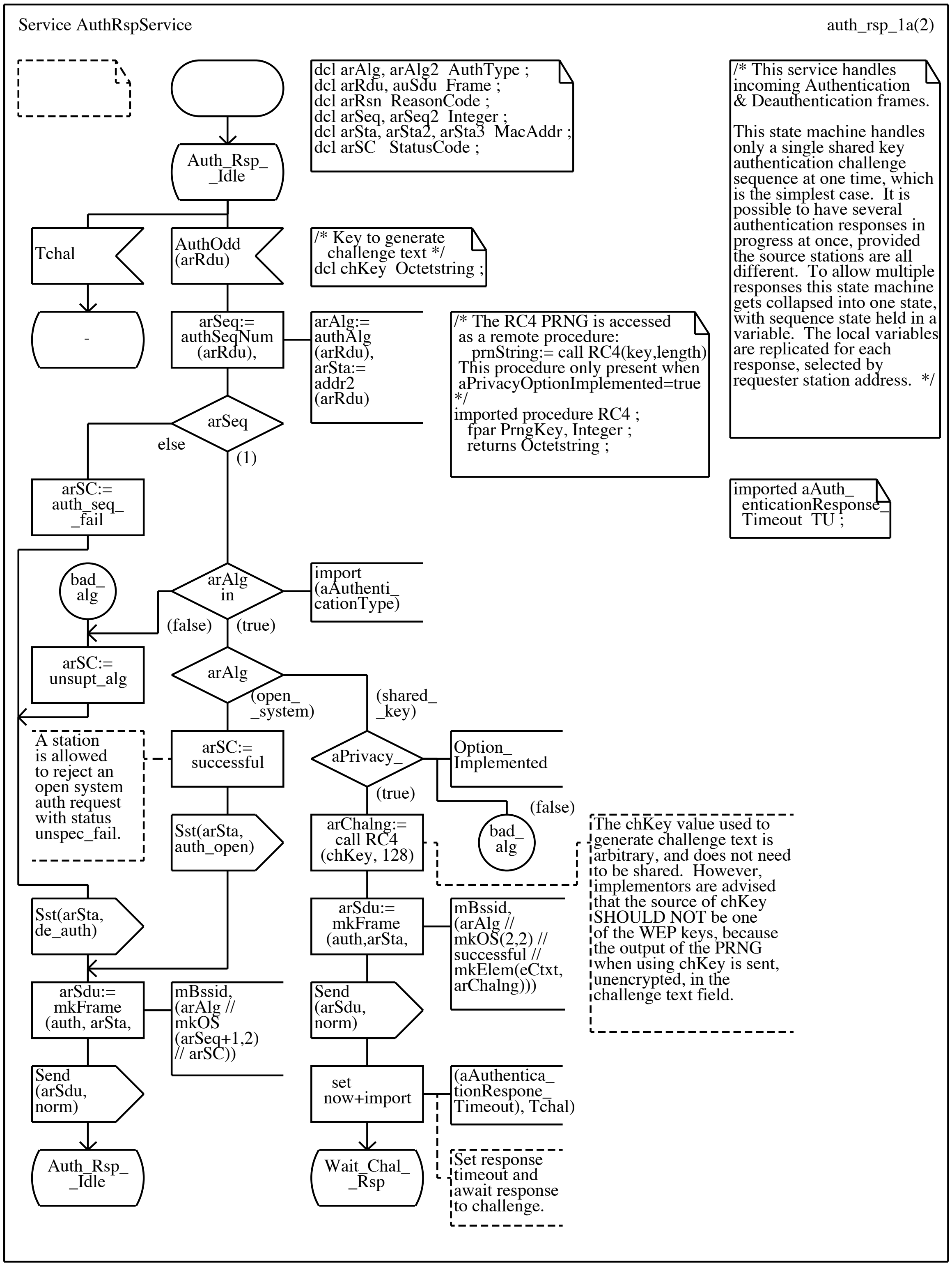

Mop
Process Mlme_Sta_Services
MlmeAuthenticate.confirm,
MlmeDeauthenticate.confirm
sta_Mm_svc_1b(1)
/* Each of these ovals represents a SERVICE. Each service contains the state transitions to handle a DISJOINT SUBSET of the input signal set of this process. Services share local variables and the input queue. At any instant, a state transition can occur in, at most, one service -- the service which handles the kind of signal at the head of the process input queue. */
MlmeAssociate.confirm,
MlmeDisassociate.confirm,
MlmeDisassociate.indication,
MlmeReassociate.confirm
MlmeAuthenticate.indication,
MlmeDeauthenticate.indication
MlmeJoin.confirm,
MlmePowermgt.confirm,
MlmeScan.confirm,
MlmeStart.confirm
/* Intra-MAC remote variables */
dcl exported mAId AsocId:= 0,
mAssoc Boolean:= false,
mAtimW Boolean:= false,
mBrates Ratestring:=mkOS(2,1),
mBssId MacAddr:= nullAddr,
mCap Octetstring:= O2,
mCfp Boolean:= false,
mDisable Boolean:= true,
mDtimCount Integer:= 1,
mDtimPeriod Integer:= 1,
mIbss Boolean:= false,
mNextBdry Time:= 0,
mNextTbtt Time:= 0,
mOrates Ratestring:=mkOS(2,1),
mPcAvail Boolean:= false,
mPcPoll Boolean:= false,
mPdly Usec:= 0,
mPsm PwrSave:= sta_active,
mPss PsState:= awake,
mSsId Octetstring:= null ;
AuthEven,
Cls2err
AuthReqService_
_Sta
ArqMop
MlmeAuthenticate._
request,
MlmeDeauthenticate._
request
ArqDs
AsocService_Sta
AsMop
MlmeAssociate.request,
MlmeReassociate.request,
MlmeDisassociate.request
AsDs
To_
Mtx
MmRequest
Sst,
Send,
Xport
Sst,
Send,
Xport
AsocReq, ReasocReq,
AsocRsp, ReasocRsp,
Disasoc, Cls3err
DsTx
MmConfirm
Distribute_
_Mmpdus
Sst,
Send,
Xport
ArsInd
Signal Atim(Frame),
AsocReq(Frame),
AsocRsp(Frame),
AuthEven(Frame),
AuthOdd(Frame),
Beacon(Frame,
Time,Time),
Cls2err(MacAddr),
Cls3err(MacAddr),
Deauth(Frame),
Disasoc(Frame),
ProbeReq(Frame),
ProbeRsp(Frame,
Time,Time),
ReasocReq(Frame),
ReasocRsp(Frame),
Send(Frame,Imed),
Sent(Frame,TxStatus),
Sst(MacAddr,
StationState),
Xport ;
DsSs
ArsDs
Send,
Xport
AuthOdd,
Deauth
AuthRspService
Ssu
StaState
Mm_
Indicate
DsRx
SyDs
ProbeReq,
ProbeRsp,
Beacon,
Sent, Atim
Timer Tasoc,
Tauth, Tchal,
Tbcn, Tatim ;
ResetMAC
handled by
Sync service.
SyCtl
SyMop
To_
Mct
Synchronization_
_Sta
Doze, Wake,
MmCancel,
SwChnl, Tbtt
PsmDone,
SwDone
MlmeJoin.request,
MlmePowermgt.request,
MlmeScan.request,
MlmeStart.request
SyRx
ChangeNav
ToRx

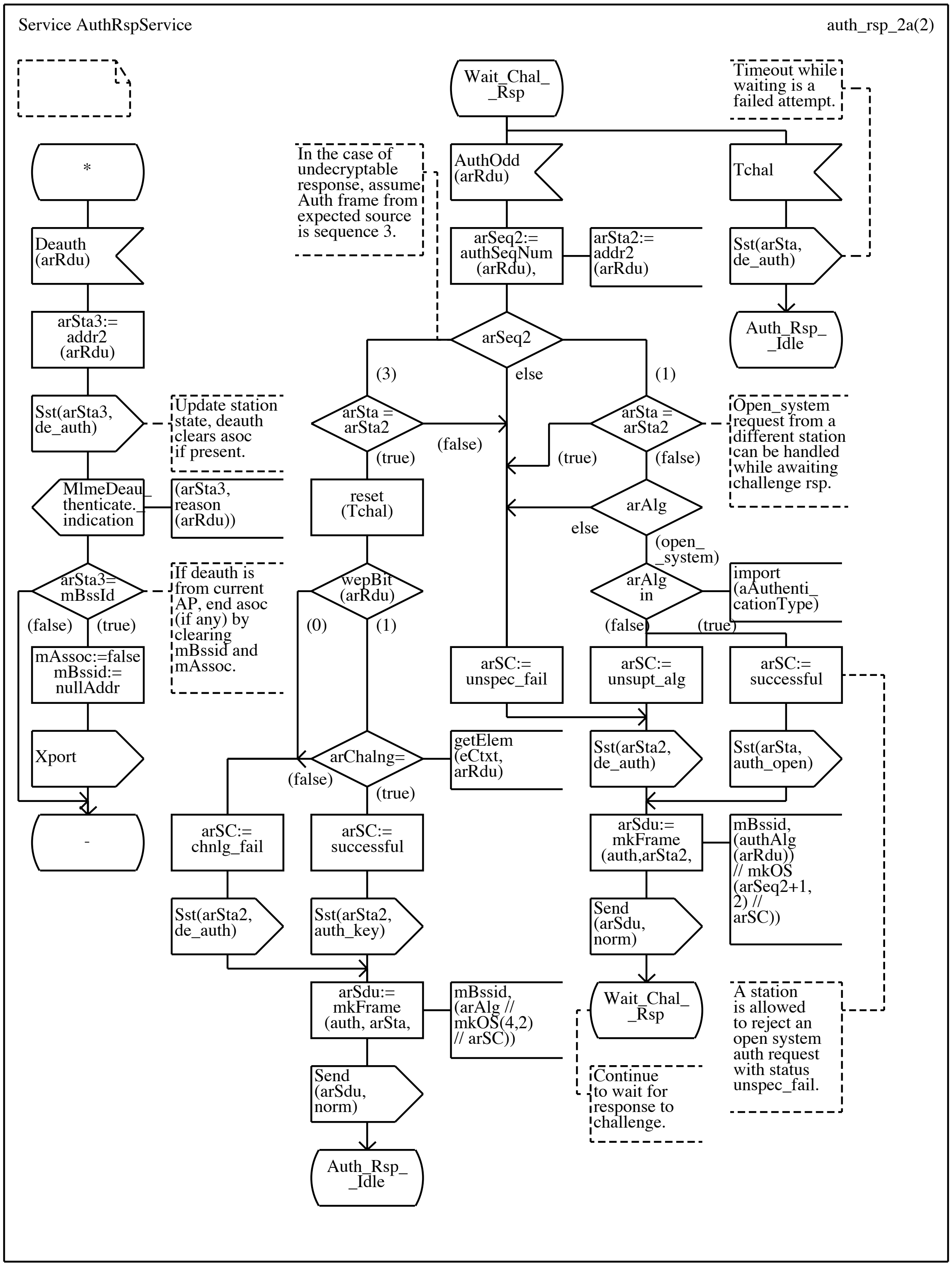

Service AsocService_Sta
sta_disasoc_1a(2)
asoc_ err
reset(Tasoc)
/* This service handles Associate, Reassociate and Disassociate requests at non-AP stations. This service also generates responses for class 3 errors and incoming (re)association requests. */
dcl asCap Capability ;
dcl asRsn ReasonCode ;
dcl asSta MacAddr ;
dcl asSts TxResult ;
dcl asTmot TU ;
dcl asRdu, asSdu Frame ;
Asoc_Idle
On this page are Disassociate request, incoming Disassociation frame, class 3 error, and incoming (Re)Association request frames. More on next page.
Disasoc (asRdu)
AsocReq (asRdu)
ReasocReq (asRdu)
Cls3err (asSta)
MlmeDis_ associate._ request
(asSta, asRsn)
mAssoc
(false)
(true)
addr2(asRdu) = mBssid
(false)
(true)
asRsn:= class3_err
asSdu:= mkFrame (disasoc,
asSta, mBssid, asRsn)
Send (asSdu, norm)
-
Ignore incoming association frames at non-AP station, and disassociation frames from all but current AP.
MlmeDis_ associate._ indication
(addr2(asRdu), reason(asRdu))
Sst(asSta, dis_asoc)
Local station state updated even if notification frame is undeliverable.
Sst(asSta, dis_asoc)
Update station state regarding this association.
asSta= mBssId
If destination is the current AP clear mBssid and mAssoc.
(false)
(true)
mAssoc:=false mBssid:= nullAddr
mAssoc:=false mBssid:= nullAddr
Xport
Xport
-
asRsn= class3_err
Don't confirm class 3 error notifications.
(true)
(false)
MlmeDis_ associate._ confirm
(successful)
-

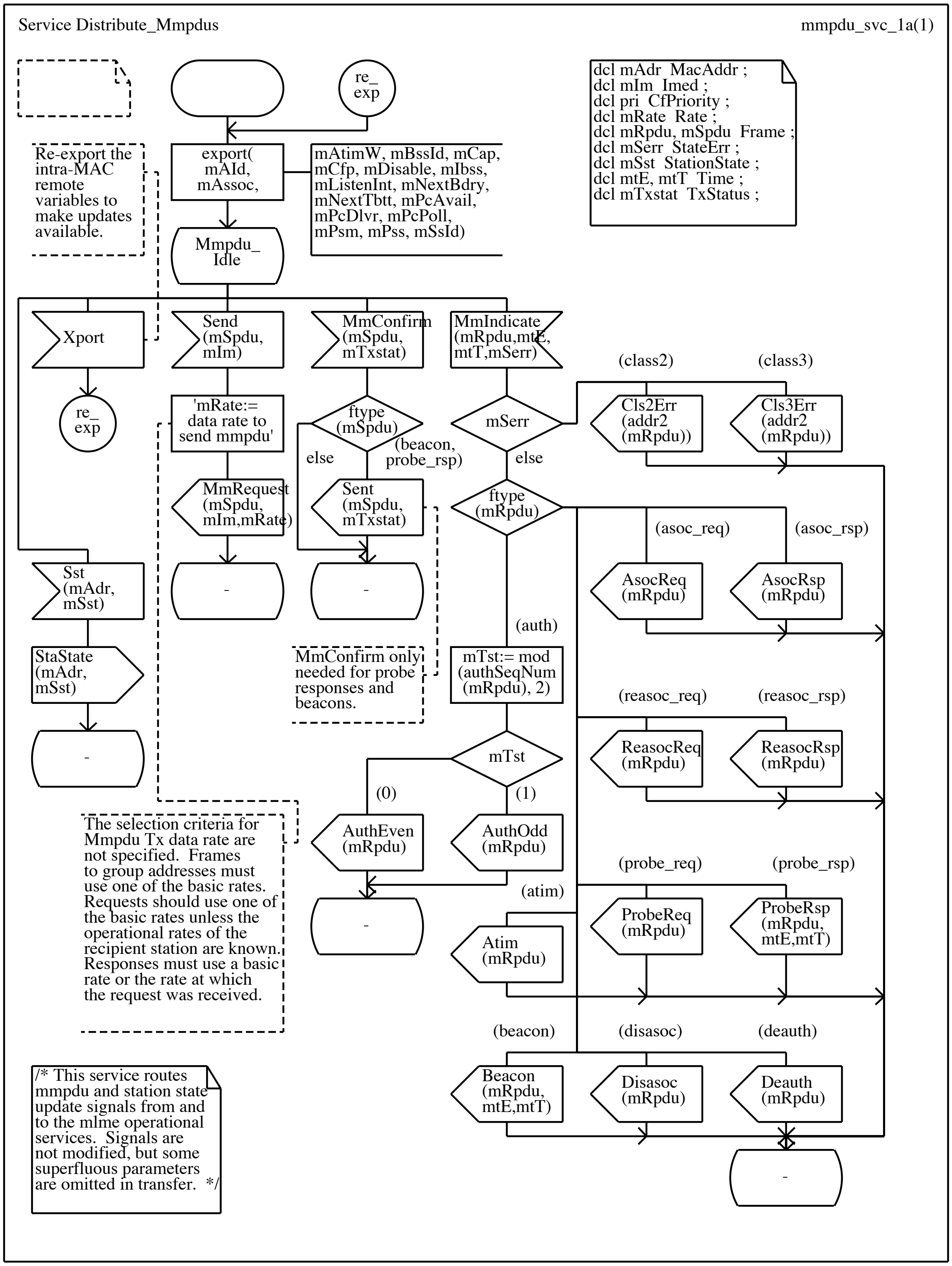

Service AsocService_Sta
sta_asoc_2a(2)
Asoc_Idle
On this page are associate request and reassociate request. More of this state on previous page.
old_ asoc
new_ asoc
Mlme_ Associate._ request
(asSta, asTmot, asCap)
MlmeRe_ associate._ request
(asSta, asTmot, asCap)
Sst(mBssid, dis_asoc)
asSdu:= mkFrame (asoc_req,
asSta, mBssid, (asCap // mkOs(mLis_ tenInt,2) // mkElem (eSsId,mSsid) // mkElem (eSupRates, mBrates)))
asSdu:= mkFrame (reasoc_req,
asSta, mBssid, (asCap // mkOS(mLis_ tenInt,2) // mBssid // mkElem (eSsId,mSsid) // mkElem (eSupRates, mBrates)))
Remove old association before saving data on new association.
Sst(asSta, asoc)
set(now + dTU(as_ Tmot),Tasoc)
reset(Tasoc)
Send (asSdu, norm)
mCap:= CapA(asRdu)
Wait_Asoc_ _Response
Only accept response from request target.
if (mCap and cPollable)=cPollable then true else false fi
mPcPoll:=
*
AsocRsp (asRdu)
Tasoc
if (mCap and cPollReq)=cPollReq then true else false fi
mPcAvail:= mPcPoll or
(false)
-
asSta= addr2(asRdu)
MlmeAsso_ ciate.confirm (timeout)
mAId:= AId(asRdu)
(true)
status (asRdu)
asoc_ err
Wait_ _Reasoc_ _Response
Only accept response from request target.
mOrates:= getElem(asRdu, eSupRates)
else
(successful)
MlmeAsso_ ciate.confirm (refused)
MlmeAsso_ ciate.confirm (successful)
Reasoc_ Rsp(asRdu)
mBssid:= addr2(asRdu)
asoc_ err
new_ asoc
mAsoc:= true
MlmeReas_ sociate.con_ firm(timeout)
MlmeReas_ sociate.con_ firm(refused)
MlmeReas_ sociate.con_ firm(successful)
Re-export intra-MAC variables.
Xport
old_ asoc
Asoc_Idle

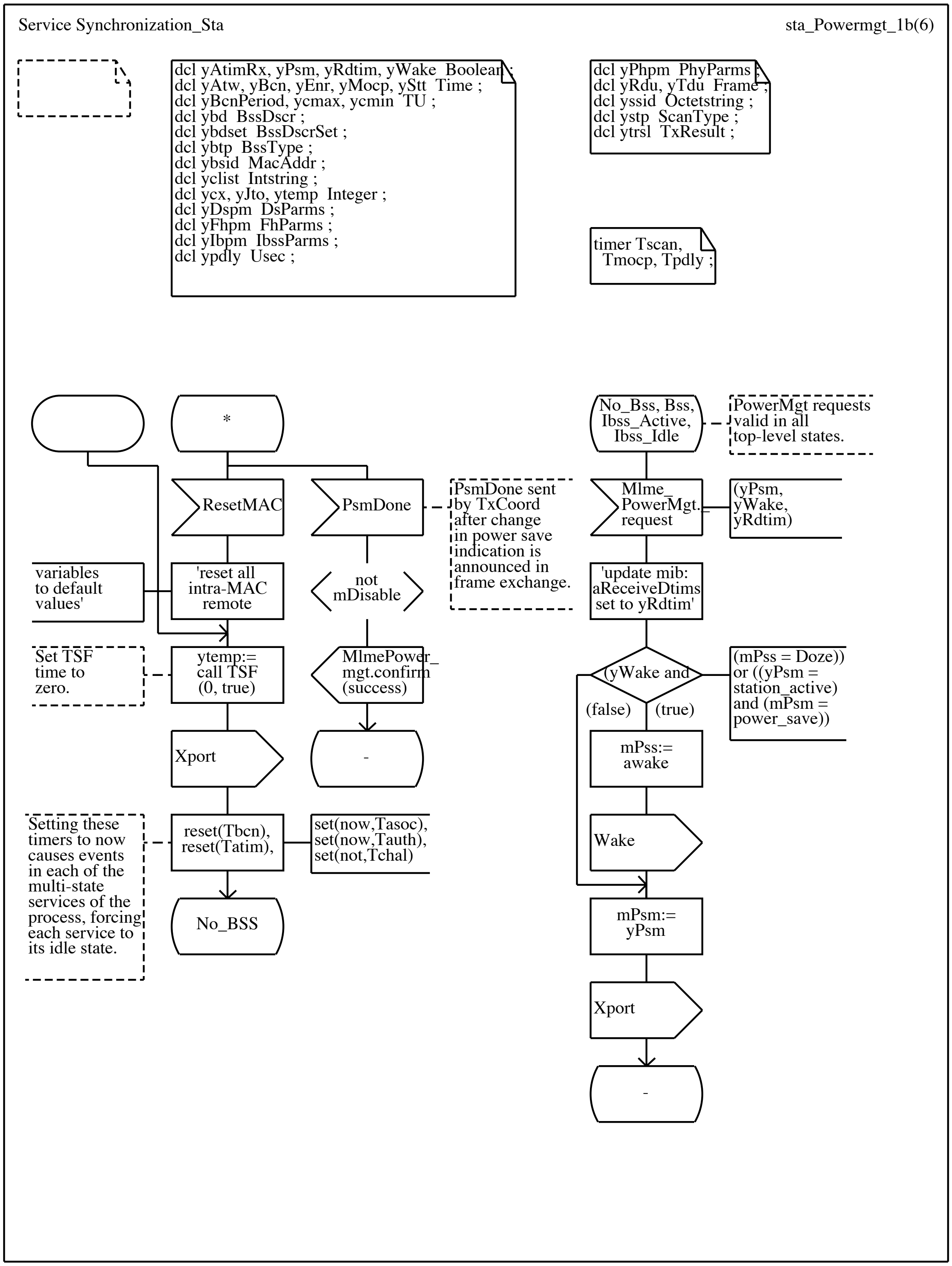

Service AuthReqService_Sta
auth_req_1a(2)
dcl auAlg AuthType ;
dcl auCap Capability ;
dcl auRdu, auSdu Frame ;
dcl auRsn ReasonCode ;
dcl auSta MacAddr ;
dcl auSts TxResult ;
dcl auTmot TU ;
/* This service handles (De)Authenticate requests. This service also handles incoming the generation of responses for class 2 errors.
This state machine handles Mlme requests sequentially, which is the simplest case. It is permissible to have several authentications in progress at once, provided the destination stations are all different. To support concurrent sequences this state machine gets collapsed into one state, with sequence state held in a variable. The local variables are replicated for each sequence, selected by responder address. */
Auth_Req_ _Idle
Authenticate Request is on this page, Deauthenticate and class 2 error on next page.
MlmeAu_ thenticate._ request
(auSta, auAlg, auTmot)
(auAlg in
(false)
aAuthentication_ Algorithms and (not isGroup(auSta))
(true)
auSdu:= mkFrame (auth, auSta,
mBssid, (auAlg // mkOS(1,2) // O2)
MlmeAuth_ enticate.con_ firm(invalid)
-
set(now + dTU(au_ Tmot),Tauth)
Send (auSdu, norm)
Cannot authenticate using group address.
Wait_Auth_ _Seq_2
Ignore auth frames from other stations.
*
AuthEven (auRdu)
Tauth
(false)
auSta= addr2(auRdu)
(true)
MlmeAuth_ enticate.con_ firm(timeout)
else
auth_ SeqNum (auRdu)
-
auth_ err
(2)
authStat (auRdu)
else
(succes_ sful)
(open_system)
MlmeAuth_ enticate.con_ firm(refused)
auAlg
Sst(auSta, auth_open)
(shared_ _key)
auth_ err
auth_ cont
Auth_Req_ _Idle
auth_ cont
Copy challenge text from auth seq #2 frame.
auSdu:= mkFrame (auth, auSta,
mBssid, (auAlg // mkOS(3,2) // O2 // substr(auRdu, 31,128)))
auSdu:= setWepBit (mmpdu,1)
Send (auSdu, norm)
Mark shared key frame #3 for encryption.
auth_ err
Wait_Auth_ _Seq_4
*
AuthEven (auRdu)
Tauth
(false)
auSta= addr2(auRdu)
(true)
MlmeAuth_ enticate.con_ firm(timeout)
else
auth_ SeqNum (auRdu)
-
(4)
Sst(auSta, de_auth)
authStat (auRdu)
Auth_Req_ _Idle
else
(succes_ sful)
MlmeAuth_ enticate.con_ firm(refused)
Sst(auSta, auth_key)
Ignore response sequence errors, which may be from requests that timed out. Also, there is no status in odd to inform the sender.
auth_ err
Auth_Req_ _Idle

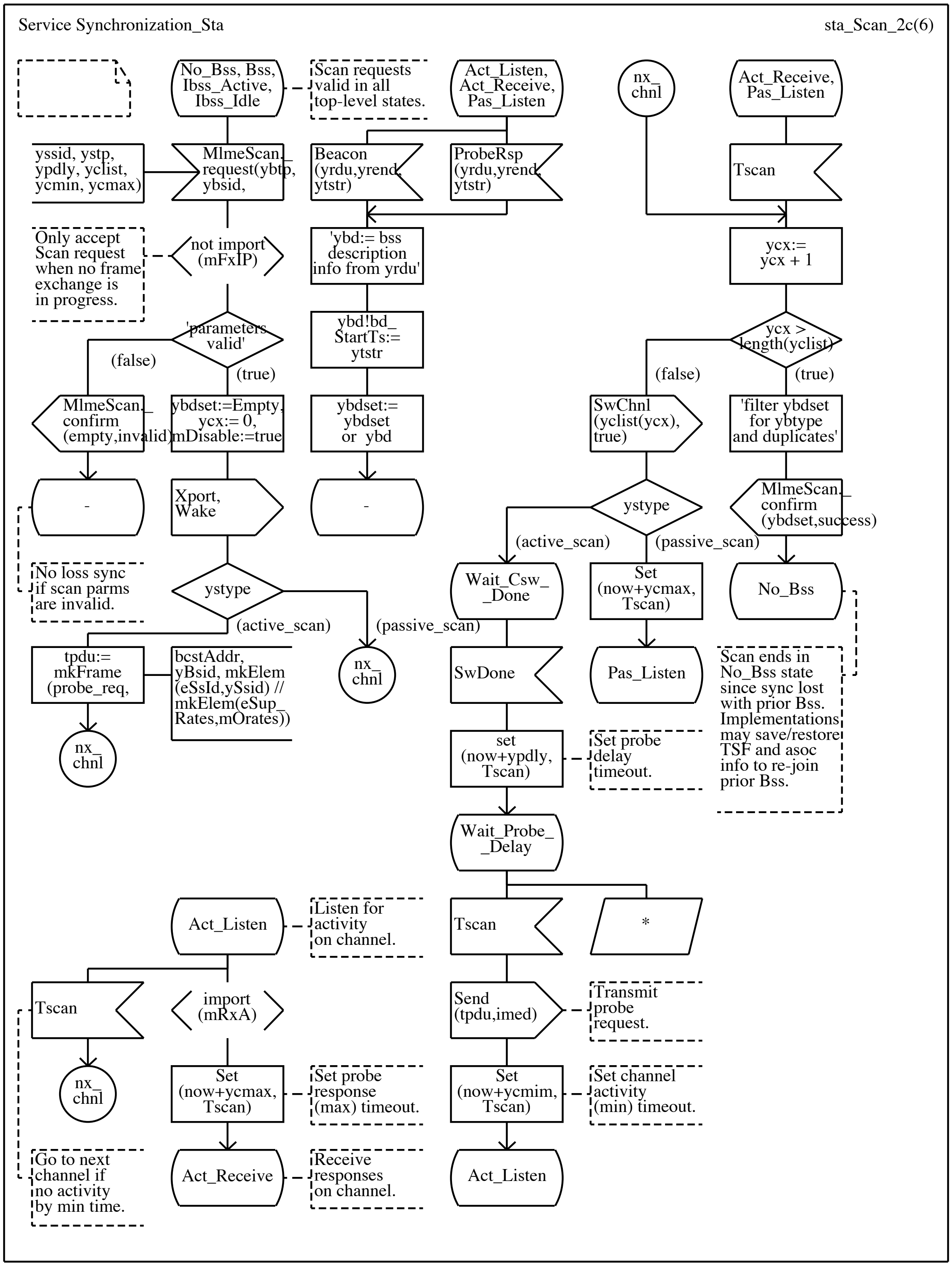

Service AuthReqService_Sta
deauth_2a(2)
Auth_Req_
Idle
Deauthenticate request and
class 2 error are on this page.
Authentication on previous page.
Cls2err
(auSta)
MlmeDeau_
thenticate._
request
(auSta,
auRsn)
asRsn:=
class2_err
auSdu:=
mkFrame
(deauth,
auSta,
mBssid,
auRsn)
Send
(auSdu,
norm)
Send notification,
do not wait for
delivery confirmation.
Sst(asSta,
de_auth)
Update local stations state
records. Sending deauth also
clears asoc state if present.
If deauthenticating
the current AP, or
deauthenticating
everyone, end the
association (if
any) by clearing
mBssid and mAssoc.
auSta=
mBssId
or
isGroup
(auSta)
(false)
(true)
mAssoc:=false
mBssid:=
nullAddr
Xport
auRsn=
class2_err
Don't confirm
class 2 error
notifications.
(true)
(false)
MlmeDis_
associate._
confirm
(successful)
-

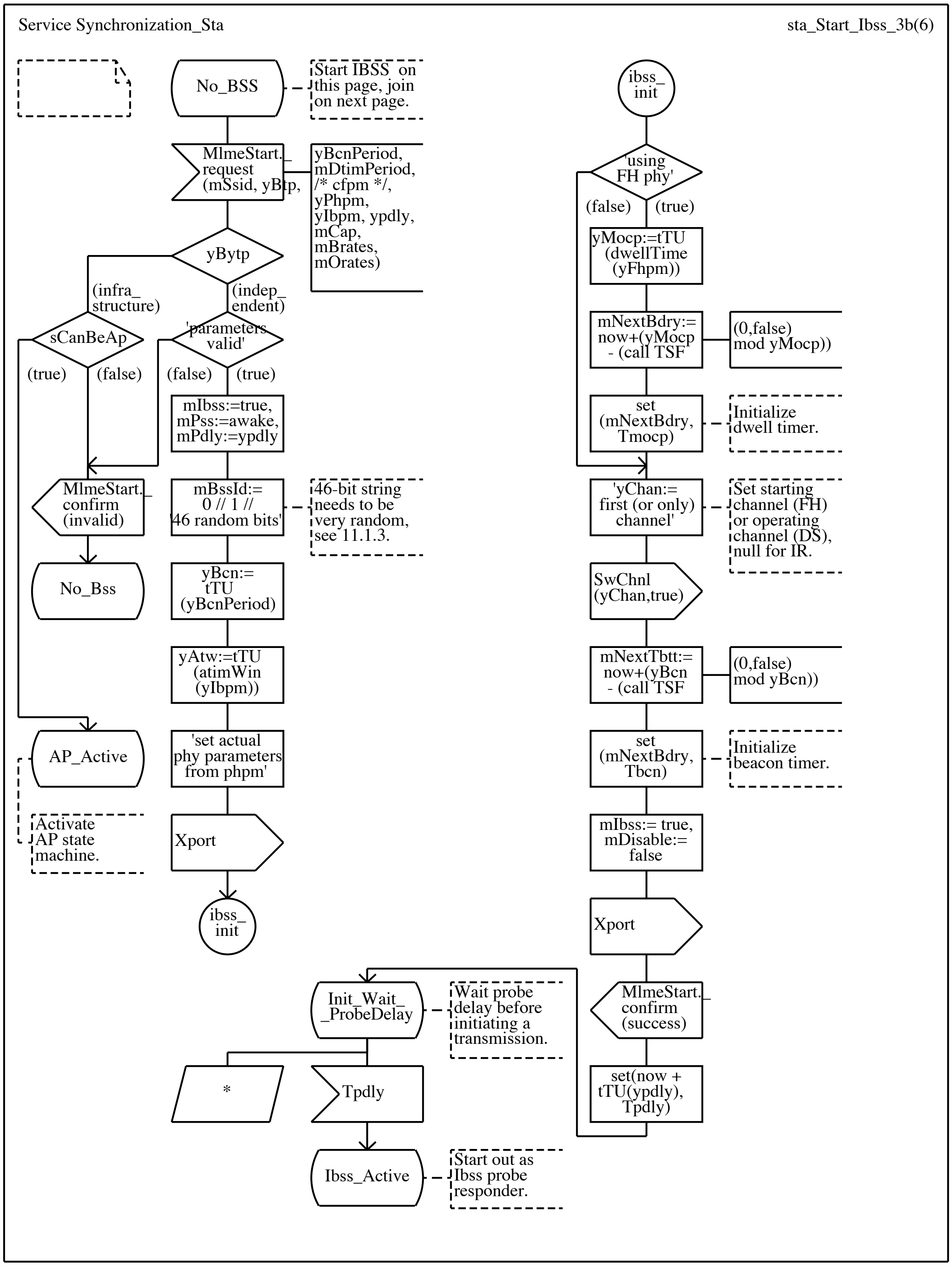

Service AuthRspService
auth_rsp_1a(2)
dcl arAlg, arAlg2 AuthType ;
dcl arRdu, auSdu Frame ;
dcl arRsn ReasonCode ;
dcl arSeq, arSeq2 Integer ;
dcl arSta, arSta2, arSta3 MacAddr ;
dcl arSC StatusCode ;
/* This service handles incoming Authentication & Deauthentication frames.
This state machine handles only a single shared key authentication challenge sequence at one time, which is the simplest case. It is possible to have several authentication responses in progress at once, provided the source stations are all different. To allow multiple responses this state machine gets collapsed into one state, with sequence state held in a variable. The local variables are replicated for each response, selected by requester station address. */
Auth_Rsp_ _Idle
Tchal
AuthOdd (arRdu)
/* Key to generate challenge text */
dcl chKey Octetstring ;
-
arSeq:= authSeqNum (arRdu),
arAlg:= authAlg (arRdu), arSta:= addr2 (arRdu)
/* The RC4 PRNG is accessed as a remote procedure:
prnString:= call RC4(key,length)
This procedure only present when aPrivacyOptionImplemented=true */
imported procedure RC4 ;
fpar PrngKey, Integer ;
returns Octetstring ;
arSeq
else
(1)
arSC:= auth_seq_ _fail
imported aAuth_ enticationResponse_ Timeout TU ;
bad_ alg
arAlg in
import (aAuthenti_ cationType)
(false)
(true)
arSC:= unsupt_alg
arAlg
(open_ _system)
(shared_ _key)
A station is allowed to reject an open system auth request with status unspec_fail.
arSC:= successful
aPrivacy_
Option_ Implemented
(true)
(false)
Sst(arSta, auth_open)
arChalng:= call RC4 (chKey, 128)
bad_ alg
The chKey value used to generate challenge text is arbitrary, and does not need to be shared. However, implementors are advised that the source of chKey SHOULD NOT be one of the WEP keys, because the output of the PRNG when using chKey is sent, unencrypted, in the challenge text field.
Sst(arSta, de_auth)
arSdu:= mkFrame (auth,arSta,
mBssid, (arAlg // mkOS(2,2) // successful // mkElem(eCtxt, arChalng)))
arSdu:= mkFrame (auth, arSta,
mBssid, (arAlg // mkOS (arSeq+1,2) // arSC))
Send (arSdu, norm)
Send (arSdu, norm)
set now+import
(aAuthentica_ tionRespone_ Timeout), Tchal)
Auth_Rsp_ _Idle
Wait_Chal_ _Rsp
Set response timeout and await response to challenge.

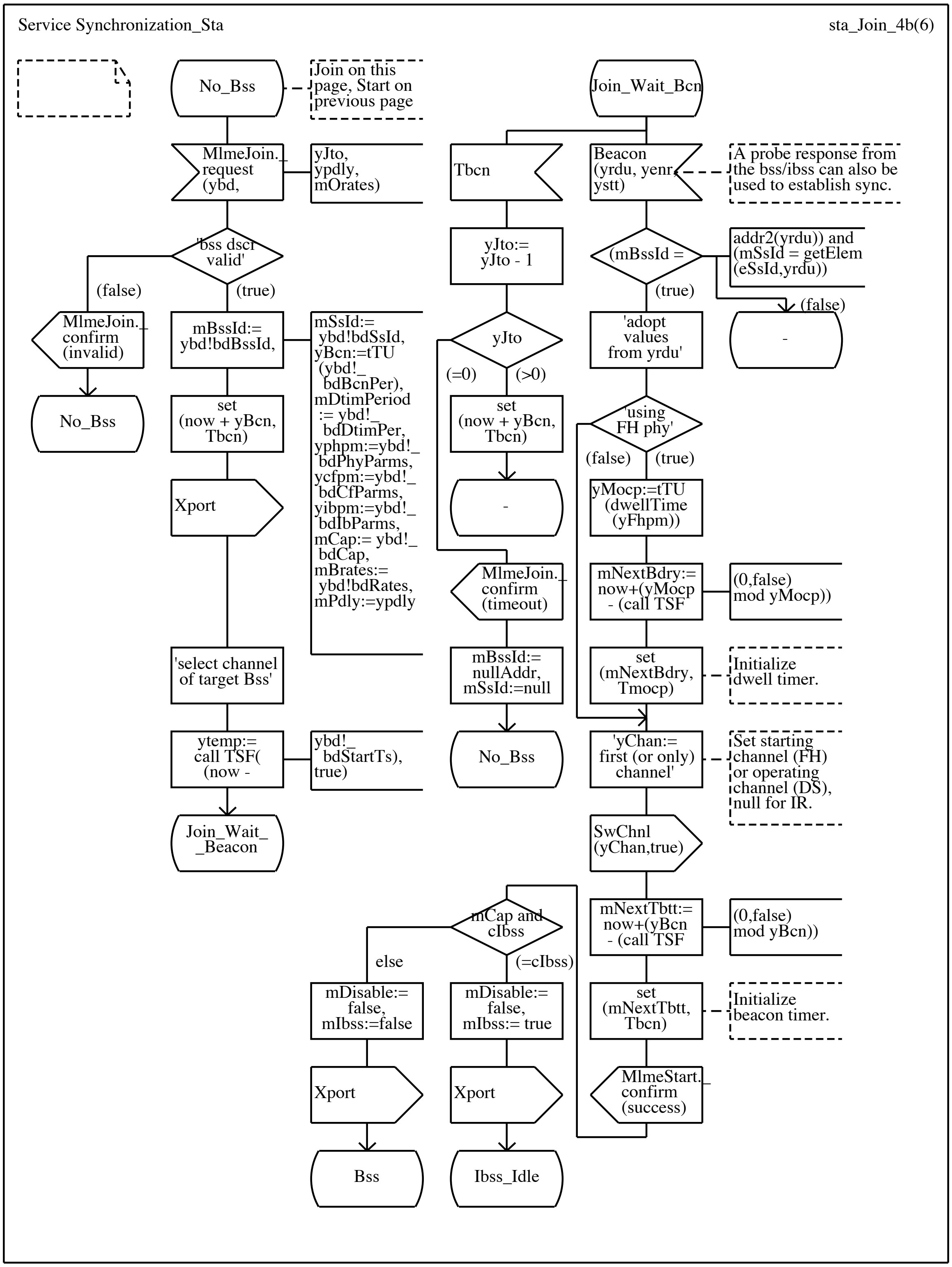

Service AuthRspService
auth_rsp_2a(2)
*
Deauth (arRdu)
arSta3:= addr2 (arRdu)
Sst(arSta3, de_auth)
Update station state, deauth clears asoc if present.
MlmeDeauthenticate.indication
(arSta3, reason (arRdu))
arSta3= mBssId
If deauth is from current AP, end asoc (if any) by clearing mBssid and mAssoc.
(false)
(true)
mAssoc:=false mBssid:= nullAddr
Xport
-
Wait_Chal_Rsp
Timeout while waiting is a failed attempt.
In the case of undecryptable response, assume Auth frame from expected source is sequence 3.
AuthOdd (arRdu)
Tchal
arSeq2:= authSeqNum (arRdu),
arSta2:= addr2 (arRdu)
Sst(arSta, de_auth)
Auth_Rsp_Idle
arSeq2
(3)
else
(1)
arSta = arSta2
(true)
(false)
reset (Tchal)
Open_system request from a different station can be handled while awaiting challenge rsp.
arAlg
(open_system)
wepBit (arRdu)
(0)
(1)
arAlg in
import (aAuthenticationType)
arSC:= unspec_fail
arSC:= unsupt_alg
arSC:= successful
arChalng=
getElem (eCtxt, arRdu)
Sst(arSta2, de_auth)
Sst(arSta, auth_open)
arSC:= chnlg_fail
arSdu:= mkFrame (auth,arSta2,
mBssid, (authAlg (arRdu)) // mkOS (arSeq2+1, 2) // arSC))
Sst(arSta2, auth_key)
Send (arSdu, norm)
arSdu:= mkFrame (auth, arSta,
mBssid, (arAlg // mkOS(4,2) // arSC))
A station is allowed to reject an open system auth request with status unspec_fail.
Continue to wait for response to challenge.

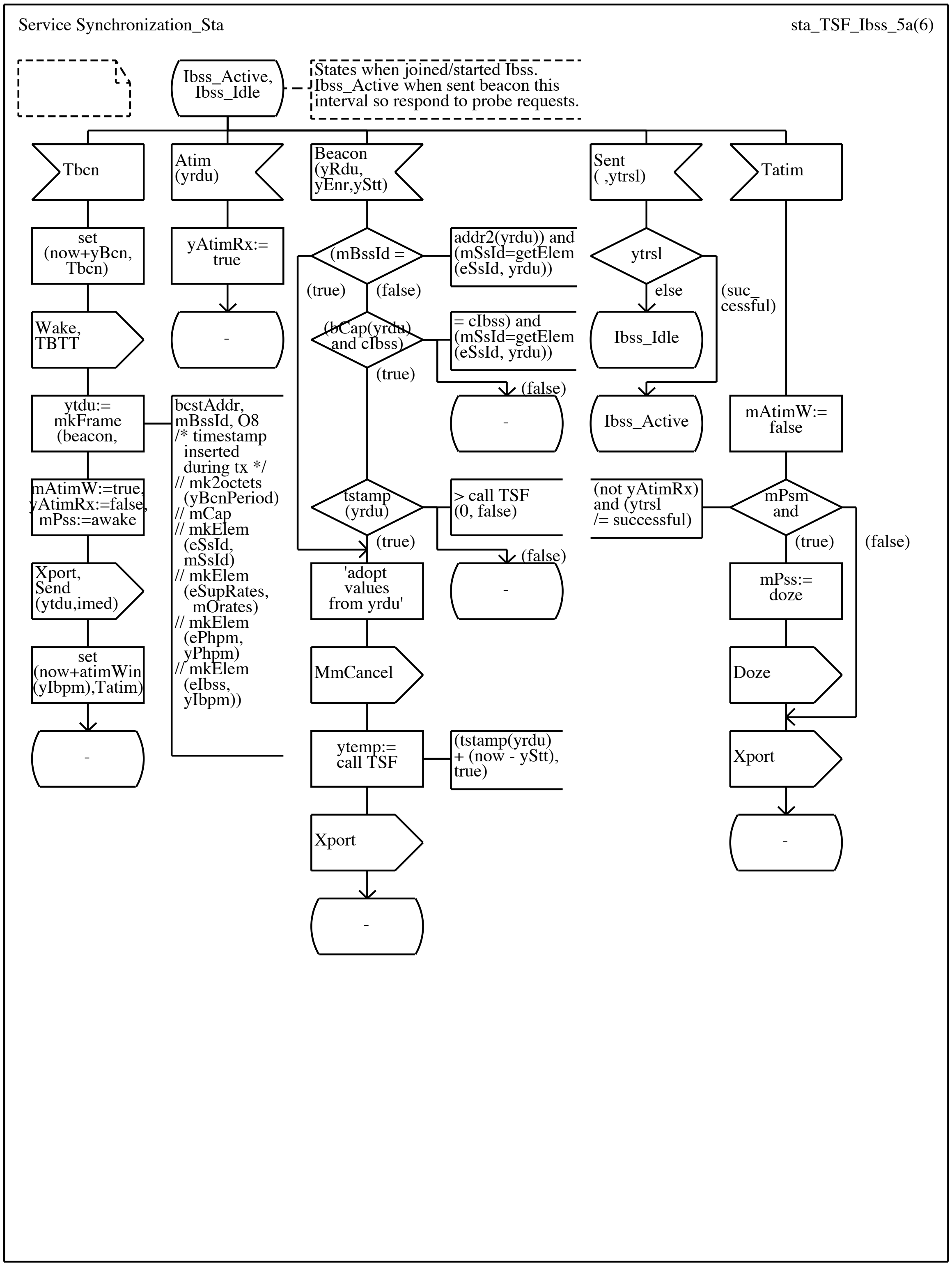

Service Distribute_Mmpdus
mmpdu_svc_1a(1)
re_
exp
export(
mAId,
mAssoc,
mAtimW, mBssId, mCap,
mCfp, mDisable, mIbss,
mListenInt, mNextBdry,
mNextTbtt, mPcAvail,
mPcDlvr, mPcPoll,
mPsm, mPss, mSsId)
dcl mAdr MacAddr ;
dcl mIm Imed ;
dcl pri CfPriority ;
dcl mRate Rate ;
dcl mRpdu, mSpdu Frame ;
dcl mSerr StateErr ;
dcl mSst StationState ;
dcl mtE, mtT Time ;
dcl mTxstat TxStatus ;
Re-export the
intra-MAC
remote
variables to
make updates
available.
Mmpdu_
Idle
Xport
re_
exp
Send
(mSpdu,
mIm)
'mRate:=
data rate to
send mmpdu'
MmRequest
(mSpdu,
mIm,mRate)
-
MmConfirm
(mSpdu,
mTxstat)
ftype
(mSpdu)
else
(beacon,
probe_rsp)
Sent
(mSpdu,
mTxstat)
-
MmConfirm only
needed for probe
responses and
beacons.
MmIndicate
(mRpdu,mtE,
mtT,mSerr)
mSerr
else
(class2)
(class3)
Cls2Err
(addr2
(mRpdu))
Cls3Err
(addr2
(mRpdu))
ftype
(mRpdu)
(asoc_req)
(asoc_rsp)
AsocReq
(mRpdu)
AsocRsp
(mRpdu)
(auth)
mTst:= mod
(authSeqNum
(mRpdu), 2)
mTst
(0)
(1)
AuthEven
(mRpdu)
AuthOdd
(mRpdu)
-
(reasoc_req)
(reasoc_rsp)
ReasocReq
(mRpdu)
ReasocRsp
(mRpdu)
(probe_req)
(probe_rsp)
(atim)
ProbeReq
(mRpdu)
ProbeRsp
(mRpdu,
mtE,mtT)
Atim
(mRpdu)
(beacon)
(disasoc)
(deauth)
Beacon
(mRpdu,
mtE,mtT)
Disasoc
(mRpdu)
Deauth
(mRpdu)
-
Sst
(mAdr,
mSst)
StaState
(mAdr,
mSst)
-
The selection criteria for
Mmpdu Tx data rate are
not specified. Frames
to group addresses must
use one of the basic rates.
Requests should use one of
the basic rates unless the
operational rates of the
recipient station are known.
Responses must use a basic
rate or the rate at which
the request was received.
/* This service routes
mmpdu and station state
update signals from and
to the mlme operational
services. Signals are
not modified, but some
superfluous parameters
are omitted in transfer. */

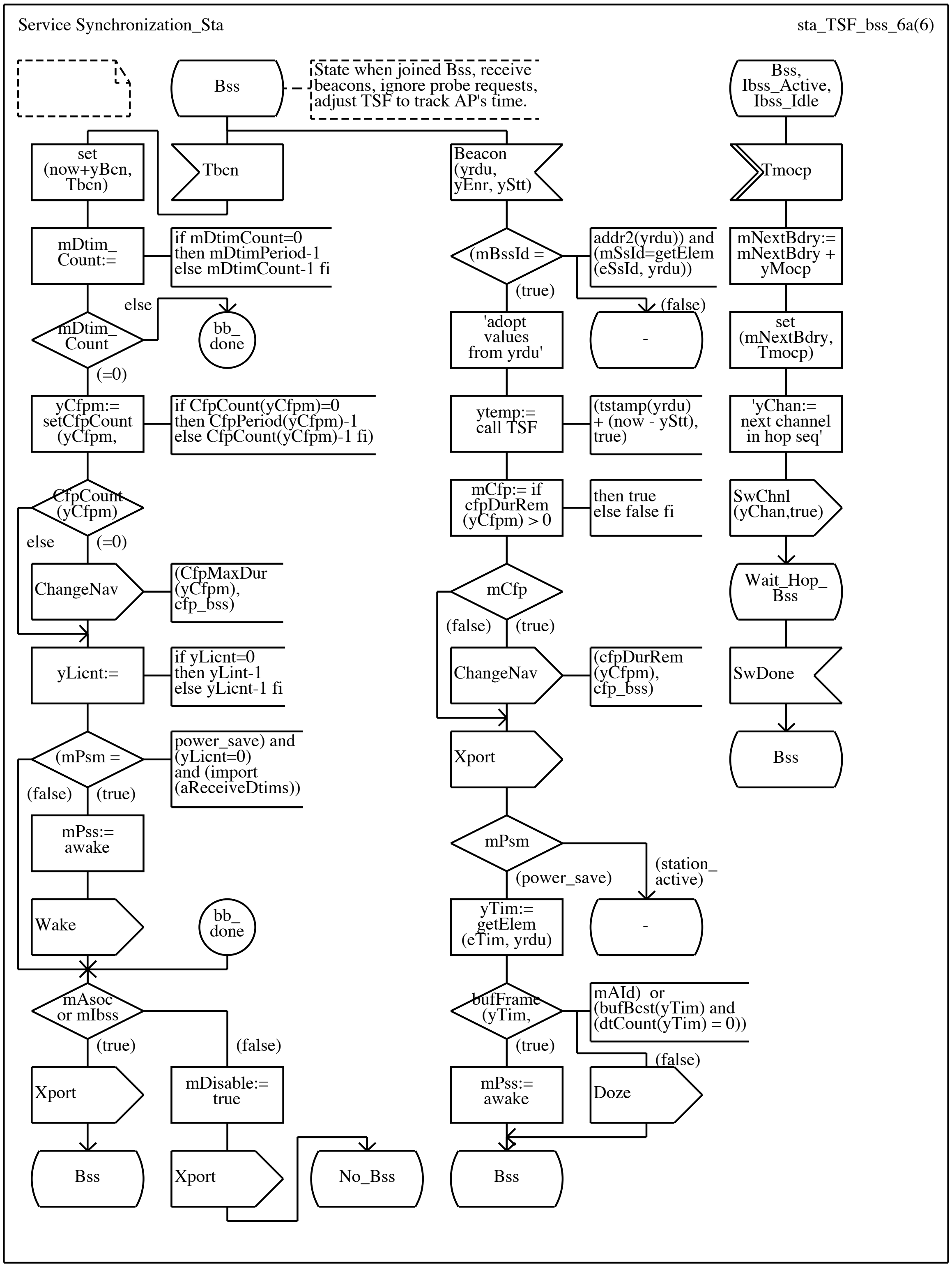

Service Synchronization_Sta
sta_Powermgt_1b(6)
dcl yAtimRx, yPsm, yRdtim, yWake Boolean ;
dcl yAtw, yBcn, yEnr, yMocp, yStt Time ;
dcl yBcnPeriod, ycmax, ycmin TU ;
dcl ybd BssDscr ;
dcl ybdset BssDscrSet ;
dcl ybtp BssType ;
dcl ybsid MacAddr ;
dcl yclist Intstring ;
dcl ycx, yJto, ytemp Integer ;
dcl yDspm DsParms ;
dcl yFhpm FhParms ;
dcl yIbpm IbssParms ;
dcl ypdly Usec ;
dcl yPhpm PhyParms ;
dcl yRdu, yTdu Frame ;
dcl yssid Octetstring ;
dcl ystp ScanType ;
dcl ytrsl TxResult ;
timer Tscan, Tmocp, Tpdly ;
*
ResetMAC
PsmDone
PsmDone sent by TxCoord after change in power save indication is announced in frame exchange.
variables to default values'
'reset all intra-MAC remote
not mDisable
Set TSF time to zero.
ytemp:= call TSF (0, true)
MlmePower_ mgt.confirm (success)
Xport
-
Setting these timers to now causes events in each of the multi-state services of the process, forcing each service to its idle state.
reset(Tbcn), reset(Tatim),
set(now,Tasoc), set(now,Tauth), set(not,Tchal)
No_BSS
No_Bss, Bss, Ibss_Active, Ibss_Idle
PowerMgt requests valid in all top-level states.
Mlme_ PowerMgt. request
(yPsm, yWake, yRdtim)
'update mib: aReceiveDtims set to yRdtim'
(yWake and
(mPss = Doze)) or ((yPsm = station_active) and (mPsm = power_save))
(false)
(true)
mPss:= awake
Wake
mPsm:= yPsm
Xport
-

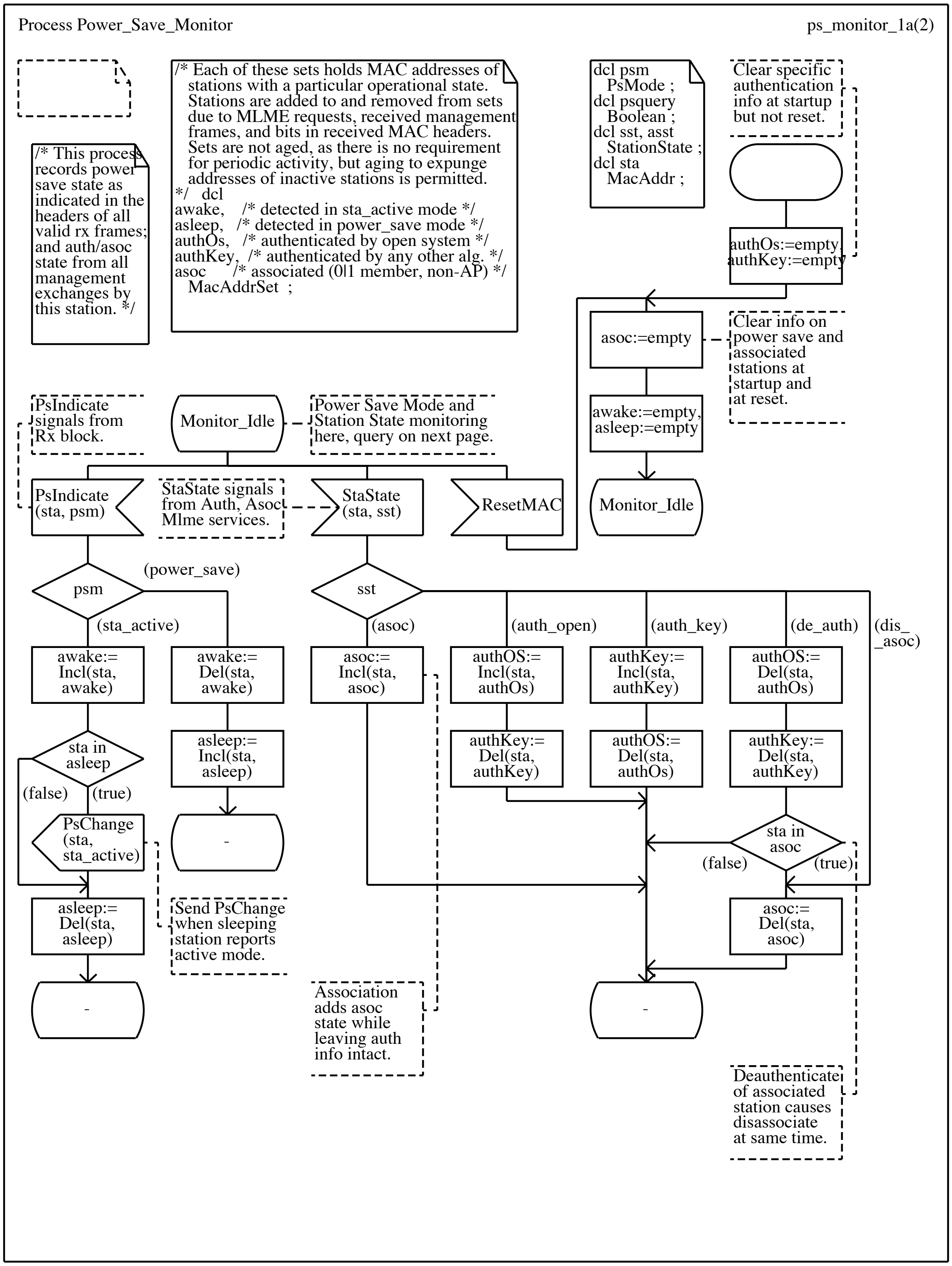

Service Synchronization_Sta
sta_Scan_2c(6)
No_Bss, Bss, Ibss_Active, Ibss_Idle
Scan requests valid in all top-level states.
Act_Listen, Act_Receive, Pas_Listen
nx_ chnl
Act_Receive, Pas_Listen
yssid, ystp, ypdly, yclist, ycmin, ycmax)
MlmeScan_ request(ybtp, ybsid,
Beacon (yrdu,yrend, ytstr)
ProbeRsp (yrdu,yrend, ytstr)
Tscan
Only accept Scan request when no frame exchange is in progress.
not import (mFxIP)
'ybd:= bss description info from yrdu'
ycx:= ycx + 1
'parameters valid'
(false)
(true)
ybd!bd_ StartTs:= ytstr
ycx > length(yclist)
(false)
(true)
MlmeScan_ confirm (empty,invalid)
ybdset:=Empty, ycx:= 0, mDisable:=true
ybdset:= ybdset or ybd
SwChnl (yclist(ycx), true)
'filter ybdset for ybtype and duplicates'
-
Xport, Wake
-
ystype
MlmeScan_ confirm (ybdset,success)
(active_scan)
(passive_scan)
No loss sync if scan parms are invalid.
ystype
Wait_Csw_ _Done
Set (now+ycmax, Tscan)
No_Bss
(active_scan)
(passive_scan)
tpdu:= mkFrame (probe_req,
bcstAddr, yBsid, mkElem (eSsId,ySsid) // mkElem(eSup_ Rates,mOrates))
nx_ chnl
SwDone
Pas_Listen
Scan ends in No_Bss state since sync lost with prior Bss. Implementations may save/restore TSF and asoc info to re-join prior Bss.
nx_ chnl
set (now+ypdly, Tscan)
Set probe delay timeout.
Wait_Probe_ _Delay
Act_Listen
Listen for activity on channel.
Tscan
*
Tscan
import (mRxA)
Send (tpdu,imed)
Transmit probe request.
nx_ chnl
Set (now+ycmax, Tscan)
Set probe response (max) timeout.
Set (now+ycmim, Tscan)
Set channel activity (min) timeout.
Go to next channel if no activity by min time.
Act_Receive
Receive responses on channel.
Act_Listen

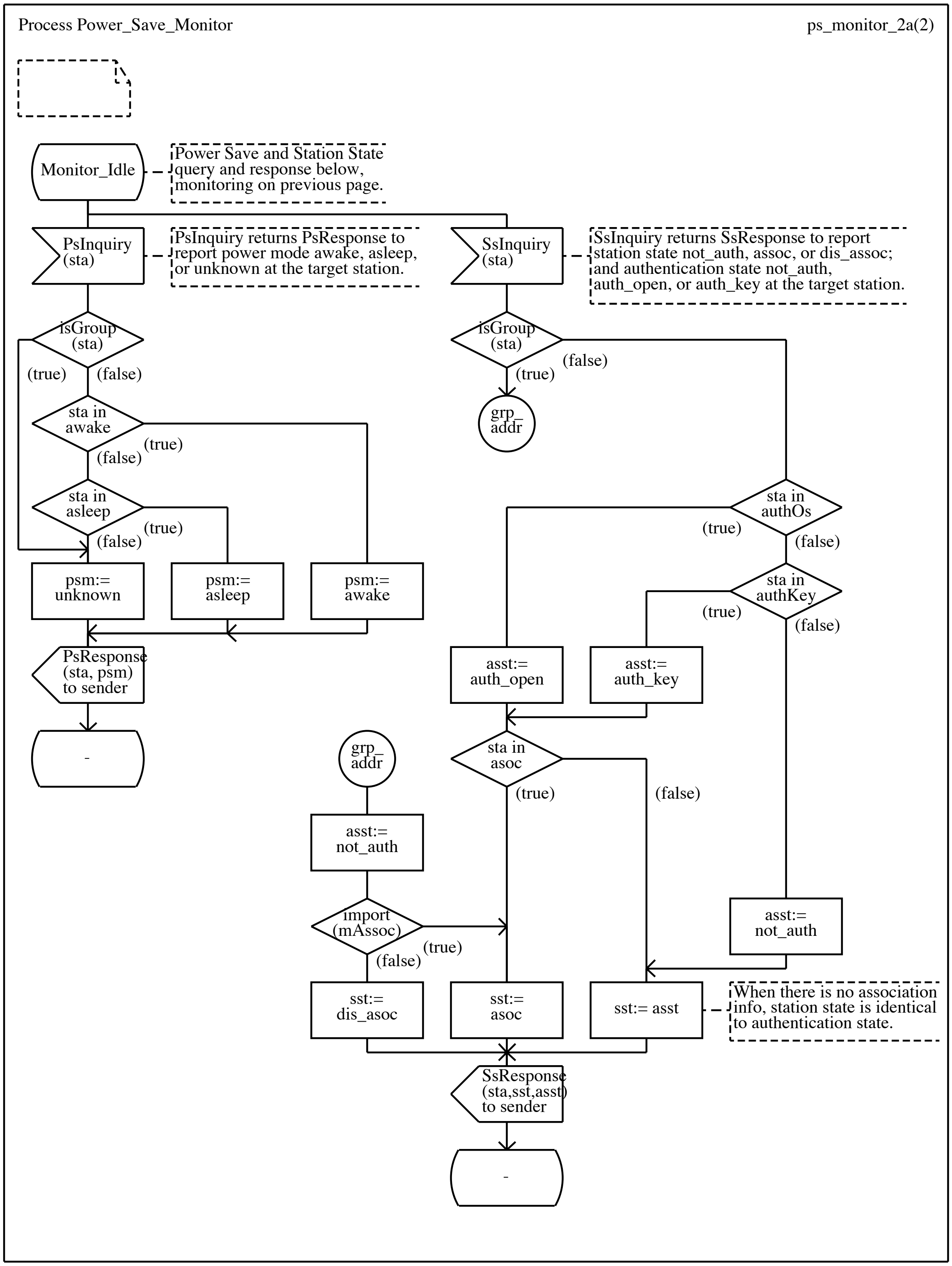

Service Synchronization_Sta
sta_Start_Ibss_3b(6)
No_BSS
Start IBSS on this page, join on next page.
MlmeStart._ request (mSsid, yBtp,
yBcnPeriod, mDtimPeriod, /* cfpm */, yPhpm, yIbpm, ypdly, mCap, mBrates, mOrates)
yBytp
(infra_ structure)
(indep_ endent)
sCanBeAp
(true)
(false)
'parameters valid'
(false)
(true)
mIbss:=true, mPss:=awake, mPdly:=ypdly
MlmeStart._ confirm (invalid)
No_Bss
mBssId:= 0 // 1 // '46 random bits'
46-bit string needs to be very random, see 11.1.3.
yBcn:= tTU (yBcnPeriod)
yAtw:=tTU (atimWin (yIbpm))
AP_Active
Activate AP state machine.
'set actual phy parameters from phpm'
Xport
ibss_ init
ibss_ init
'using FH phy'
(false)
(true)
yMocp:=tTU (dwellTime (yFhpm))
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
set (mNextBdry, Tmocp)
Initialize dwell timer.
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
SwChnl (yChan,true)
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
set (mNextBdry, Tbcn)
Initialize beacon timer.
mIbss:= true, mDisable:= false
Xport
MlmeStart._ confirm (success)
set(now + tTU(ypdly), Tpdly)
Init_Wait_ _ProbeDelay
Wait probe delay before initiating a transmission.
*
Tpdly
Ibss_Active
Start out as Ibss probe responder.

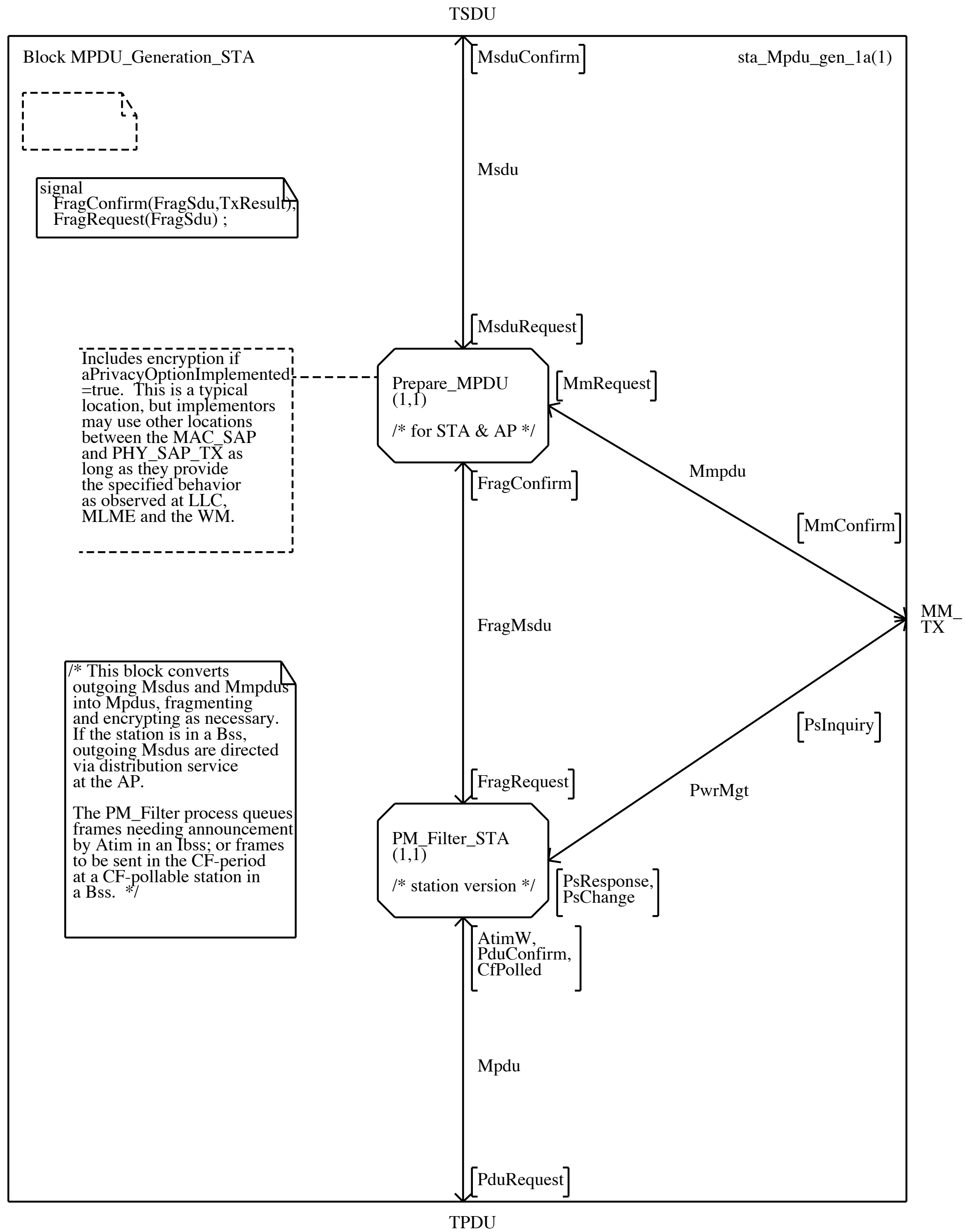

Service Synchronization_Sta
sta_Join_4b(6)
No_Bss
Join on this page, Start on previous page
Join_Wait_Bcn
MlmeJoin._ request (ybd,
yJto, ypdly, mOrates)
Tbcn
Beacon (yrdu, yenr, ystt)
A probe response from the bss/ibss can also be used to establish sync.
'bss dscr valid'
(false)
(true)
yJto:= yJto - 1
(mBssId =
addr2(yrdu)) and (mSsId = getElem (eSsId,yrdu))
(true)
(false)
MlmeJoin._ confirm (invalid)
mBssId:= ybd!bdBssId,
mSsId:= ybd!bdSsId, yBcn:=tTU (ybd!_ bdBcnPer), mDtimPeriod := ybd!_ bdDtimPer, yphpm:=ybd!_ bdPhyParms, ycfpm:=ybd!_ bdCfParms, yibpm:=ybd!_ bdIbParms, mCap:= ybd!_ bdCap, mBrates:= ybd!bdRates, mPdly:=ypdly
yJto
'adopt values from yrdu'
-
No_Bss
set (now + yBcn, Tbcn)
(=0)
(>0)
set (now + yBcn, Tbcn)
'using FH phy'
(false)
(true)
Xport
-
yMocp:=tTU (dwellTime (yFhpm))
MlmeJoin._ confirm (timeout)
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
'select channel of target Bss'
mBssId:= nullAddr, mSsId:=null
set (mNextBdry, Tmocp)
Initialize dwell timer.
ytemp:= call TSF((now -
ybd!_ bdStartTs), true)
No_Bss
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
Join_Wait_ _Beacon
SwChnl (yChan,true)
mCap and cIbss
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
else
(=cIbss)
mDisable:= false, mIbss:=false
mDisable:= false, mIbss:= true
set (mNextTbtt, Tbcn)
Initialize beacon timer.
Xport
Xport
MlmeStart._ confirm (success)
Bss
Ibss_Idle

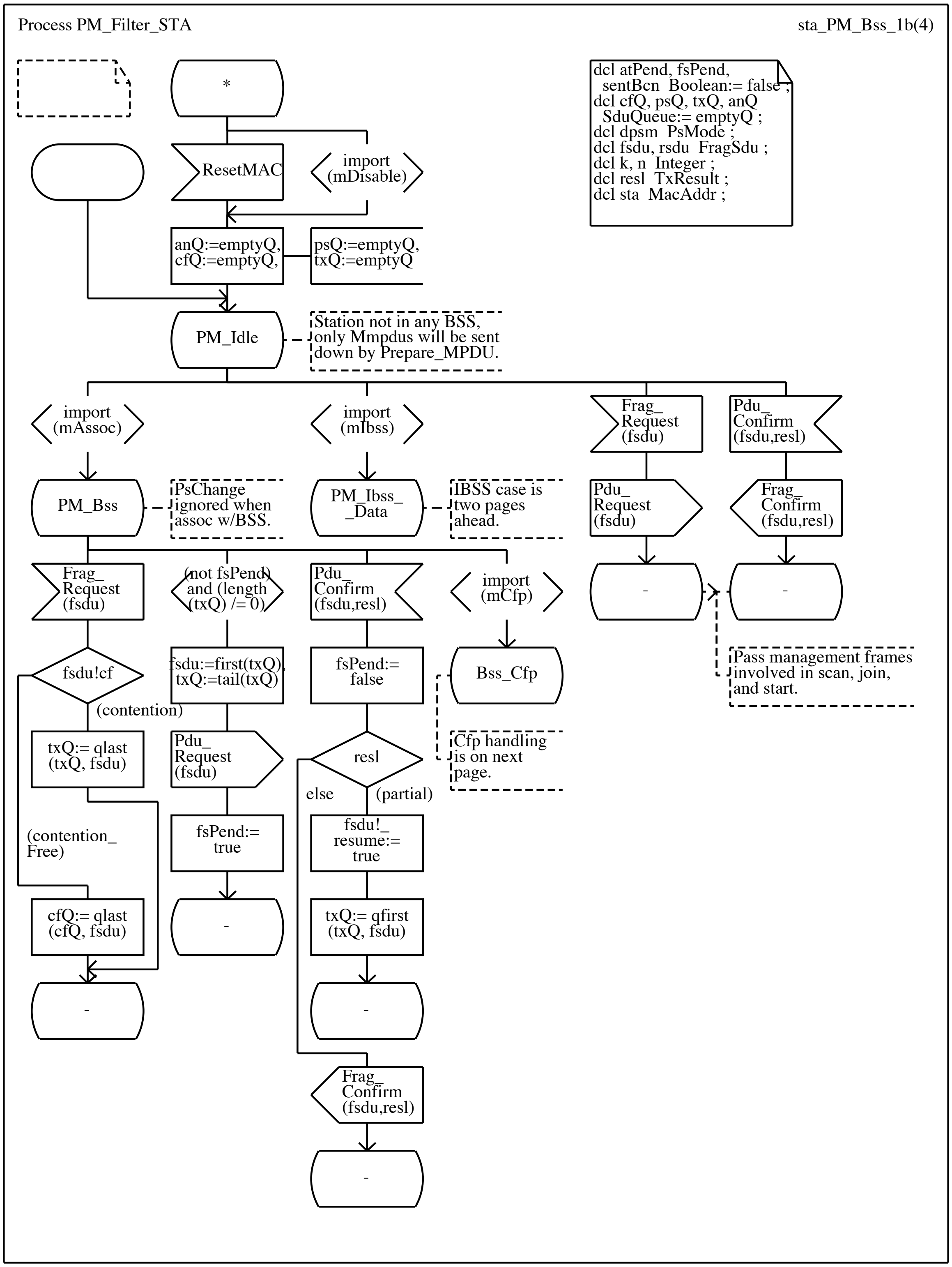

Service Synchronization_Sta
sta_TSF_Ibss_5a(6)
Ibss_Active, Ibss_Idle
States when joined/started Ibss. Ibss_Active when sent beacon this interval so respond to probe requests.
Tbcn
set (now+yBcn, Tbcn)
Wake, TBTT
ytdu:= mkFrame (beacon,
bcstAddr, mBssId, O8 /* timestamp inserted during tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId, mSsId) // mkElem (eSupRates, mOrates) // mkElem (ePhpm, yPhpm) // mkElem (eIbss, yIbpm))
mAtimW:=true, yAtimRx:=false, mPss:=awake
Xport, Send (ytdu,imed)
set (now+atimWin (yIbpm),Tatim)
-
Atim (yrdu)
yAtimRx:= true
-
Beacon (yRdu, yEnr,ySt)
(mBssId =
addr2(yrdu)) and (mSsId=getElem (eSsId, yrdu))
(true)
(false)
(bCap(yrdu) and cIbss)
= cIbss) and (mSsId=getElem (eSsId, yrdu))
(true)
(false)
-
tstamp (yrdu)
> call TSF (0, false)
(true)
(false)
-
'adopt values from yrdu'
MmCancel
ytemp:= call TSF
(tstamp(yrdu) + (now - yStt), true)
Xport
-
Sent (,ytrsl)
ytrsl
else
(suc_ cessful)
Ibss_Idle
Ibss_Active
Tatim
mAtimW:= false
mPsm and
(not yAtimRx) and (ytrsl /= successful)
(true)
(false)
mPss:= doze
Doze
Xport
-

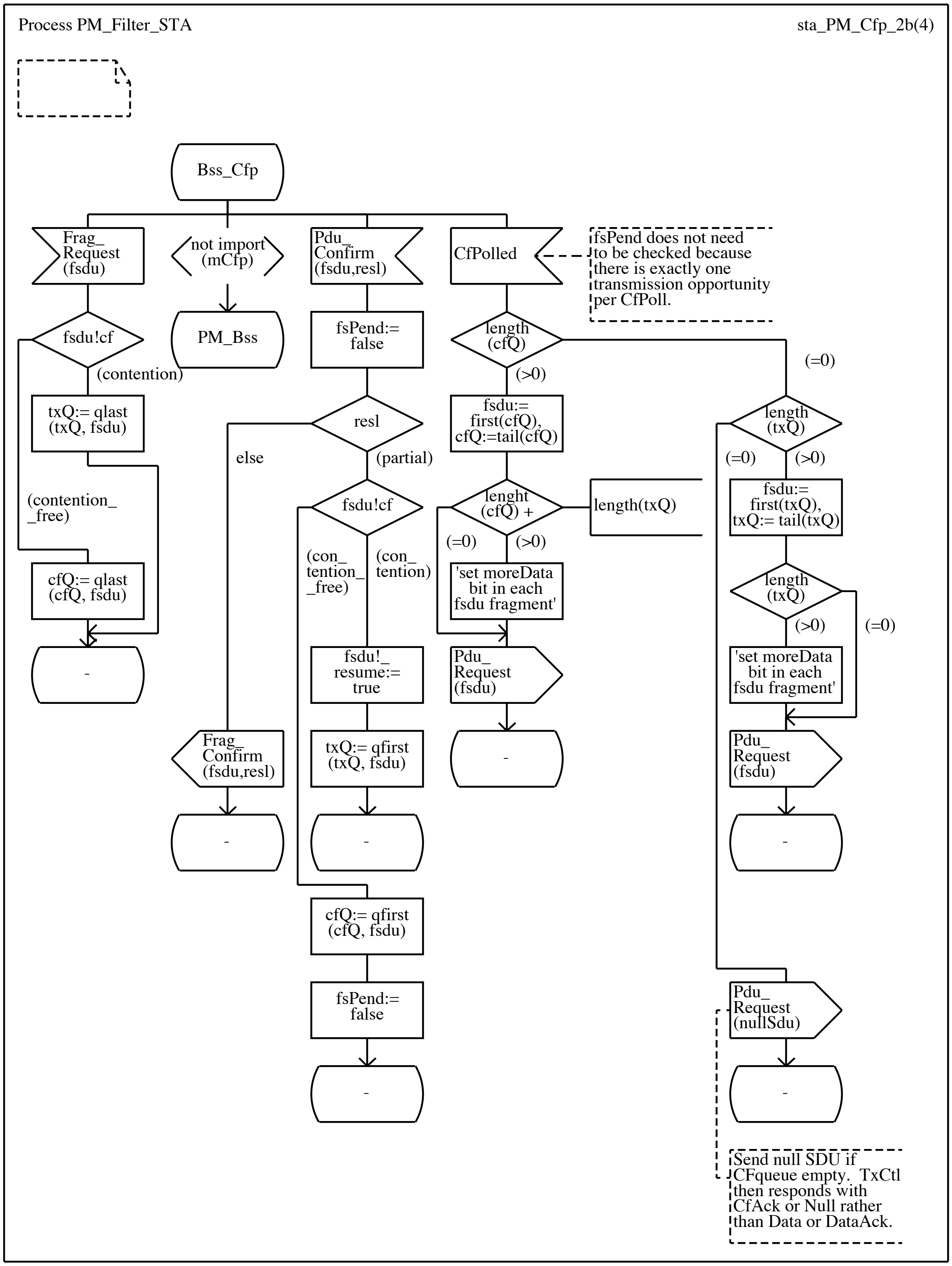

Service Synchronization_Sta
sta_TSF_bss_6a(6)
Bss
State when joined Bss, receive beacons, ignore probe requests, adjust TSF to track AP's time.
Bss, Ibss_Active, Ibss_Idle
set (now+yBcn, Tbcn)
Tbcn
Beacon (yrdu, yEnr, yStt)
Tmocp
mDtim_ Count:=
if mDtimCount=0 then mDtimPeriod-1 else mDtimCount-1 fi
(mBssId =
addr2(yrdu)) and (mSsId=getElem (eSsId, yrdu))
mNextBdry:= mNextBdry + yMocp
else
mDtim_ Count
bb_ done
(true)
(false)
'adopt values from yrdu'
-
set (mNextBdry, Tmocp)
(=0)
yCfpm:= setCfpCount (yCfpm,
if CfpCount(yCfpm)=0 then CfpPeriod(yCfpm)-1 else CfpCount(yCfpm)-1 fi)
ytemp:= call TSF
(tstamp(yrdu) + (now - yStt), true)
'yChan:= next channel in hop seq'
CfpCount (yCfpm)
mCfp:= if cfpDurRem (yCfpm) > 0
then true else false fi
SwChnl (yChan,true)
else
(=0)
ChangeNav
(CfpMaxDur (yCfpm), cfp_bss)
mCfp
Wait_Hop_ Bss
(false)
(true)
yLicnt:=
if yLicnt=0 then yLint-1 else yLicnt-1 fi
ChangeNav
(cfpDurRem (yCfpm), cfp_bss)
SwDone
(mPsm =
power_save) and (yLicnt=0) and (import (aReceiveDtims))
Xport
Bss
(false)
(true)
mPss:= awake
mPsm
(power_save)
(station_ active)
Wake
bb_ done
yTim:= getElem (eTim, yrdu)
-
mAsoc or mIbss
bufFrame (yTim,
mAId) or (bufBcst(yTim) and (dtCount(yTim) = 0))
(true)
(false)
(true)
(false)
Xport
mDisable:= true
mPss:= awake
Doze
Bss
Xport
No_Bss
Bss

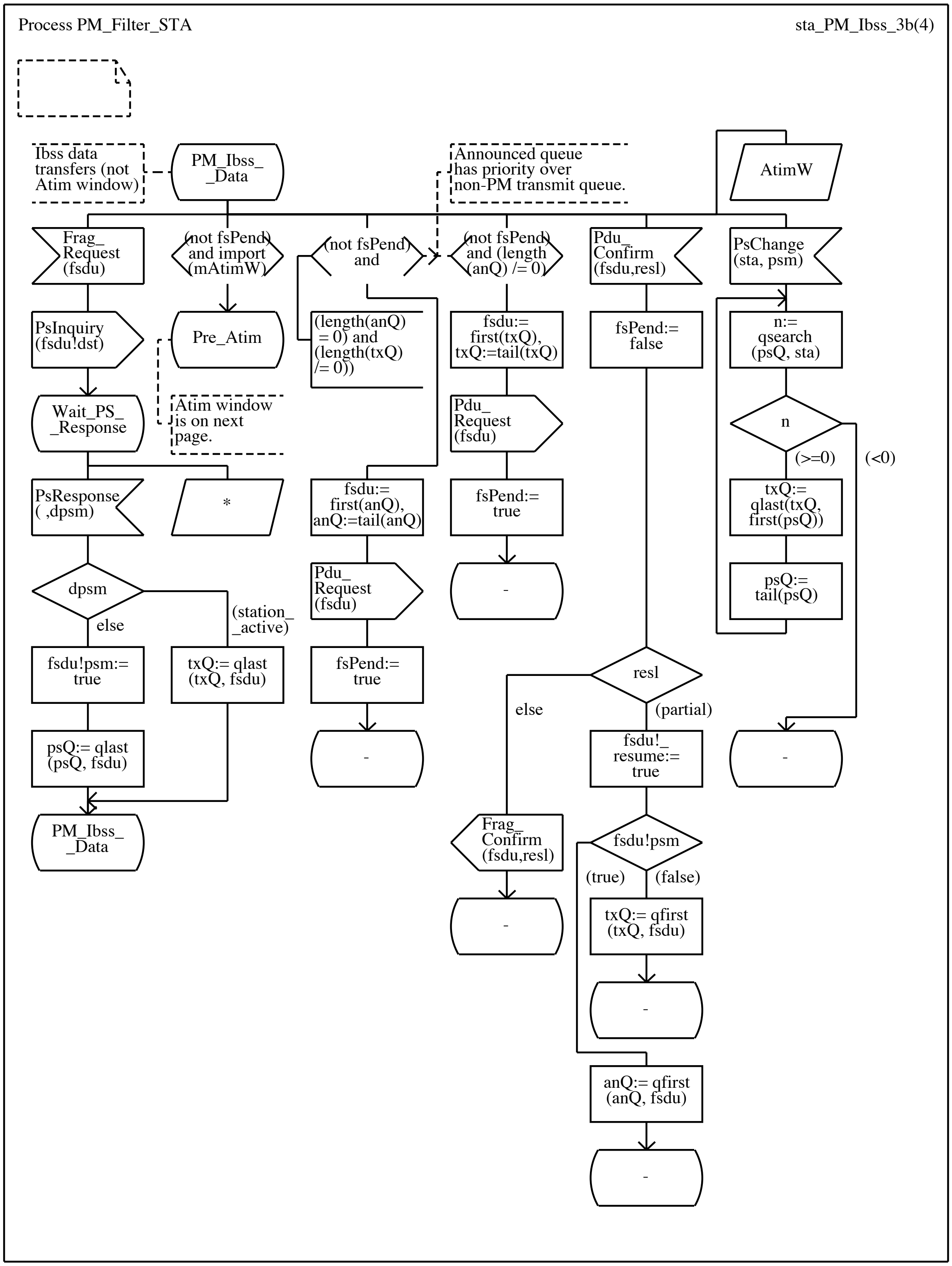

Process Power_Save_Monitor
ps_monitor_1a(2)
/* This process records power save state as indicated in the headers of all valid rx frames; and auth/asoc state from all management exchanges by this station. */
/* Each of these sets holds MAC addresses of stations with a particular operational state. Stations are added to and removed from sets due to MLME requests, received management frames, and bits in received MAC headers. Sets are not aged, as there is no requirement for periodic activity, but aging to expunge addresses of inactive stations is permitted. */ dcl
awake, /* detected in sta_active mode */
asleep, /* detected in power_save mode */
authOs, /* authenticated by open system */
authKey, /* authenticated by any other alg. */
asoc /* associated (0|1 member, non-AP) */
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
asoc:=empty
Clear info on power save and associated stations at startup and at reset.
awake:=empty, asleep:=empty
Monitor_Idle
PsIndicate signals from Rx block.
Monitor_Idle
Power Save Mode and Station State monitoring here, query on next page.
PsIndicate (sta, psm)
StaState signals from Auth, Asoc Mlme services.
StaState (sta, sst)
ResetMAC
psm
(power_save)
(sta_active)
awake:= Incl(sta, awake)
awake:= Del(sta, awake)
asleep:= Incl(sta, asleep)
-
sta in asleep
(false)
(true)
PsChange (sta, sta_active)
asleep:= Del(sta, asleep)
-
Send PsChange when sleeping station reports active mode.
sst
(asoc)
(auth_open)
(auth_key)
(de_auth)
(dis_ _asoc)
asoc:= Incl(sta, asoc)
authOS:= Incl(sta, authOs)
authKey:= Incl(sta, authKey)
authOS:= Del(sta, authOs)
authKey:= Del(sta, authKey)
authOS:= Del(sta, authOs)
authKey:= Del(sta, authKey)
sta in asoc
(false)
(true)
asoc:= Del(sta, asoc)
-
Association adds asoc state while leaving auth info intact.
Deauthenticate of associated station causes disassociate at same time.

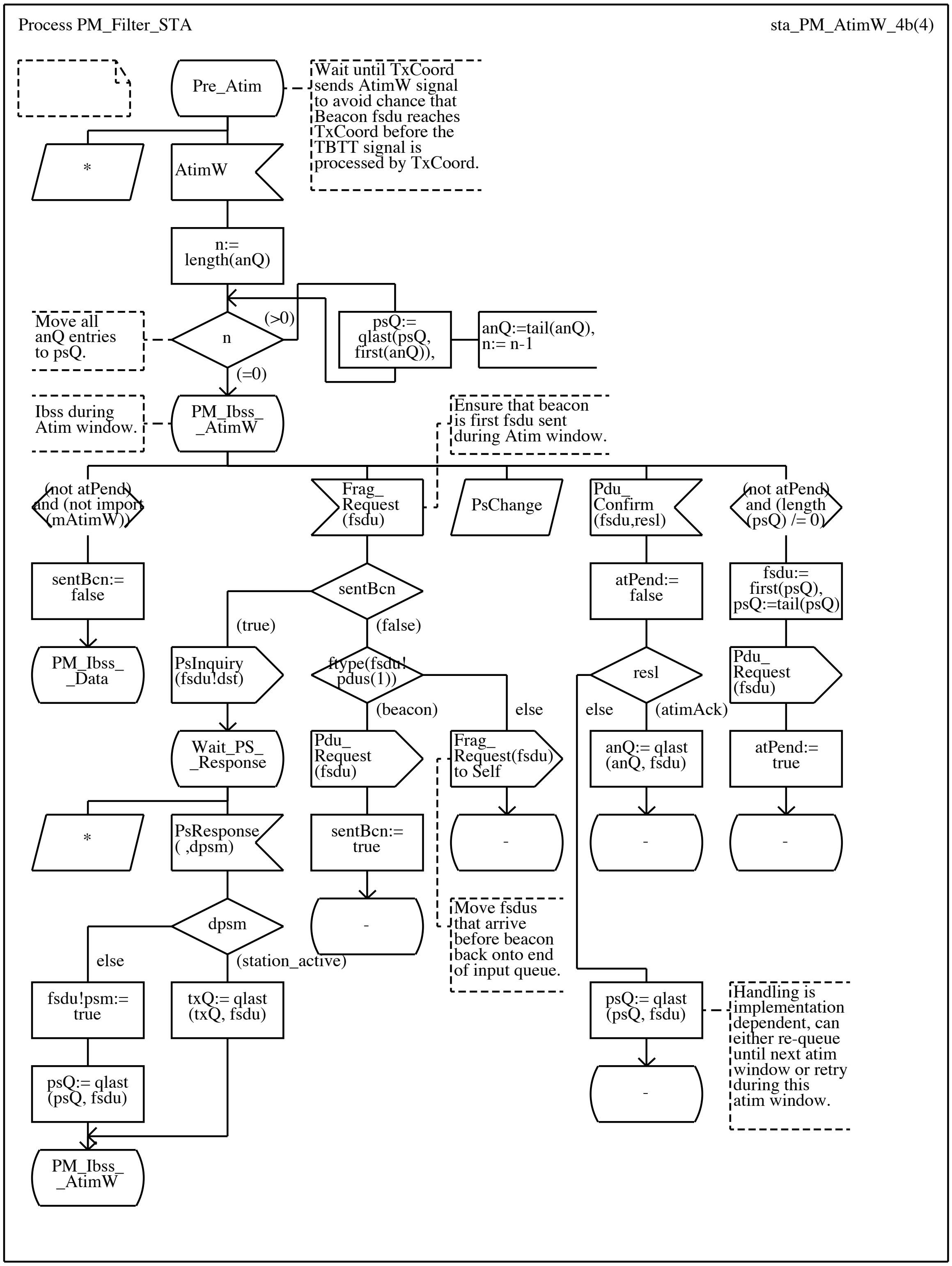

Process Power_Save_Monitor
ps_monitor_2a(2)
Monitor_Idle
Power Save and Station State query and response below, monitoring on previous page.
PsInquiry (sta)
PsInquiry returns PsResponse to report power mode awake, asleep, or unknown at the target station.
SsInquiry (sta)
SsInquiry returns SsResponse to report station state not_auth, assoc, or dis_assoc; and authentication state not_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 asoc
asst:= not_auth
import (mAssoc)
sst:= dis_asoc
sst:= asoc
sst:= asst
When there is no association info, station state is identical to authentication state.
SsResponse (sta,sst,asst) to sender

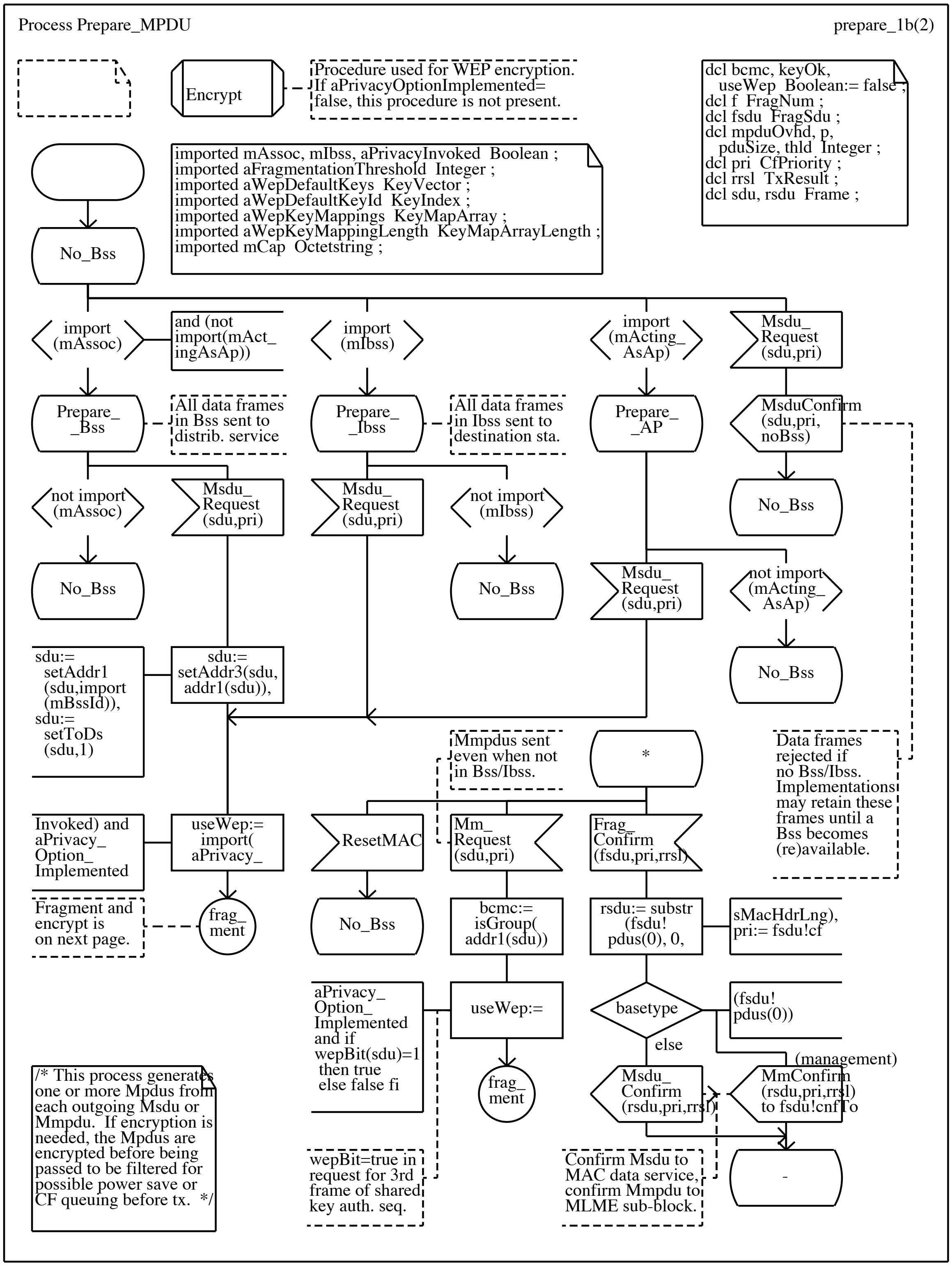

TSDU
Block MPDU_Generation_STA
sta_Mpdu_gen_1a(1)
MsduConfirm
Msdu
signal
FragConfirm(FragSdu,TxResult),
FragRequest(FragSdu) ;
MsduRequest
Includes encryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the MAC_SAP and PHY_SAP_TX as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Prepare_MPDU
(1,1)
/* for STA & AP */
MmRequest
Mmpdu
FragConfirm
MmConfirm
MM_
TX
FragMsdu
/* This block converts outgoing Msdus and Mmpdus into Mpdus, fragmenting and encrypting as necessary. If the station is in a Bss, outgoing Msdus are directed via distribution service at the AP.
The PM_Filter process queues frames needing announcement by Atim in an Ibss; or frames to be sent in the CF-period at a CF-pollable station in a Bss. */
PsInquiry
FragRequest
PwrMgt
PM_Filter_STA
(1,1)
/* station version */
PsResponse,
PsChange
AtimW,
PduConfirm,
CfPolled
Mpdu
PduRequest
TPDU

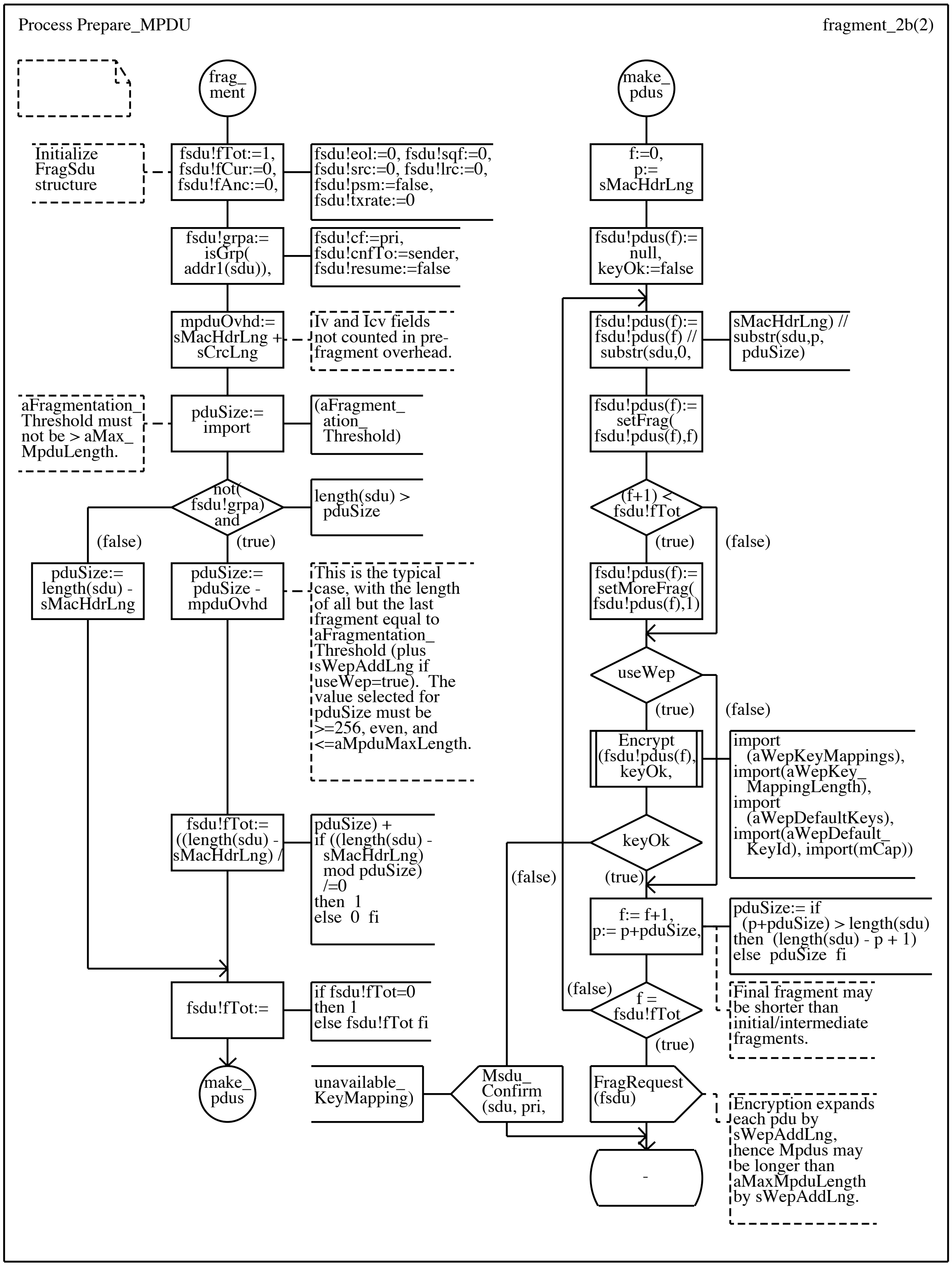

Process PM_Filter_STA
sta_PM_Bss_1b(4)
*
ResetMAC
import
(mDisable)
dcl atPend, fsPend,
sentBcn Boolean:= false ;
dcl cfQ, psQ, txQ, anQ
SduQueue:= emptyQ ;
dcl dpsm PsMode ;
dcl fsdu, rsdu FragSdu ;
dcl k, n Integer ;
dcl resl TxResult ;
dcl sta MacAddr ;
anQ:=emptyQ,
cfQ:=emptyQ,
psQ:=emptyQ,
txQ:=emptyQ
PM_Idle
Station not in any BSS,
only Mmpdus will be sent
down by Prepare_MPDU.
import
(mAssoc)
import
(mIbss)
Frag_
Request
(fsdu)
Pdu_
Confirm
(fsdu,resl)
PM_Bss
PsChange
ignored when
assoc w/BSS.
PM_Ibss_
_Data
IBSS case is
two pages
ahead.
Pdu_
Request
(fsdu)
Frag_
Confirm
(fsdu,resl)
Frag_
Request
(fsdu)
(not fsPend)
and (length
(txQ) /= 0)
Pdu_
Confirm
(fsdu,resl)
import
(mCfp)
-
-
Pass management frames
involved in scan, join,
and start.
fsdu!cf
(contention)
fsdu:=first(txQ),
txQ:=tail(txQ)
fsPend:=
false
Bss_Cfp
txQ:= qlast
(txQ, fsdu)
Pdu_
Request
(fsdu)
resl
Cfp handling
is on next
page.
else
(partial)
(contention_
Free)
fsPend:=
true
fsdu!_
resume:=
true
cfQ:= qlast
(cfQ, fsdu)
-
txQ:= qfirst
(txQ, fsdu)
-
-
Frag_
Confirm
(fsdu,resl)
-

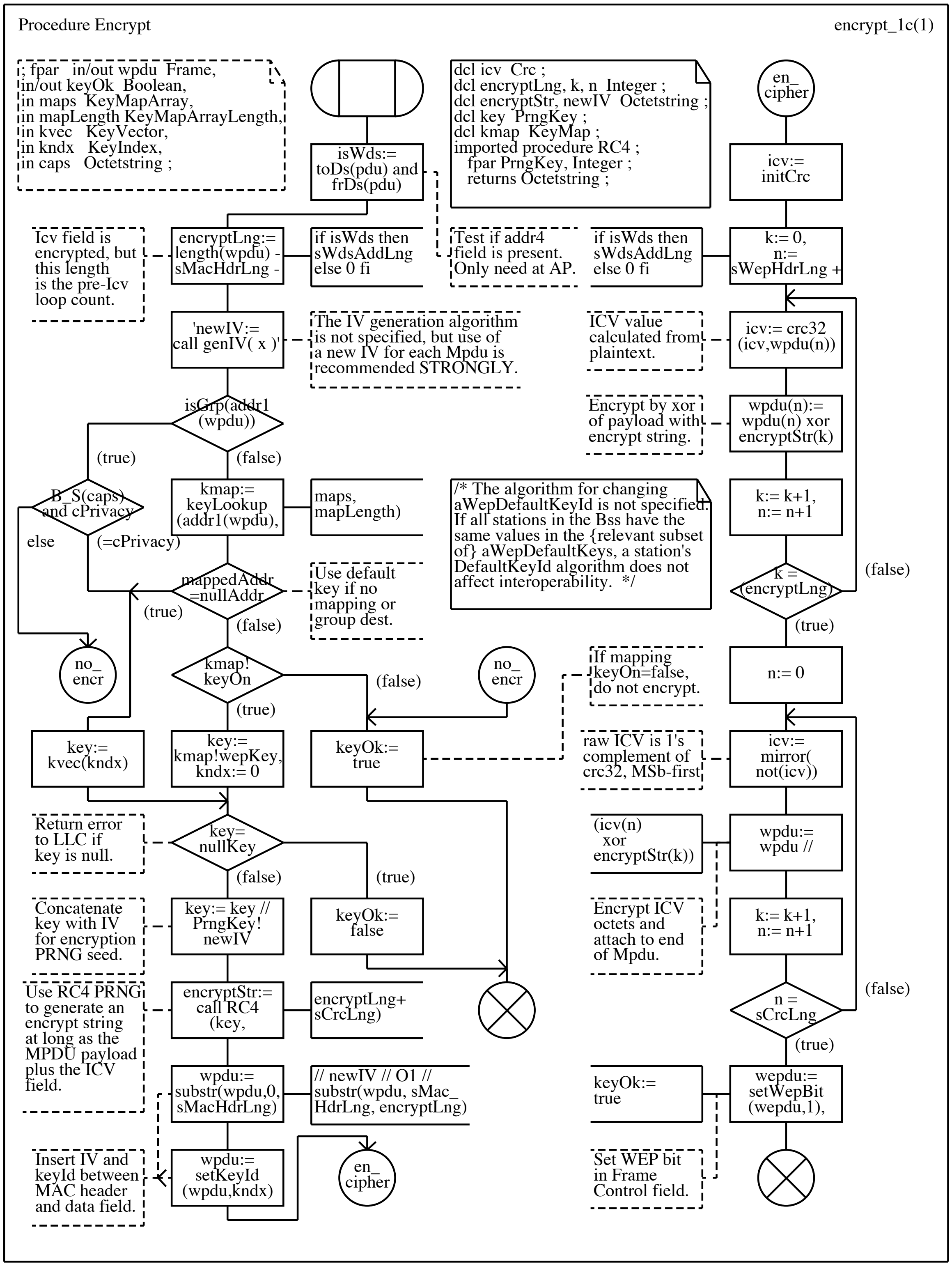

Process PM_Filter_STA
sta_PM_Cfp_2b(4)
Bss_Cfp
Frag_ Request (fsdu)
not import (mCfp)
Pdu_ Confirm (fsdu,resl)
CfPolled
fsPend does not need to be checked because there is exactly one transmission opportunity per CfPoll.
fsdu!cf
PM_Bss
fsPend:= false
length (cfQ)
(contention)
(>0)
(=0)
txQ:= qlast (txQ, fsdu)
resl
fsdu:= first(cfQ), cfQ:=tail(cfQ)
length (txQ)
else
(partial)
(=0)
(>0)
(contention_ _free)
fsdu!cf
lenght (cfQ) +
length(txQ)
fsdu:= first(txQ), txQ:= tail(txQ)
(con_ tention_ _free)
(con_ tention)
(=0)
(>0)
cfQ:= qlast (cfQ, fsdu)
'set moreData bit in each fsdu fragment'
length (txQ)
(>0)
(=0)
-
fsdu!_ resume:= true
Pdu_ Request (fsdu)
'set moreData bit in each fsdu fragment'
Frag_ Confirm (fsdu,resl)
txQ:= qfirst (txQ, fsdu)
-
Pdu_ Request (fsdu)
-
-
-
cfQ:= qfirst (cfQ, fsdu)
fsPend:= false
Pdu_ Request (nullSdu)
-
-
Send null SDU if CFqueue empty. TxCtl then responds with CfAck or Null rather than Data or DataAck.

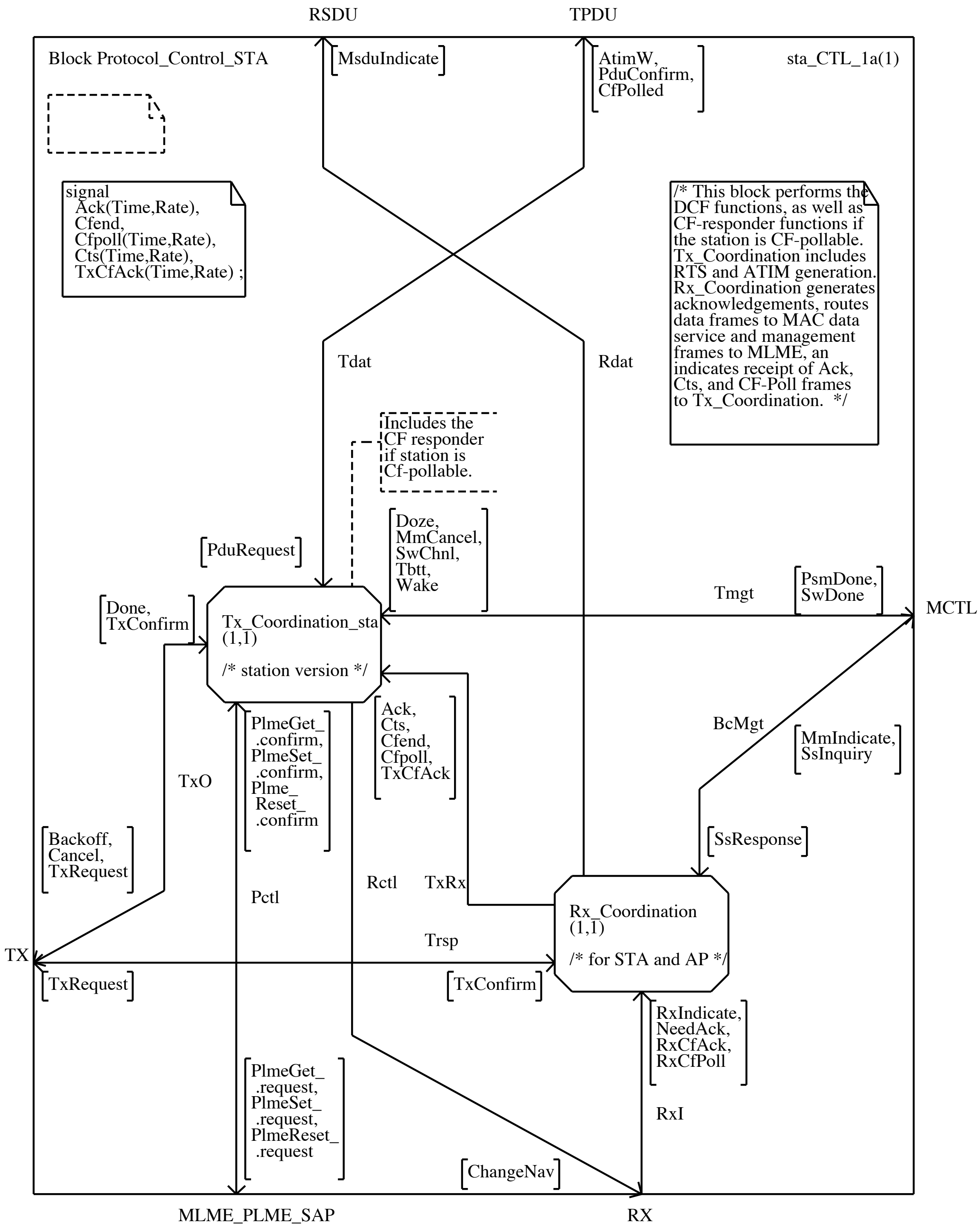

Process PM_Filter_STA
sta_PM_Ibss_3b(4)
Ibss data transfers (not Atim window)
PM_Ibss_ _Data
Announced queue has priority over non-PM transmit queue.
AtimW
Frag_ Request (fsdu)
(not fsPend) and import (mAtimW)
(not fsPend) and
(not fsPend) and (length (anQ) /= 0)
Pdu_ Confirm (fsdu,resl)
PsChange (sta, psm)
PsInquiry (fsdu!dst)
Pre_Atim
(length(anQ) = 0) and (length(txQ) /= 0))
fsdu:= first(txQ), txQ:=tail(txQ)
fsPend:= false
n:= qsearch (psQ, sta)
Wait_PS_ _Response
Atim window is on next page.
Pdu_ Request (fsdu)
n
(>=0)
(<0)
PsResponse (,dpsm)
*
fsdu:= first(anQ), anQ:=tail(anQ)
fsPend:= true
txQ:= qlast(txQ, first(psQ))
dpsm
Pdu_ Request (fsdu)
-
psQ:= tail(psQ)
else
(station_ _active)
fsdu!psm:= true
txQ:= qlast (txQ, fsdu)
fsPend:= true
resl
else
(partial)
psQ:= qlast (psQ, fsdu)
-
fsdu!_ resume:= true
-
PM_Ibss_ _Data
Frag_ Confirm (fsdu,resl)
fsdu!psm
(true)
(false)
-
txQ:= qfirst (txQ, fsdu)
-
anQ:= qfirst (anQ, fsdu)
-

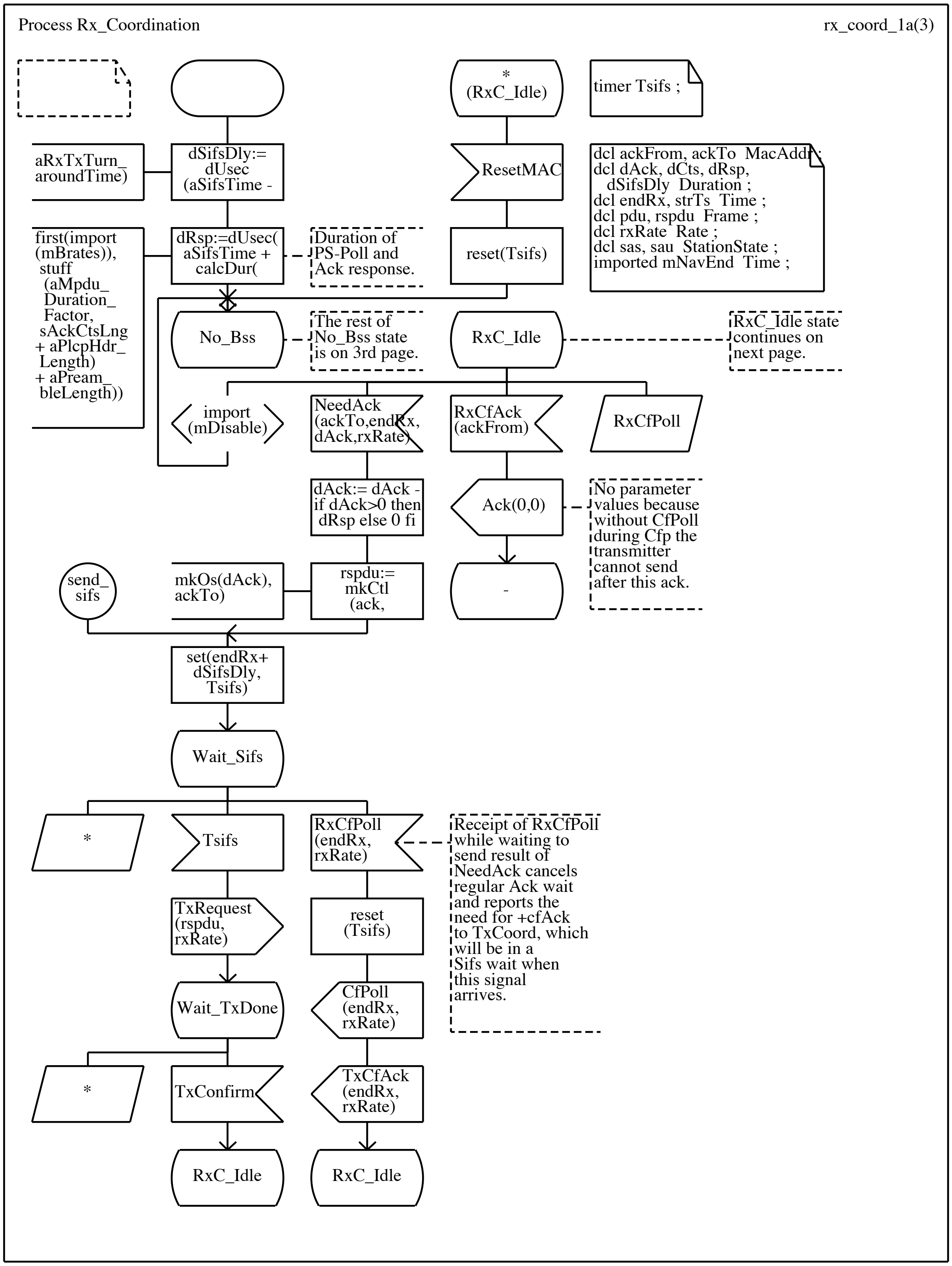

Process PM_Filter_STA
sta_PM_AtimW_4b(4)
Pre_Atim
Wait until TxCoord sends AtimW signal to avoid chance that Beacon fsdu reaches TxCoord before the TBTT signal is processed by TxCoord.
*
AtimW
n:= length(anQ)
Move all anQ entries to psQ.
n
(>0)
psQ:= qlast(psQ, first(anQ)),
anQ:=tail(anQ), n:= n-1
(=0)
Ibss during Atim window.
PM_Ibss_ _AtimW
Ensure that beacon is first fsdu sent during Atim window.
(not atPend) and (not import (mAtimW))
Frag_ Request (fsdu)
PsChange
Pdu_ Confirm (fsdu,resl)
(not atPend) and (length (psQ) /= 0)
sentBcn:= false
sentBcn
(true)
(false)
atPend:= false
fsdu:= first(psQ), psQ:=tail(psQ)
PM_Ibss_ _Data
PsInquiry (fsdu!dst)
ftype(fsdu! pdus(1))
resl
Pdu_ Request (fsdu)
(beacon)
else
else
(atimAck)
Wait_PS_ _Response
Pdu_ Request (fsdu)
Frag_ Request(fsdu) to Self
anQ:= qlast (anQ, fsdu)
atPend:= true
*
PsResponse (,dpsm)
sentBcn:= true
-
-
-
-
Move fsdus that arrive before beacon back onto end of input queue.
dpsm
else
(station_active)
fsdu!psm:= true
txQ:= qlast (txQ, fsdu)
psQ:= qlast (psQ, fsdu)
Handling is implementation dependent, can either re-queue until next atim window or retry during this atim window.
psQ:= qlast (psQ, fsdu)
-
PM_Ibss_ _AtimW

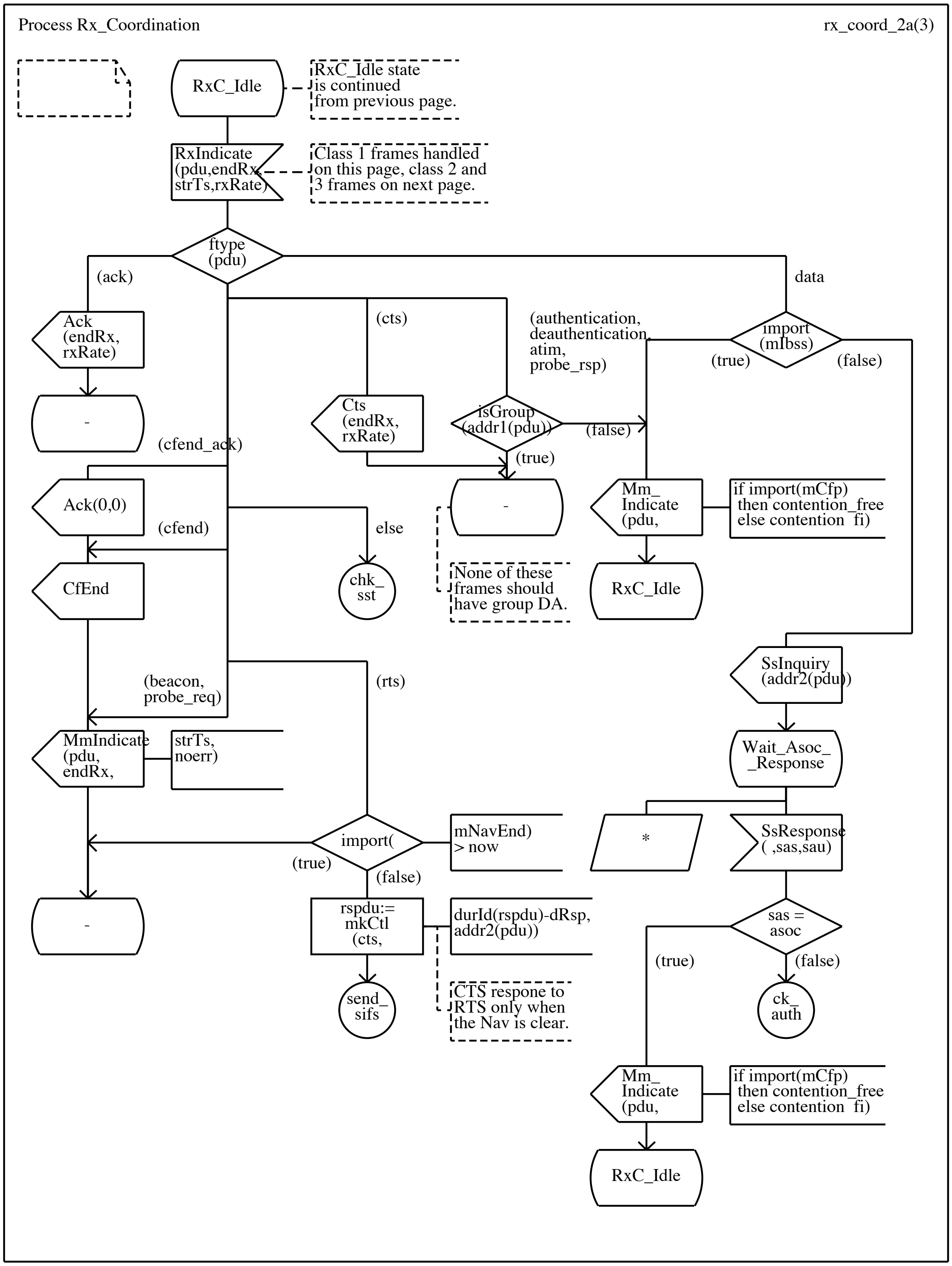

Process Prepare_MPDU
prepare_1b(2)
Encrypt
Procedure used for WEP encryption. If aPrivacyOptionImplemented= false, this procedure is not present.
dcl bcmc, keyOk, useWep Boolean:= false ; dcl f FragNum ; dcl fsdu FragSdu ; dcl mpduOvhd, p, pduSize, thld Integer ; dcl pri CfPriority ; dcl rrsl TxResult ; dcl sdu, rsdu Frame ;
imported mAssoc, mIbss, aPrivacyInvoked Boolean ; imported aFragmentationThreshold Integer ; imported aWepDefaultKeys KeyVector ; imported aWepDefaultKeyId KeyIndex ; imported aWepKeyMappings KeyMapArray ; imported aWepKeyMappingLength KeyMapArrayLength ; imported mCap Octetstring ;
No_Bss
import (mAssoc)
and (not import(mActingAsAp))
import (mIbss)
import (mActing_AsAp)
Msdu_Request (sdu,pri)
Prepare_Bss
All data frames in Bss sent to distrib. service
Prepare_Ibss
All data frames in Ibss sent to destination sta.
Prepare_AP
MsduConfirm (sdu,pri, noBss)
not import (mAssoc)
Msdu_Request (sdu,pri)
Msdu_Request (sdu,pri)
not import (mIbss)
No_Bss
No_Bss
No_Bss
Msdu_Request (sdu,pri)
not import (mActing_AsAp)
No_Bss
sdu:= setAddr1 (sdu,import (mBssId)), sdu:= setToDs (sdu,1)
sdu:= setAddr3(sdu, addr1(sdu)),
Mmpdus sent even when not in Bss/Ibss.
*
Data frames rejected if no Bss/Ibss. Implementations may retain these frames until a Bss becomes (re)available.
Invoked) and aPrivacy_Option_Implemented
useWep:= import(aPrivacy_
ResetMAC
Mm_Request (sdu,pri)
Frag_Confirm (fsdu,pri,rrsl)
Fragment and encrypt is on next page.
frag_ment
No_Bss
bcmc:= isGroup(addr1(sdu))
rsdu:= substr (fsdu! pdus(0), 0,
sMacHdrLng), pri:= fsdu!cf
aPrivacy_Option_Implemented and if wepBit(sdu)=1 then true else false fi
useWep:=
basetype
(fsdu! pdus(0))
else
(management)
/* This process generates one or more Mpdus from each outgoing Msdu or Mmpdu. If encryption is needed, the Mpdus are encrypted before being passed to be filtered for possible power save or CF queuing before tx. */
frag_ment
Msdu_Confirm (rsdu,pri,rrsl)
MmConfirm (rsdu,pri,rrsl) to fsdu!cnfTo
wepBit=true in request for 3rd frame of shared key auth. seq.
Confirm Msdu to MAC data service, confirm Mmpdu to MLME sub-block.
-

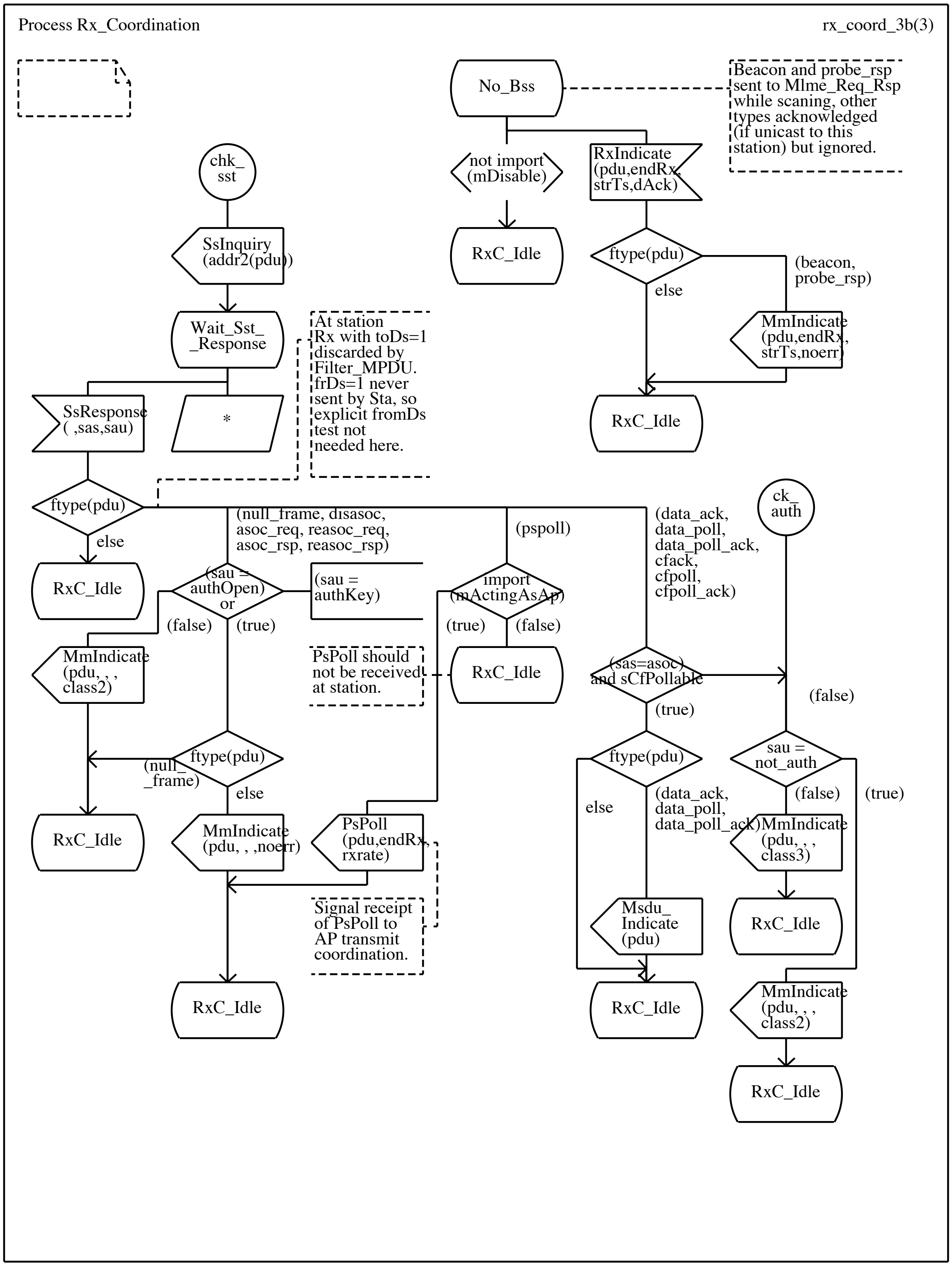

Process Prepare_MPDU
fragment_2b(2)
frag_ ment
Initialize FragSdu structure
fsdu!fTot:=1, fsdu!fCur:=0, fsdu!fAnc:=0,
fsdu!eol:=0, fsdu!sqf:=0, fsdu!src:=0, fsdu!lrc:=0, fsdu!psm:=false, fsdu!txrate:=0
fsdu!grpa:= isGrp(addr1(sdu)),
fsdu!cf:=pri, fsdu!cnfTo:=sender, fsdu!resume:=false
mpduOvhd:= sMacHdrLng + sCrcLng
Iv and Icv fields not counted in pre-fragment overhead.
aFragmentation_ Threshold must not be > aMax_ MpduLength.
pduSize:= import
(aFragment_ ation_ Threshold)
not(fsdu!grpa) and
length(sdu) > pduSize
(false)
(true)
pduSize:= length(sdu) - sMacHdrLng
pduSize:= pduSize - mpduOvhd
This is the typical case, with the length of all but the last fragment equal to aFragmentation_ Threshold (plus sWepAddLng if useWep=true). The value selected for pduSize must be >=256, even, and <=aMpduMaxLength.
fsdu!fTot:= ((length(sdu) - sMacHdrLng) /
pduSize) + if ((length(sdu) - sMacHdrLng) mod pduSize) /=0 then 1 else 0 fi
fsdu!fTot:=
if fsdu!fTot=0 then 1 else fsdu!fTot fi
make_ pdus
unavailable_ KeyMapping)
Msdu_ Confirm (sdu, pri,
make_ pdus
f:=0, p:= sMacHdrLng
fsdu!pdus(f):= null, keyOk:=false
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), keyOk,
import (aWepKeyMappings), import(aWepKey_ MappingLength), import (aWepDefaultKeys), import(aWepDefault_ KeyId), import(mCap))
keyOk
(false)
(true)
f:= f+1, p:= p+pduSize,
pduSize:= if (p+pduSize) > length(sdu) then (length(sdu) - p + 1) else pduSize fi
Final fragment may be shorter than initial/intermediate fragments.
f = fsdu!fTot
(false)
(true)
FragRequest (fsdu)
Encryption expands each pdu by sWepAddLng, hence Mpdus may be longer than aMaxMpduLength by sWepAddLng.
-

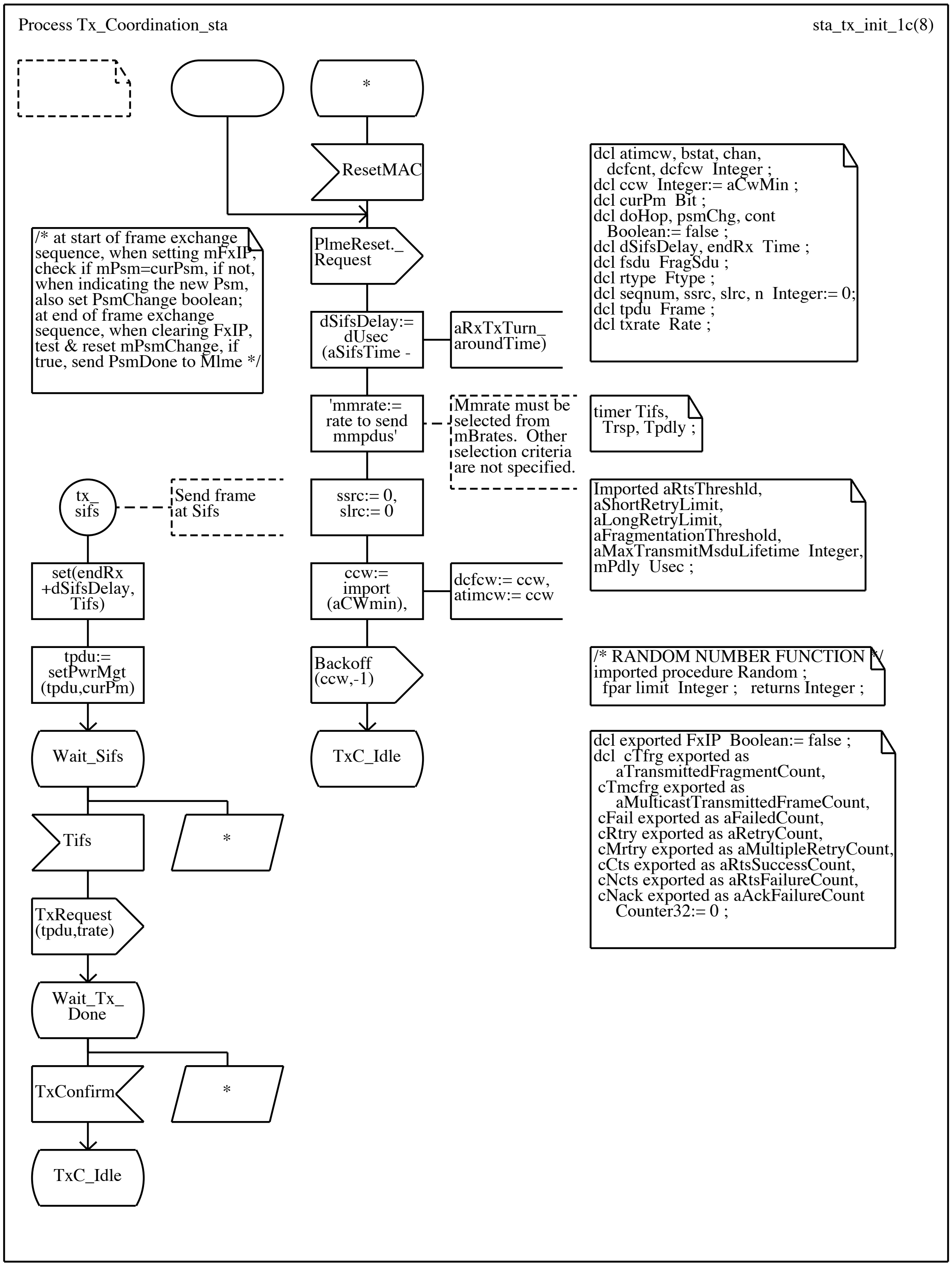

Procedure Encrypt
encrypt_1c(1)
; fpar in/out wpdu Frame, in/out keyOk Boolean, in maps KeyMapArray, in mapLength KeyMapArrayLength, in kvec KeyVector, in kndx KeyIndex, in caps Octetstring ;
dcl icv Crc ; dcl encryptLng, k, n Integer ; dcl encryptStr, newIV Octetstring ; dcl key PrngKey ; dcl kmap KeyMap ; imported procedure RC4 ; fpar PrngKey, Integer ; returns Octetstring ;
isWds:= toDs(pdu) and frDs(pdu)
Test if addr4 field is present. Only need at AP.
Icv field is encrypted, but this length is the pre-Icv loop count.
encryptLng:= length(wpdu) - sMacHdrLng -
if isWds then sWdsAddLng else 0 fi
'newIV:= call genIV(x)'
The IV generation algorithm is not specified, but use of a new IV for each Mpdu is recommended STRONGLY.
isGrp(addr1 (wpdu))
(true)
(false)
B_S(caps) and cPrivacy
else
(=cPrivacy)
kmap:= keyLookup (addr1(wpdu),
maps, mapLength)
/* The algorithm for changing aWepDefaultKeyId is not specified. If all stations in the Bss have the same values in the {relevant subset of} aWepDefaultKeys, a station's DefaultKeyId algorithm does not affect interoperability. */
mappedAddr =nullAddr
(true)
(false)
Use default key if no mapping or group dest.
no_ encr
kmap! keyOn
(false)
(true)
no_ encr
If mapping keyOn=false, do not encrypt.
key:= kvec(kndx)
key:= kmap!wepKey, kndx:= 0
keyOk:= true
Return error to LLC if key is null.
key= nullKey
(false)
(true)
Concatenate key with IV for encryption PRNG seed.
key:= key // PrngKey! newIV
keyOk:= false
Use RC4 PRNG to generate an encrypt string at long as the MPDU payload plus the ICV field.
encryptStr:= call RC4 (key,
encryptLng+ sCrcLng)
wpdu:= substr(wpdu,0, sMacHdrLng)
// newIV // O1 // substr(wpdu, sMac_ HdrLng, encryptLng)
Insert IV and keyId between MAC header and data field.
wpdu:= setKeyId (wpdu,kndx)
en_ cipher
en_ cipher
icv:= initCrc
k:= 0, n:= sWepHdrLng +
if isWds then sWdsAddLng else 0 fi
ICV value calculated from plaintext.
icv:= crc32 (icv,wpdu(n))
Encrypt by xor of payload with encrypt string.
wpdu(n):= wpdu(n) xor encryptStr(k)
k:= k+1, n:= n+1
k = (encryptLng)
(false)
(true)
n:= 0
raw ICV is 1's complement of crc32, MSb-first
icv:= mirror(not(icv))
(icv(n) xor encryptStr(k))
wpdu:= wpdu //
Encrypt ICV octets and attach to end of Mpdu.
k:= k+1, n:= n+1
n = sCrcLng
(false)
(true)
keyOk:= true
wepdu:= setWepBit (wepdu,1),
Set WEP bit in Frame Control field.

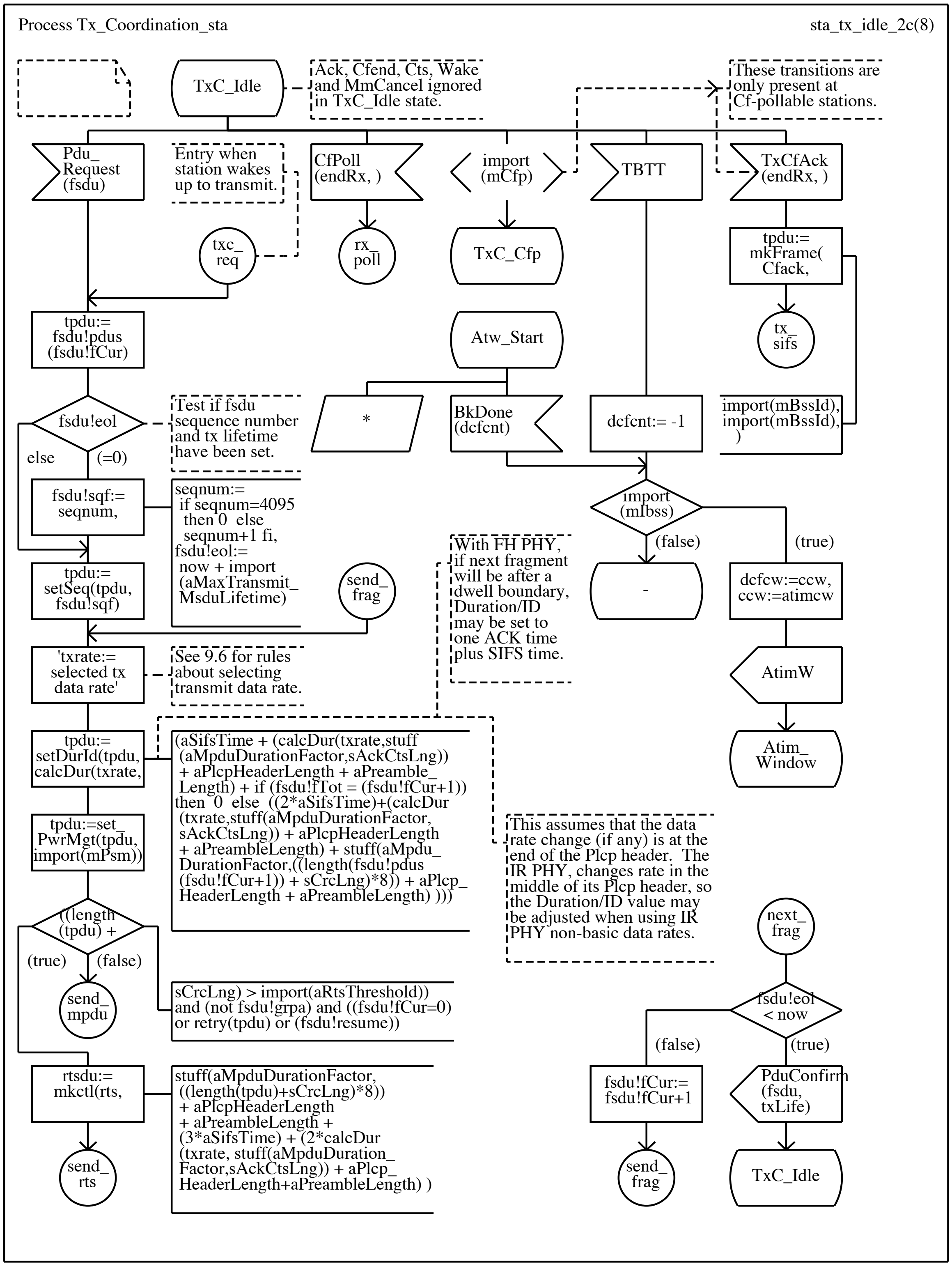

RSDU
TPDU
Block Protocol_Control_STA
MsduIndicate
AtimW,
PduConfirm,
CfPolled
sta_CTL_1a(1)
signal
Ack(Time,Rate),
Cfend,
Cfpoll(Time,Rate),
Cts(Time,Rate),
TxCfAck(Time,Rate) ;
/* This block performs the DCF functions, as well as CF-responder functions if the station is CF-pollable. Tx_Coordination includes RTS and ATIM generation. Rx_Coordination generates acknowledgements, routes data frames to MAC data service and management frames to MLME, an indicates receipt of Ack, Cts, and CF-Poll frames to Tx_Coordination. */
Tdat
Rdat
Includes the CF responder if station is Cf-pollable.
PduRequest
Doze,
MmCancel,
SwChnl,
Tbtt,
Wake
Tmgt
PsmDone,
SwDone
MCTL
Done,
TxConfirm
Tx_Coordination_sta
(1,1)
/* station version */
PlmeGet_
.confirm,
PlmeSet_
.confirm,
Plme_
Reset_
.confirm
Ack,
Cts,
Cfend,
Cfpoll,
TxCfAck
BcMgt
MmIndicate,
SsInquiry
TxO
SsResponse
Backoff,
Cancel,
TxRequest
Pctl
Rctl
TxRx
Rx_Coordination
(1,1)
/* for STA and AP */
Trsp
TX
TxRequest
TxConfirm
RxIndicate,
NeedAck,
RxCfAck,
RxCfPoll
PlmeGet_
.request,
PlmeSet_
.request,
PlmeReset_
.request
RxI
ChangeNav
MLME_PLME_SAP
RX

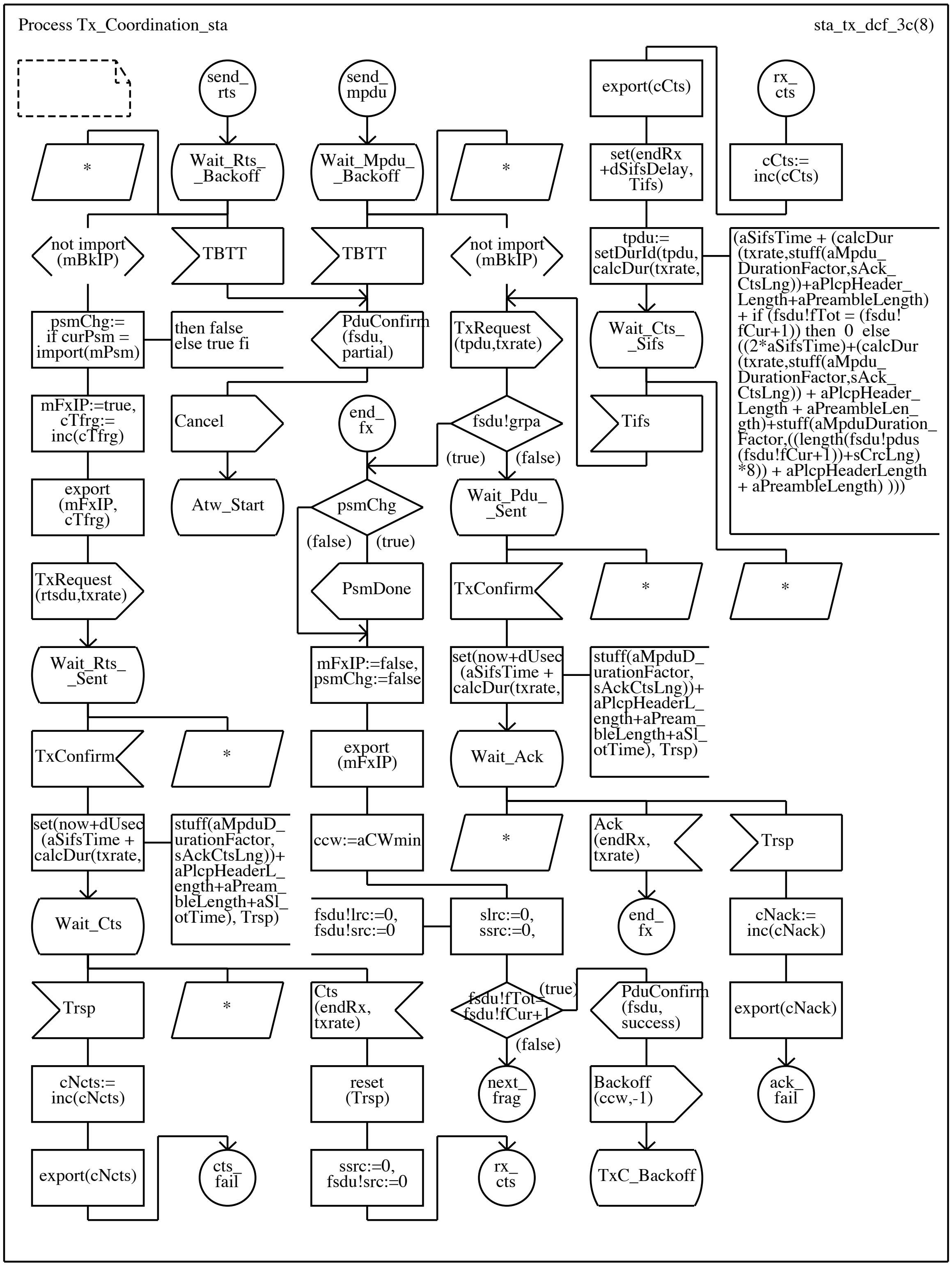

Process Rx_Coordination
rx_coord_1a(3)
* (RxC_Idle)
timer Tsifs ;
aRxTxTurn_ aroundTime)
dSifsDly:= dUsec (aSifsTime -
ResetMAC
dcl ackFrom, ackTo MacAddr ;
dcl dAck, dCts, dRsp, dSifsDly Duration ;
dcl endRx, strTs Time ;
dcl pdu, rspdu Frame ;
dcl rxRate Rate ;
dcl sas, sau StationState ;
imported mNavEnd Time ;
first(import (mBrates)), stuff (aMpdu_ Duration_ Factor, sAckCtsLng + aPlcpHdr_ Length) + aPream_ bleLength))
dRsp:=dUsec(aSifsTime + calcDur(
Duration of PS-Poll and Ack response.
reset(Tsifs)
No_Bss
The rest of No_Bss state is on 3rd page.
RxC_Idle
RxC_Idle state continues on next page.
import (mDisable)
NeedAck (ackTo,endRx, dAck,rxRate)
RxCfAck (ackFrom)
RxCfPoll
dAck:= dAck - if dAck>0 then dRsp else 0 fi
Ack(0,0)
No parameter values because without CfPoll during Cfp the transmitter cannot send after this ack.
send_ sifs
mkOs(dAck), ackTo)
rspdu:= mkCtl (ack,
-
set(endRx+ dSifsDly, Tsifs)
Wait_Sifs
*
Tsifs
RxCfPoll (endRx, rxRate)
Receipt of RxCfPoll while waiting to send result of NeedAck cancels regular Ack wait and reports the need for +cfAck to TxCoord, which will be in a Sifs wait when this signal arrives.
TxRequest (rspdu, rxRate)
reset (Tsifs)
Wait_TxDone
CfPoll (endRx, rxRate)
*
TxConfirm
TxCfAck (endRx, rxRate)
RxC_Idle
RxC_Idle

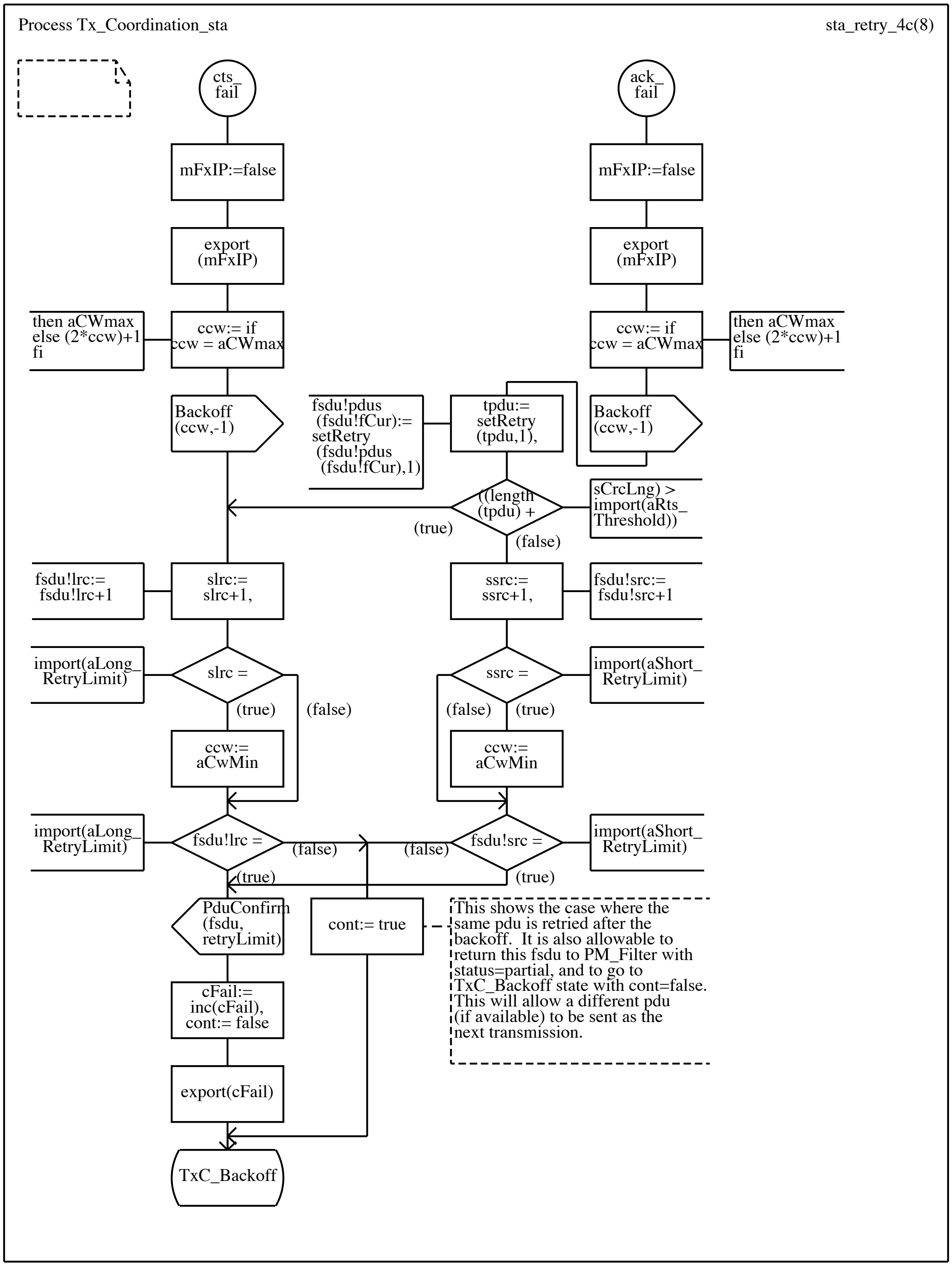

Process Rx_Coordination
rx_coord_2a(3)
RxC_Idle
RxC_Idle state
is continued
from previous page.
RxIndicate
(pdu,endRx,
strTs,rxRate)
Class 1 frames handled
on this page, class 2 and
3 frames on next page.
ftype
(pdu)
(ack)
data
Ack
(endRx,
rxRate)
(cts)
(authentication,
deauthentication,
atim,
probe_rsp)
import
(mIbss)
(true)
(false)
-
Cts
(endRx,
rxRate)
isGroup
(addr1(pdu))
(false)
(cfend_ack)
(true)
Ack(0,0)
-
Mm_
Indicate
(pdu,
if import(mCfp)
then contention_free
else contention fi)
(cfend)
else
CfEnd
chk_
sst
None of these
frames should
have group DA.
RxC_Idle
SsInquiry
(addr2(pdu))
(beacon,
probe_req)
(rts)
Wait_Asoc_
_Response
MmIndicate
(pdu,
endRx,
strTs,
noerr)
*
SsResponse
(,sas,sau)
import(
mNavEnd)
> now
(true)
(false)
-
rspdu:=
mkCtl
(cts,
durId(rspdu)-dRsp,
addr2(pdu))
sas =
asoc
(true)
(false)
send_
sifs
CTS respone to
RTS only when
the Nav is clear.
ck_
auth
Mm_
Indicate
(pdu,
if import(mCfp)
then contention_free
else contention fi)
RxC_Idle

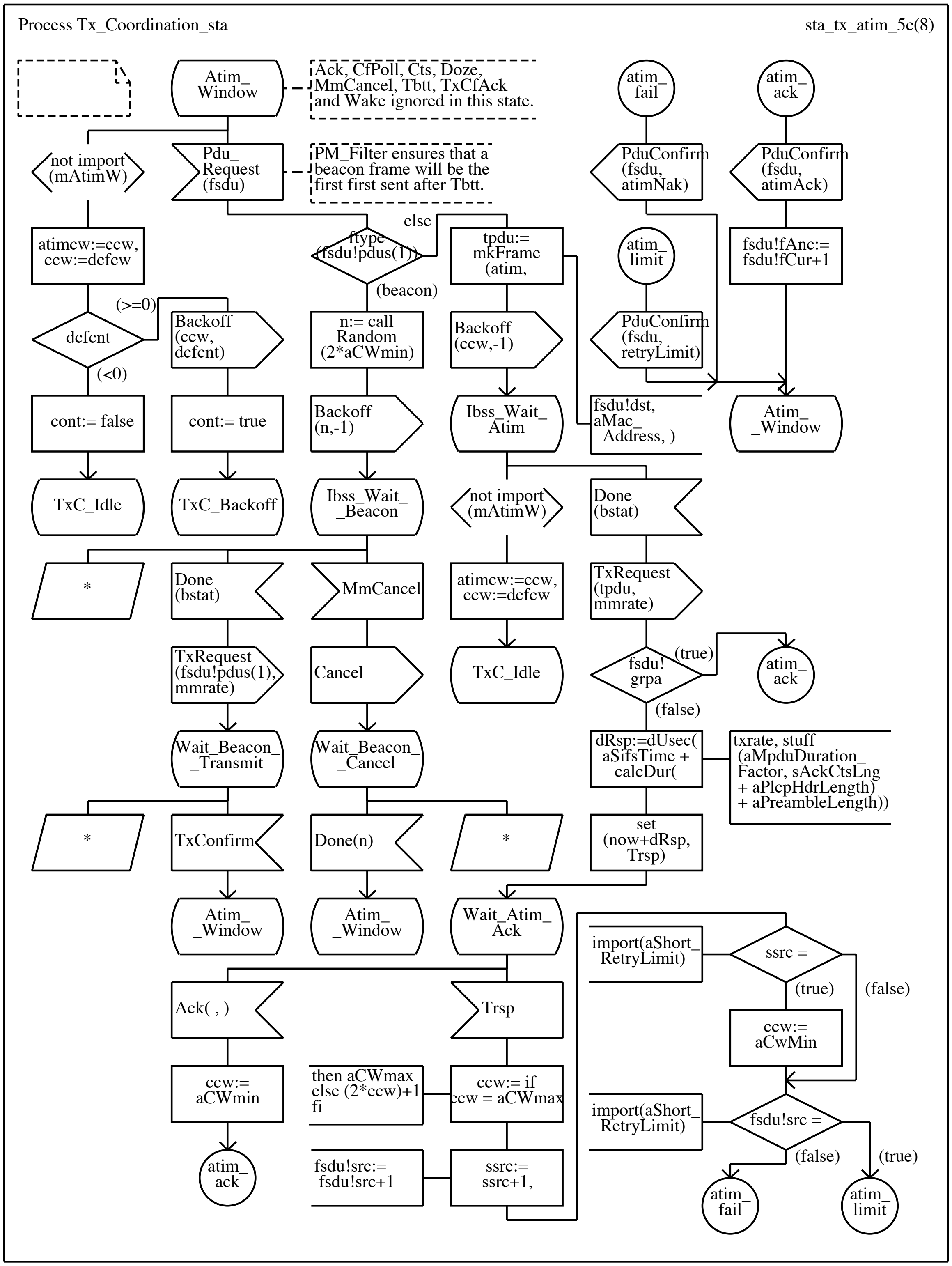

Process Rx_Coordination
rx_coord_3b(3)
No_Bss
Beacon and probe_rsp sent to Mlme_Req_Rsp while scaning, other types acknowledged (if unicast to this station) but ignored.
not import (mDisable)
RxIndicate (pdu,endRx, strTs,dAck)
RxC_Idle
ftype(pdu)
else
(beacon, probe_rsp)
MmIndicate (pdu,endRx, strTs,noerr)
RxC_Idle
chk_ sst
SsInquiry (addr2(pdu))
Wait_Sst_ _Response
At station Rx with toDs=1 discarded by Filter_MPDU. frDs=1 never sent by Sta, so explicit fromDs test not needed here.
SsResponse (,sas,sau)
*
ftype(pdu)
else
RxC_Idle
(null_frame, disasoc, asoc_req, reasoc_req, asoc_rsp, reasoc_rsp)
(pspoll)
(data_ack, data_poll, data_poll_ack, cfack, cfpoll, cfpoll_ack)
ck_ auth
(sau = authOpen) or
(sau = authKey)
(false)
(true)
import (mActingAsAp)
(true)
(false)
MmIndicate (pdu, , , class2)
PsPoll should not be received at station.
RxC_Idle
(sas=asoc) and sCfPollable
(true)
(false)
ftype(pdu)
(null_ _frame)
else
RxC_Idle
MmIndicate (pdu, , ,noerr)
PsPoll (pdu,endRx, rxrate)
ftype(pdu)
else
(data_ack, data_poll, data_poll_ack)
sau = not_auth
(false)
(true)
MmIndicate (pdu, , , class3)
Signal receipt of PsPoll to AP transmit coordination.
Msdu_ Indicate (pdu)
RxC_Idle
RxC_Idle
RxC_Idle
MmIndicate (pdu, , , class2)
RxC_Idle

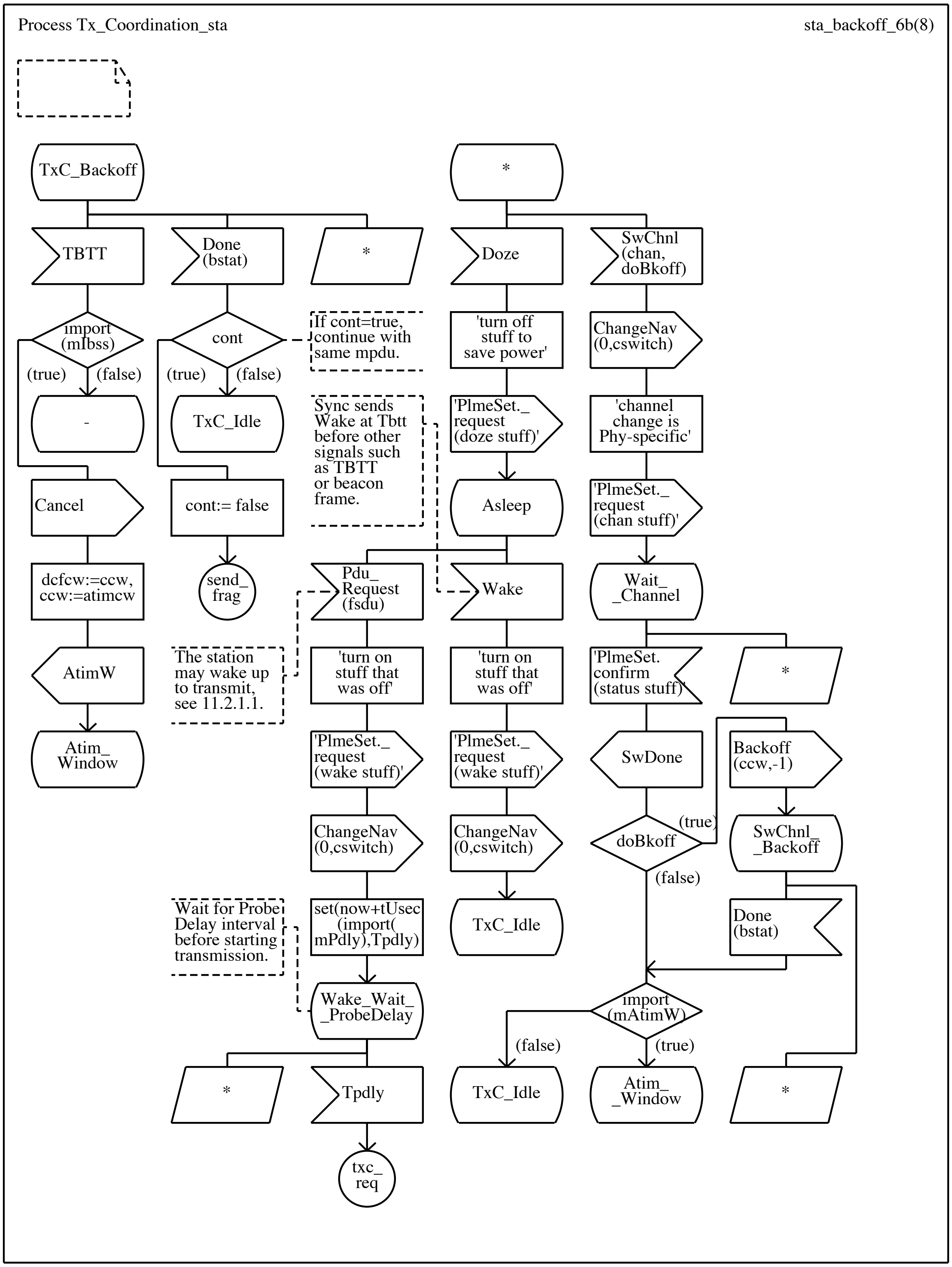

Process Tx_Coordination_sta
sta_tx_init_1c(8)
*
ResetMAC
PlmeReset._ Request
dSifsDelay:= dUsec (aSifsTime -
aRxTxTurn_ aroundTime)
'mmrate:= rate to send mmpdus'
Mmrate must be selected from mBrates. Other selection criteria are not specified.
ssrc:= 0, slrc:= 0
ccw:= import (aCWmin),
dcfcw:= ccw, atimcw:= ccw
Backoff (ccw,-1)
TxC_Idle
/* at start of frame exchange sequence, when setting mFxIP, check if mPsm=curPsm, if not, when indicating the new Psm, also set PsmChange boolean; at end of frame exchange sequence, when clearing FxIP, test & reset mPsmChange, if true, send PsmDone to Mlme */
tx_ sifs
Send frame at Sifs
set(endRx +dSifsDelay, Tifs)
tpdu:= setPwrMgt (tpdu,curPm)
Wait_Sifs
Tifs
*
TxRequest (tpdu,trate)
Wait_Tx_ Done
TxConfirm
*
TxC_Idle
dcl atimcw, bstat, chan, dcfcnt, dcfcw Integer ; dcl ccw Integer:= aCwMin ; dcl curPm Bit ; dcl doHop, psmChg, cont Boolean:= false ; dcl dSifsDelay, endRx Time ; dcl fsdu FragSdu ; dcl rtype Ftype ; dcl seqnum, ssrc, slrc, n Integer:= 0; dcl tpdu Frame ; dcl txrate Rate ;
timer Tifs, Trsp, Tpdly ;
Imported aRtsThreshld, aShortRetryLimit, aLongRetryLimit, aFragmentationThreshold, aMaxTransmitMsduLifetime Integer, mPdly Usec ;
/* RANDOM NUMBER FUNCTION */ imported procedure Random ; fpar limit Integer ; returns Integer ;
dcl exported FxIP Boolean:= false ; dcl cTfrg exported as aTransmittedFragmentCount, cTmcfrg exported as aMulticastTransmittedFrameCount, cFail exported as aFailedCount, cRtry exported as aRetryCount, cMrtry exported as aMultipleRetryCount, cCts exported as aRtsSuccessCount, cNcts exported as aRtsFailureCount, cNack exported as aAckFailureCount Counter32:= 0 ;


Process Tx_Coordination_sta
sta_tx_idle_2c(8)
TxC_Idle
Ack, Cfend, Cts, Wake and MmCancel ignored in TxC_Idle state.
These transitions are only present at Cf-pollable stations.
Pdu_ Request (fsdu)
Entry when station wakes up to transmit.
CfPoll (endRx,)
import (mCfp)
TBTT
TxCfAck (endRx,)
txc_ req
rx_ poll
TxC_Cfp
tpdu:= mkFrame(Cfack,
tx_ sifs
tpdu:= fsdu!pdus (fsdu!fCur)
Atw_Start
fsdu!eol
Test if fsdu sequence number and tx lifetime have been set.
*
BkDone (dcfcnt)
dcfcnt:= -1
import(mBssId), import(mBssId),)
else
(=0)
fsdu!sqf:= seqnum,
seqnum:= if seqnum=4095 then 0 else seqnum+1 fi, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
import (mIbss)
(false)
(true)
With FH PHY, if next fragment will be after a dwell boundary, Duration/ID may be set to one ACK time plus SIFS time.
tpdu:= setSeq(tpdu, fsdu!sqf)
send_ frag
-
dcfcw:=ccw, ccw:=atimcw
'txrate:= selected tx data rate'
See 9.6 for rules about selecting transmit data rate.
AtimW
tpdu:= setDurId(tpdu, calcDur(txrate,
(aSifsTime + (calcDur(txrate,stuff (aMpduDurationFactor,sAckCtsLng)) + aPlcpHeaderLength + aPreamble_ Length) + if (fsdu!fTot = (fsdu!fCur+1)) then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpduDurationFactor, sAckCtsLng)) + aPlcpHeaderLength + aPreambleLength) + stuff(aMpdu_ DurationFactor,((length(fsdu!pdus (fsdu!fCur+1)) + sCrcLng)*8)) + aPlcp_ HeaderLength + aPreambleLength))))
Atim_ Window
tpdu:=set_ PwrMgt(tpdu, import(mPsm))
This assumes that the data rate change (if any) is at the end of the Plcp header. The IR PHY, changes rate in the middle of its Plcp header, so the Duration/ID value may be adjusted when using IR PHY non-basic data rates.
((length (tpdu) +
(true)
(false)
next_ frag
send_ mpdu
sCrcLng) > import(aRtsThreshold)) and (not fsdu!grpa) and ((fsdu!fCur=0) or retry(tpdu) or (fsdu!resume))
fsdu!eol < now
(false)
(true)
rtsdu:= mkctl(rts,
stuff(aMpduDurationFactor, ((length(tpdu)+sCrcLng)*8)) + aPlcpHeaderLength + aPreambleLength + (3*aSifsTime) + (2*calcDur (txrate, stuff(aMpduDuration_ Factor,sAckCtsLng)) + aPlcp_ HeaderLength+aPreambleLength))
fsdu!fCur:= fsdu!fCur+1
PduConfirm (fsdu, txLife)
send_ rts
send_ frag
TxC_Idle

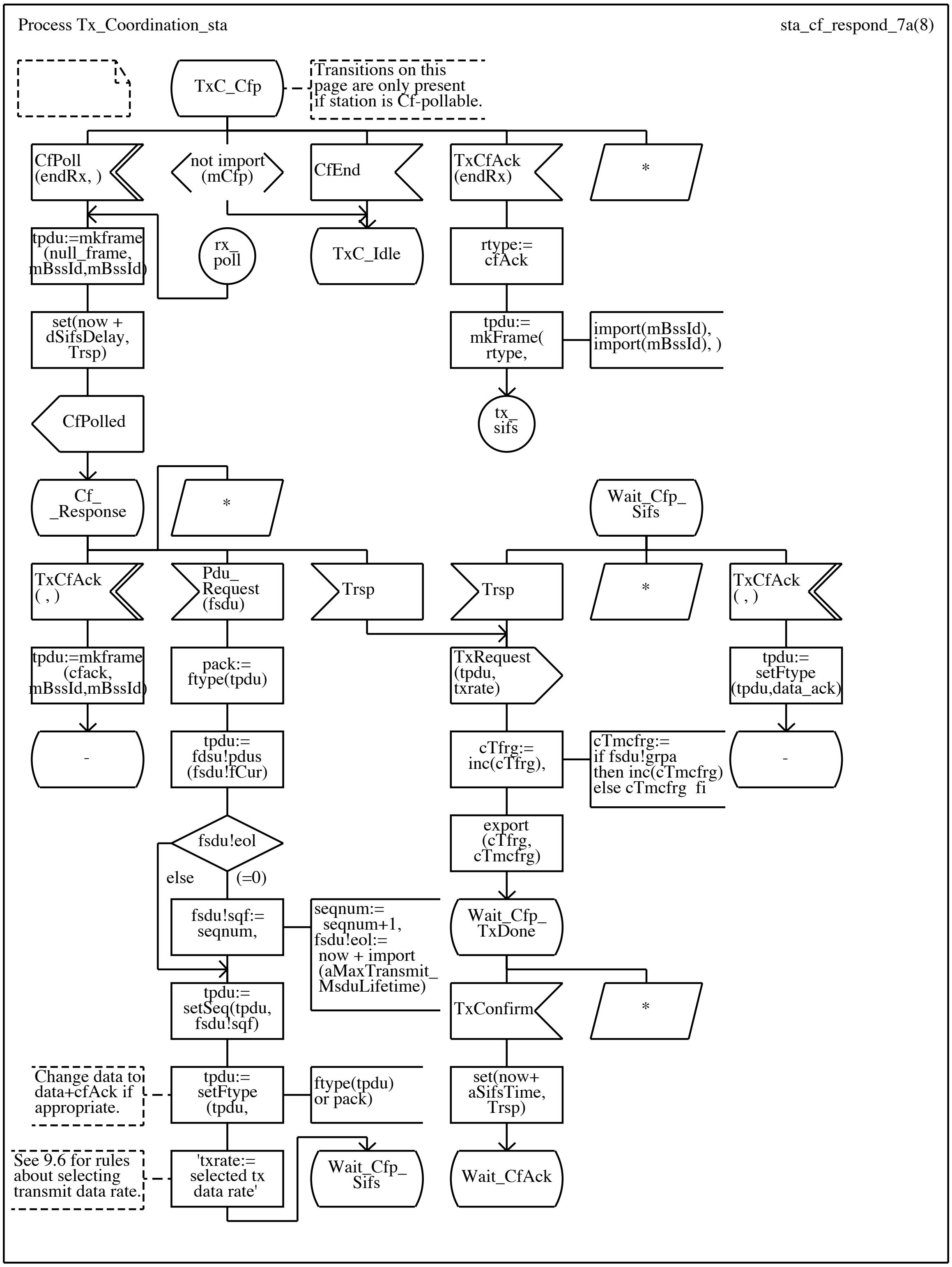

Process Tx_Coordination_sta
sta_tx_dcf_3c(8)
send_ rts
send_ mpdu
export(cCts)
rx_ cts
*
Wait_Rts_ _Backoff
Wait_Mpdu_ _Backoff
*
set(endRx +dSifsDelay, Tifs)
cCts:= inc(cCts)
not import (mBkIP)
TBTT
TBTT
not import (mBkIP)
tpdu:= setDurId(tpdu, calcDur(txrate,
(aSifsTime + (calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng))+aPlcpHeader_ Length+aPreambleLength) + if (fsdu!fTot = (fsdu! fCur+1)) then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng)) + aPlcpHeader_ Length + aPreambleLen_ gth)+stuff(aMpduDuration_ Factor,((length(fsdu!pdus (fsdu!fCur+1))+sCrcLng) *8)) + aPlcpHeaderLength + aPreambleLength))))
psmChg:= if curPsm = import(mPsm)
then false else true fi
PduConfirm (fsdu, partial)
TxRequest (tpdu,txrate)
Wait_Cts_ _Sifs
mFxIP:=true, cTfrg:= inc(cTfrg)
Cancel
end_ fx
fsdu!grpa
Tifs
(true)
(false)
export (mFxIP, cTfrg)
Atw_Start
psmChg
Wait_Pdu_ _Sent
(false)
(true)
TxRequest (rtsdu,txrate)
PsmDone
TxConfirm
*
*
Wait_Rts_ _Sent
mFxIP:=false, psmChg:=false
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD_ urationFactor, sAckCtsLng))+ aPlcpHeaderL_ ength+aPream_ bleLength+aSl_ otTime), Trsp)
TxConfirm
*
export (mFxIP)
Wait_Ack
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD_ urationFactor, sAckCtsLng))+ aPlcpHeaderL_ ength+aPream_ bleLength+aSl_ otTime), Trsp)
ccw:=aCWmin
*
Ack (endRx, txrate)
Trsp
Wait_Cts
fsdu!lrc:=0, fsdu!src:=0
slrc:=0, ssrc:=0,
end_ fx
cNack:= inc(cNack)
Trsp
*
Cts (endRx, txrate)
fsdu!fTot= fsdu!fCur+1
(true)
PduConfirm (fsdu, success)
export(cNack)
(false)
cNcts:= inc(cNcts)
reset (Trsp)
next_ frag
Backoff (ccw,-1)
ack_ fail
export(cNcts)
cts_ fail
ssrc:=0, fsdu!src:=0
rx_ cts
TxC_Backoff

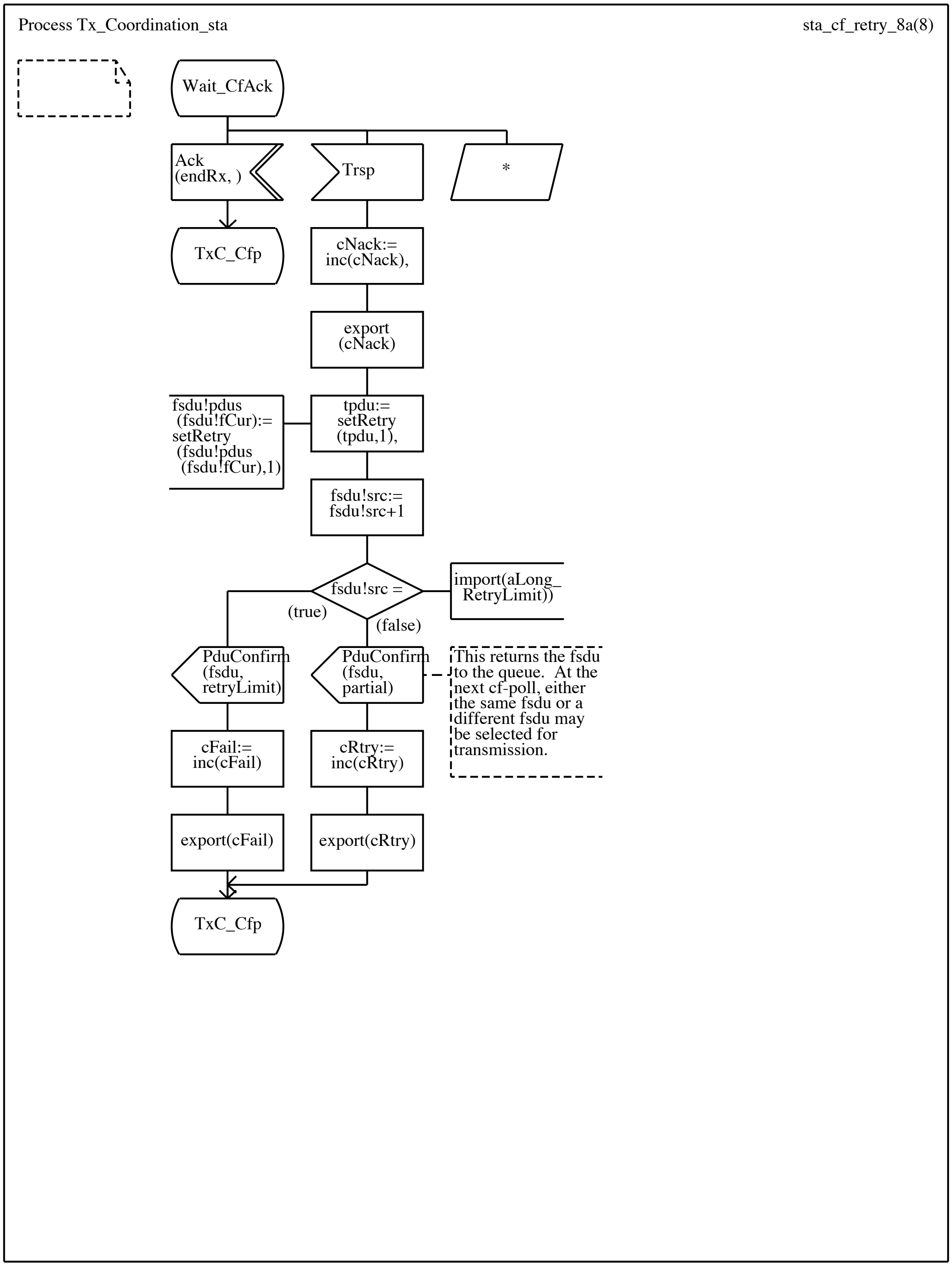

Process Tx_Coordination_sta
sta_retry_4c(8)
cts_ fail
ack_ fail
mFxIP:=false
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw,-1)
fsdu!pdus (fsdu!fCur):= setRetry (fsdu!pdus (fsdu!fCur),1)
tpdu:= setRetry (tpdu,1),
((length (tpdu) +
sCrcLng) > import(aRts_ Threshold))
(true)
(false)
fsdu!lrc:= fsdu!lrc+1
slrc:= slrc+1,
ssrc:= ssrc+1,
fsdu!src:= fsdu!src+1
import(aLong_ RetryLimit)
slrc =
ssrc =
import(aShort_ RetryLimit)
ccw:= aCwMin
fsdu!lrc =
fsdu!src =
PduConfirm (fsdu, retryLimit)
cont:= true
This shows the case where the same pdu is retried after the backoff. It is also allowable to return this fsdu to PM_Filter with status=partial, and to go to TxC_Backoff state with cont=false. This will allow a different pdu (if available) to be sent as the next transmission.
cFail:= inc(cFail), cont:= false
export(cFail)
TxC_Backoff

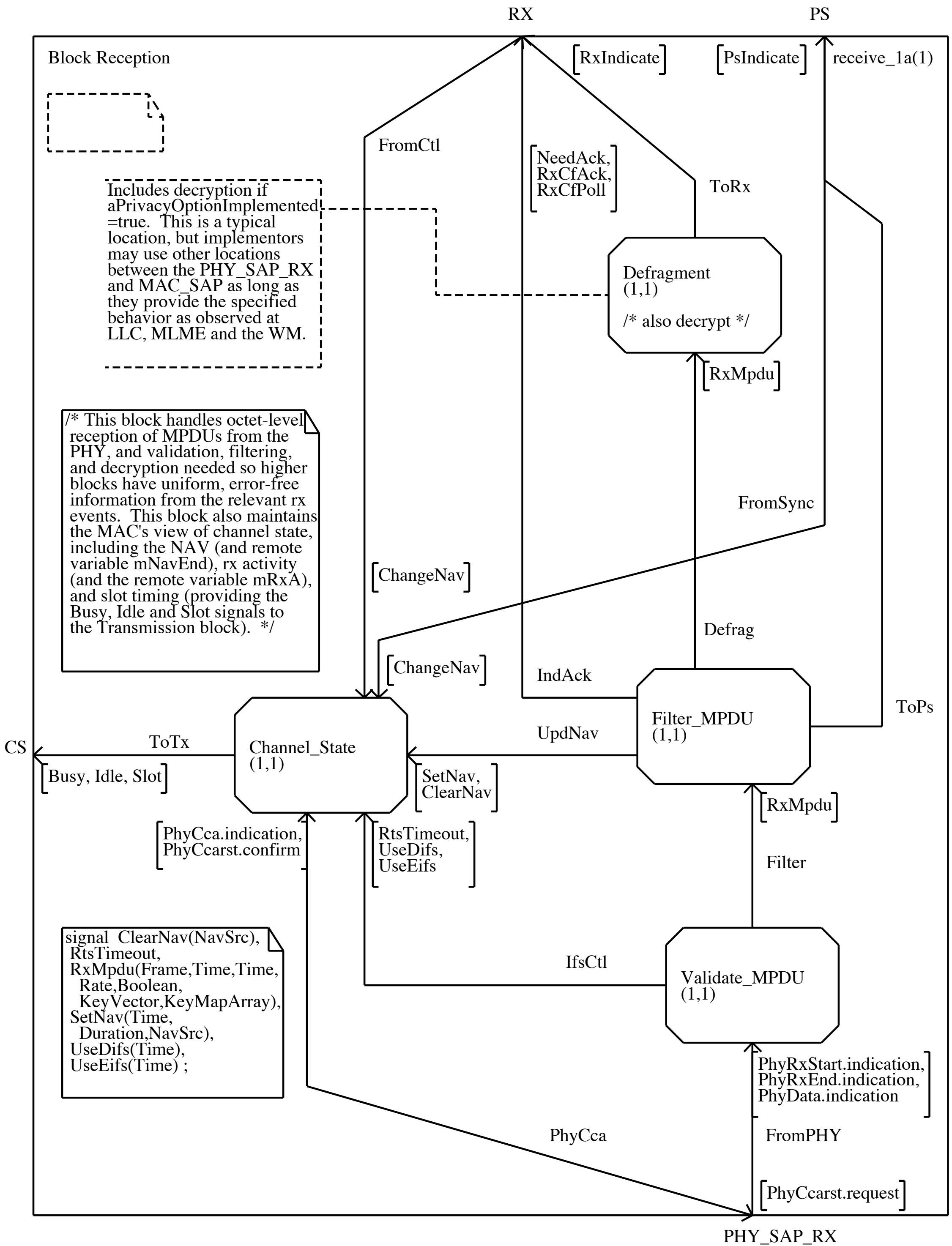

Process Tx_Coordination_sta
sta_tx_atim_5c(8)
Atim_ Window
Ack, CfPoll, Cts, Doze, MmCancel, Tbtt, TxCfAck and Wake ignored in this state.
atim_ fail
atim_ ack
not import (mAtimW)
Pdu_ Request (fsdu)
PM_Filter ensures that a beacon frame will be the first first sent after Tbtt.
PduConfirm (fsdu, atimNak)
PduConfirm (fsdu, atimAck)
else
ftype (fsdu!pdus(1))
tpdu:= mkFrame (atim,
atimcw:=ccw, ccw:=dcfcw
atim_ limit
fsdu!fAnc:= fsdu!fCur+1
(beacon)
(>=0)
dcfcnt
Backoff (ccw, dcfcnt)
n:= call Random (2*aCWmin)
Backoff (ccw,-1)
PduConfirm (fsdu, retryLimit)
(<0)
cont:= false
cont:= true
Backoff (n,-1)
Ibss_Wait_ Atim
fsdu!dst, aMac_ Address,)
Atim_ _Window
TxC_Idle
TxC_Backoff
Ibss_Wait_ _Beacon
not import (mAtimW)
Done (bstat)
*
Done (bstat)
MmCancel
atimcw:=ccw, ccw:=dcfcw
TxRequest (tpdu, mmrate)
TxRequest (fsdu!pdus(1), mmrate)
Cancel
TxC_Idle
fsdu! grpa
(true)
atim_ ack
(false)
Wait_Beacon_ _Transmit
Wait_Beacon_ _Cancel
dRsp:=dUsec(aSifsTime + calcDur(
txrate, stuff (aMpduDuration_ Factor, sAckCtsLng + aPlcpHdrLength) + aPreambleLength))
*
TxConfirm
Done(n)
*
set (now+dRsp, Trsp)
Atim_ _Window
Atim_ _Window
Wait_Atim_ Ack
import(aShort_ RetryLimit)
ssrc =
(true)
(false)
Ack(,)
Trsp
ccw:= aCwMin
ccw:= aCWmin
then aCWmax else (2*ccw)+1 fi
ccw:= if ccw = aCWmax
import(aShort_ RetryLimit)
fsdu!src =
(false)
(true)
atim_ ack
fsdu!src:= fsdu!src+1
ssrc:= ssrc+1,
atim_ fail
atim_ limit

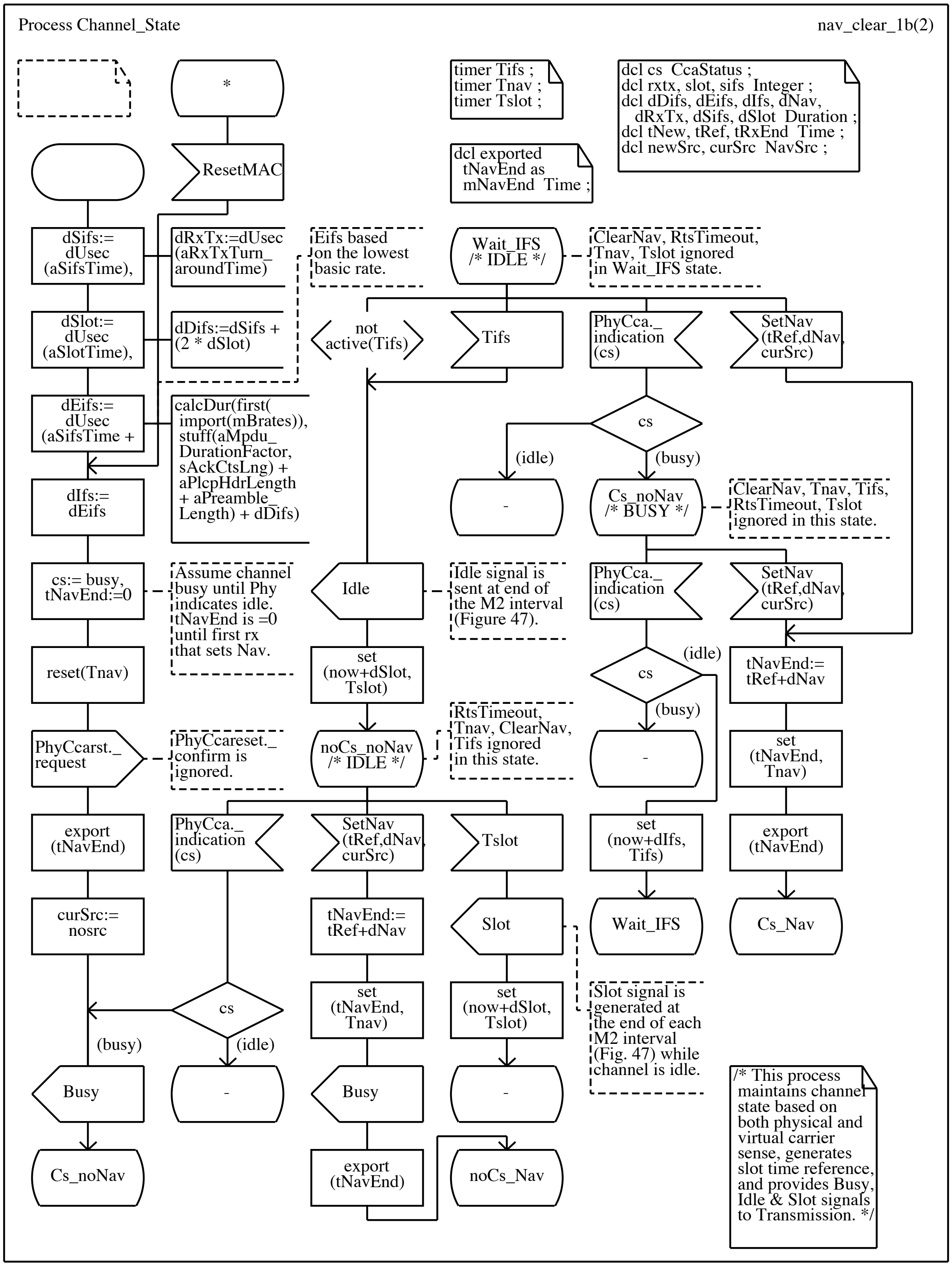


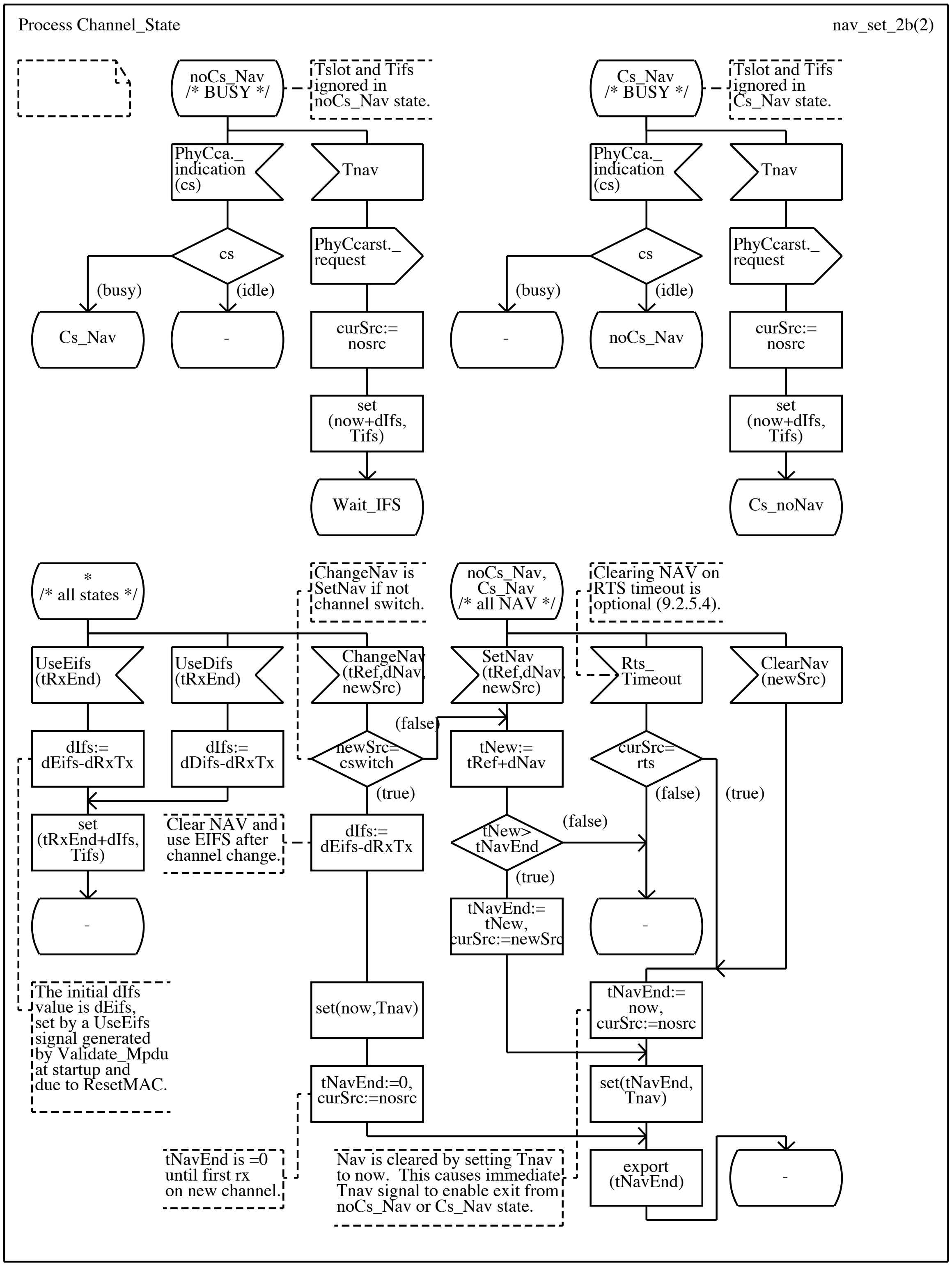
Process Tx_Coordination_sta
sta_backoff_6b(8)
TxC_Backoff
*
TBTT
Done (bstat)
*
Doze
SwChnl (chan, doBkoff)
import (mIbss)
(true)
(false)
cont
(true)
(false)
If cont=true, continue with same mpdu.
'turn off stuff to save power'
ChangeNav (0,cswitch)
-
TxC_Idle
Sync sends Wake at Tbtt before other signals such as TBTT or beacon frame.
'PlmeSet._ request (doze stuff)'
'channel change is Phy-specific'
Cancel
cont:= false
Asleep
'PlmeSet._ request (chan stuff)'
dcfcw:=ccw, ccw:=atimcw
send_ frag
Pdu_ Request (fsdu)
Wake
Wait_ _Channel
AtimW
The station may wake up to transmit, see 11.2.1.1.
'turn on stuff that was off'
'turn on stuff that was off'
'PlmeSet. confirm (status stuff)'
*
Atim_ Window
'PlmeSet._ request (wake stuff)'
'PlmeSet._ request (wake stuff)'
SwDone
Backoff (ccw,-1)
ChangeNav (0,cswitch)
ChangeNav (0,cswitch)
doBkoff
(true)
(false)
SwChnl_ _Backoff
Wait for Probe Delay interval before starting transmission.
set(now+tUsec (import(mPdly),Tpdly)
TxC_Idle
Done (bstat)
Wake_Wait_ _ProbeDelay
import (mAtimW)
(false)
(true)
*
Tpdly
TxC_Idle
Atim_ _Window
*
txc_ req

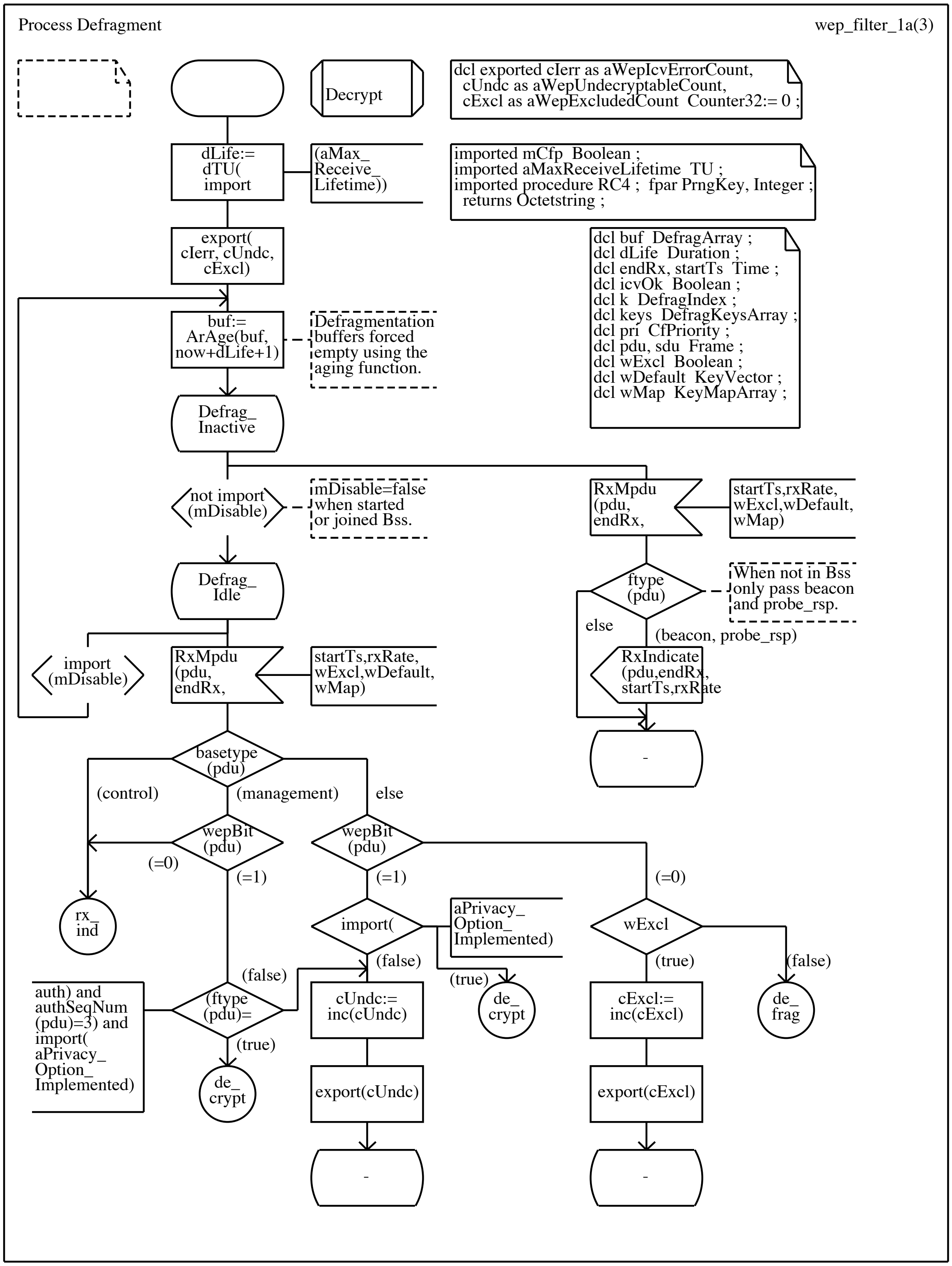

Process Tx_Coordination_sta
sta_cf_respond_7a(8)
TxC_Cfp
Transitions on this page are only present if station is Cf-pollable.
CfPoll (endRx,)
not import (mCfp)
CfEnd
TxCfAck (endRx)
*
tpdu:=mkframe (null_frame, mBssId,mBssId)
rx_ poll
TxC_Idle
rtype:= cfAck
set(now + dSifsDelay, Trsp)
tpdu:= mkFrame(rtype,
import(mBssId), import(mBssId),)
tx_ sifs
CfPolled
Cf_ _Response
*
Wait_Cfp_ Sifs
TxCfAck (,)
Pdu_ Request (fsdu)
Trsp
Trsp
*
TxCfAck (,)
tpdu:=mkframe (cfack, mBssId,mBssId)
pack:= ftype(tpdu)
TxRequest (tpdu, txrate)
tpdu:= setFtype (tpdu,data_ack)
-
tpdu:= fdsu!pdus (fsdu!fCur)
cTfrg:= inc(cTfrg),
cTmcfrg:= if fsdu!grpa then inc(cTmcfrg) else cTmcfrg fi
-
fsdu!eol
else
(=0)
export (cTfrg, cTmcfrg)
fsdu!sqf:= seqnum,
seqnum:= seqnum+1, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
Wait_Cfp_ TxDone
tpdu:= setSeq(tpdu, fsdu!sqf)
TxConfirm
*
Change data to data+cfAck if appropriate.
tpdu:= setFtype (tpdu,
ftype(tpdu) or pack)
set(now+ aSifsTime, Trsp)
See 9.6 for rules about selecting transmit data rate.
'txrate:= selected tx data rate'
Wait_Cfp_ Sifs
Wait_CfAck

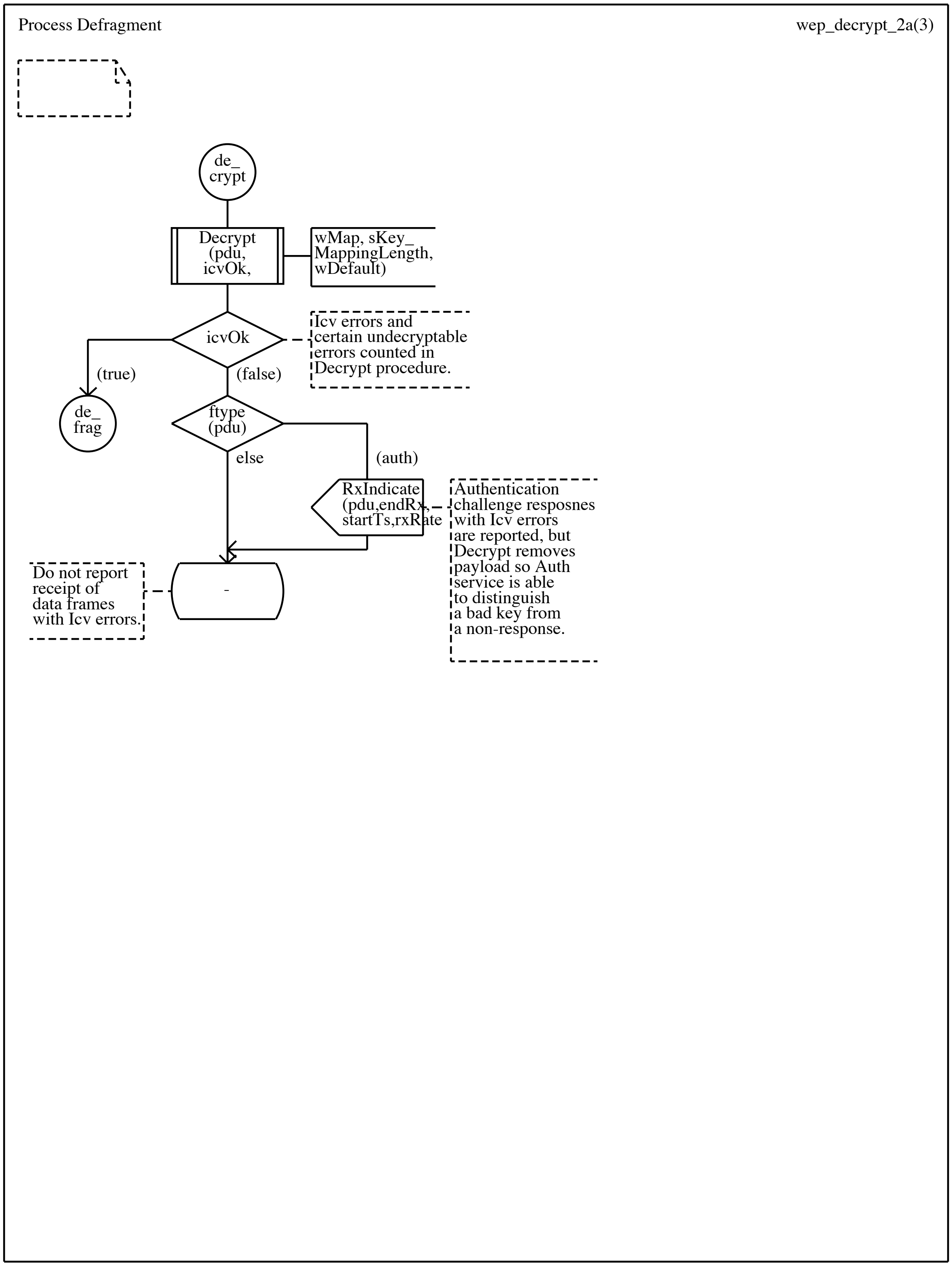

Process Tx_Coordination_sta
sta_cf_retry_8a(8)
Wait_CfAck
Ack
(endRx,)
Trsp
*
TxC_Cfp
cNack:=
inc(cNack),
export
(cNack)
fsdu!pdus
(fsdu!fCur):=
setRetry
(fsdu!pdus
(fsdu!fCur),1)
tpdu:=
setRetry
(tpdu,1),
fsdu!src:=
fsdu!src+1
fsdu!src =
import(aLong_
RetryLimit))
(true)
(false)
PduConfirm
(fsdu,
retryLimit)
PduConfirm
(fsdu,
partial)
This returns the fsdu
to the queue. At the
next cf-poll, either
the same fsdu or a
different fsdu may
be selected for
transmission.
cFail:=
inc(cFail)
cRtry:=
inc(cRtry)
export(cFail)
export(cRtry)
TxC_Cfp

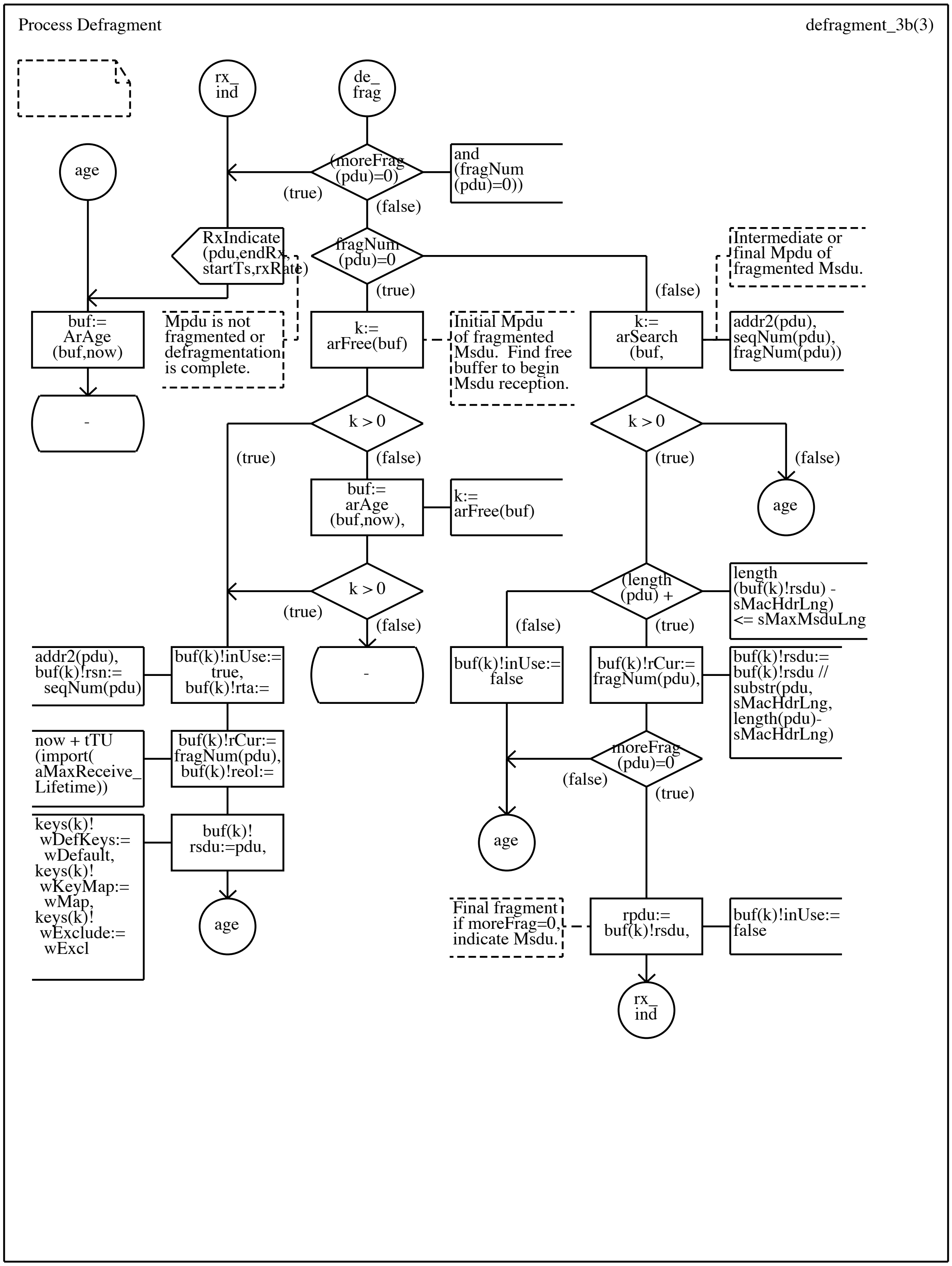

RX
PS
Block Reception
RxIndicate
PsIndicate
receive_1a(1)
FromCtl
NeedAck,
RxCfAck,
RxCfPoll
ToRx
Includes decryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the PHY_SAP_RX and MAC_SAP as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Defragment
(1,1)
/* also decrypt */
RxMpdu
/* This block handles octet-level reception of MPDUs from the PHY, and validation, filtering, and decryption needed so higher blocks have uniform, error-free information from the relevant rx events. This block also maintains the MAC's view of channel state, including the NAV (and remote variable mNavEnd), rx activity (and the remote variable mRxA), and slot timing (providing the Busy, Idle and Slot signals to the Transmission block). */
FromSync
ChangeNav
Defrag
ChangeNav
IndAck
Filter_MPDU
(1,1)
ToPs
CS
ToTx
Channel_State
(1,1)
UpdNav
Busy, Idle, Slot
SetNav,
ClearNav
RxMpdu
PhyCca.indication,
PhyCcarst.confirm
RtsTimeout,
UseDifs,
UseEifs
Filter
signal ClearNav(NavSrc),
RtsTimeout,
RxMpdu(Frame,Time,Time,
Rate,Boolean,
KeyVector,KeyMapArray),
SetNav(Time,
Duration,NavSrc),
UseDifs(Time),
UseEifs(Time) ;
IfsCtl
Validate_MPDU
(1,1)
PhyRxStart.indication,
PhyRxEnd.indication,
PhyData.indication
FromPHY
PhyCca
PhyCcarst.request
PHY_SAP_RX

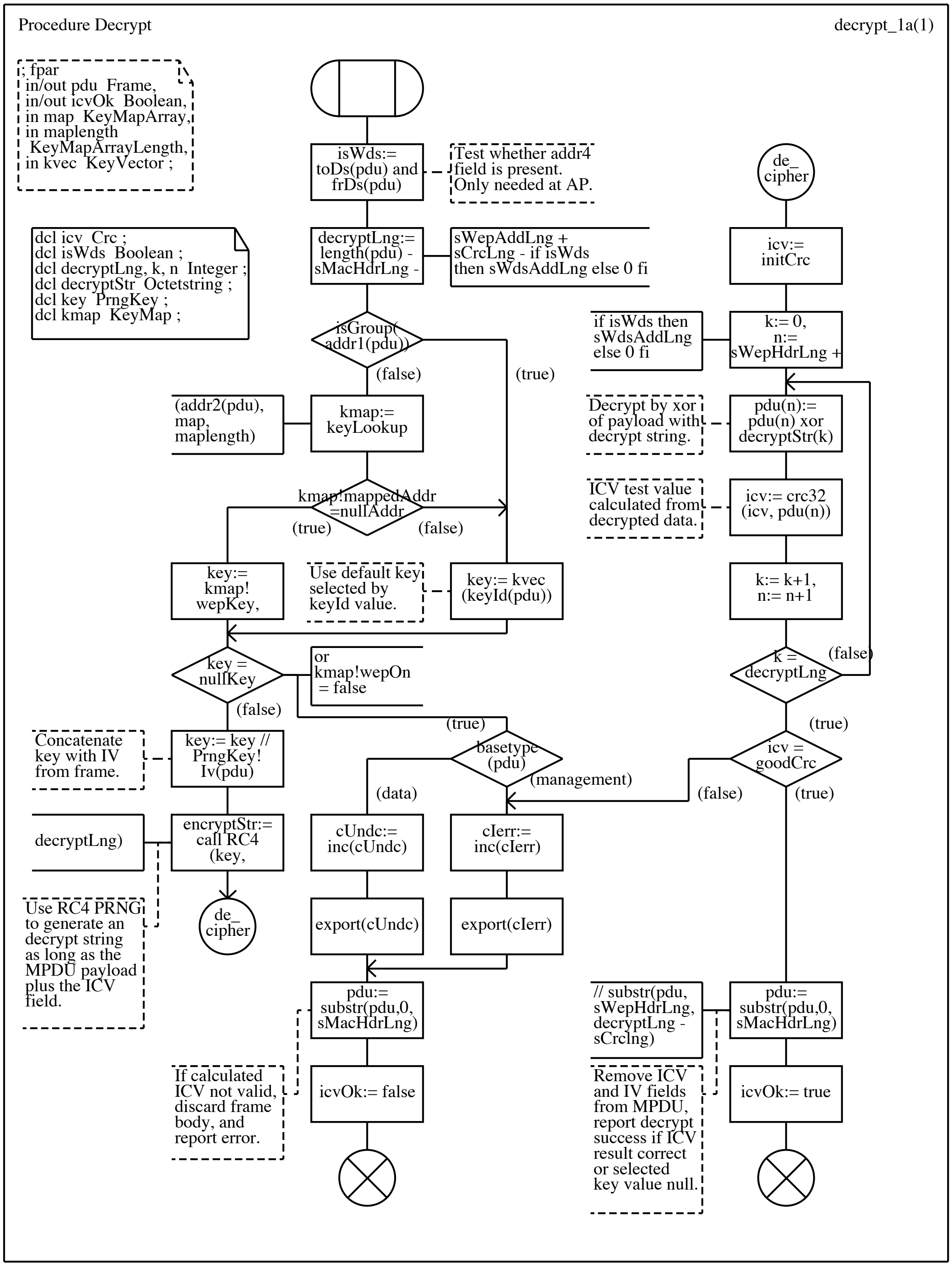

Process Channel_State
nav_clear_1b(2)
*
ResetMAC
timer Tifs ;
timer Tnav ;
timer Tslot ;
dcl cs CcaStatus ;
dcl rxtx, slot, sifs Integer ;
dcl dDifs, dEifs, dIfs, dNav,
dRxTx, dSifs, dSlot Duration ;
dcl tNew, tRef, tRxEnd Time ;
dcl newSrc, curSrc NavSrc ;
dcl exported
tNavEnd as
mNavEnd Time ;
dSifs:=
dUsec
(aSifsTime),
dRxTx:=dUsec
(aRxTxTurn_
aroundTime)
Eifs based
on the lowest
basic rate.
Wait_IFS
/* IDLE */
ClearNav, RtsTimeout,
Tnav, Tslot ignored
in Wait_IFS state.
dSlot:=
dUsec
(aSlotTime),
dDifs:=dSifs +
(2 * dSlot)
not
active(Tifs)
Tifs
PhyCca._
indication
(cs)
SetNav
(tRef,dNav,
curSrc)
dEifs:=
dUsec
(aSifsTime +
calcDur(first(
import(mBrates)),
stuff(aMpdu_
DurationFactor,
sAckCtsLng) +
aPlcpHdrLength
+ aPreamble_
Length) + dDifs)
cs
(idle)
(busy)
dIfs:=
dEifs
-
Cs_noNav
/* BUSY */
ClearNav, Tnav, Tifs,
RtsTimeout, Tslot
ignored in this state.
cs:= busy,
tNavEnd:=0
Assume channel
busy until Phy
indicates idle.
tNavEnd is =0
until first rx
that sets Nav.
Idle
Idle signal is
sent at end of
the M2 interval
(Figure 47).
PhyCca._
indication
(cs)
SetNav
(tRef,dNav,
curSrc)
reset(Tnav)
set
(now+dSlot,
Tslot)
cs
(idle)
(busy)
tNavEnd:=
tRef+dNav
PhyCcarst._
request
PhyCcareset._
confirm is
ignored.
noCs_noNav
/* IDLE */
RtsTimeout,
Tnav, ClearNav,
Tifs ignored
in this state.
-
set
(tNavEnd,
Tnav)
export
(tNavEnd)
PhyCca._
indication
(cs)
SetNav
(tRef,dNav,
curSrc)
Tslot
set
(now+dIfs,
Tifs)
export
(tNavEnd)
curSrc:=
nosrc
tNavEnd:=
tRef+dNav
Slot
Wait_IFS
Cs_Nav
cs
(busy)
(idle)
set
(tNavEnd,
Tnav)
set
(now+dSlot,
Tslot)
Slot signal is
generated at
the end of each
M2 interval
(Fig. 47) while
channel is idle.
Busy
-
Busy
-
/* This process
maintains channel
state based on
both physical and
virtual carrier
sense, generates
slot time reference,
and provides Busy,
Idle & Slot signals
to Transmission. */
Cs_noNav
export
(tNavEnd)
noCs_Nav

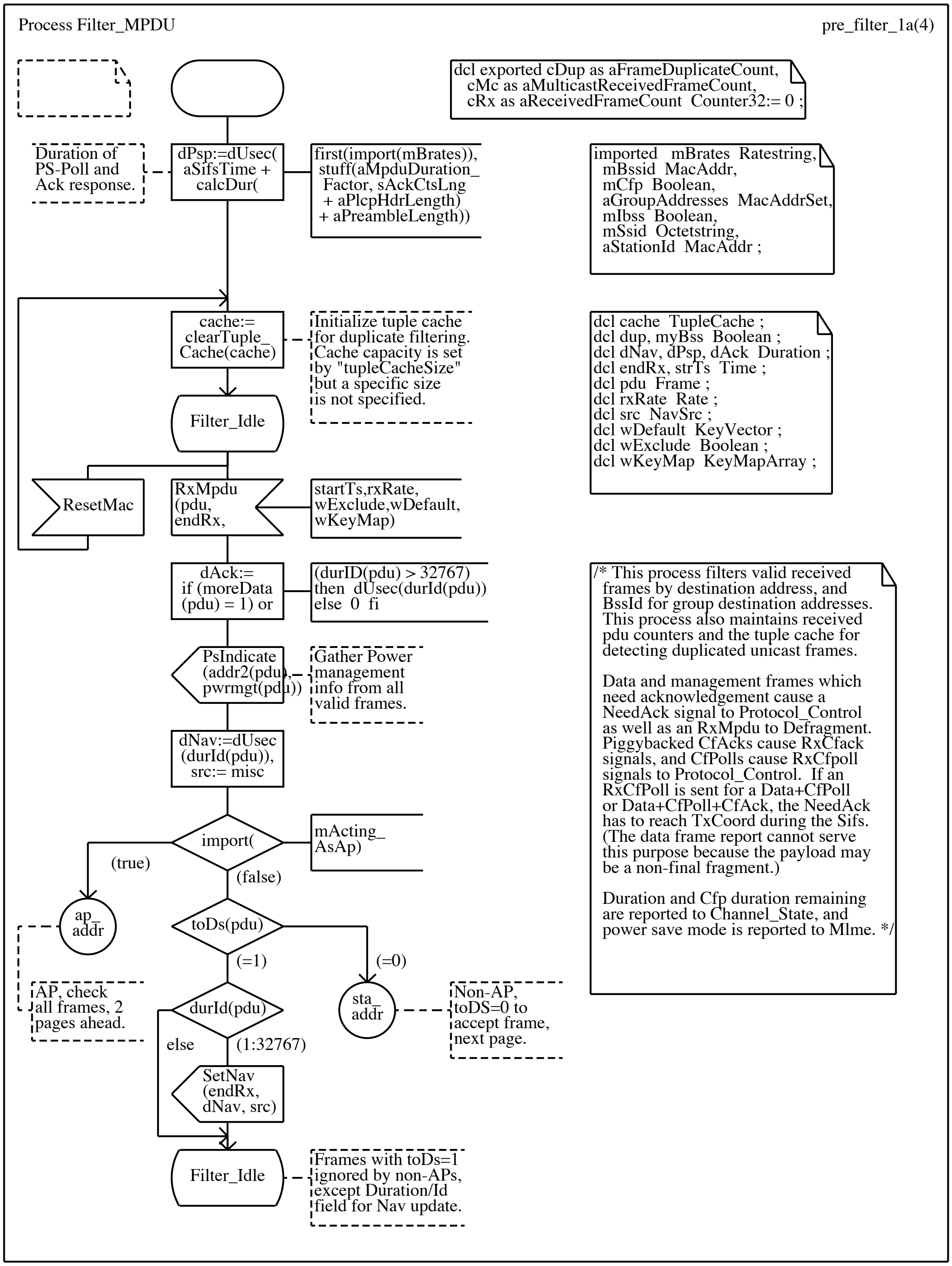

Process Channel_State
nav_set_2b(2)
noCs_Nav /* BUSY */
Tslot and Tifs ignored in noCs_Nav state.
PhyCca._ indication (cs)
Tnav
cs
(busy)
(idle)
Cs_Nav
-
PhyCcarst._ request
curSrc:= nosrc
set (now+dIfs, Tifs)
Wait_IFS
Cs_Nav /* BUSY */
Tslot and Tifs ignored in Cs_Nav state.
PhyCca._ indication (cs)
Tnav
cs
(busy)
(idle)
-
noCs_Nav
PhyCcarst._ request
curSrc:= nosrc
set (now+dIfs, Tifs)
Cs_noNav
* /* all states */
UseEifs (tRxEnd)
UseDifs (tRxEnd)
ChangeNav is SetNav if not channel switch.
ChangeNav (tRef,dNav, newSrc)
noCs_Nav, Cs_Nav /* all NAV */
Clearing NAV on RTS timeout is optional (9.2.5.4).
SetNav (tRef,dNav, newSrc)
Rts_ Timeout
ClearNav (newSrc)
dIfs:= dEifs-dRxTx
dIfs:= dDifs-dRxTx
newSrc= cswitch
(false)
(true)
tNew:= tRef+dNav
curSrc= rts
(false)
(true)
set (tRxEnd+dIfs, Tifs)
Clear NAV and use EIFS after channel change.
dIfs:= dEifs-dRxTx
tNew> tNavEnd
(false)
(true)
-
tNavEnd:= tNew, curSrc:=newSrc
-
The initial dIfs value is dEifs, set by a UseEifs signal generated by Validate_Mpdu at startup and due to ResetMAC.
set(now,Tnav)
tNavEnd:= now, curSrc:=nosrc
tNavEnd:=0, curSrc:=nosrc
set(tNavEnd, Tnav)
tNavEnd is =0 until first rx on new channel.
Nav is cleared by setting Tnav to now. This causes immediate Tnav signal to enable exit from noCs_Nav or Cs_Nav state.
export (tNavEnd)
-

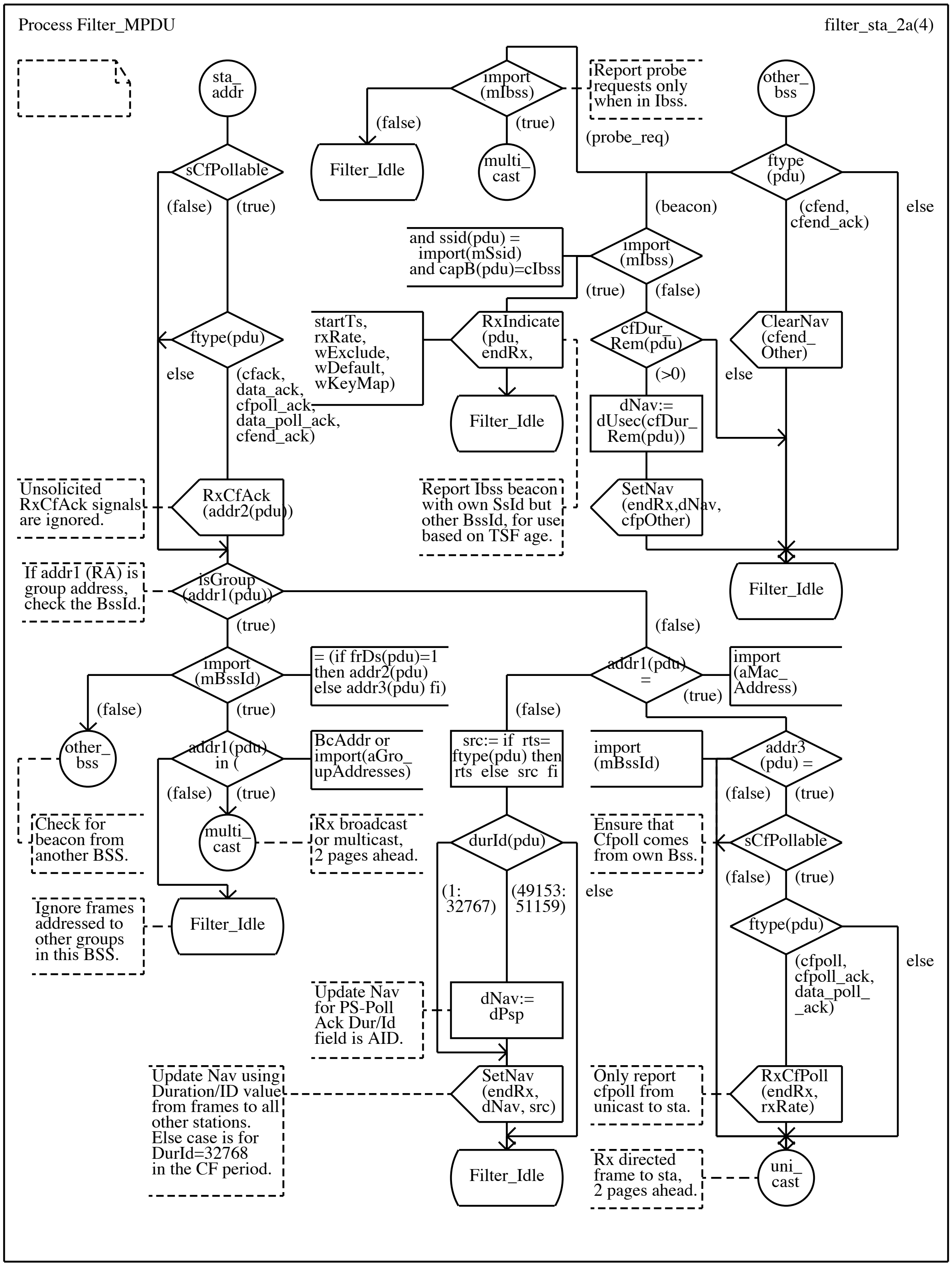

Process Defragment
wep_filter_1a(3)
Decrypt
dcl exported cIerr as aWepIcvErrorCount, cUndc as aWepUndecryptableCount, cExcl as aWepExcludedCount Counter32:= 0 ;
dLife:= dTU(import
(aMax_ Receive_ Lifetime))
imported mCfp Boolean ; imported aMaxReceiveLifetime TU ; imported procedure RC4 ; fpar PrngKey, Integer ; returns Octetstring ;
export(cIerr, cUndc, cExcl)
dcl buf DefragArray ; dcl dLife Duration ; dcl endRx, startTs Time ; dcl icvOk Boolean ; dcl k DefragIndex ; dcl keys DefragKeysArray ; dcl pri CfPriority ; dcl pdu, sdu Frame ; dcl wExcl Boolean ; dcl wDefault KeyVector ; dcl wMap KeyMapArray ;
buf:= ArAge(buf, now+dLife+1)
Defragmentation buffers forced empty using the aging function.
Defrag_ Inactive
not import (mDisable)
mDisable=false when started or joined Bss.
RxMpdu (pdu, endRx,
startTs,rxRate, wExcl,wDefault, wMap)
ftype (pdu)
When not in Bss only pass beacon and probe_rsp.
else
(beacon, probe_rsp)
RxIndicate (pdu,endRx, startTs,rxRate
-
Defrag_ Idle
import (mDisable)
RxMpdu (pdu, endRx,
startTs,rxRate, wExcl,wDefault, wMap)
basetype (pdu)
(control)
(management)
else
wepBit (pdu)
(=0)
(=1)
wepBit (pdu)
(=1)
(=0)
rx_ ind
import(
aPrivacy_ Option_ Implemented)
wExcl
(false)
(true)
(true)
(false)
auth) and authSeqNum (pdu)=3) and import(aPrivacy_ Option_ Implemented)
(ftype (pdu)=
(false)
(true)
cUndc:= inc(cUndc)
de_ crypt
cExcl:= inc(cExcl)
de_ frag
de_ crypt
export(cUndc)
export(cExcl)
-
-

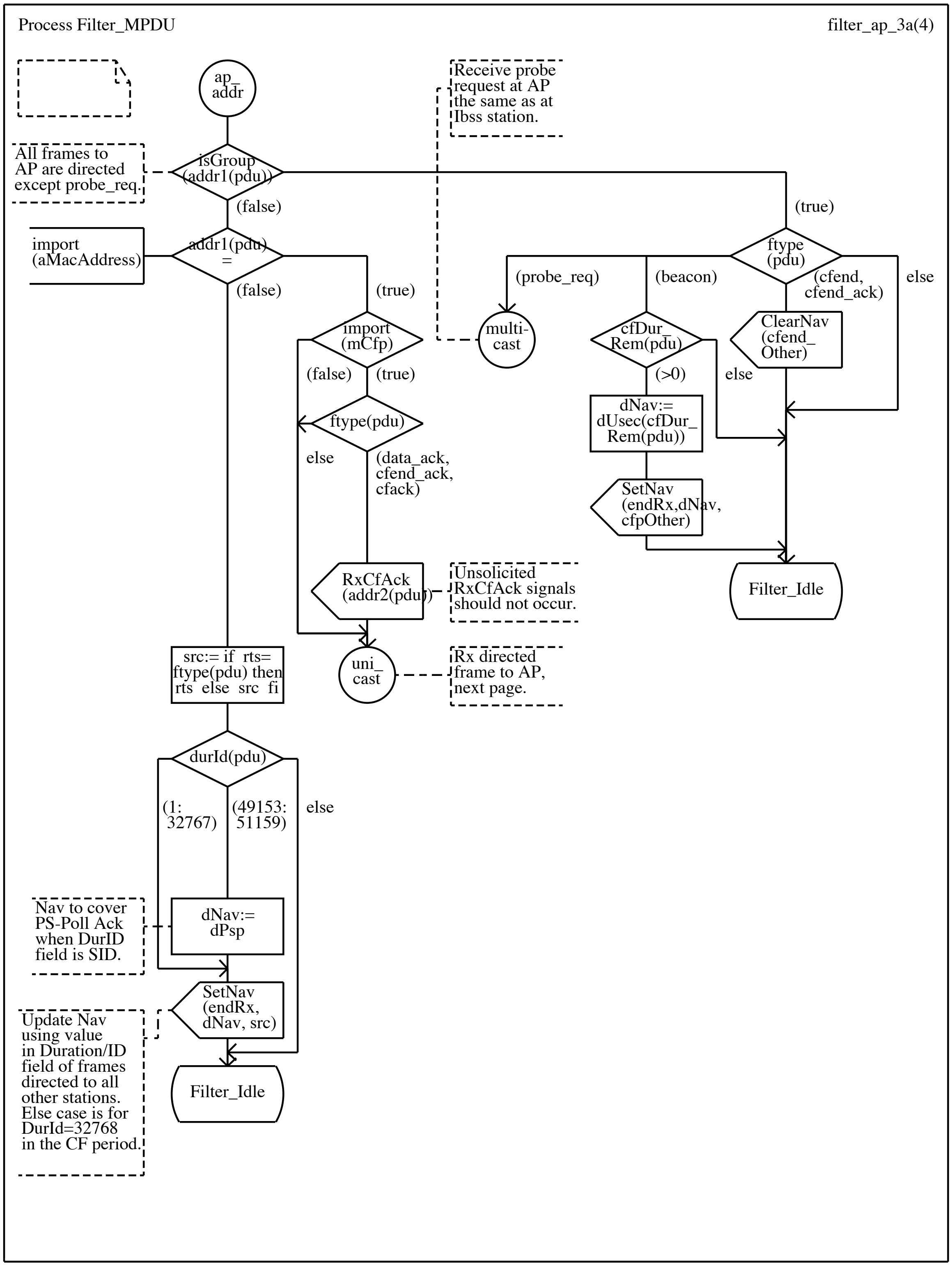

Process Defragment
wep_decrypt_2a(3)
de_
crypt
Decrypt
(pdu,
icvOk,
wMap, sKey_
MappingLength,
wDefault)
icvOk
Icv errors and
certain undecryptable
errors counted in
Decrypt procedure.
(true)
(false)
de_
frag
ftype
(pdu)
else
(auth)
RxIndicate
(pdu,endRx,
startTs,rxRate
Authentication
challenge resposnes
with Icv errors
are reported, but
Decrypt removes
payload so Auth
service is able
to distinguish
a bad key from
a non-response.
Do not report
receipt of
data frames
with Icv errors.
-

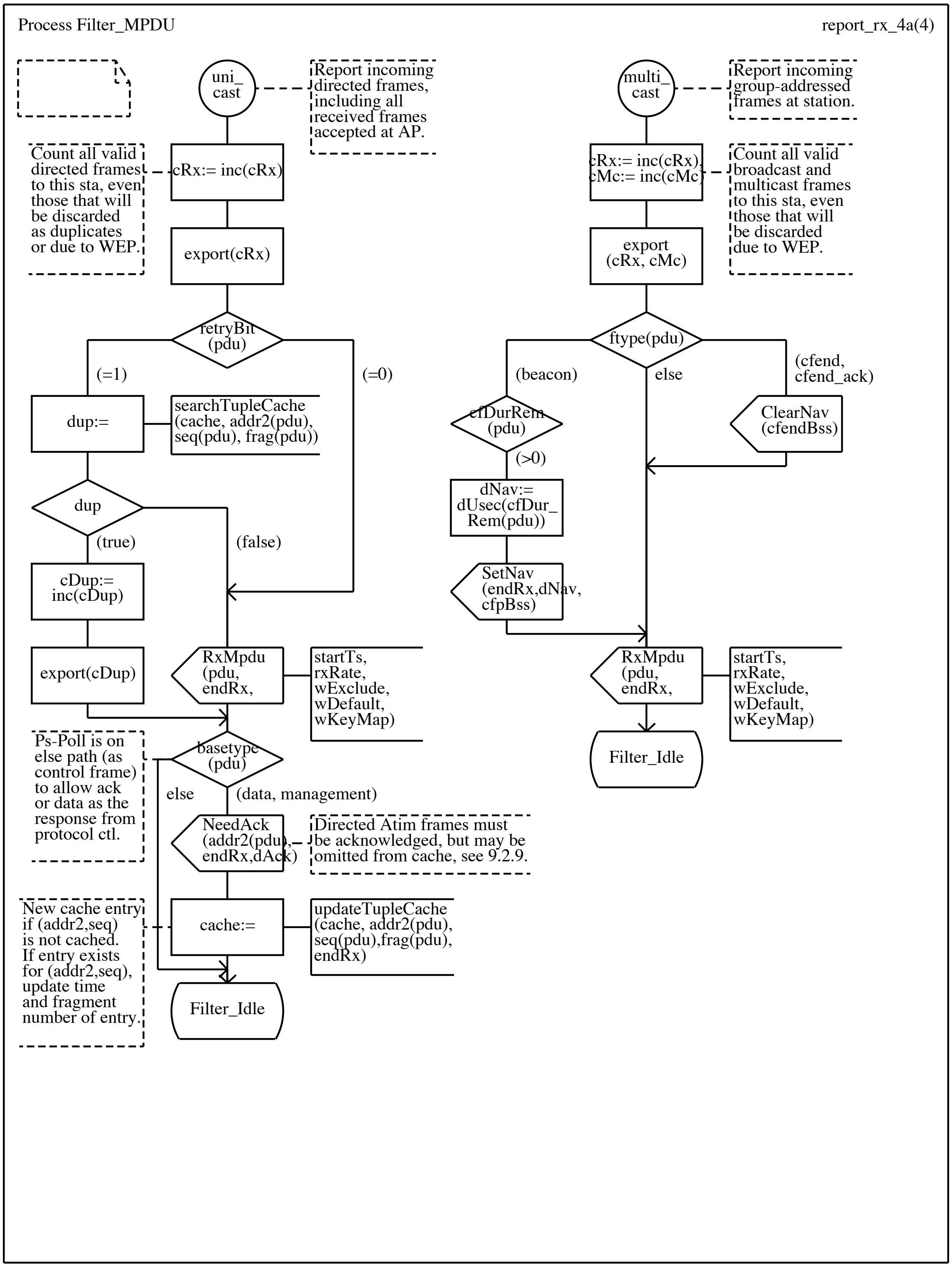

Process Defragment
defragment_3b(3)
rx_ ind
de_ frag
age
(moreFrag (pdu)=0)
and (fragNum (pdu)=0))
(true)
(false)
RxIndicate (pdu,endRx, startTs,rxRate)
fragNum (pdu)=0
(true)
(false)
Intermediate or final Mpdu of fragmented Msdu.
buf:= ArAge (buf,now)
Mpdu is not fragmented or defragmentation is complete.
k:= arFree(buf)
Initial Mpdu of fragmented Msdu. Find free buffer to begin Msdu reception.
k:= arSearch (buf,
addr2(pdu), seqNum(pdu), fragNum(pdu))
-
k > 0
(true)
(false)
k > 0
(true)
(false)
buf:= arAge (buf,now),
k:= arFree(buf)
age
k > 0
(true)
(false)
(length (pdu) +
length (buf(k)!rsdu) - sMacHdrLng) <= sMaxMsduLng
(false)
(true)
addr2(pdu), buf(k)!rsn:= seqNum(pdu)
buf(k)!inUse:= true, buf(k)!rta:=
-
buf(k)!inUse:= false
buf(k)!rCur:= fragNum(pdu),
buf(k)!rsdu:= buf(k)!rsdu // substr(pdu, sMacHdrLng, length(pdu)- sMacHdrLng)
now + tTU (import(aMaxReceive_ Lifetime))
buf(k)!rCur:= fragNum(pdu), buf(k)!reol:=
moreFrag (pdu)=0
(false)
(true)
keys(k)! wDefKeys:= wDefault, keys(k)! wKeyMap:= wMap, keys(k)! wExclude:= wExcl
buf(k)! rsdu:=pdu,
age
age
Final fragment if moreFrag=0, indicate Msdu.
rpdu:= buf(k)!rsdu,
buf(k)!inUse:= false
rx_ ind

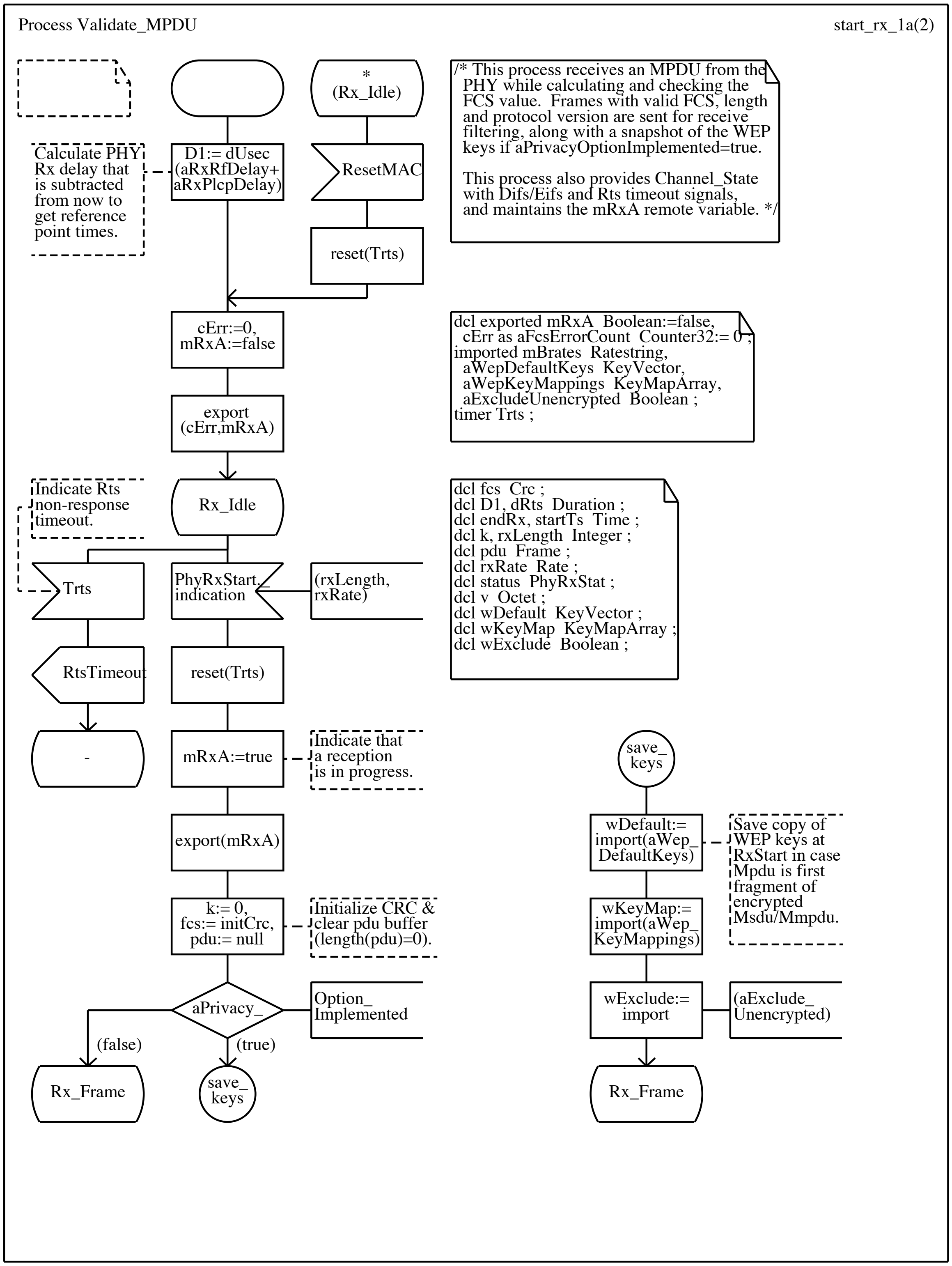

Procedure Decrypt
decrypt_1a(1)
; fpar
in/out pdu Frame,
in/out icvOk Boolean,
in map KeyMapArray,
in maplength
KeyMapArrayLength,
in kvec KeyVector ;
dcl icv Crc ;
dcl isWds Boolean ;
dcl decryptLng, k, n Integer ;
dcl decryptStr Octetstring ;
dcl key PrngKey ;
dcl kmap KeyMap ;
isWds:=
toDs(pdu) and
frDs(pdu)
Test whether addr4
field is present.
Only needed at AP.
decryptLng:=
length(pdu) -
sMacHdrLng -
sWepAddLng +
sCrcLng - if isWds
then sWdsAddLng else 0 fi
isGroup(
addr1(pdu))
(false)
(true)
(addr2(pdu),
map,
maplength)
kmap:=
keyLookup
kmap!mappedAddr
=nullAddr
(true)
(false)
key:=
kmap!
wepKey,
Use default key
selected by
keyId value.
key:= kvec
(keyId(pdu))
key =
nullKey
or
kmap!wepOn
= false
(false)
(true)
Concatenate
key with IV
from frame.
key:= key //
PrngKey!
Iv(pdu)
decryptLng)
encryptStr:=
call RC4
(key,
Use RC4 PRNG
to generate an
decrypt string
as long as the
MPDU payload
plus the ICV
field.
de_
cipher
basetype
(pdu)
(data)
(management)
cUndc:=
inc(cUndc)
cIerr:=
inc(cIerr)
export(cUndc)
export(cIerr)
pdu:=
substr(pdu,0,
sMacHdrLng)
If calculated
ICV not valid,
discard frame
body, and
report error.
icvOk:= false
de_
cipher
icv:=
initCrc
if isWds then
sWdsAddLng
else 0 fi
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 =
goodCrc
(false)
(true)
// substr(pdu,
sWepHdrLng,
decryptLng -
sCrclng)
pdu:=
substr(pdu,0,
sMacHdrLng)
Remove ICV
and IV fields
from MPDU,
report decrypt
success if ICV
result correct
or selected
key value null.
icvOk:= true

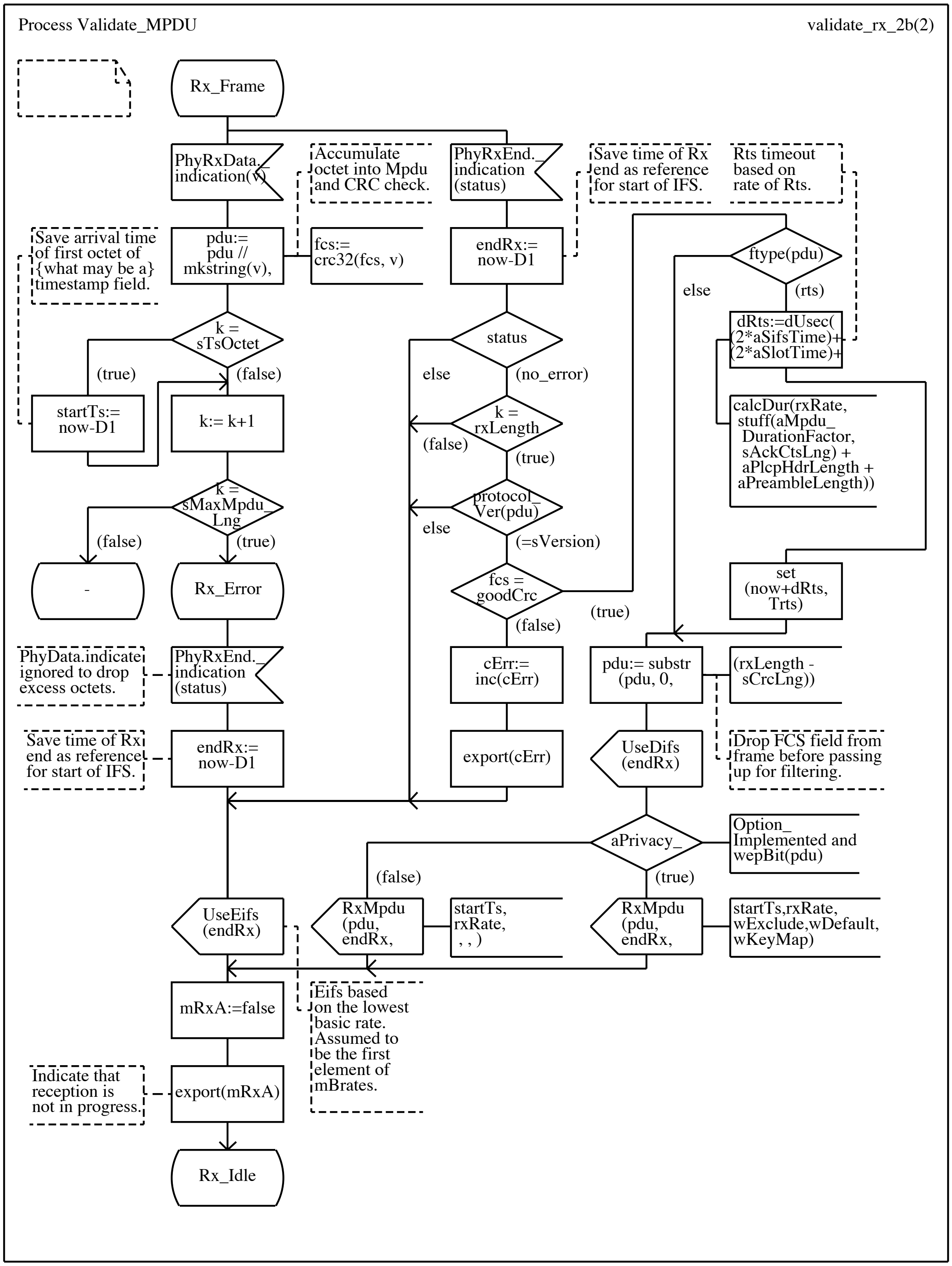

Process Filter_MPDU
pre_filter_1a(4)
dcl exported cDup as aFrameDuplicateCount, cMc as aMulticastReceivedFrameCount, cRx as aReceivedFrameCount Counter32:= 0 ;
Duration of PS-Poll and Ack response.
dPsp:=dUsec(aSifsTime + calcDur(
first(import(mBrates)), stuff(aMpduDuration_ Factor, sAckCtsLng + aPlcpHdrLength) + aPreambleLength))
imported mBrates Ratestring, mBssid MacAddr, mCfp Boolean, aGroupAddresses MacAddrSet, mIbss Boolean, mSsid Octetstring, aStationId MacAddr ;
cache:= clearTuple_ Cache(cache)
Initialize tuple cache for duplicate filtering. Cache capacity is set by "tupleCacheSize" but a specific size is not specified.
dcl cache TupleCache ;
dcl dup, myBss Boolean ;
dcl dNav, dPsp, dAck Duration ;
dcl endRx, strTs Time ;
dcl pdu Frame ;
dcl rxRate Rate ;
dcl src NavSrc ;
dcl wDefault KeyVector ;
dcl wExclude Boolean ;
dcl wKeyMap KeyMapArray ;
Filter_Idle
ResetMac
RxMpdu (pdu, endRx,
startTs,rxRate, wExclude,wDefault, wKeyMap)
dAck:= if (moreData (pdu) = 1) or
(durID(pdu) > 32767) then dUsec(durId(pdu)) else 0 fi
/* This process filters valid received frames by destination address, and BssId for group destination addresses. This process also maintains received pdu counters and the tuple cache for detecting duplicated unicast frames.
Data and management frames which need acknowledgement cause a NeedAck signal to Protocol_Control as well as an RxMpdu to Defragment. Piggybacked CfAcks cause RxCfack signals, and CfPolls cause RxCfpoll signals to Protocol_Control. If an RxCfPoll is sent for a Data+CfPoll or Data+CfPoll+CfAck, the NeedAck has to reach TxCoord during the Sifs. (The data frame report cannot serve this purpose because the payload may be a non-final fragment.)
Duration and Cfp duration remaining are reported to Channel_State, and power save mode is reported to Mlme. */
PsIndicate (addr2(pdu), pwrmgt(pdu))
Gather Power management info from all valid frames.
dNav:=dUsec (durId(pdu)), src:= misc
import(
mActing_ AsAp)
(true)
(false)
ap_ addr
toDs(pdu)
(=1)
(=0)
AP, check all frames, 2 pages ahead.
durId(pdu)
sta_ addr
Non-AP, toDS=0 to accept frame, next page.
else
(1:32767)
SetNav (endRx, dNav, src)
Filter_Idle
Frames with toDs=1 ignored by non-APs, except Duration/Id field for Nav update.

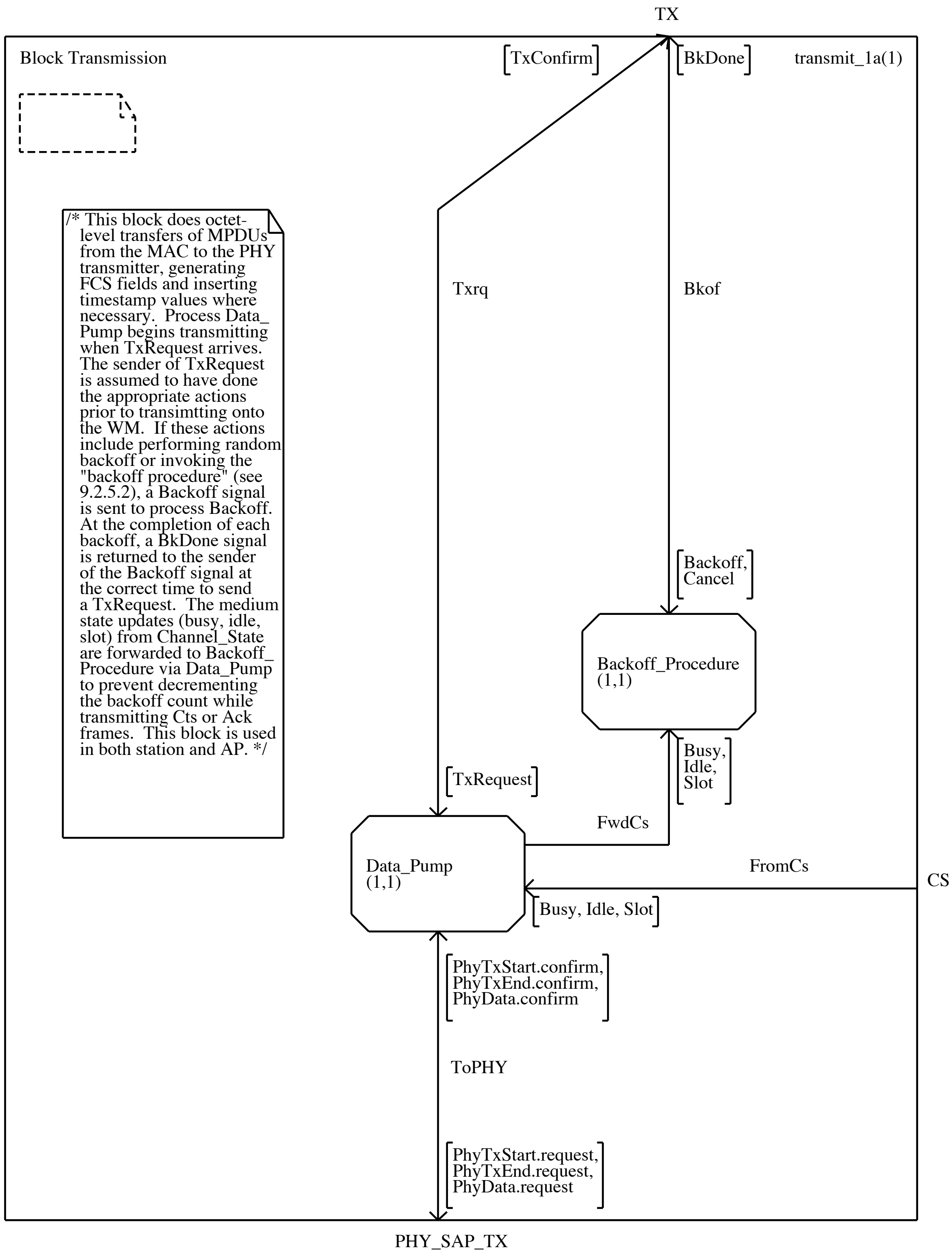

Process Filter_MPDU
filter_sta_2a(4)
sta_
addr
sCfPollable
(false)
(true)
ftype(pdu)
else
(cfack,
data_ack,
cfpoll_ack,
data_poll_ack,
cfend_ack)
Unsolicited
RxCfAck signals
are ignored.
RxCfAck
(addr2(pdu))
If addr1 (RA) is
group address,
check the BssId.
isGroup
(addr1(pdu))
(true)
import
(mBssId)
= (if frDs(pdu)=1
then addr2(pdu)
else addr3(pdu) fi)
(false)
other_
bss
Check for
beacon from
another BSS.
addr1(pdu)
in (
BcAddr or
import(aGro_
upAddresses)
multi_
cast
Rx broadcast
or multicast,
2 pages ahead.
Ignore frames
addressed to
other groups
in this BSS.
Filter_Idle
import
(mIbss)
Report probe
requests only
when in Ibss.
(probe_req)
multi_
cast
other_
bss
ftype
(pdu)
(beacon)
(cfend,
cfend_ack)
else
and ssid(pdu) =
import(mSsid)
and capB(pdu)=cIbss
import
(mIbss)
startTs,
rxRate,
wExclude,
wDefault,
wKeyMap)
RxIndicate
(pdu,
endRx,
cfDur_
Rem(pdu)
(>0)
dNav:=
dUsec(cfDur_
Rem(pdu))
ClearNav
(cfend_
Other)
Report Ibss beacon
with own SsId but
other BssId, for use
based on TSF age.
SetNav
(endRx,dNav,
cfpOther)
addr1(pdu)
=
import
(aMac_
Address)
src:= if rts=
ftype(pdu) then
rts else src fi
import
(mBssId)
addr3
(pdu) =
durId(pdu)
(1:
32767)
(49153:
51159)
Ensure that
Cfpoll comes
from own Bss.
Update Nav
for PS-Poll
Ack Dur/Id
field is AID.
dNav:=
dPsp
(cfpoll,
cfpoll_ack,
data_poll_
_ack)
Update Nav using
Duration/ID value
from frames to all
other stations.
Else case is for
DurId=32768
in the CF period.
SetNav
(endRx,
dNav, src)
Only report
cfpoll from
unicast to sta.
RxCfPoll
(endRx,
rxRate)
Rx directed
frame to sta,
2 pages ahead.
uni_
cast

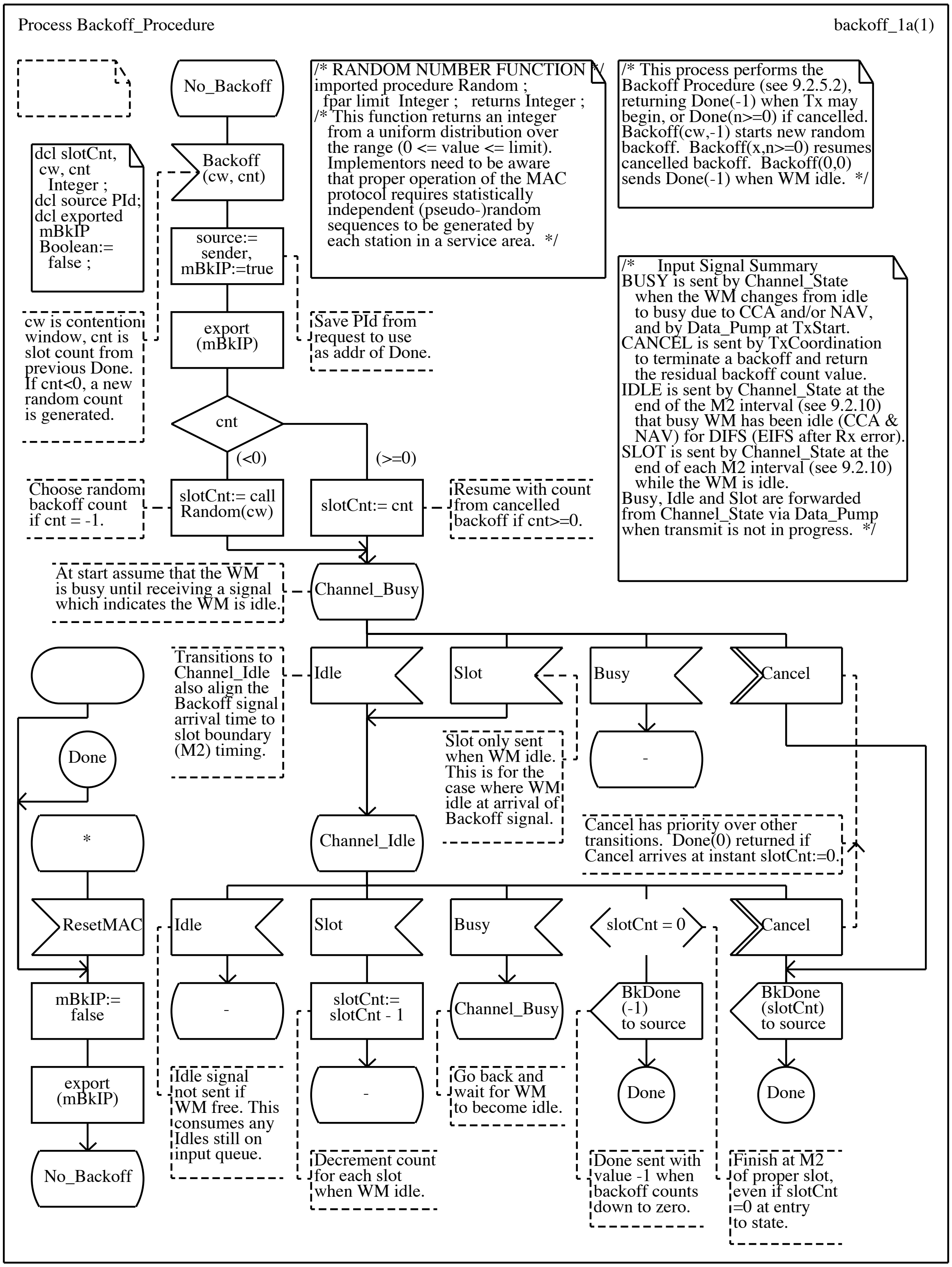

Process Filter_MPDU
filter_ap_3a(4)
ap_
addr
All frames to
AP are directed
except probe_req.
isGroup
(addr1(pdu))
(false)
(true)
import
(aMacAddress)
addr1(pdu)
=
(false)
(true)
Receive probe
request at AP
the same as at
Ibss station.
ftype
(pdu)
(probe_req)
(beacon)
(cfend,
cfend_ack)
else
import
(mCfp)
multi-
cast
cfDur_
Rem(pdu)
ClearNav
(cfend_
Other)
(false)
(true)
(>0)
else
ftype(pdu)
dNav:=
dUsec(cfDur_
Rem(pdu))
else
(data_ack,
cfend_ack,
cfack)
SetNav
(endRx,dNav,
cfpOther)
RxCfAck
(addr2(pdu))
Unsolicited
RxCfAck signals
should not occur.
Filter_Idle
src:= if rts=
ftype(pdu) then
rts else src fi
uni_
cast
Rx directed
frame to AP,
next page.
durId(pdu)
(1:
32767)
(49153:
51159)
else
Nav to cover
PS-Poll Ack
when DurID
field is SID.
dNav:=
dPsp
SetNav
(endRx,
dNav, src)
Update Nav
using value
in Duration/ID
field of frames
directed to all
other stations.
Else case is for
DurId=32768
in the CF period.
Filter_Idle

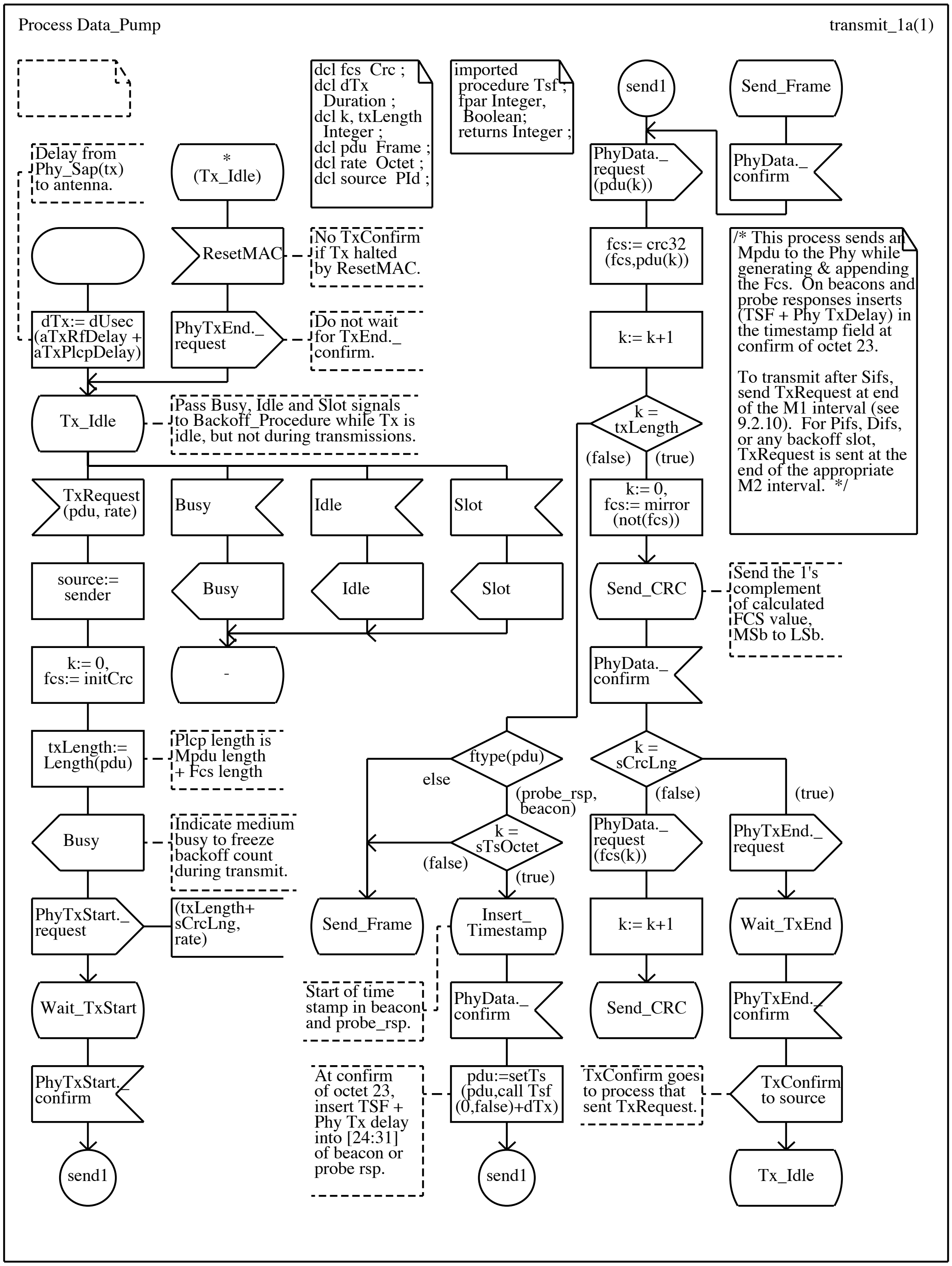

Process Filter_MPDU
report_rx_4a(4)
uni_ cast
Report incoming directed frames, including all received frames accepted at AP.
Count all valid directed frames to this sta, even those that will be discarded as duplicates or due to WEP.
cRx:= inc(cRx)
export(cRx)
retryBit (pdu)
(=1)
(=0)
dup:=
searchTupleCache (cache, addr2(pdu), seq(pdu), frag(pdu))
dup
(true)
(false)
cDup:= inc(cDup)
export(cDup)
RxMpdu (pdu, endRx,
startTs, rxRate, wExclude, wDefault, wKeyMap)
Ps-Poll is on else path (as control frame) to allow ack or data as the response from protocol ctl.
basetype (pdu)
else
(data, management)
NeedAck (addr2(pdu), endRx,dAck)
Directed Atim frames must be acknowledged, but may be omitted from cache, see 9.2.9.
New cache entry if (addr2,seq) is not cached. If entry exists for (addr2,seq), update time and fragment number of entry.
cache:=
updateTupleCache (cache, addr2(pdu), seq(pdu),frag(pdu), endRx)
Filter_Idle
multi_ cast
Report incoming group-addressed frames at station.
cRx:= inc(cRx), cMc:= inc(cMc)
Count all valid broadcast and multicast frames to this sta, even those that will be discarded due to WEP.
export (cRx, cMc)
ftype(pdu)
(beacon)
else
(cfend, cfend_ack)
cfDurRem (pdu)
ClearNav (cfendBss)
(>0)
dNav:= dUsec(cfDur_ Rem(pdu))
SetNav (endRx,dNav, cfpBss)
RxMpdu (pdu, endRx,
startTs, rxRate, wExclude, wDefault, wKeyMap)
Filter_Idle


Process Validate_MPDU
start_rx_1a(2)
* (Rx_Idle)
ResetMAC
reset(Trts)
/* This process receives an MPDU from the PHY while calculating and checking the FCS value. Frames with valid FCS, length and protocol version are sent for receive filtering, along with a snapshot of the WEP keys if aPrivacyOptionImplemented=true.
This process also provides Channel_State with Difs/Eifs and Rts timeout signals, and maintains the mRxA remote variable. */
Calculate PHY Rx delay that is subtracted from now to get reference point times.
D1:= dUsec (aRxRfDelay+ aRxPlcpDelay)
cErr:=0, mRxA:=false
export (cErr,mRxA)
dcl exported mRxA Boolean:=false, cErr as aFcsErrorCount Counter32:= 0 ; imported mBrates Ratestring, aWepDefaultKeys KeyVector, aWepKeyMappings KeyMapArray, aExcludeUnencrypted Boolean ; timer Trts ;
Indicate Rts non-response timeout.
Rx_Idle
dcl fcs Crc ;
dcl D1, dRts Duration ;
dcl endRx, startTs Time ;
dcl k, rxLength Integer ;
dcl pdu Frame ;
dcl rxRate Rate ;
dcl status PhyRxStat ;
dcl v Octet ;
dcl wDefault KeyVector ;
dcl wKeyMap KeyMapArray ;
dcl wExclude Boolean ;
Trts
PhyRxStart_ indication
(rxLength, rxRate)
RtsTimeout
reset(Trts)
-
mRxA:=true
Indicate that a reception is in progress.
export(mRxA)
k:= 0, fcs:= initCrc, pdu:= null
Initialize CRC & clear pdu buffer (length(pdu)=0).
aPrivacy_
Option_ Implemented
(false)
(true)
Rx_Frame
save_ keys
save_ keys
wDefault:= import(aWep_ DefaultKeys)
Save copy of WEP keys at RxStart in case Mpdu is first fragment of encrypted Msdu/Mmpdu.
wKeyMap:= import(aWep_ KeyMappings)
wExclude:= import
(aExclude_ Unencrypted)
Rx_Frame


Process Validate_MPDU
validate_rx_2b(2)
Rx_Frame
PhyRxData._ indication(v)
Accumulate octet into Mpdu and CRC check.
PhyRxEnd._ indication (status)
Save time of Rx end as reference for start of IFS.
Rts timeout based on rate of Rts.
Save arrival time of first octet of {what may be a} timestamp field.
pdu:= pdu // mkstring(v),
fcs:= crc32(fcs, v)
endRx:= now-D1
ftype(pdu)
else
(rts)
k = sTsOctet
(true)
(false)
status
else
(no_error)
dRts:=dUsec((2*aSifsTime)+ (2*aSlotTime)+
startTs:= now-D1
k:= k+1
k = rxLength
(false)
(true)
calcDur(rxRate, stuff(aMpdu_ DurationFactor, sAckCtsLng) + aPlcpHdrLength + aPreambleLength))
k = sMaxMpdu_ Lng
protocol_ Ver(pdu)
else
(false)
(true)
(=sVersion)
-
Rx_Error
fcs = goodCrc
set (now+dRts, Trts)
(true)
(false)
PhyData.indicate ignored to drop excess octets.
PhyRxEnd._ indication (status)
cErr:= inc(cErr)
pdu:= substr (pdu, 0,
(rxLength - sCrcLng))
Save time of Rx end as reference for start of IFS.
endRx:= now-D1
export(cErr)
UseDifs (endRx)
Drop FCS field from frame before passing up for filtering.
aPrivacy_
Option_ Implemented and wepBit(pdu)
(false)
(true)
UseEifs (endRx)
RxMpdu (pdu, endRx,
startTs, rxRate, , ,)
RxMpdu (pdu, endRx,
startTs,rxRate, wExclude,wDefault, wKeyMap)
mRxA:=false
Eifs based on the lowest basic rate. Assumed to be the first element of mBrates.
Indicate that reception is not in progress.
export(mRxA)
Rx_Idle


TX
Block Transmission
TxConfirm
BkDone
transmit_1a(1)
/* This block does octet-level transfers of MPDUs from the MAC to the PHY transmitter, generating FCS fields and inserting timestamp values where necessary. Process Data_ Pump begins transmitting when TxRequest arrives. The sender of TxRequest is assumed to have done the appropriate actions prior to transimtting onto the WM. If these actions include performing random backoff or invoking the "backoff procedure" (see 9.2.5.2), a Backoff signal is sent to process Backoff. At the completion of each backoff, a BkDone signal is returned to the sender of the Backoff signal at the correct time to send a TxRequest. The medium state updates (busy, idle, slot) from Channel_State are forwarded to Backoff_ Procedure via Data_Pump to prevent decrementing the backoff count while transmitting Cts or Ack frames. This block is used in both station and AP. */
Txrq
Bkof
Backoff, Cancel
Backoff_Procedure (1,1)
Busy, Idle, Slot
TxRequest
FwdCs
Data_Pump (1,1)
FromCs
CS
Busy, Idle, Slot
PhyTxStart.confirm, PhyTxEnd.confirm, PhyData.confirm
ToPHY
PhyTxStart.request, PhyTxEnd.request, PhyData.request
PHY_SAP_TX

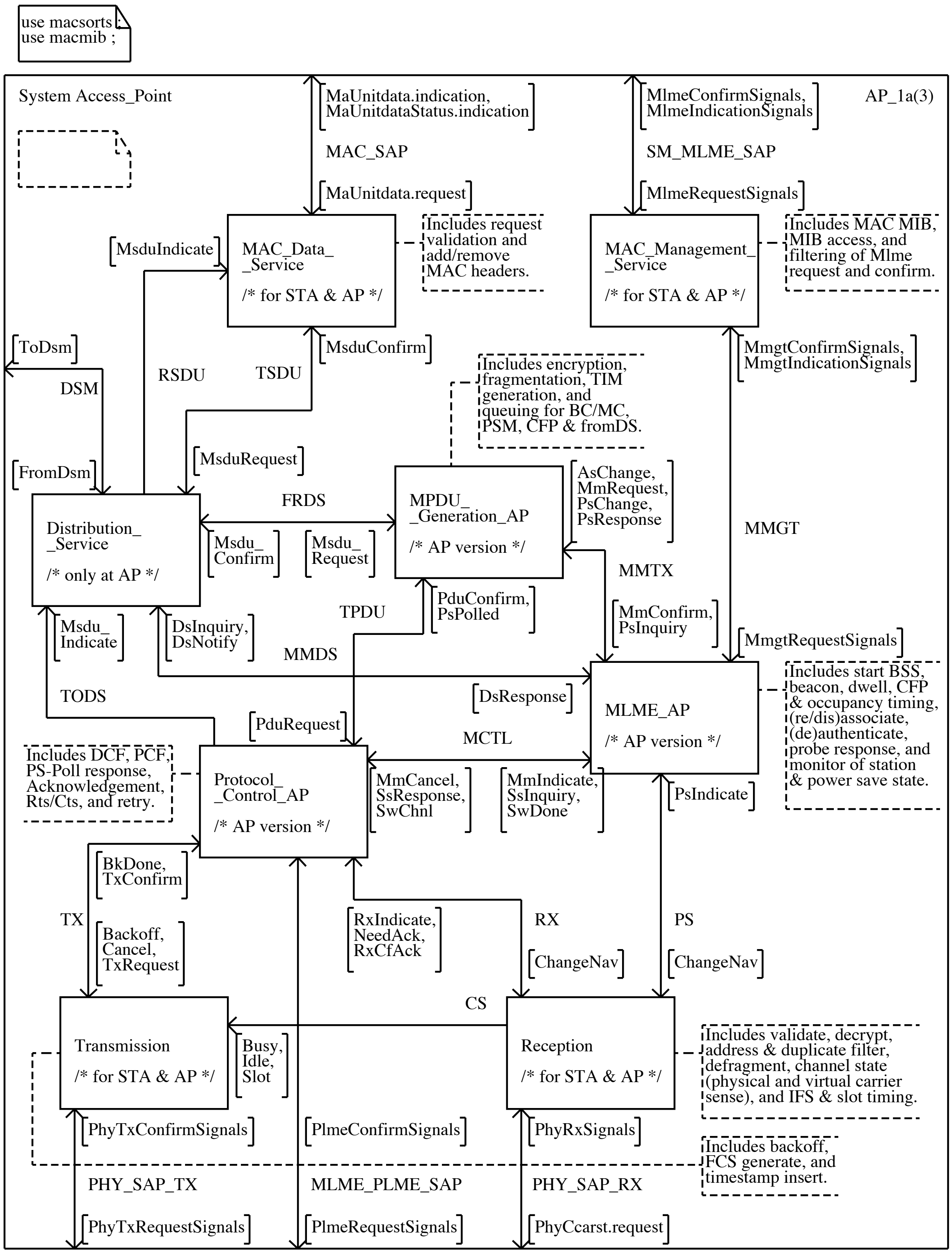

Process Backoff_Procedure
backoff_1a(1)
No_Backoff
/* RANDOM NUMBER FUNCTION */
imported procedure Random ;
fpar limit Integer ; returns Integer ;
/* This function returns an integer from a uniform distribution over the range (0 <= value <= limit). Implementors need to be aware that proper operation of the MAC protocol requires statistically independent (pseudo-)random sequences to be generated by each station in a service area. */
/* This process performs the Backoff Procedure (see 9.2.5.2), returning Done(-1) when Tx may begin, or Done(n>=0) if cancelled. Backoff(cw,-1) starts new random backoff. Backoff(x,n>=0) resumes cancelled backoff. Backoff(0,0) sends Done(-1) when WM idle. */
dcl slotCnt, cw, cnt Integer ; dcl source PId; dcl exported mBkIP Boolean:= false ;
Backoff (cw, cnt)
source:= sender, mBkIP:=true
export (mBkIP)
Save PId from request to use as addr of Done.
cw is contention window, cnt is slot count from previous Done. If cnt<0, a new random count is generated.
cnt
(<0)
(>=0)
Choose random backoff count if cnt = -1.
slotCnt:= call Random(cw)
slotCnt:= cnt
Resume with count from cancelled backoff if cnt>=0.
/* Input Signal Summary
BUSY is sent by Channel_State when the WM changes from idle to busy due to CCA and/or NAV, and by Data_Pump at TxStart.
CANCEL is sent by TxCoordination to terminate a backoff and return the residual backoff count value.
IDLE is sent by Channel_State at the end of the M2 interval (see 9.2.10) that busy WM has been idle (CCA & NAV) for DIFS (EIFS after Rx error).
SLOT is sent by Channel_State at the end of each M2 interval (see 9.2.10) while the WM is idle.
Busy, Idle and Slot are forwarded from Channel_State via Data_Pump when transmit is not in progress. */
At start assume that the WM is busy until receiving a signal which indicates the WM is idle.
Channel_Busy
Transitions to Channel_Idle also align the Backoff signal arrival time to slot boundary (M2) timing.
Idle
Slot
Busy
Cancel
-
Slot only sent when WM idle. This is for the case where WM idle at arrival of Backoff signal.
Done
Cancel has priority over other transitions. Done(0) returned if Cancel arrives at instant slotCnt:=0.
*
Channel_Idle
ResetMAC
Idle
Slot
Busy
slotCnt = 0
Cancel
mBkIP:= false
-
slotCnt:= slotCnt - 1
Channel_Busy
BkDone (-1) to source
BkDone (slotCnt) to source
export (mBkIP)
Idle signal not sent if WM free. This consumes any Idles still on input queue.
-
Go back and wait for WM to become idle.
Done
Done
No_Backoff
Decrement count for each slot when WM idle.
Done sent with value -1 when backoff counts down to zero.
Finish at M2 of proper slot, even if slotCnt =0 at entry to state.


Process Data_Pump
transmit_1a(1)
dcl fcs Crc ;
dcl dTx Duration ;
dcl k, txLength Integer ;
dcl pdu Frame ;
dcl rate Octet ;
dcl source PId ;
imported procedure Tsf ;
fpar Integer, Boolean;
returns Integer ;
send1
Send_Frame
Delay from Phy_Sap(tx) to antenna.
*
(Tx_Idle)
PhyData._request (pdu(k))
PhyData._confirm
ResetMAC
No TxConfirm if Tx halted by ResetMAC.
fcs:= crc32 (fcs,pdu(k))
/* This process sends an Mpdu to the Phy while generating & appending the Fcs. On beacons and probe responses inserts (TSF + Phy TxDelay) in the timestamp field at confirm of octet 23.
To transmit after Sifs, send TxRequest at end of the M1 interval (see 9.2.10). For Pifs, Difs, or any backoff slot, TxRequest is sent at the end of the appropriate M2 interval. */
dTx:= dUsec (aTxRfDelay + aTxPlcpDelay)
PhyTxEnd._request
Do not wait for TxEnd._confirm.
k:= k+1
Tx_Idle
Pass Busy, Idle and Slot signals to Backoff_Procedure while Tx is idle, but not during transmissions.
k = txLength
(false)
(true)
TxRequest (pdu, rate)
Busy
Idle
Slot
k:= 0, fcs:= mirror (not(fcs))
source:= sender
Send_CRC
Send the 1's complement of calculated FCS value, MSb to LSb.
k:= 0, fcs:= initCrc
-
txLength:= Length(pdu)
Plcp length is Mpdu length + Fcs length
ftype(pdu)
else
(probe_rsp, beacon)
k = sCrcLng
Indicate medium busy to freeze backoff count during transmit.
k = sTsOctet
PhyData._request (fcs(k))
PhyTxEnd._request
PhyTxStart._request
(txLength+ sCrcLng, rate)
Insert_Timestamp
Wait_TxEnd
Wait_TxStart
Start of time stamp in beacon and probe_rsp.
PhyTxEnd._confirm
PhyTxStart._confirm
At confirm of octet 23, insert TSF + Phy Tx delay into [24:31] of beacon or probe rsp.
pdu:=setTs (pdu,call Tsf (0,false)+dTx)
TxConfirm goes to process that sent TxRequest.
TxConfirm to source

C.4 State machines for MAC access point

The following SDL-92 system specification defines operation of the MAC protocol at an IEEE 802.11 AP. Many aspects of AP operation are identical to STA operation. These are defined in blocks and processes referenced from both the STA and AP system specifications. Blocks and processes used in both STA and AP are identifiable by the SDL comment /* for STA & AP */ below the block or process name. The definitions of these blocks and processes appear in Clause C.3. Blocks and processes specific to AP operation are identifiable by the SDL comment /* AP version */ below the block or process name. The definitions of these blocks and processes appear in this subclause.

The remainder of Clause C.4 is the formal description, in SDL/GR, of an IEEE 802.11 AP.


use macsorts ;
use macmib ;
System Access_Point
AP_1a(3)
MaUnitdata.indication,
MaUnitdataStatus.indication
MAC_SAP
MaUnitdata.request
MlmeConfirmSignals,
MlmeIndicationSignals
SM_MLME_SAP
MlmeRequestSignals
MsduIndicate
MAC_Data_
_Service
/* for STA & AP */
Includes request validation and add/remove MAC headers.
MAC_Management_
_Service
/* for STA & AP */
Includes MAC MIB, MIB access, and filtering of Mlme request and confirm.
ToDsm
DSM
RSDU
TSDU
MsduConfirm
MmgtConfirmSignals,
MmgtIndicationSignals
Includes encryption, fragmentation, TIM generation, and queuing for BC/MC, PSM, CFP & fromDS.
MsduRequest
FromDsm
FRDS
Distribution_
_Service
/* only at AP */
Msdu_
Confirm
Msdu_
Request
MPDU_
_Generation_AP
/* AP version */
AsChange,
MmRequest,
PsChange,
PsResponse
MMGT
MMTX
TPDU
PduConfirm,
PsPolled
Msdu_
Indicate
DsInquiry,
DsNotify
MmConfirm,
PsInquiry
MmgtRequestSignals
MMDS
TODS
DsResponse
MLME_AP
/* AP version */
Includes start BSS, beacon, dwell, CFP & occupancy timing, (re/dis)associate, (de)authenticate, probe response, and monitor of station & power save state.
PduRequest
MCTL
Includes DCF, PCF, PS-Poll response, Acknowledgement, Rts/Cts, and retry.
Protocol_
_Control_AP
/* AP version */
MmCancel,
SsResponse,
SwChnl
MmIndicate,
SsInquiry,
SwDone
PsIndicate
BkDone,
TxConfirm
TX
Backoff,
Cancel,
TxRequest
RxIndicate,
NeedAck,
RxCfAck
RX
PS
ChangeNav
ChangeNav
CS
Transmission
/* for STA & AP */
Busy,
Idle,
Slot
Reception
/* for STA & AP */
Includes validate, decrypt, address & duplicate filter, defragment, channel state (physical and virtual carrier sense), and IFS & slot timing.
PhyTxConfirmSignals
PlmeConfirmSignals
PhyRxSignals
Includes backoff, FCS generate, and timestamp insert.
PHY_SAP_TX
MLME_PLME_SAP
PHY_SAP_RX
PhyTxRequestSignals
PlmeRequestSignals
PhyCcarst.request

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

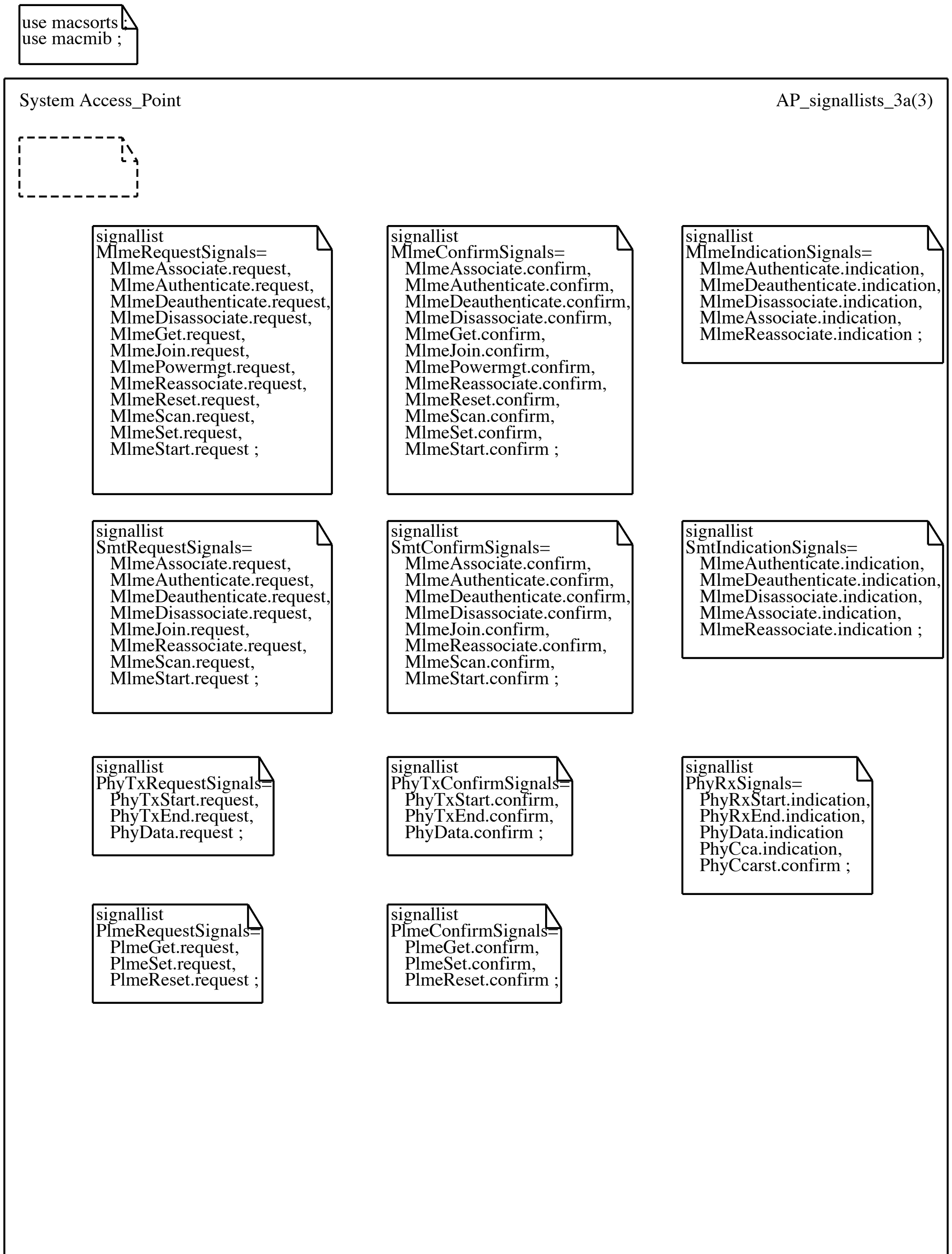
System Access_Point

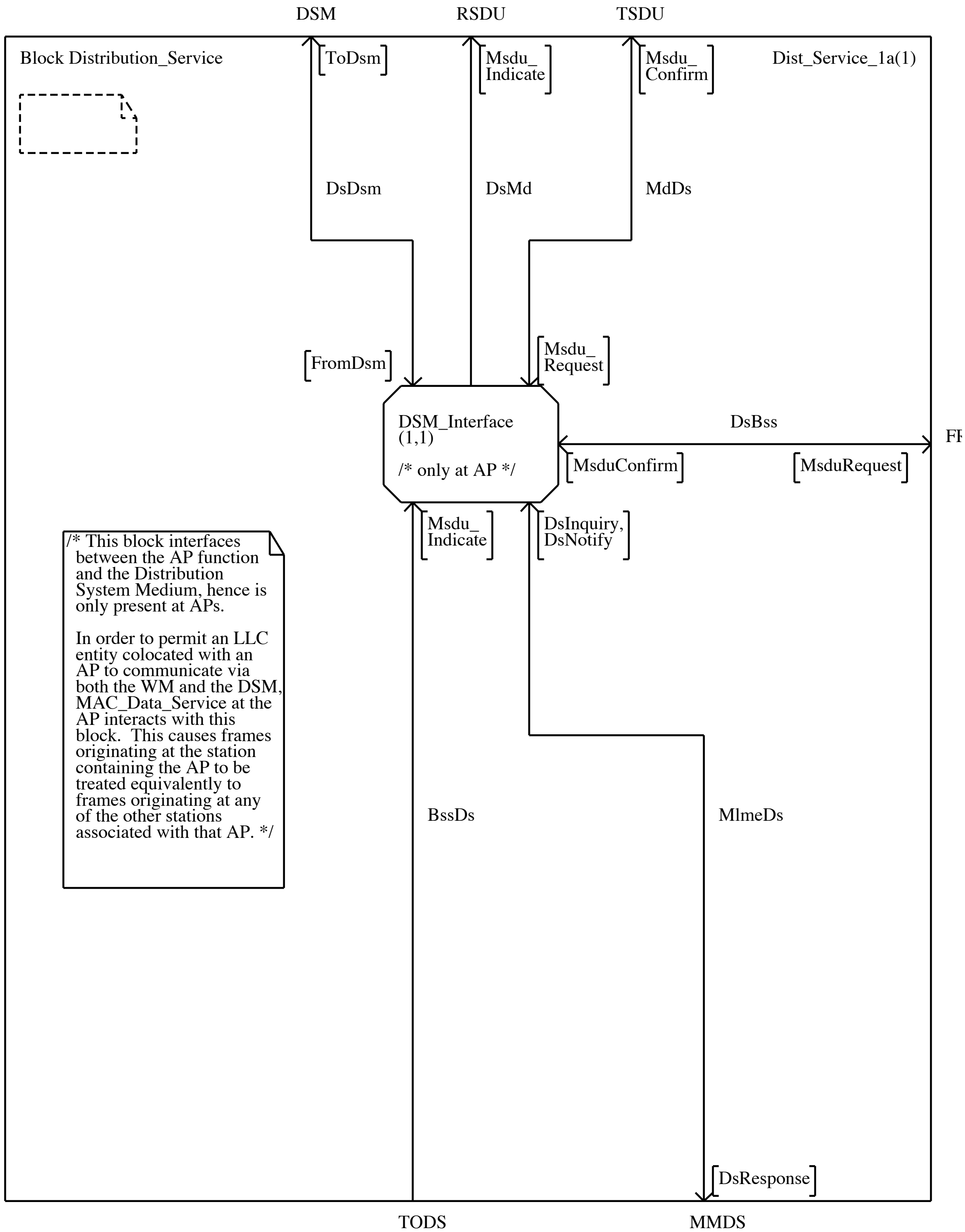
AP_signals_2b(3)

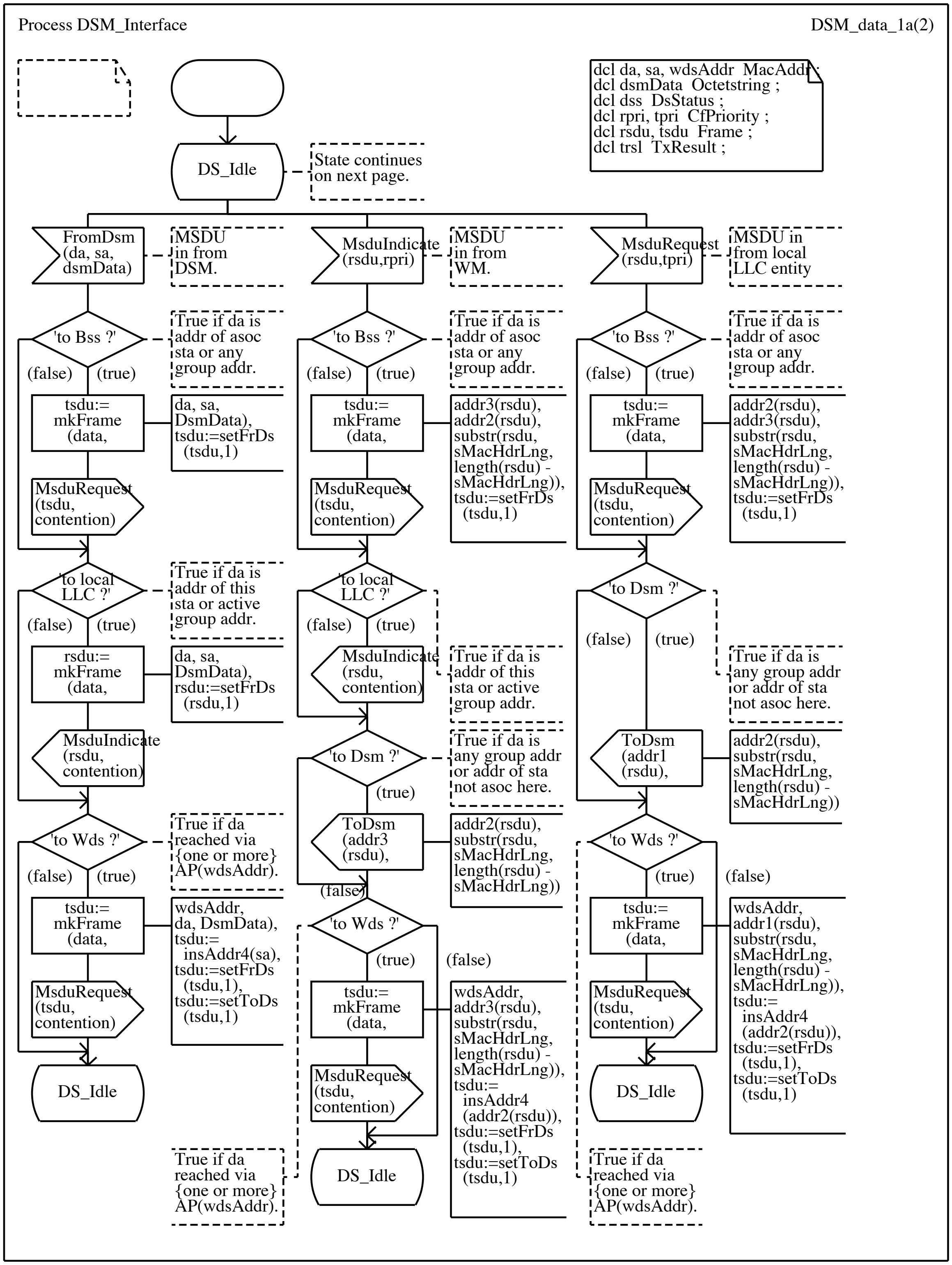
```
newtype DsStatus  literals
  assoc, disassoc, reassoc, unknown
endnewtype DsStatus ;
```

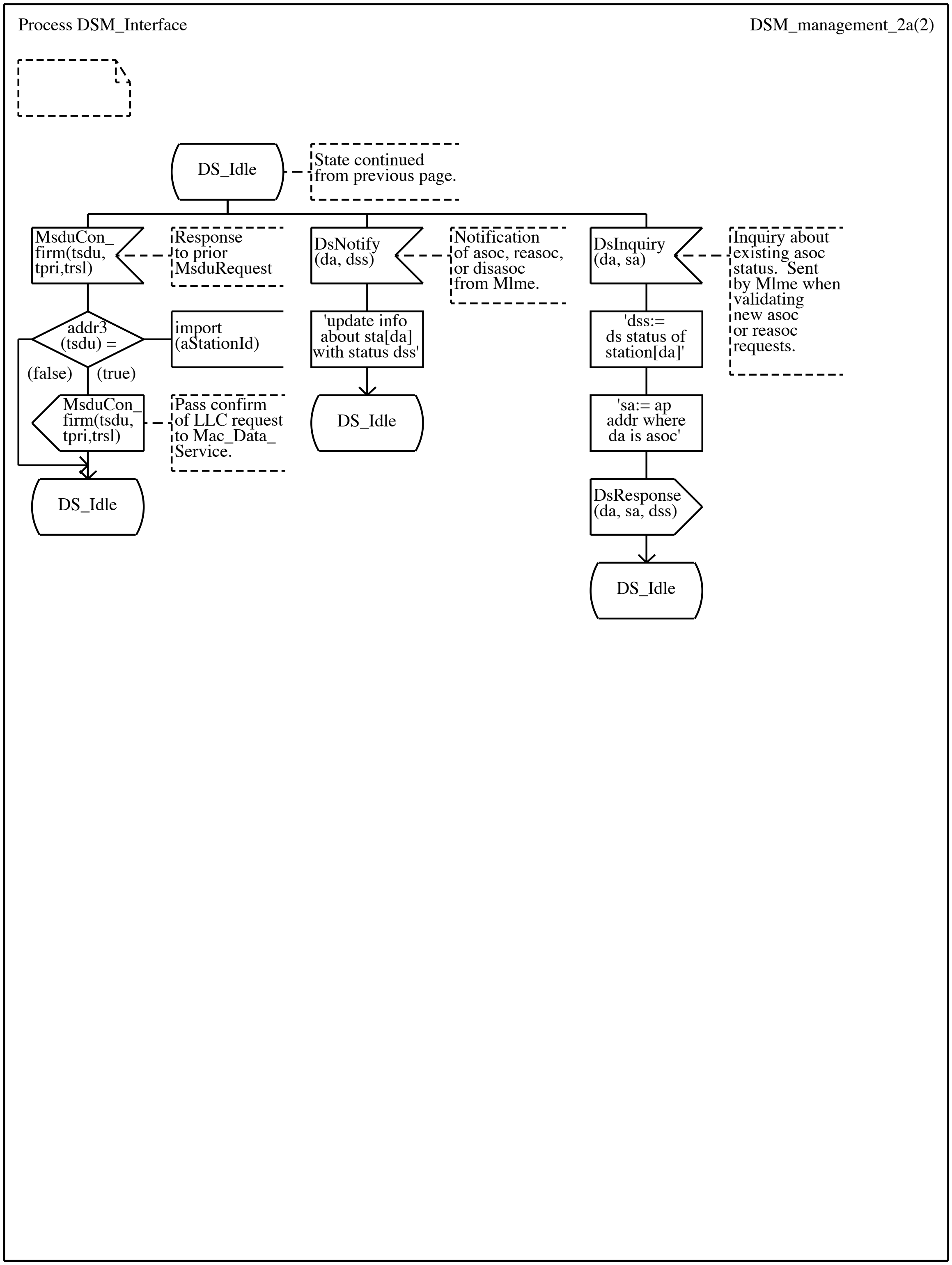
```
signal
  AsChange(Frame,DsStatus)
  Backoff(Integer,Integer),
  BkDone(Integer),
  Busy,
  Cancel,
  ChangeNav(Time,Duration,NavSrc),
  DsInquiry(MacAddr,MacAddr),
  DsNotify(MacAddr,DsStatus)
  DsResponse(MacAddr,MacAddr,DsStatus),
  FromDsm(MacAddr,MacAddr,Octetstring),
  Idle,
  MaUnitdata.indication(MacAddr,MacAddr,
    Routing,Octetstring,RxStatus,
    CfPriority,ServiceClass),
  MaUnitdata.request(MacAddr,MacAddr,
    Routing,Octetstring,CfPriority,ServiceClass),
  MaUnitdataStatus.indication(MacAddr,
    MacAddr,TxStatus,CfPriority,ServiceClass),
  MlmeAssociate.confirm(MlmeStatus),
  MlmeAssociate.indication(MacAddr),
  MlmeAssociate.request(MacAddr,TU,Capability,Integer),
  MlmeAuthenticate.confirm
    (MacAddr,AuthType,MlmeStatus),
  MlmeAuthenticate.indication(MacAddr,AuthType),
  MlmeAuthenticate.request(MacAddr,AuthType,TU),
  MlmeDeauthenticate.confirm(MacAddr,MlmeStatus),
  MlmeDeauthenticate.indication(MacAddr,ReasonCode),
  MlmeDeauthenticate.request(MacAddr,ReasonCode),
  MlmeDisassociate.confirm(MlmeStatus),
  MlmeDisassociate.indication(MacAddr,ReasonCode),
  MlmeDisassociate.request(MacAddr,ReasonCode),
  MlmeGet.confirm(MibStatus,MibAtrib,MibValue),
  MlmeGet.request(MibAtrib),
  MlmeJoin.confirm(MlmeStatus),
  MlmeJoin.request(BssDscr,Integer,Usec,Ratestring),
  MlmePowermgt.confirm(MlmeStatus),
  MlmePowermgt.request(PwrSave,Boolean,Boolean),
  MlmeReassociate.confirm(MlmeStatus),
  MlmeReassociate.indication(MacAddr),
  MlmeReassociate.request(MacAddr,TU,Capability,Integer),
  MlmeReset.confirm(MlmeStatus),
  MlmeReset.request,
  MlmeScan.confirm(BssDscrSet,MlmeStatus),
  MlmeScan.request(BssTypeSet,MacAddr,Octetstring,
    ScanType,Usec,Intstring,TU,TU),
  MlmeSet.confirm(MibStatus,MibAtrib),
  MlmeSet.request(MibAtrib,MibValue),
  MlmeStart.confirm(MlmeStatus),
  MlmeStart.request(Octetstring,BssType,TU,
    Integer,CfParms,PhyParms,IbssParms,Usec,
    Capability,Ratestring,Ratestring) ;
```

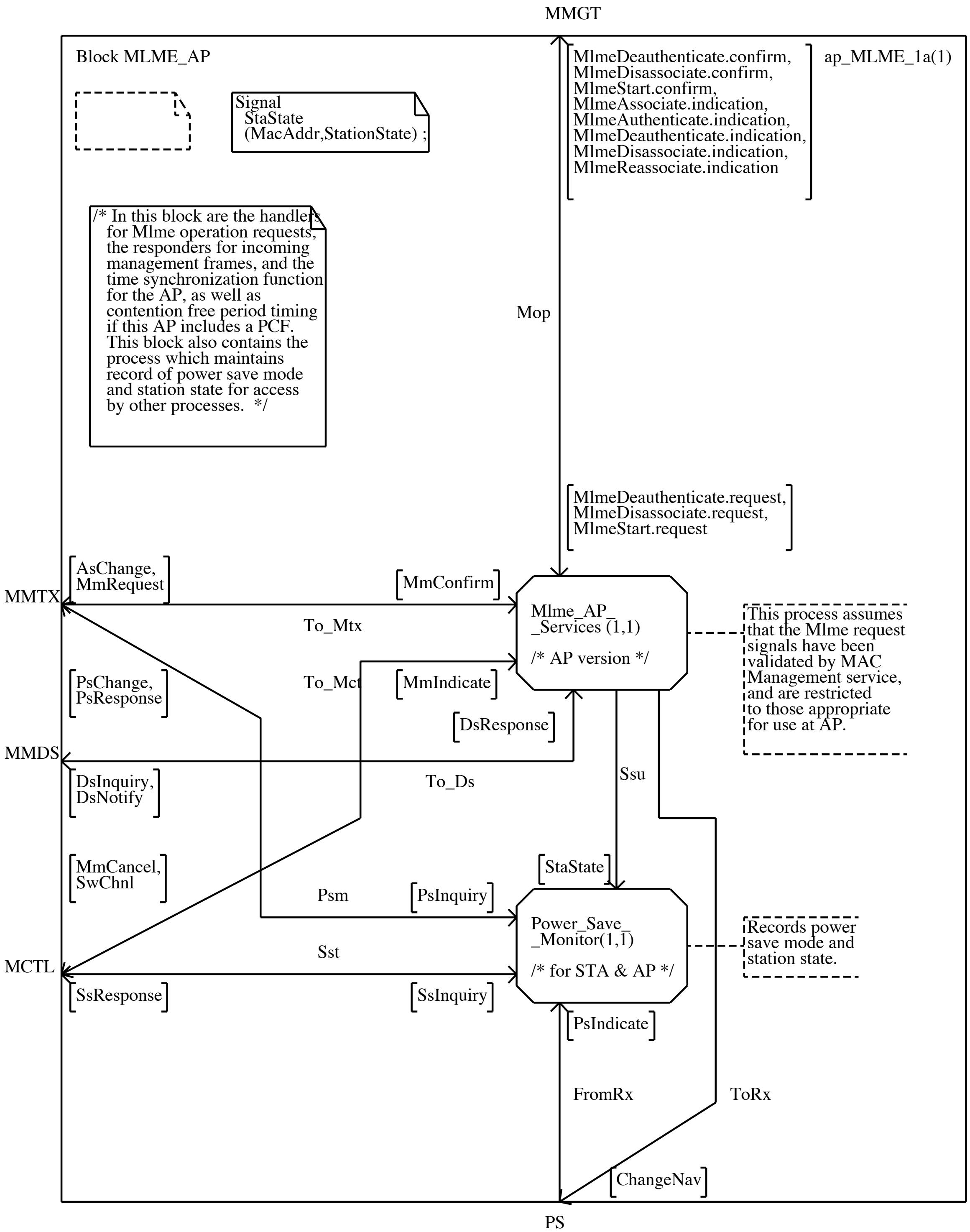
```
signal
  MmCancel,
  MmConfirm(Frame,TxStatus),
  MmIndicate(Frame,Time,Time,StateErr),
  MmRequest(Frame,Imed,Rate),
  MsduConfirm(Frame,CfPriority,TxStatus),
  MsduIndicate(Frame,CfPriority),
  MsduRequest(Frame,CfPriority),
  NeedAck(MacAddr,Time,Duration,Rate),
  PduConfirm(FragSdu,TxResult),
  PduRequest(FragSdu),
  PhyCca.indication(Ccastatus),
  PhyCcarst.confirm,
  PhyCcarst.request,
  PhyData.confirm,
  PhyData.indication(Octet),
  PhyData.request(Octet),
  PhyRxEnd.indication(PhyRxStat),
  PhyRxStart.indication(Integer,Rate),
  PhyTxEnd.confirm,
  PhyTxEnd.request,
  PhyTxStart.confirm,
  PhyTxStart.request(Integer,Rate),
  PlmeGet.confirm(MibStatus,
    MibAtrib,MibValue),
  PlmeGet.request(MibAtrib),
  PlmeReset.confirm(Boolean),
  PlmeReset.request,
  PlmeSet.confirm(MibStatus,MibAtrib),
  PlmeSet.request(MibAtrib,MibValue),
  PsmDone,
  PsPolled(MacAddr,AsocId),
  PsChange(MacAddr,PsMode),
  PsIndicate(MacAddr,PsMode),
  PsInquiry(MacAddr),
  PsResponse(MacAddr,PsMode),
  ResetMAC,
  RxCfAck(MacAddr),
  RxIndicate(Frame,Time,Time,Rate),
  Slot,
  SsInquiry(MacAddr),
  SsResponse(MacAddr,
    StationState,StationState),
  SwChnl(Integer,Boolean),
  SwDone,
  ToDsm(MacAddr,MacAddr,Octetstring),
  TxConfirm,
  TxRequest(Frame,Rate) ;
```

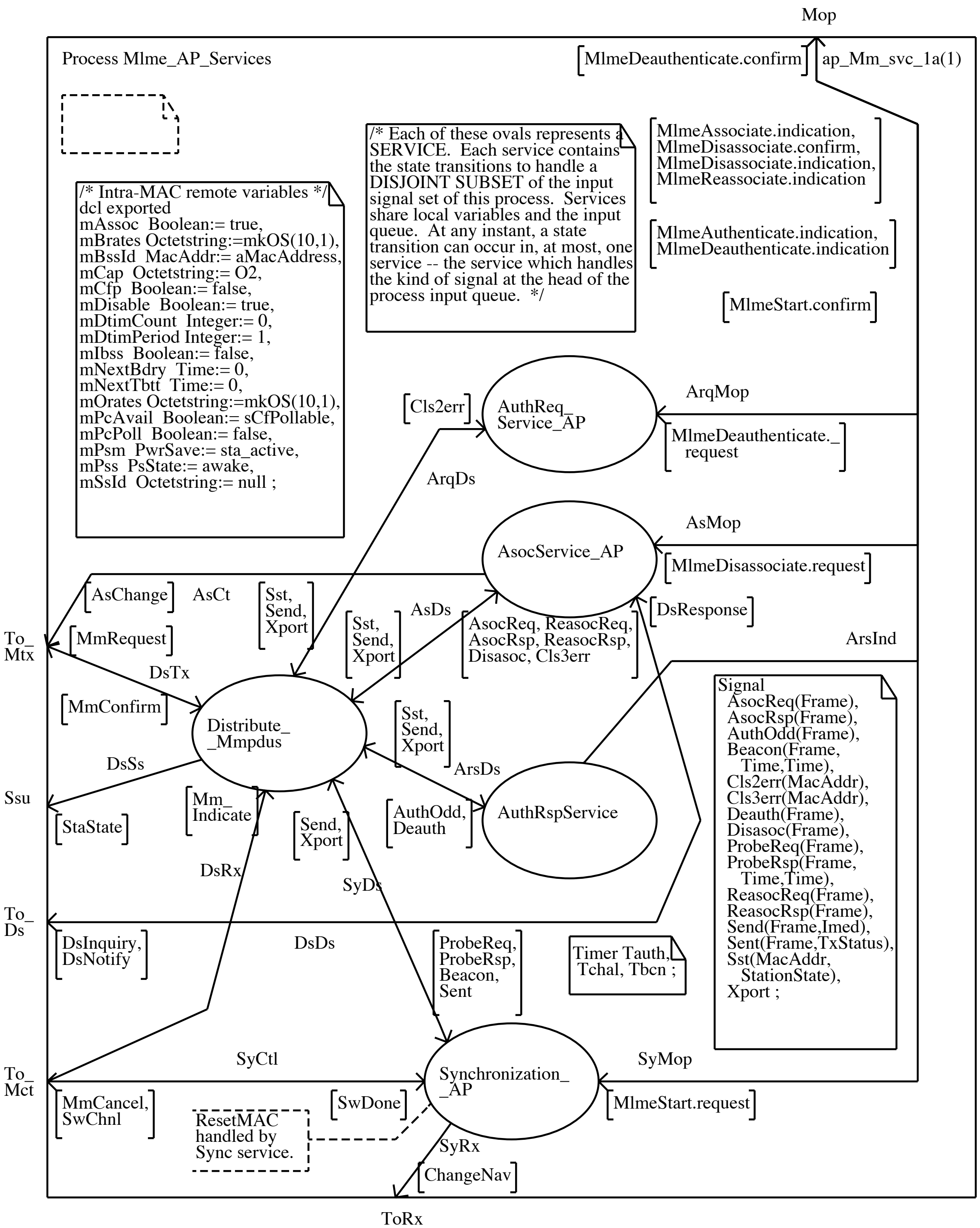

use macsorts ;
use macmib ;
System Access_Point
AP_signallists_3a(3)
signallist MlmeRequestSignals= MlmeAssociate.request, MlmeAuthenticate.request, MlmeDeauthenticate.request, MlmeDisassociate.request, MlmeGet.request, MlmeJoin.request, MlmePowermgt.request, MlmeReassociate.request, MlmeReset.request, 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 MlmeIndicationSignals= MlmeAuthenticate.indication, MlmeDeauthenticate.indication, MlmeDisassociate.indication, MlmeAssociate.indication, MlmeReassociate.indication ;
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 SmtIndicationSignals= MlmeAuthenticate.indication, MlmeDeauthenticate.indication, MlmeDisassociate.indication, MlmeAssociate.indication, MlmeReassociate.indication ;
signallist PhyTxRequestSignals= PhyTxStart.request, PhyTxEnd.request, PhyData.request ;
signallist PhyTxConfirmSignals= PhyTxStart.confirm, PhyTxEnd.confirm, PhyData.confirm ;
signallist PhyRxSignals= PhyRxStart.indication, PhyRxEnd.indication, PhyData.indication PhyCca.indication, PhyCcarst.confirm ;
signallist PlmeRequestSignals= PlmeGet.request, PlmeSet.request, PlmeReset.request ;
signallist PlmeConfirmSignals= PlmeGet.confirm, PlmeSet.confirm, PlmeReset.confirm ;

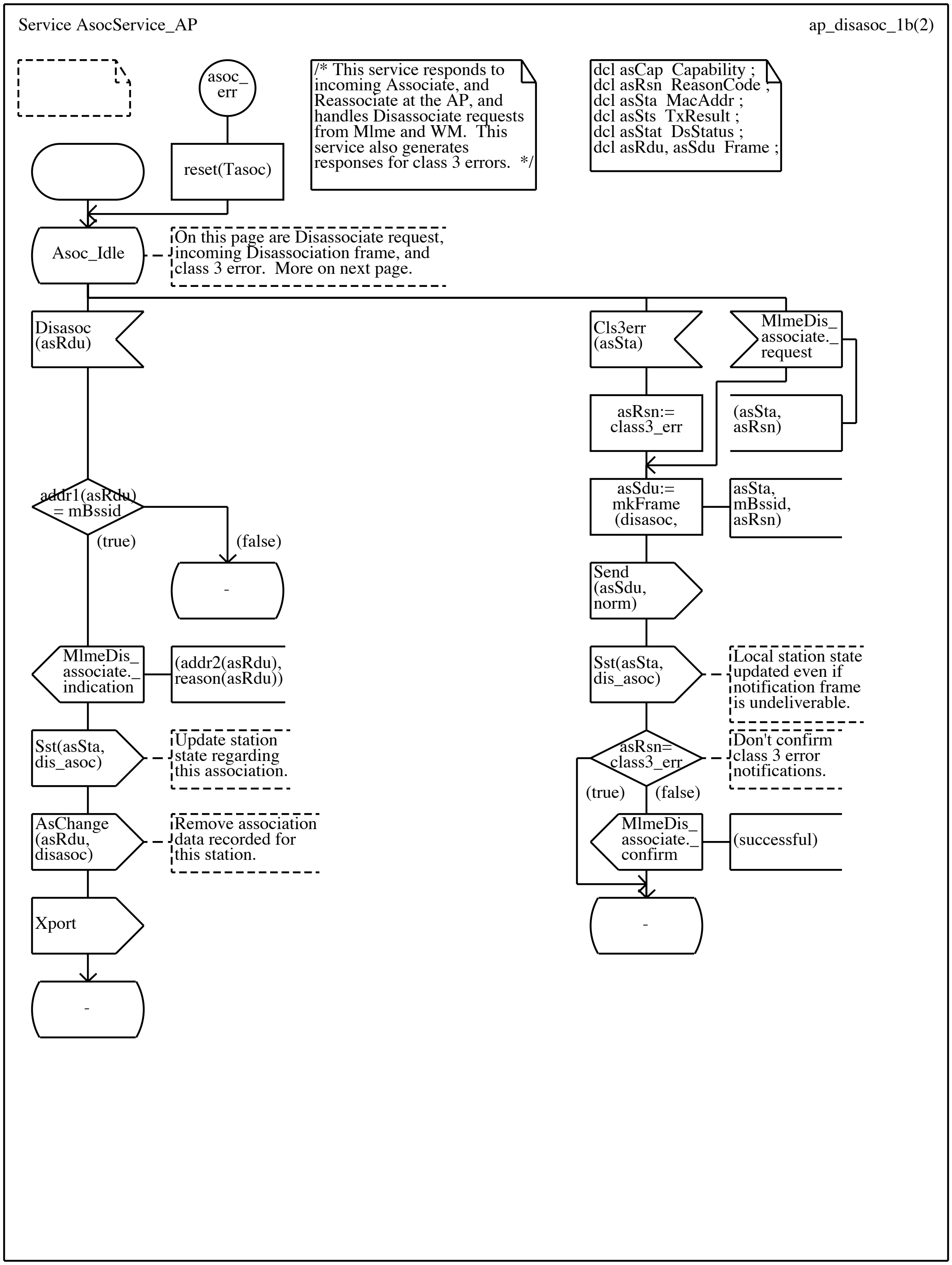

DSM
RSDU
TSDU
Block Distribution_Service
ToDsm
Msdu_ Indicate
Msdu_ Confirm
Dist_Service_1a(1)
DsDsm
DsMd
MdDs
FromDsm
Msdu_ Request
DSM_Interface (1,1)
/* only at AP */
DsBss
MsduConfirm
MsduRequest
Msdu_ Indicate
DsInquiry, DsNotify
/* This block interfaces between the AP function and the Distribution System Medium, hence is only present at APs.
In order to permit an LLC entity colocated with an AP to communicate via both the WM and the DSM, MAC_Data_Service at the AP interacts with this block. This causes frames originating at the station containing the AP to be treated equivalently to frames originating at any of the other stations associated with that AP. */
BssDs
MlmeDs
DsResponse
TODS
MMDS

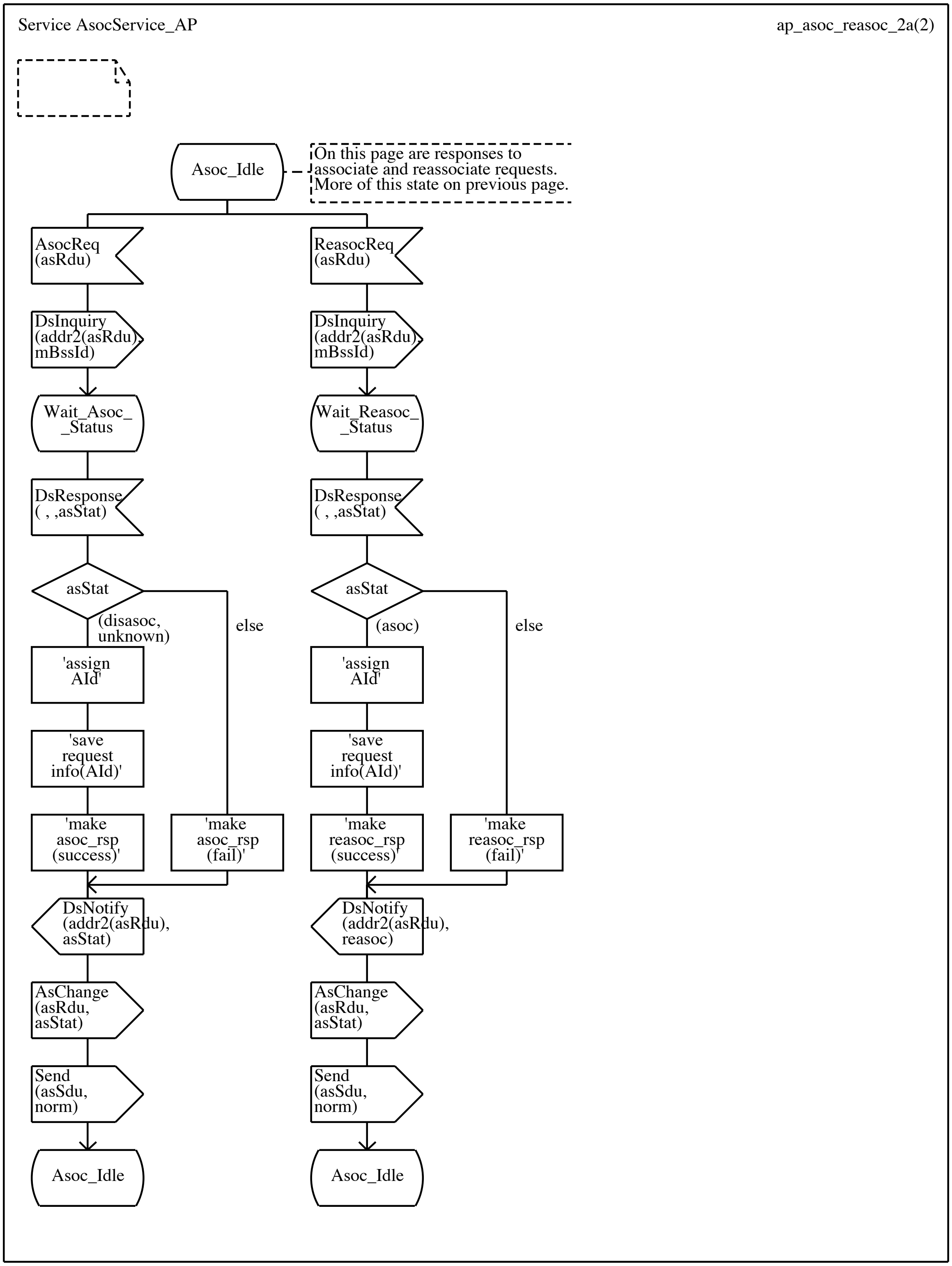

Process DSM_Interface
DSM_data_1a(2)
dcl da, sa, wdsAddr MacAddr ;
dcl dsmData Octetstring ;
dcl dss DsStatus ;
dcl rpri, tpri CfPriority ;
dcl rsdu, tsdu Frame ;
dcl trsl TxResult ;
DS_Idle
State continues on next page.
FromDsm (da, sa, dsmData)
MSDU in from DSM.
MsduIndicate (rsdu,rpri)
MSDU in from WM.
MsduRequest (rsdu,tpri)
MSDU in from local LLC entity
'to Bss ?'
True if da is addr of asoc sta or any group addr.
(false)
(true)
tsdu:= mkFrame (data,
da, sa, DsmData), tsdu:=setFrDs (tsdu,1)
addr3(rsdu), addr2(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:=setFrDs (tsdu,1)
addr2(rsdu), addr3(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:=setFrDs (tsdu,1)
MsduRequest (tsdu, contention)
'to local LLC ?'
True if da is addr of this sta or active group addr.
'to Dsm ?'
True if da is any group addr or addr of sta not asoc here.
rsdu:= mkFrame (data,
da, sa, DsmData), rsdu:=setFrDs (rsdu,1)
MsduIndicate (rsdu, contention)
ToDsm (addr3 (rsdu),
addr2(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng))
ToDsm (addr1 (rsdu),
'to Wds ?'
True if da reached via {one or more} AP(wdsAddr).
wdsAddr, da, DsmData), tsdu:= insAddr4(sa), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)
wdsAddr, addr3(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:= insAddr4 (addr2(rsdu)), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)
wdsAddr, addr1(rsdu), substr(rsdu, sMacHdrLng, length(rsdu) - sMacHdrLng)), tsdu:= insAddr4 (addr2(rsdu)), tsdu:=setFrDs (tsdu,1), tsdu:=setToDs (tsdu,1)

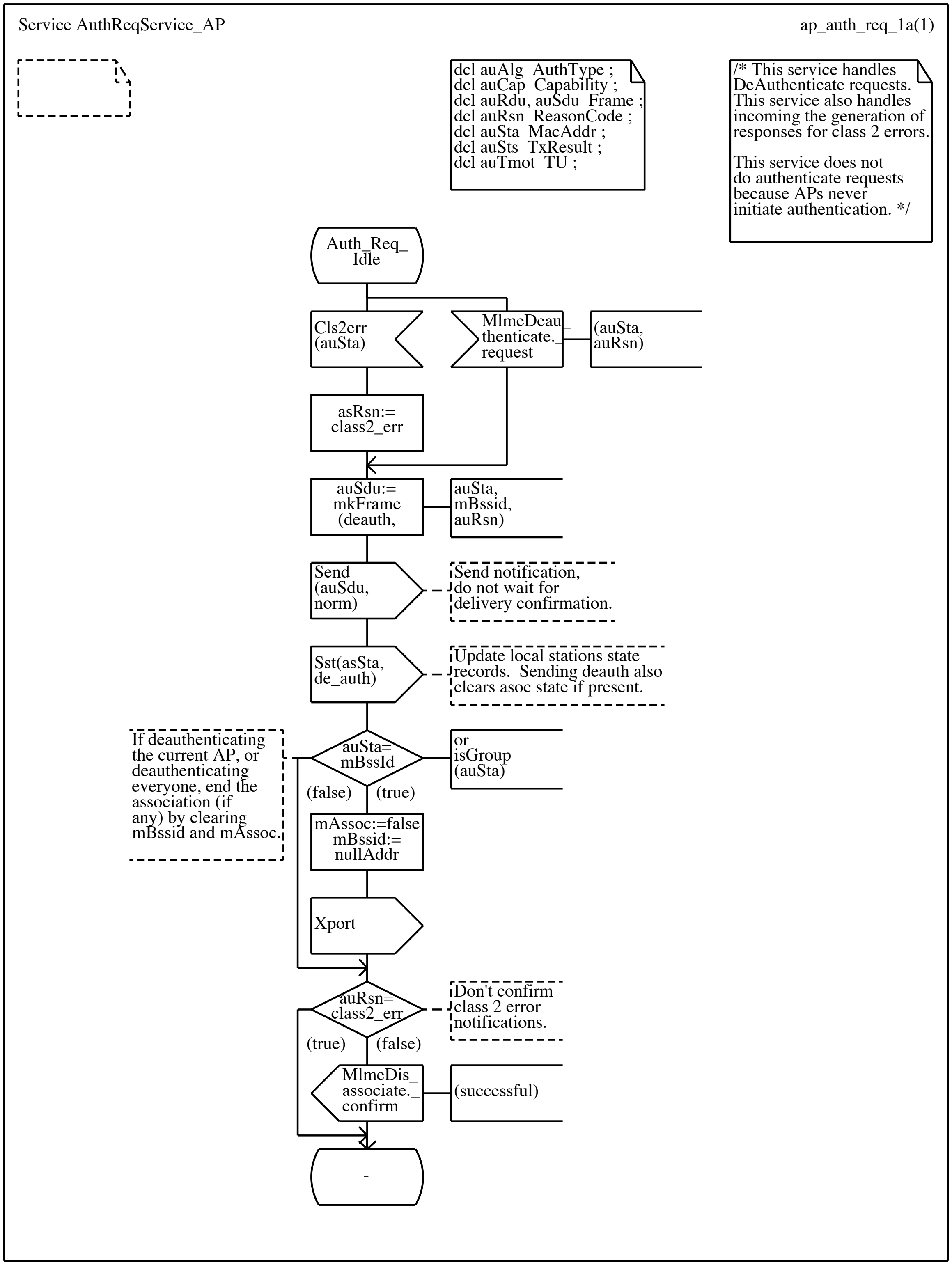

Process DSM_Interface
DSM_management_2a(2)
DS_Idle
State continued from previous page.
MsduCon_ firm(tsdu, tpri,trsl)
Response to prior MsduRequest
DsNotify (da, dss)
Notification of asoc, reasoc, or disasoc from Mlme.
DsInquiry (da, sa)
Inquiry about existing asoc status. Sent by Mlme when validating new asoc or reasoc requests.
addr3 (tsdu) =
import (aStationId)
(false)
(true)
MsduCon_ firm(tsdu, tpri,trsl)
Pass confirm of LLC request to Mac_Data_ Service.
DS_Idle
'update info about sta[da] with status dss'
DS_Idle
'dss:= ds status of station[da]'
'sa:= ap addr where da is asoc'
DsResponse (da, sa, dss)
DS_Idle

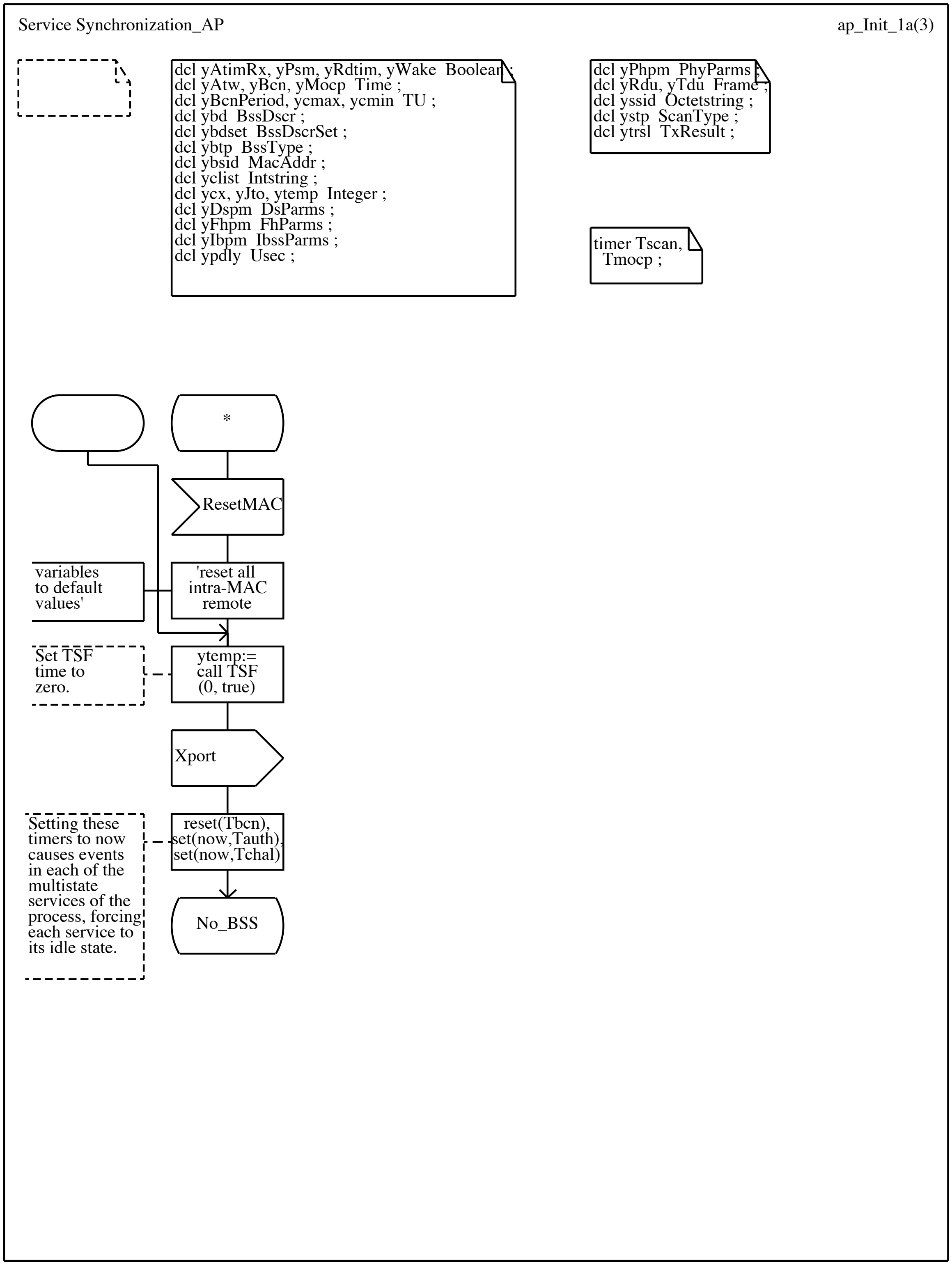

MMGT
Block MLME_AP
ap_MLME_1a(1)
Signal
StaState
(MacAddr,StationState) ;
MlmeDeauthenticate.confirm,
MlmeDisassociate.confirm,
MlmeStart.confirm,
MlmeAssociate.indication,
MlmeAuthenticate.indication,
MlmeDeauthenticate.indication,
MlmeDisassociate.indication,
MlmeReassociate.indication
/* In this block are the handlers for Mlme operation requests, the responders for incoming management frames, and the time synchronization function for the AP, as well as contention free period timing if this AP includes a PCF. This block also contains the process which maintains record of power save mode and station state for access by other processes. */
Mop
MlmeDeauthenticate.request,
MlmeDisassociate.request,
MlmeStart.request
MMTX
AsChange,
MmRequest
MmConfirm
To_Mtx
Mlme_AP_
_Services (1,1)
/* AP version */
This process assumes that the Mlme request signals have been validated by MAC Management service, and are restricted to those appropriate for use at AP.
PsChange,
PsResponse
To_Mct
MmIndicate
DsResponse
MMDS
DsInquiry,
DsNotify
To_Ds
Ssu
MmCancel,
SwChnl
StaState
Psm
PsInquiry
Power_Save_
_Monitor(1,1)
/* for STA & AP */
Records power save mode and station state.
Sst
MCTL
SsResponse
SsInquiry
PsIndicate
FromRx
ToRx
ChangeNav
PS

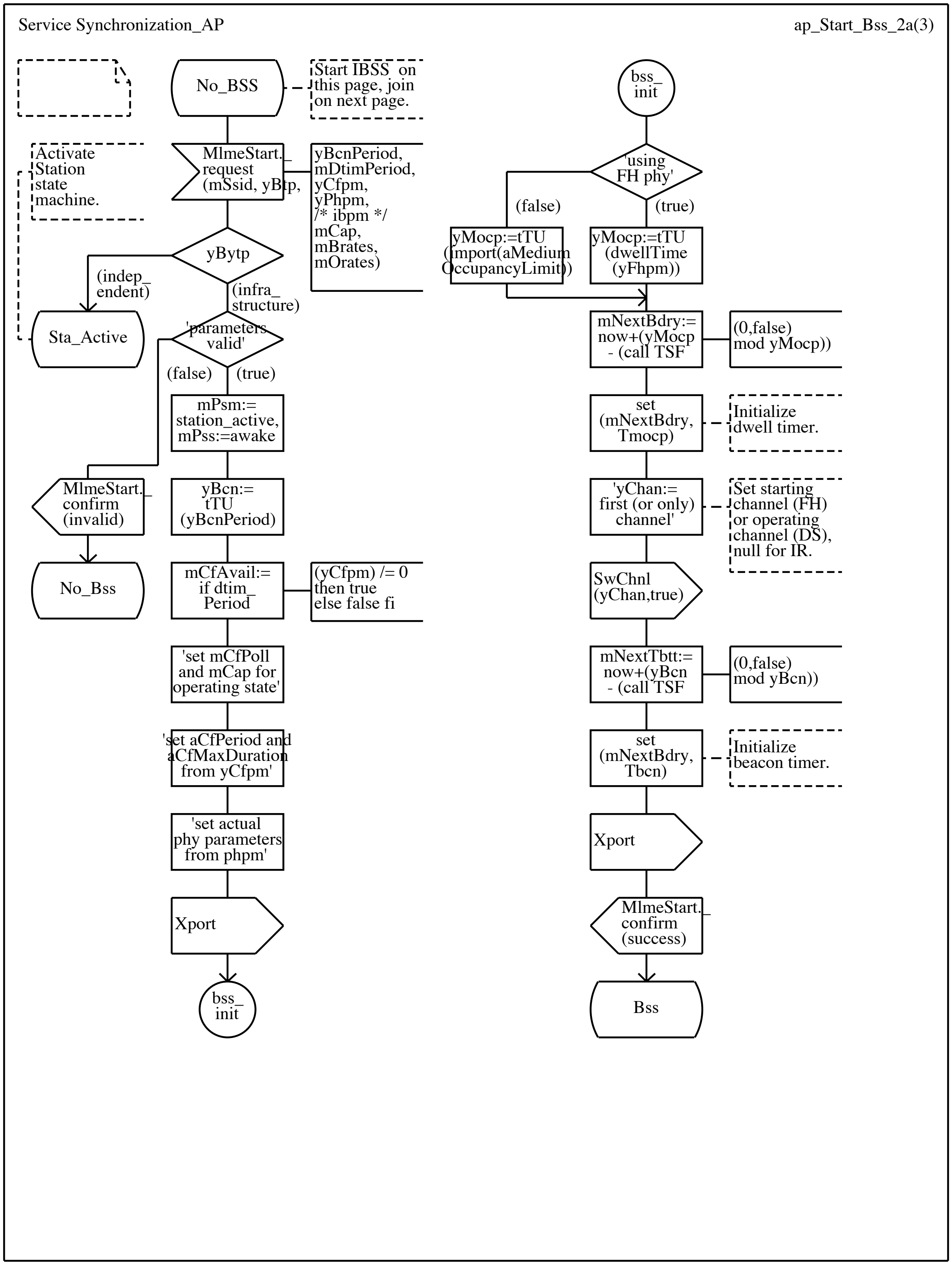

Mop
Process Mlme_AP_Services
MlmeDeauthenticate.confirm
ap_Mm_svc_1a(1)
/* Intra-MAC remote variables */
dcl exported
mAssoc Boolean:= true,
mBrates Octetstring:=mkOS(10,1),
mBssId MacAddr:= aMacAddress,
mCap Octetstring:= O2,
mCfp Boolean:= false,
mDisable Boolean:= true,
mDtimCount Integer:= 0,
mDtimPeriod Integer:= 1,
mIbss Boolean:= false,
mNextBdry Time:= 0,
mNextTbtt Time:= 0,
mOrates Octetstring:=mkOS(10,1),
mPcAvail Boolean:= sCfPollable,
mPcPoll Boolean:= false,
mPsm PwrSave:= sta_active,
mPss PsState:= awake,
mSsId Octetstring:= null ;
/* Each of these ovals represents a SERVICE. Each service contains the state transitions to handle a DISJOINT SUBSET of the input signal set of this process. Services share local variables and the input queue. At any instant, a state transition can occur in, at most, one service -- the service which handles the kind of signal at the head of the process input queue. */
MlmeAssociate.indication,
MlmeDisassociate.confirm,
MlmeDisassociate.indication,
MlmeReassociate.indication
MlmeAuthenticate.indication,
MlmeDeauthenticate.indication
MlmeStart.confirm
Cls2err
AuthReq_ Service_AP
ArqMop
MlmeDeauthenticate._ request
ArqDs
AsMop
AsocService_AP
MlmeDisassociate.request
AsChange
AsCt
Sst, Send, Xport
AsDs
DsResponse
To_ Mtx
MmRequest
Sst, Send, Xport
AsocReq, ReasocReq, AsocRsp, ReasocRsp, Disasoc, Cls3err
ArsInd
DsTx
MmConfirm
Distribute_ _Mmpdus
Sst, Send, Xport
ArsDs
Signal
AsocReq(Frame),
AsocRsp(Frame),
AuthOdd(Frame),
Beacon(Frame, Time,Time),
Cls2err(MacAddr),
Cls3err(MacAddr),
Deauth(Frame),
Disasoc(Frame),
ProbeReq(Frame),
ProbeRsp(Frame, Time,Time),
ReasocReq(Frame),
ReasocRsp(Frame),
Send(Frame,Imed),
Sent(Frame,TxStatus),
Sst(MacAddr, StationState),
Xport ;
DsSs
Ssu
StaState
Mm_ Indicate
Send, Xport
AuthOdd, Deauth
AuthRspService
DsRx
SyDs
To_ Ds
DsInquiry, DsNotify
DsDs
ProbeReq, ProbeRsp, Beacon, Sent
Timer Tauth, Tchal, Tbcn ;
To_ Mct
SyCtl
Synchronization_ _AP
SyMop
MmCancel, SwChnl
SwDone
MlmeStart.request
ResetMAC handled by Sync service.
SyRx
ChangeNav
ToRx

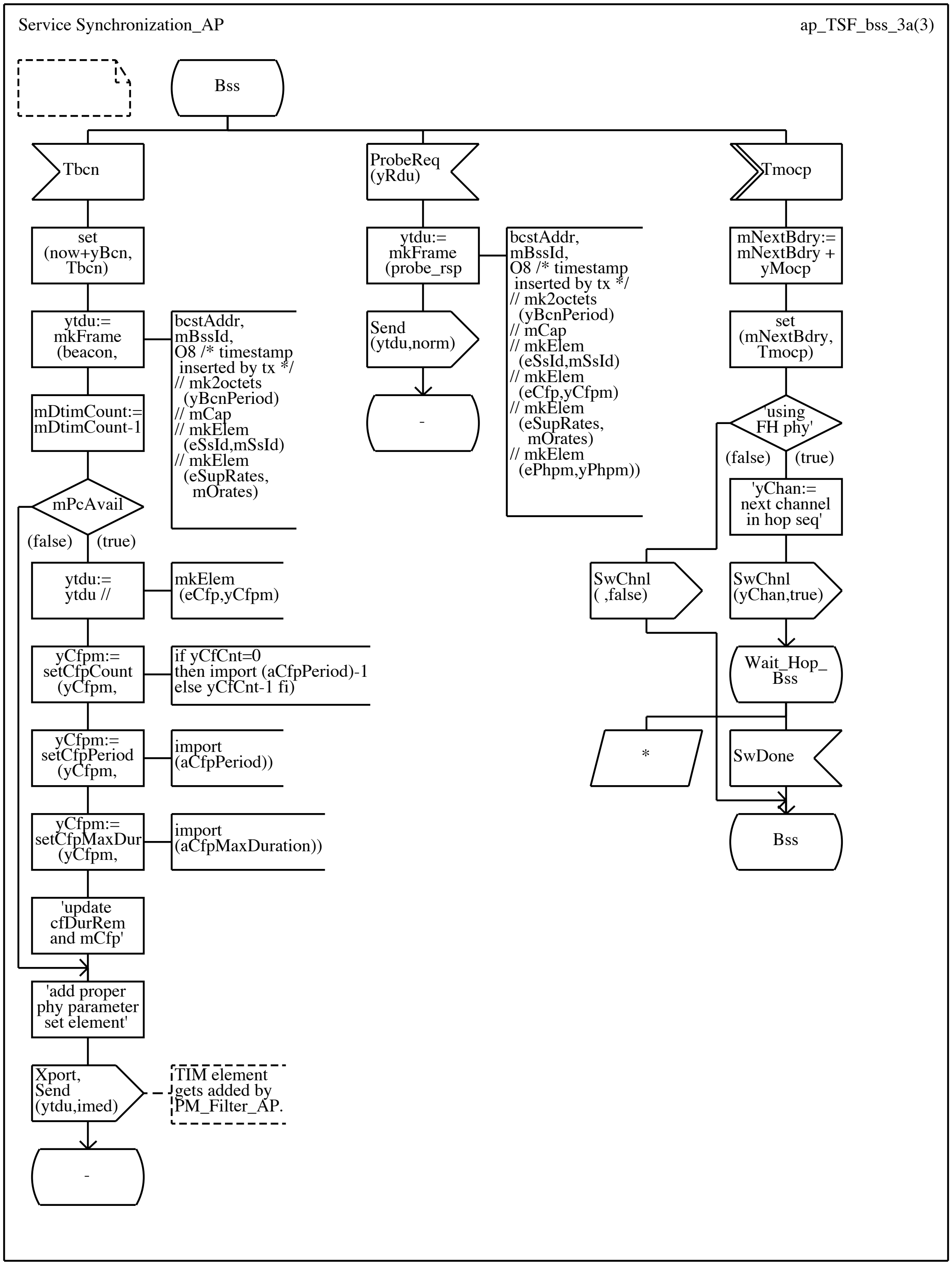

Service AsocService_AP
ap_disasoc_1b(2)
asoc_ err
reset(Tasoc)
/* This service responds to incoming Associate, and Reassociate at the AP, and handles Disassociate requests from Mlme and WM. This service also generates responses for class 3 errors. */
dcl asCap Capability ;
dcl asRsn ReasonCode ;
dcl asSta MacAddr ;
dcl asSts TxResult ;
dcl asStat DsStatus ;
dcl asRdu, asSdu Frame ;
Asoc_Idle
On this page are Disassociate request, incoming Disassociation frame, and class 3 error. More on next page.
Disasoc (asRdu)
Cls3err (asSta)
MlmeDis_ associate._ request
(asSta, asRsn)
asRsn:= class3_err
asSdu:= mkFrame (disasoc,
asSta, mBssid, asRsn)
addr1(asRdu) = mBssid
(true)
(false)
-
Send (asSdu, norm)
MlmeDis_ associate._ indication
(addr2(asRdu), reason(asRdu))
Sst(asSta, dis_asoc)
Local station state updated even if notification frame is undeliverable.
Sst(asSta, dis_asoc)
Update station state regarding this association.
asRsn= class3_err
Don't confirm class 3 error notifications.
(true)
(false)
AsChange (asRdu, disasoc)
Remove association data recorded for this station.
MlmeDis_ associate._ confirm
(successful)
Xport
-
-

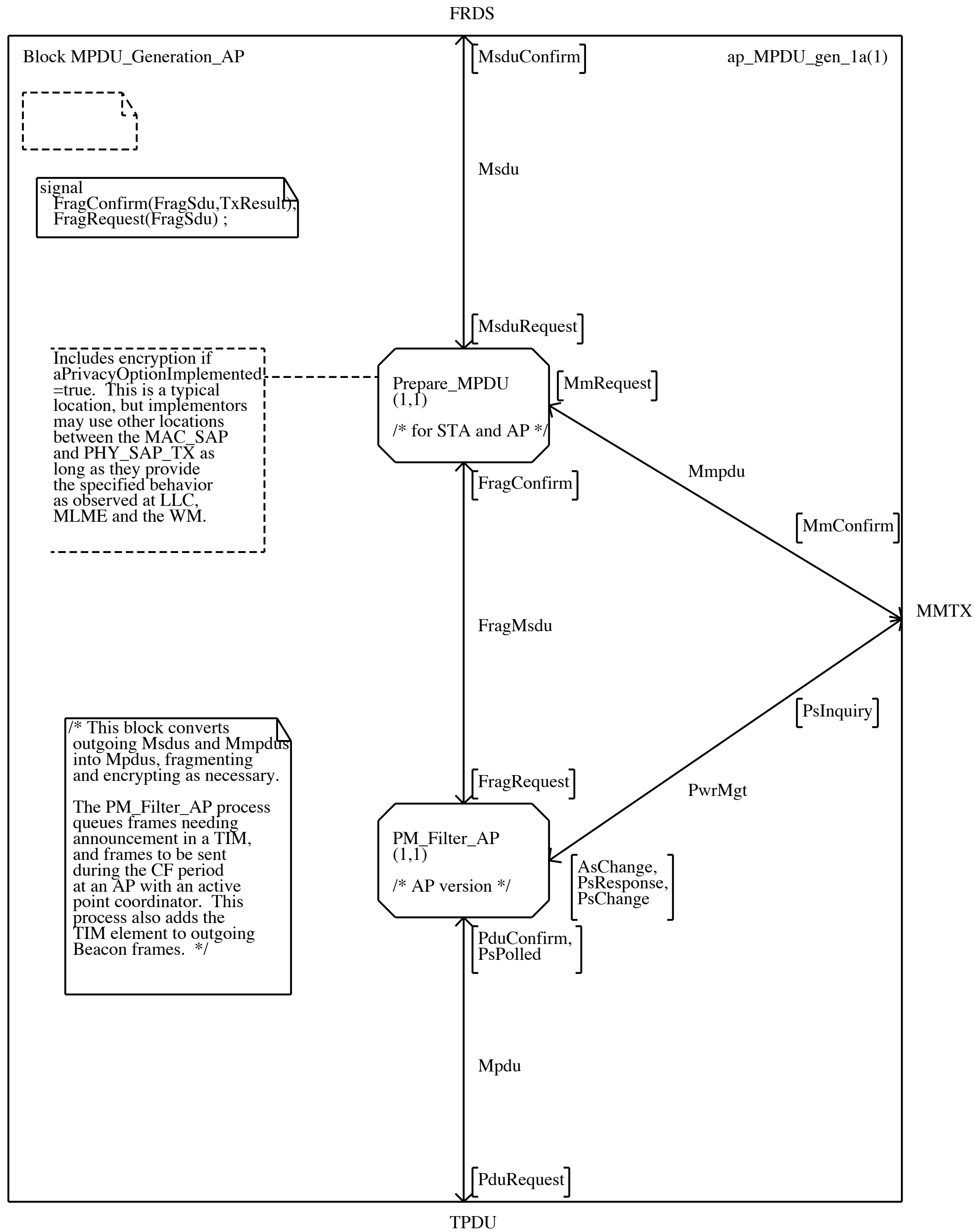

Service AsocService_AP
ap_asoc_reasoc_2a(2)
Asoc_Idle
On this page are responses to associate and reassociate requests. More of this state on previous page.
AsocReq (asRdu)
ReasocReq (asRdu)
DsInquiry (addr2(asRdu), mBssId)
DsInquiry (addr2(asRdu), mBssId)
Wait_Asoc_ _Status
Wait_Reasoc_ _Status
DsResponse (, ,asStat)
DsResponse (, ,asStat)
asStat
(disasoc, unknown)
else
asStat
(asoc)
else
'assign AId'
'assign AId'
'save request info(AId)'
'save request info(AId)'
'make asoc_rsp (success)'
'make asoc_rsp (fail)'
'make reasoc_rsp (success)'
'make reasoc_rsp (fail)'
DsNotify (addr2(asRdu), asStat)
DsNotify (addr2(asRdu), reasoc)
AsChange (asRdu, asStat)
AsChange (asRdu, asStat)
Send (asSdu, norm)
Send (asSdu, norm)
Asoc_Idle
Asoc_Idle

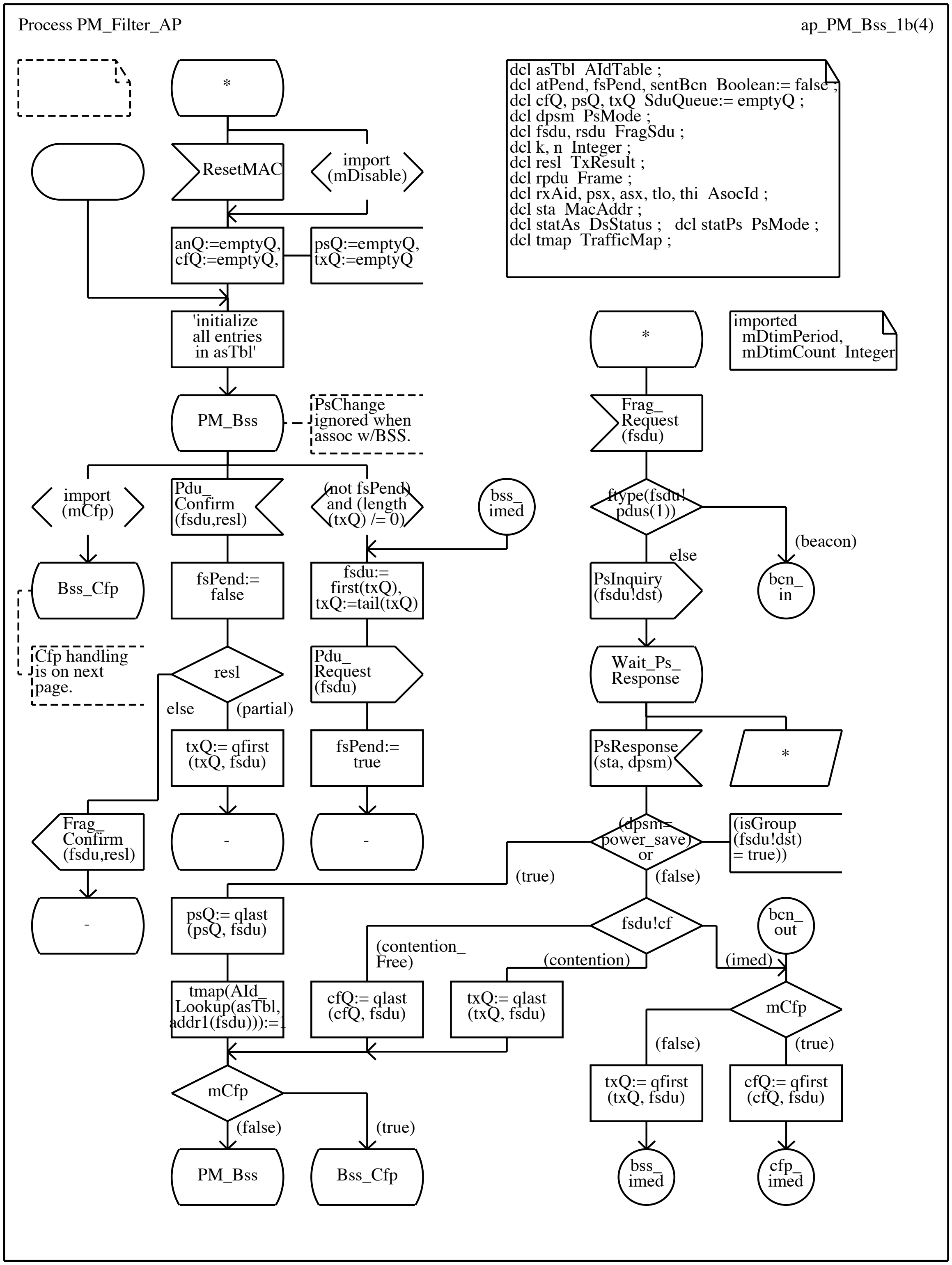

Service AuthReqService_AP
ap_auth_req_1a(1)
dcl auAlg AuthType ;
dcl auCap Capability ;
dcl auRdu, auSdu Frame ;
dcl auRsn ReasonCode ;
dcl auSta MacAddr ;
dcl auSts TxResult ;
dcl auTmot TU ;
/* This service handles DeAuthenticate requests. This service also handles incoming the generation of responses for class 2 errors.
This service does not do authenticate requests because APs never initiate authentication. */
Auth_Req_ Idle
Cls2err (auSta)
MlmeDeau_ thenticate._ request
(auSta, auRsn)
asRsn:= class2_err
auSdu:= mkFrame (deauth,
auSta, mBssid, auRsn)
Send (auSdu, norm)
Send notification, do not wait for delivery confirmation.
Sst(asSta, de_auth)
Update local stations state records. Sending deauth also clears asoc state if present.
If deauthenticating the current AP, or deauthenticating everyone, end the association (if any) by clearing mBssid and mAssoc.
auSta= mBssId
or isGroup (auSta)
(false)
(true)
mAssoc:=false mBssid:= nullAddr
Xport
auRsn= class2_err
Don't confirm class 2 error notifications.
(true)
(false)
MlmeDis_ associate._ confirm
(successful)
-

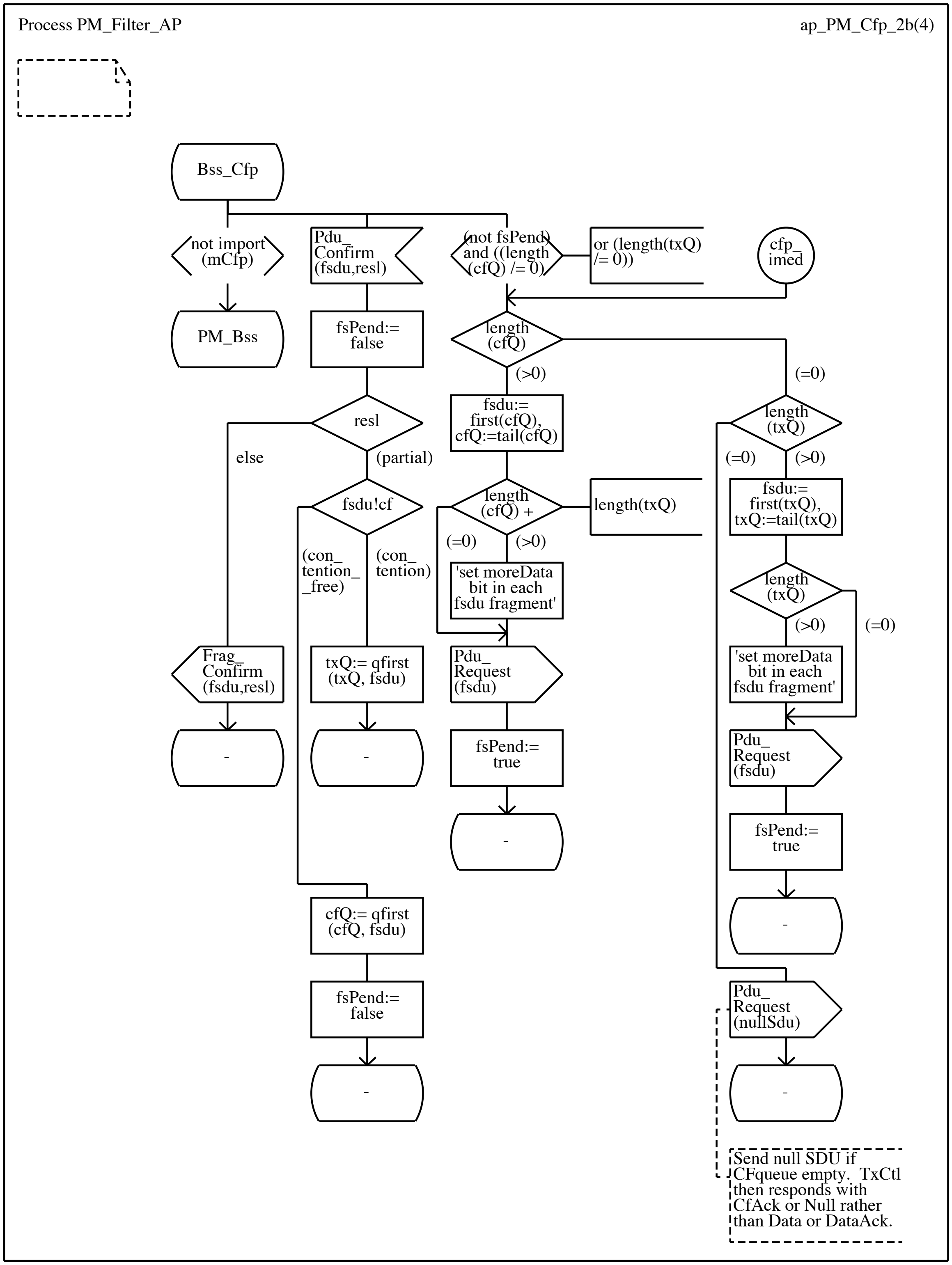

Service Synchronization_AP
ap_Init_1a(3)
dcl yAtimRx, yPsm, yRdtim, yWake Boolean ;
dcl yAtw, yBcn, yMocp Time ;
dcl yBcnPeriod, ycmax, ycmin TU ;
dcl ybd BssDscr ;
dcl ybdset BssDscrSet ;
dcl ybtp BssType ;
dcl ybsid MacAddr ;
dcl yclist Intstring ;
dcl ycx, yJto, ytemp Integer ;
dcl yDspm DsParms ;
dcl yFhpm FhParms ;
dcl yIbpm IbssParms ;
dcl ypdly Usec ;
dcl yPhpm PhyParms ;
dcl yRdu, yTdu Frame ;
dcl yssid Octetstring ;
dcl ystp ScanType ;
dcl ytrsl TxResult ;
timer Tscan, Tmocp ;
*
ResetMAC
'reset all intra-MAC remote variables to default values'
Set TSF time to zero.
ytemp:= call TSF (0, true)
Xport
Setting these timers to now causes events in each of the multistate services of the process, forcing each service to its idle state.
reset(Tbcn), set(now,Tauth), set(now,Tchal)
No_BSS

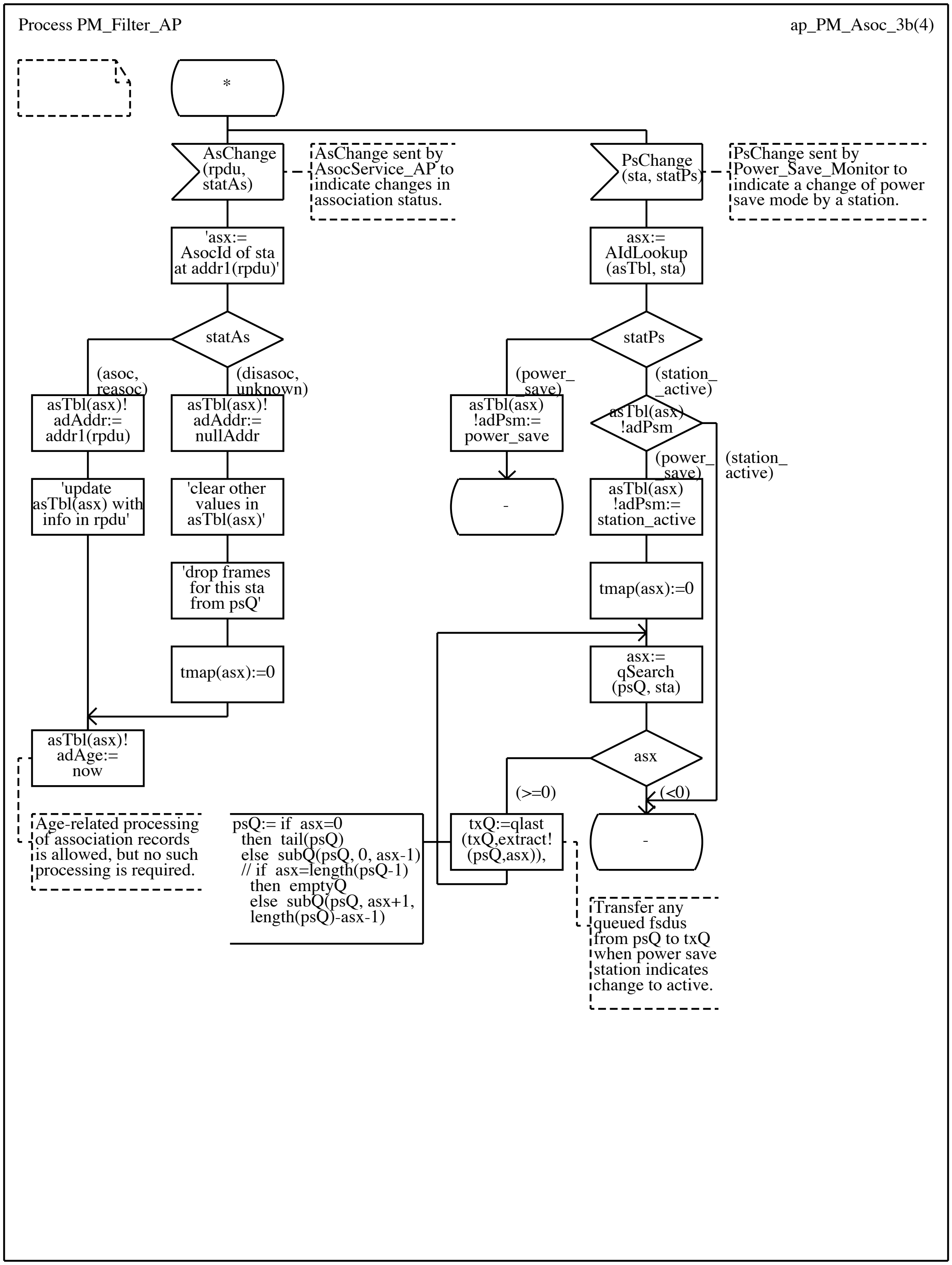

Service Synchronization_AP
ap_Start_Bss_2a(3)
No_BSS
Start IBSS on this page, join on next page.
MlmeStart_ request (mSsid, yBtp,
yBcnPeriod, mDtimPeriod, yCfpm, yPhpm, /* ibpm */ mCap, mBrates, mOrates)
Activate Station state machine.
yBytp
(indep_ endent)
(infra_ structure)
Sta_Active
'parameters valid'
(false)
(true)
mPsm:= station_active, mPss:=awake
MlmeStart_ confirm (invalid)
No_Bss
yBcn:= tTU (yBcnPeriod)
mCfAvail:= if dtim_ Period
(yCfpm) /= 0 then true else false fi
'set mCfPoll and mCap for operating state'
'set aCfPeriod and aCfMaxDuration from yCfpm'
'set actual phy parameters from phpm'
Xport
bss_ init
bss_ init
'using FH phy'
(false)
(true)
yMocp:=tTU (import(aMedium OccupancyLimit))
yMocp:=tTU (dwellTime (yFhpm))
mNextBdry:= now+(yMocp - (call TSF
(0,false) mod yMocp))
set (mNextBdry, Tmocp)
Initialize dwell timer.
'yChan:= first (or only) channel'
Set starting channel (FH) or operating channel (DS), null for IR.
SwChnl (yChan,true)
mNextTbtt:= now+(yBcn - (call TSF
(0,false) mod yBcn))
set (mNextBdry, Tbcn)
Initialize beacon timer.
Xport
MlmeStart_ confirm (success)
Bss

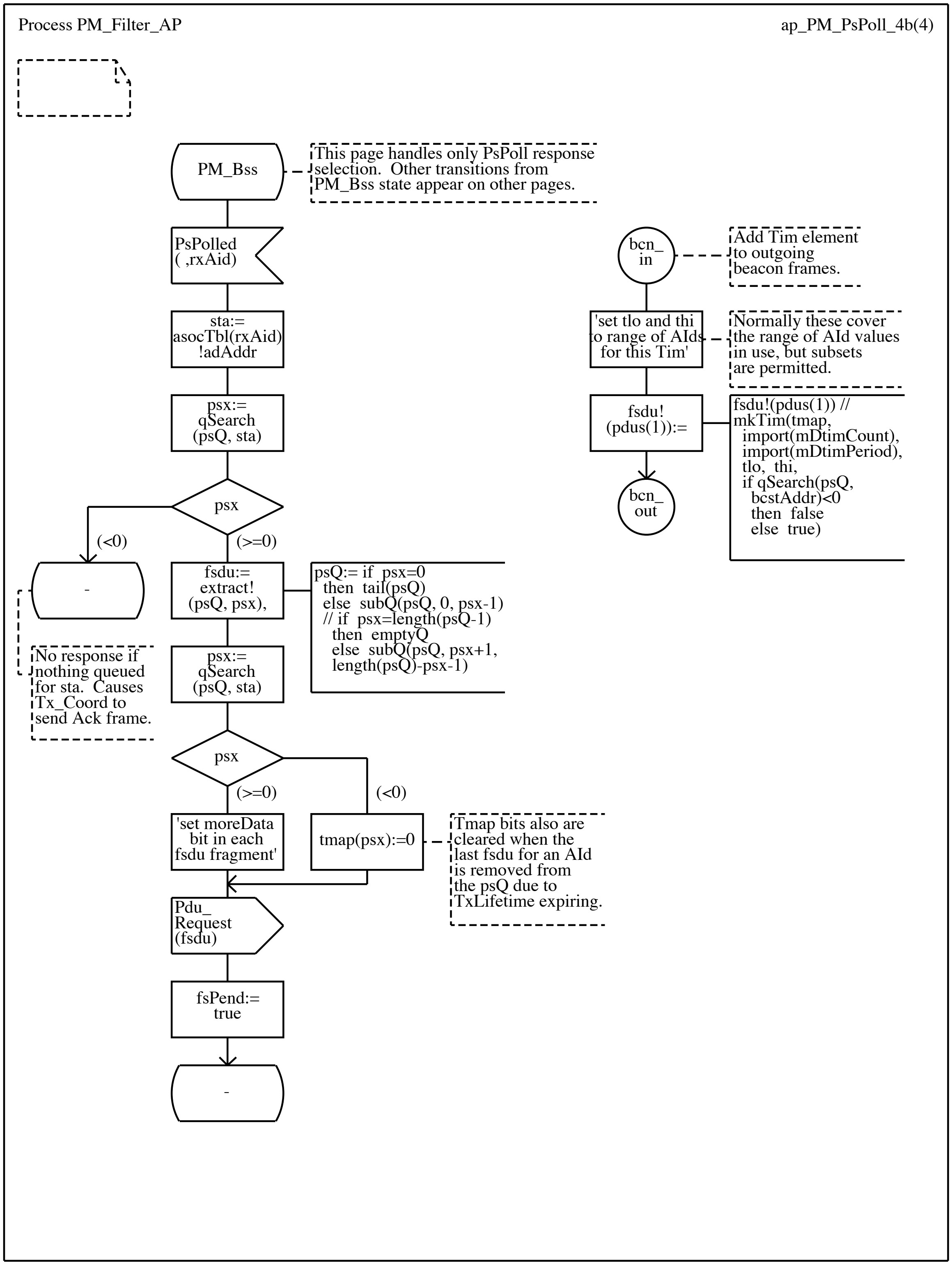

Service Synchronization_AP
ap_TSF_bss_3a(3)
Bss
Tbcn
ProbeReq (yRdu)
Tmocp
set (now+yBcn, Tbcn)
ytdu:= mkFrame (probe_rsp
bcstAddr, mBssId, O8 /* timestamp inserted by tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId,mSsId) // mkElem (eCfp,yCfpm) // mkElem (eSupRates, mOrates) // mkElem (ePhpm,yPhpm))
mNextBdry:= mNextBdry + yMocp
ytdu:= mkFrame (beacon,
bcstAddr, mBssId, O8 /* timestamp inserted by tx */ // mk2octets (yBcnPeriod) // mCap // mkElem (eSsId,mSsId) // mkElem (eSupRates, mOrates)
Send (ytdu,norm)
set (mNextBdry, Tmocp)
mDtimCount:= mDtimCount-1
-
'using FH phy'
(false)
(true)
mPcAvail
(false)
(true)
'yChan:= next channel in hop seq'
ytdu:= ytdu //
mkElem (eCfp,yCfpm)
SwChnl (,false)
SwChnl (yChan,true)
yCfpm:= setCfpCount (yCfpm,
if yCfCnt=0 then import (aCfpPeriod)-1 else yCfCnt-1 fi)
Wait_Hop_ Bss
yCfpm:= setCfpPeriod (yCfpm,
import (aCfpPeriod))
*
SwDone
yCfpm:= setCfpMaxDur (yCfpm,
import (aCfpMaxDuration))
Bss
'update cfDurRem and mCfp'
'add proper phy parameter set element'
Xport, Send (ytdu,imed)
TIM element gets added by PM_Filter_AP.
-

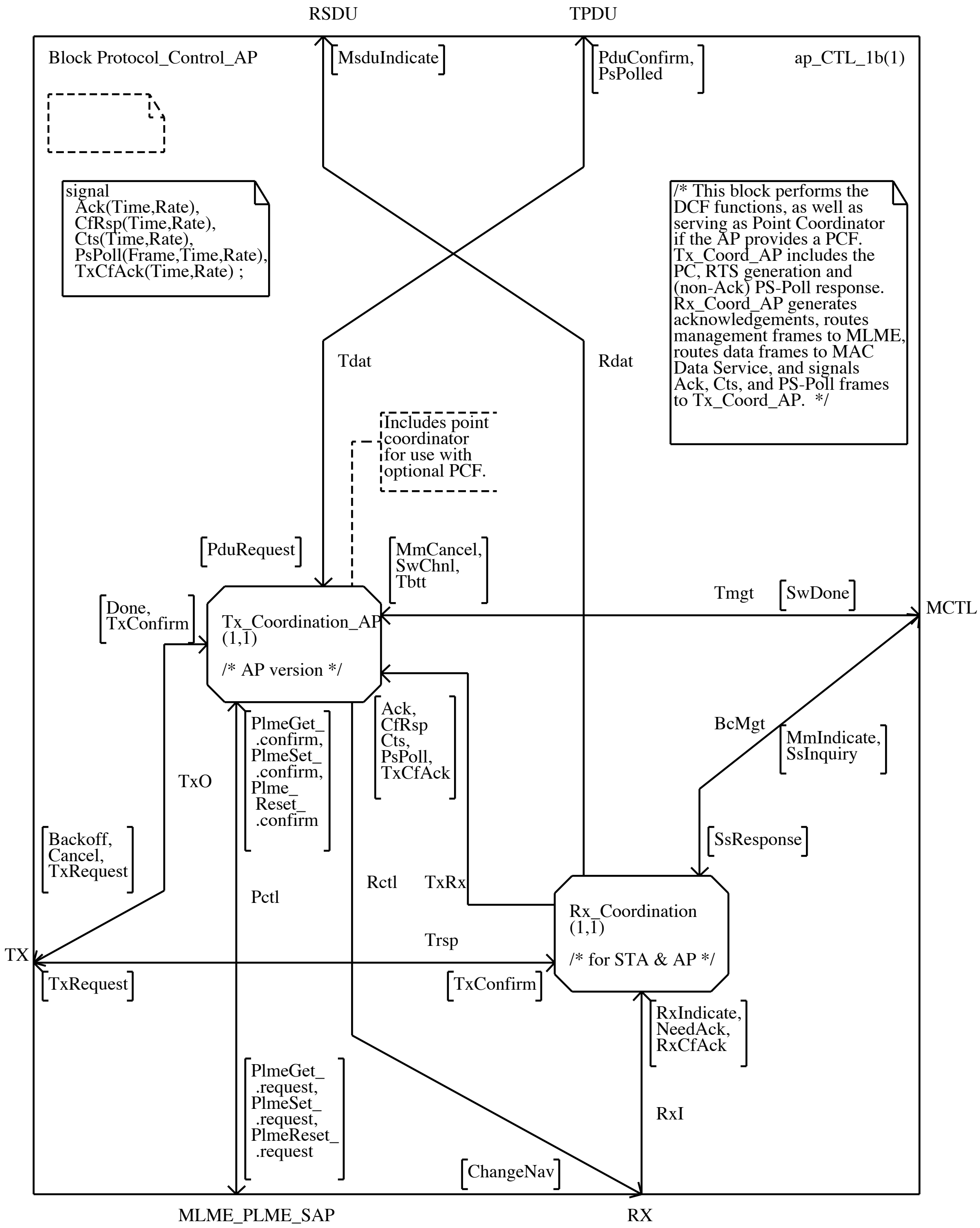

FRDS
Block MPDU_Generation_AP
ap_MPDU_gen_1a(1)
MsduConfirm
Msdu
signal
FragConfirm(FragSdu,TxResult),
FragRequest(FragSdu) ;
MsduRequest
Includes encryption if aPrivacyOptionImplemented =true. This is a typical location, but implementors may use other locations between the MAC_SAP and PHY_SAP_TX as long as they provide the specified behavior as observed at LLC, MLME and the WM.
Prepare_MPDU (1,1)
/* for STA and AP */
MmRequest
Mmpdu
FragConfirm
MmConfirm
MMTX
FragMsdu
PsInquiry
/* This block converts outgoing Msdus and Mmpdus into Mpdus, fragmenting and encrypting as necessary.
The PM_Filter_AP process queues frames needing announcement in a TIM, and frames to be sent during the CF period at an AP with an active point coordinator. This process also adds the TIM element to outgoing Beacon frames. */
FragRequest
PwrMgt
PM_Filter_AP (1,1)
/* AP version */
AsChange, PsResponse, PsChange
PduConfirm, PsPolled
Mpdu
PduRequest
TPDU

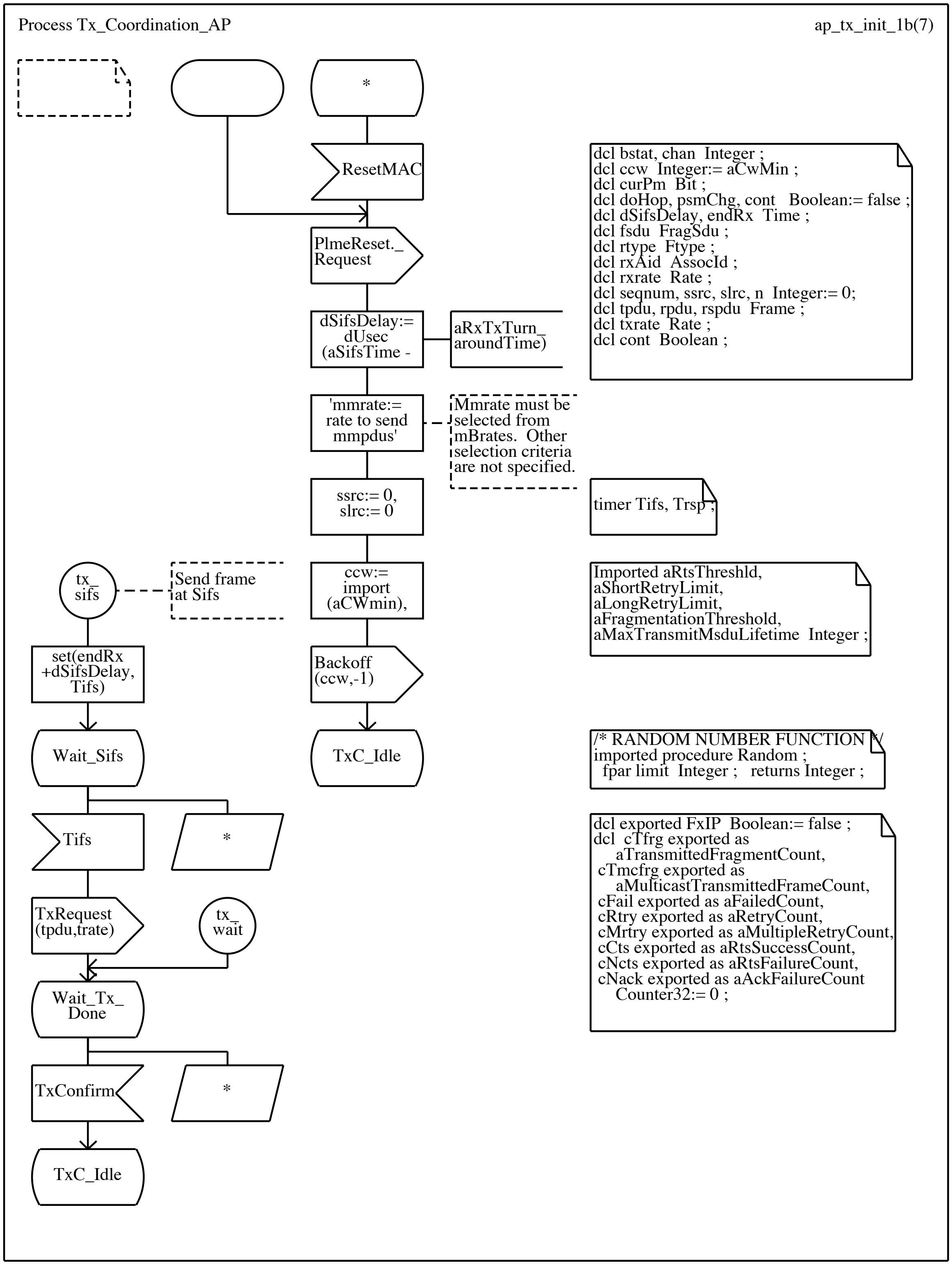

Process PM_Filter_AP
ap_PM_Bss_1b(4)
*
ResetMAC
import (mDisable)
anQ:=emptyQ, cfQ:=emptyQ,
psQ:=emptyQ, txQ:=emptyQ
'initialize all entries in asTbl'
PM_Bss
PsChange ignored when assoc w/BSS.
dcl asTbl AIdTable ;
dcl atPend, fsPend, sentBcn Boolean:= false ;
dcl cfQ, psQ, txQ SduQueue:= emptyQ ;
dcl dpsm PsMode ;
dcl fsdu, rsdu FragSdu ;
dcl k, n Integer ;
dcl resl TxResult ;
dcl rpdu Frame ;
dcl rxAid, psx, asx, tlo, thi AsocId ;
dcl sta MacAddr ;
dcl statAs DsStatus ; dcl statPs PsMode ;
dcl tmap TrafficMap ;
imported mDtimPeriod, mDtimCount Integer
import (mCfp)
Pdu_ Confirm (fsdu,resl)
(not fsPend) and (length (txQ) /= 0)
bss_ imed
Bss_Cfp
Cfp handling is on next page.
fsPend:= false
resl
else
(partial)
txQ:= qfirst (txQ, fsdu)
fsdu:= first(txQ), txQ:=tail(txQ)
Pdu_ Request (fsdu)
fsPend:= true
Frag_ Confirm (fsdu,resl)
-
Frag_ Request (fsdu)
ftype(fsdu! pdus(1))
(beacon)
else
bcn_ in
PsInquiry (fsdu!dst)
Wait_Ps_ Response
PsResponse (sta, dpsm)
(dpsm= power_save) or
(isGroup (fsdu!dst) = true))
(true)
(false)
psQ:= qlast (psQ, fsdu)
fsdu!cf
bcn_ out
(contention_ Free)
(contention)
(imed)
tmap(AId_ Lookup(asTbl, addr1(fsdu))):=1
cfQ:= qlast (cfQ, fsdu)
txQ:= qlast (txQ, fsdu)
mCfp
txQ:= qfirst (txQ, fsdu)
cfQ:= qfirst (cfQ, fsdu)
PM_Bss
Bss_Cfp
bss_ imed
cfp_ imed

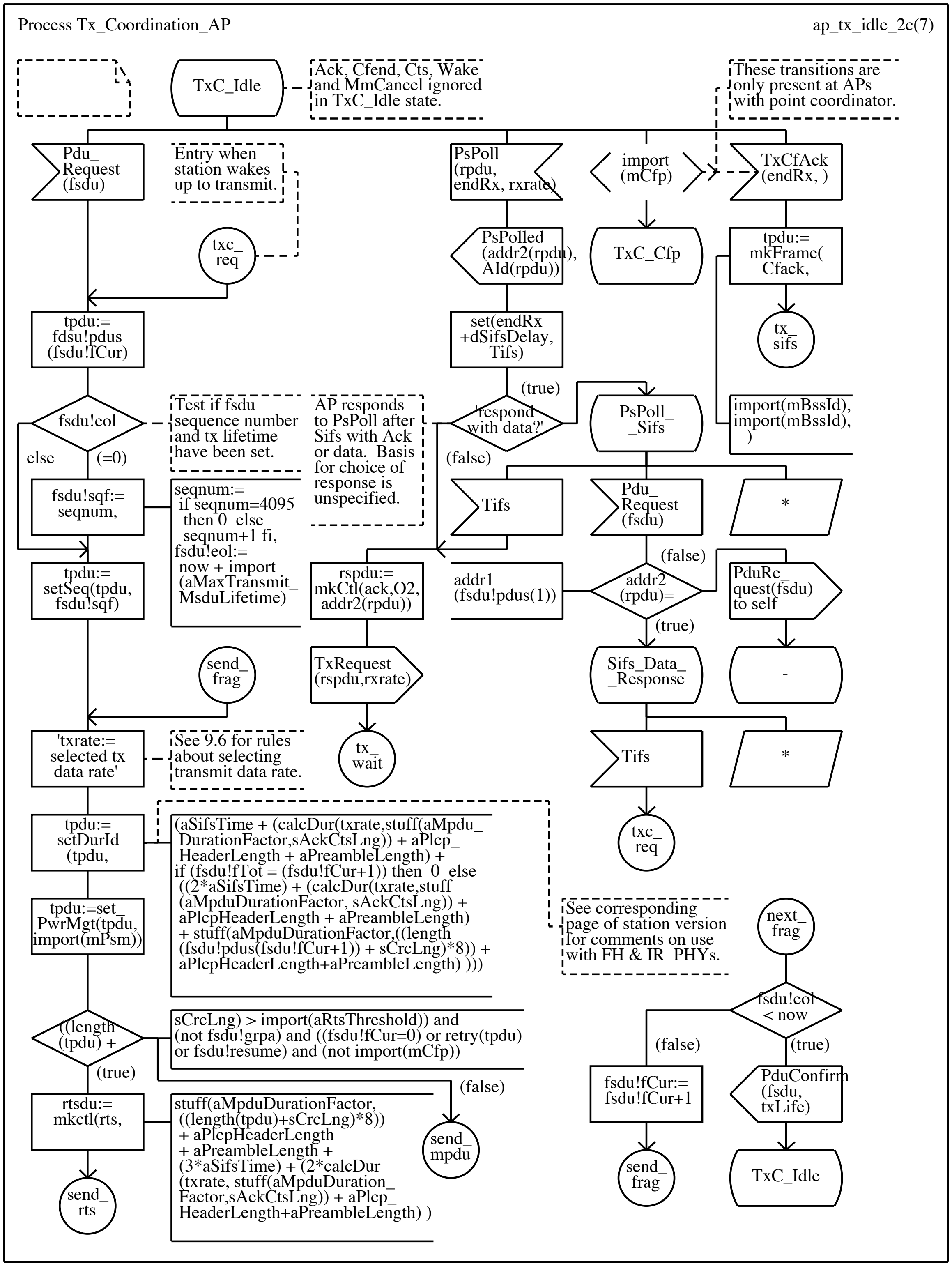

Process PM_Filter_AP
ap_PM_Cfp_2b(4)
Bss_Cfp
not import (mCfp)
PM_Bss
Pdu_ Confirm (fsdu,resl)
fsPend:= false
resl
else
(partial)
fsdu!cf
(con_ tention_ _free)
(con_ tention)
Frag_ Confirm (fsdu,resl)
-
txQ:= qfirst (txQ, fsdu)
-
cfQ:= qfirst (cfQ, fsdu)
fsPend:= false
-
(not fsPend) and ((length (cfQ) /= 0)
or (length(txQ) /= 0))
cfp_ imed
length (cfQ)
(>0)
(=0)
fsdu:= first(cfQ), cfQ:=tail(cfQ)
length (cfQ) +
length(txQ)
(=0)
(>0)
'set moreData bit in each fsdu fragment'
Pdu_ Request (fsdu)
fsPend:= true
-
length (txQ)
(=0)
(>0)
fsdu:= first(txQ), txQ:=tail(txQ)
length (txQ)
(>0)
(=0)
'set moreData bit in each fsdu fragment'
Pdu_ Request (fsdu)
fsPend:= true
-
Pdu_ Request (nullSdu)
-
Send null SDU if CFqueue empty. TxCtl then responds with CfAck or Null rather than Data or DataAck.

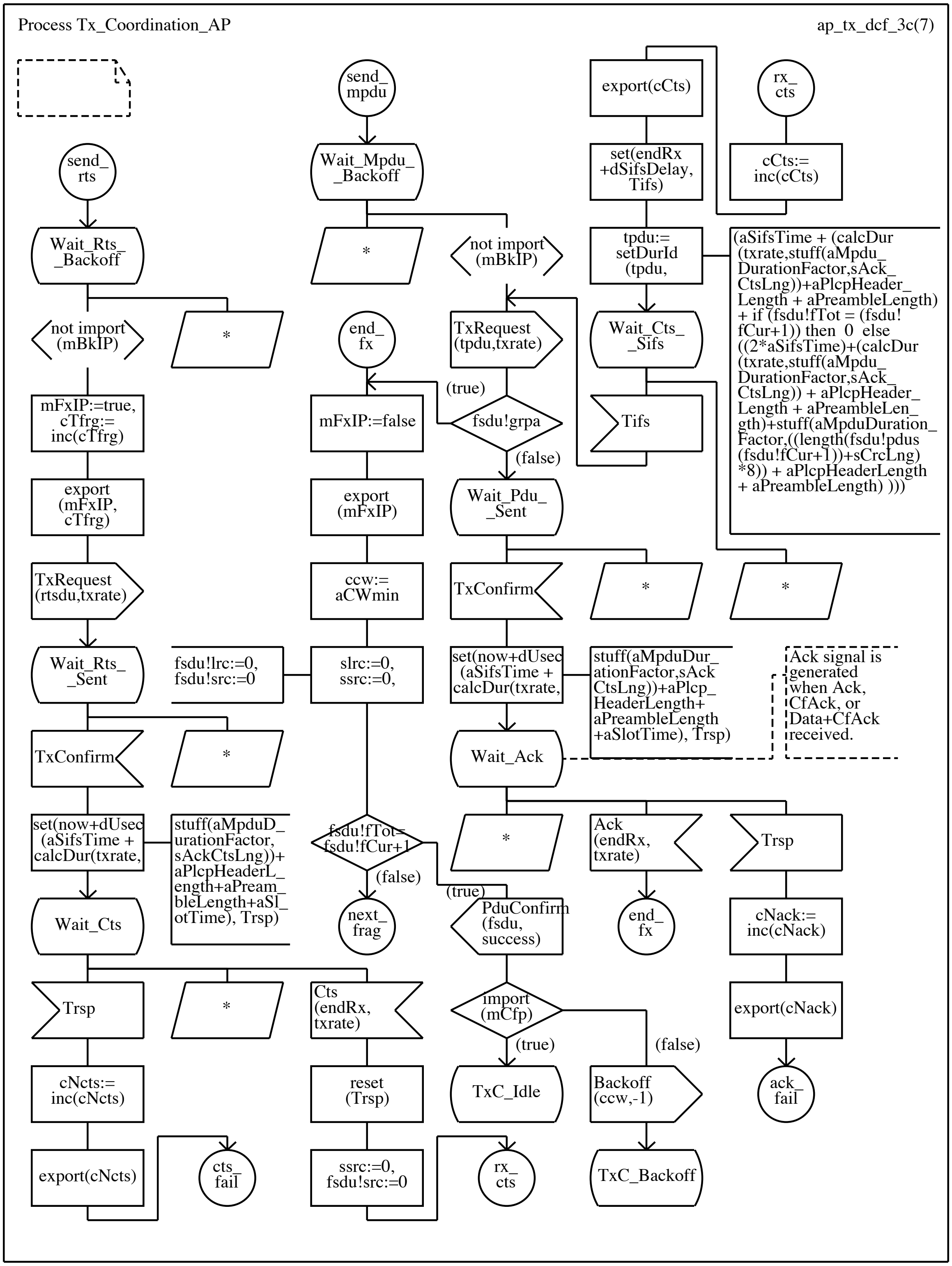

Process PM_Filter_AP
ap_PM_Asoc_3b(4)
*
AsChange (rpdu, statAs)
AsChange sent by AsocService_AP to indicate changes in association status.
PsChange (sta, statPs)
PsChange sent by Power_Save_Monitor to indicate a change of power save mode by a station.
'asx:= AsocId of sta at addr1(rpdu)'
asx:= AIdLookup (asTbl, sta)
statAs
statPs
(asoc, reasoc)
(disasoc, unknown)
(power_ save)
(station_ _active)
asTbl(asx)! adAddr:= addr1(rpdu)
asTbl(asx)! adAddr:= nullAddr
asTbl(asx) !adPsm:= power_save
asTbl(asx) !adPsm
(power_ save)
(station_ active)
'update asTbl(asx) with info in rpdu'
'clear other values in asTbl(asx)'
-
asTbl(asx) !adPsm:= station_active
'drop frames for this sta from psQ'
tmap(asx):=0
tmap(asx):=0
asx:= qSearch (psQ, sta)
asTbl(asx)! adAge:= now
asx
(>=0)
(<0)
Age-related processing of association records is allowed, but no such processing is required.
psQ:= if asx=0 then tail(psQ) else subQ(psQ, 0, asx-1) // if asx=length(psQ-1) then emptyQ else subQ(psQ, asx+1, length(psQ)-asx-1)
txQ:=qlast (txQ,extract! (psQ,asx)),
-
Transfer any queued fsdus from psQ to txQ when power save station indicates change to active.

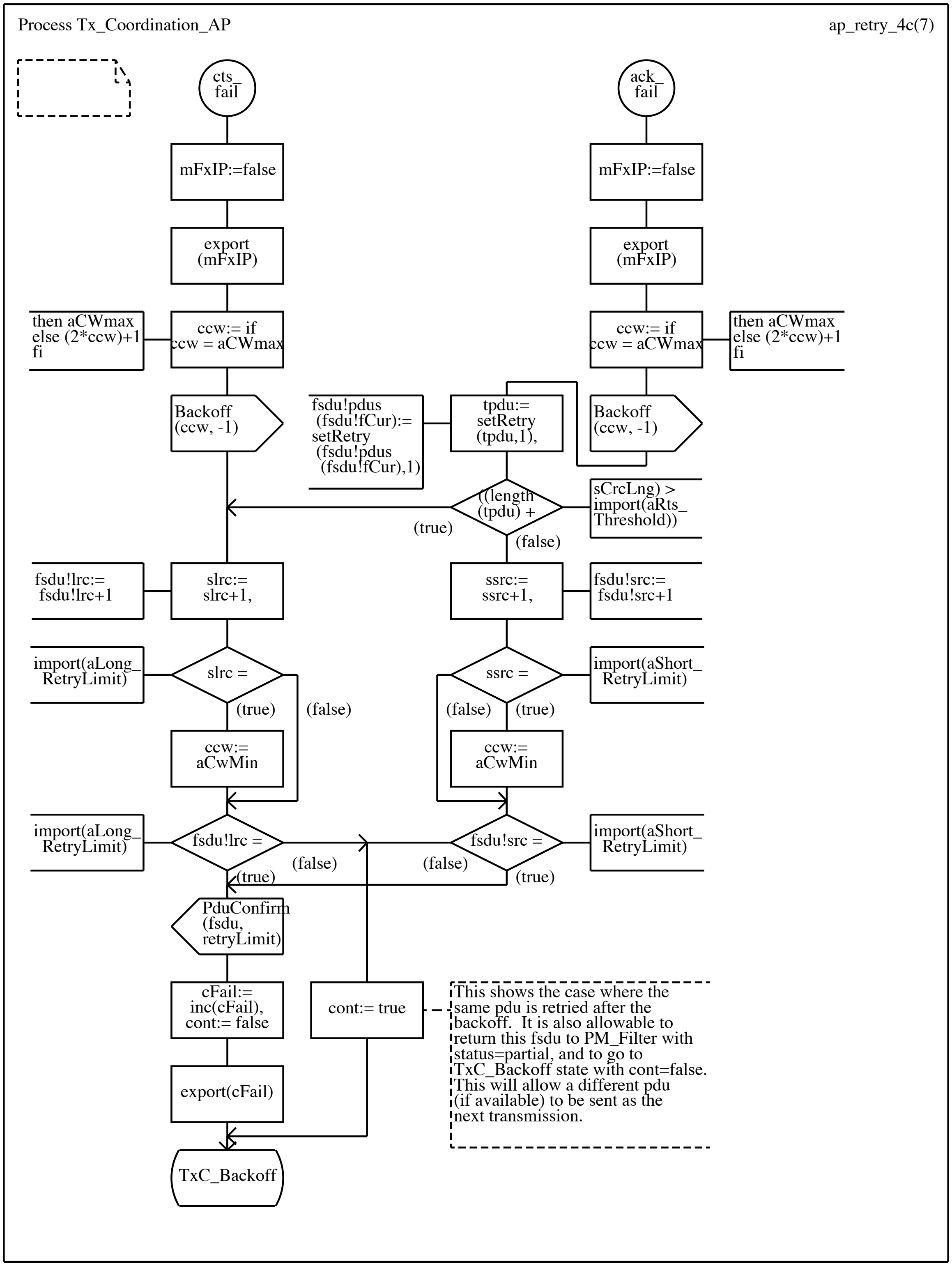

Process PM_Filter_AP
ap_PM_PsPoll_4b(4)
PM_Bss
This page handles only PsPoll response selection. Other transitions from PM_Bss state appear on other pages.
PsPolled (,rxAid)
sta:= asocTbl(rxAid) !adAddr
psx:= qSearch (psQ, sta)
psx
(<0)
(>=0)
-
No response if nothing queued for sta. Causes Tx_Coord to send Ack frame.
fsdu:= extract! (psQ, psx),
psQ:= if psx=0 then tail(psQ) else subQ(psQ, 0, psx-1) // if psx=length(psQ-1) then emptyQ else subQ(psQ, psx+1, length(psQ)-psx-1)
psx:= qSearch (psQ, sta)
psx
(>=0)
(<0)
'set moreData bit in each fsdu fragment'
tmap(psx):=0
Tmap bits also are cleared when the last fsdu for an AId is removed from the psQ due to TxLifetime expiring.
Pdu_ Request (fsdu)
fsPend:= true
-
bcn_ in
Add Tim element to outgoing beacon frames.
'set tlo and thi to range of AIds for this Tim'
Normally these cover the range of AId values in use, but subsets are permitted.
fsdu! (pdus(1)):=
fsdu!(pdus(1)) // mkTim(tmap, import(mDtimCount), import(mDtimPeriod), tlo, thi, if qSearch(psQ, bcstAddr)<0 then false else true)
bcn_ out

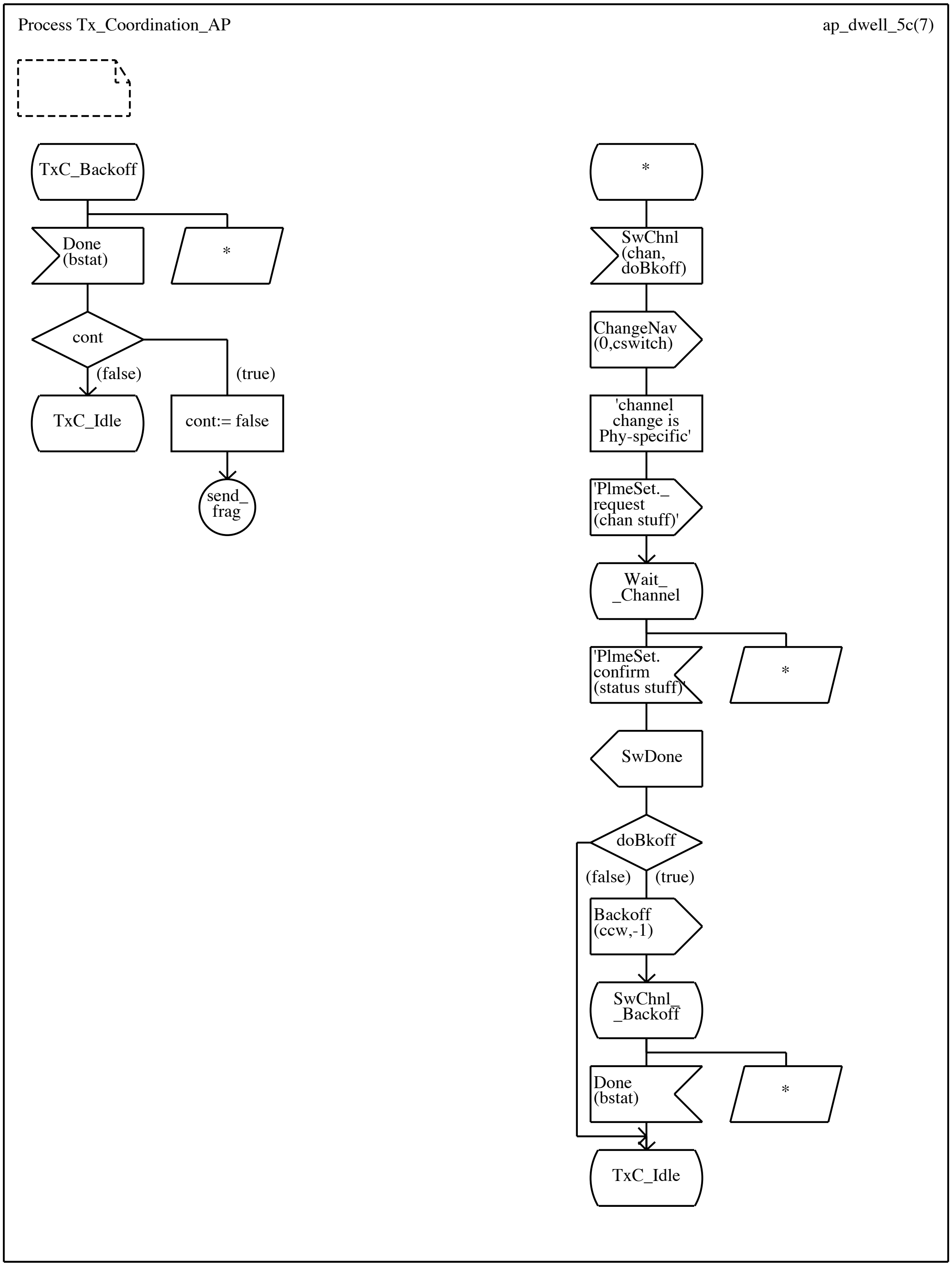

RSDU
TPDU
Block Protocol_Control_AP
MsduIndicate
PduConfirm, PsPolled
ap_CTL_1b(1)
signal
Ack(Time,Rate),
CfRsp(Time,Rate),
Cts(Time,Rate),
PsPoll(Frame,Time,Rate),
TxCfAck(Time,Rate) ;
/* This block performs the DCF functions, as well as serving as Point Coordinator if the AP provides a PCF. Tx_Coord_AP includes the PC, RTS generation and (non-Ack) PS-Poll response. Rx_Coord_AP generates acknowledgements, routes management frames to MLME, routes data frames to MAC Data Service, and signals Ack, Cts, and PS-Poll frames to Tx_Coord_AP. */
Tdat
Rdat
Includes point coordinator for use with optional PCF.
PduRequest
MmCancel, SwChnl, Tbtt
Tmgt
SwDone
MCTL
Done, TxConfirm
Tx_Coordination_AP (1,1)
/* AP version */
PlmeGet_ .confirm, PlmeSet_ .confirm, Plme_ Reset_ .confirm
Ack, CfRsp Cts, PsPoll, TxCfAck
BcMgt
MmIndicate, SsInquiry
TxO
SsResponse
Backoff, Cancel, TxRequest
Pctl
Rctl
TxRx
Rx_Coordination (1,1)
/* for STA & AP */
Trsp
TX
TxRequest
TxConfirm
RxIndicate, NeedAck, RxCfAck
PlmeGet_ .request, PlmeSet_ .request, PlmeReset_ .request
RxI
ChangeNav
MLME_PLME_SAP
RX

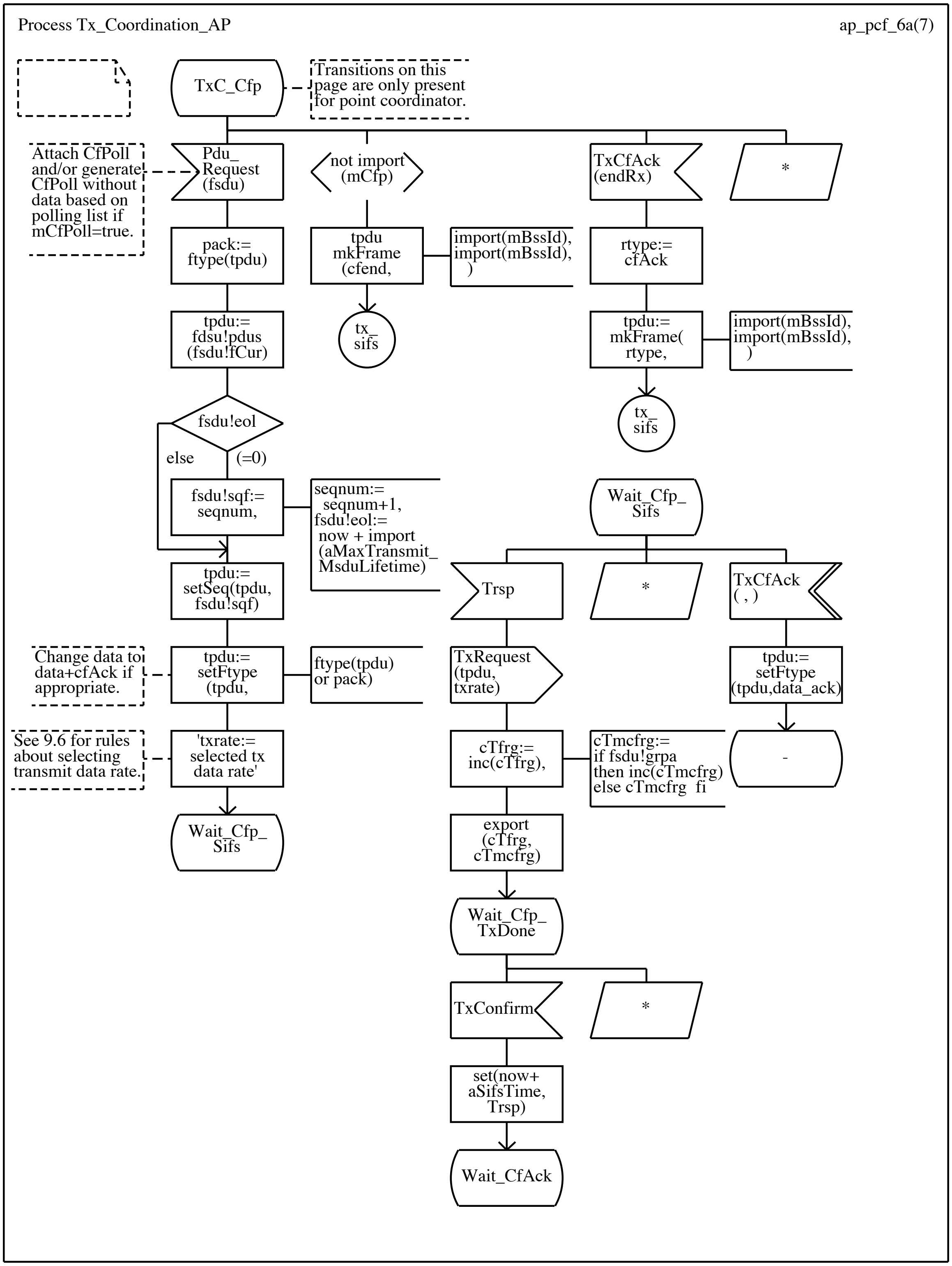

Process Tx_Coordination_AP
ap_tx_init_1b(7)
*
ResetMAC
PlmeReset._ Request
dSifsDelay:= dUsec (aSifsTime -
aRxTxTurn_ aroundTime)
'mmrate:= rate to send mmpdus'
Mmrate must be selected from mBrates. Other selection criteria are not specified.
ssrc:= 0, slrc:= 0
ccw:= import (aCWmin),
Backoff (ccw,-1)
TxC_Idle
tx_ sifs
Send frame at Sifs
set(endRx +dSifsDelay, Tifs)
Wait_Sifs
Tifs
*
TxRequest (tpdu,trate)
tx_ wait
Wait_Tx_ Done
TxConfirm
*
TxC_Idle
dcl bstat, chan Integer ;
dcl ccw Integer:= aCwMin ;
dcl curPm Bit ;
dcl doHop, psmChg, cont Boolean:= false ;
dcl dSifsDelay, endRx Time ;
dcl fsdu FragSdu ;
dcl rtype Ftype ;
dcl rxAid AssocId ;
dcl rxrate Rate ;
dcl seqnum, ssrc, slrc, n Integer:= 0;
dcl tpdu, rpdu, rspdu Frame ;
dcl txrate Rate ;
dcl cont Boolean ;
timer Tifs, Trsp ;
Imported aRtsThreshld, aShortRetryLimit, aLongRetryLimit, aFragmentationThreshold, aMaxTransmitMsduLifetime Integer ;
/* RANDOM NUMBER FUNCTION */
imported procedure Random ;
fpar limit Integer ; returns Integer ;
dcl exported FxIP Boolean:= false ;
dcl cTfrg exported as aTransmittedFragmentCount,
cTmcfrg exported as aMulticastTransmittedFrameCount,
cFail exported as aFailedCount,
cRtry exported as aRetryCount,
cMrtry exported as aMultipleRetryCount,
cCts exported as aRtsSuccessCount,
cNcts exported as aRtsFailureCount,
cNack exported as aAckFailureCount
Counter32:= 0 ;

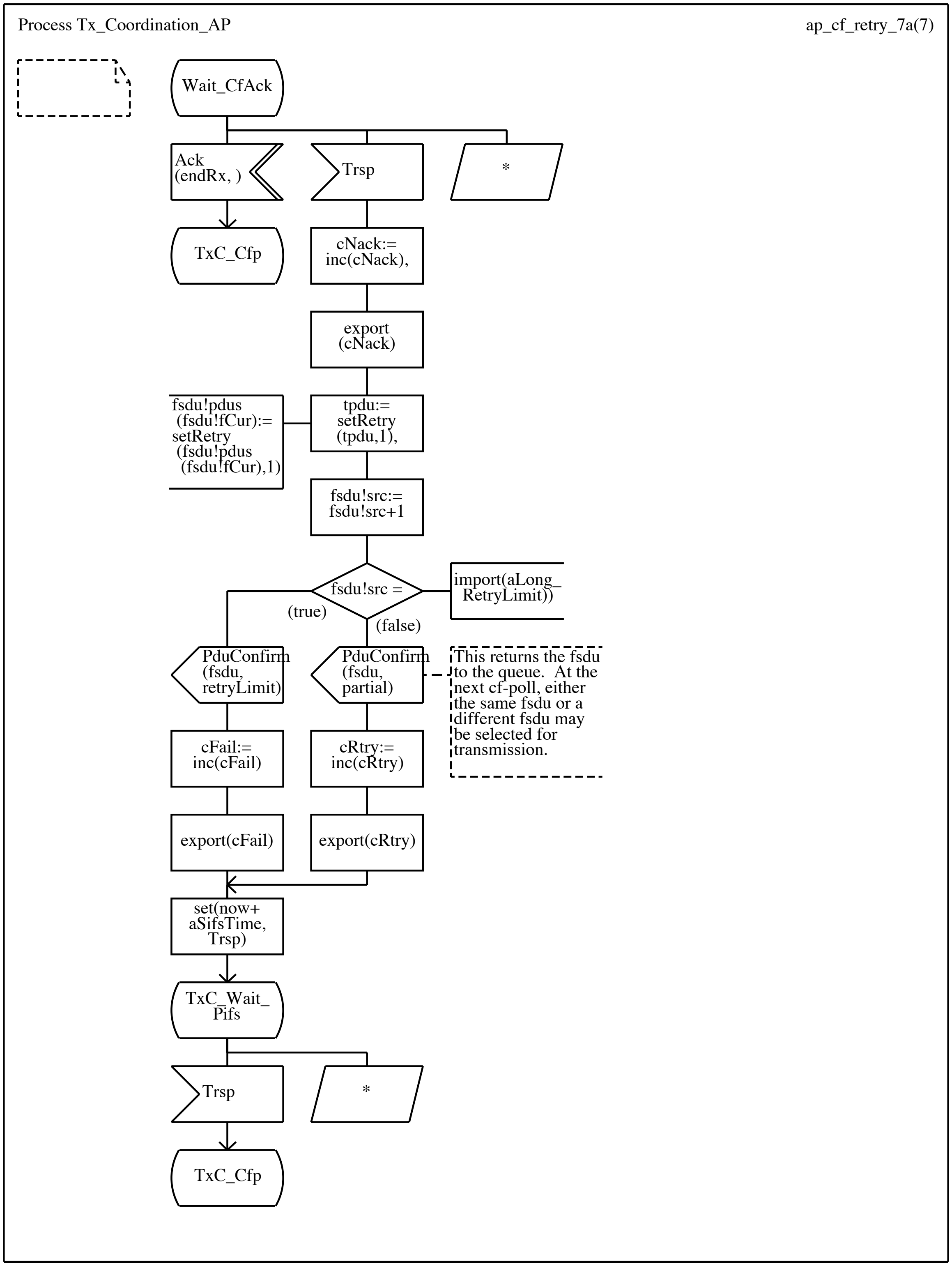

Process Tx_Coordination_AP
ap_tx_idle_2c(7)
TxC_Idle
Ack, Cfend, Cts, Wake and MmCancel ignored in TxC_Idle state.
These transitions are only present at APs with point coordinator.
Pdu_ Request (fsdu)
Entry when station wakes up to transmit.
PsPoll (rpdu, endRx, rxrate)
import (mCfp)
TxCfAck (endRx,)
txc_ req
PsPolled (addr2(rpdu), AId(rpdu))
TxC_Cfp
tpdu:= mkFrame(Cfack,
tx_ sifs
tpdu:= fdsu!pdus (fsdu!fCur)
set(endRx +dSifsDelay, Tifs)
(true)
fsdu!eol
Test if fsdu sequence number and tx lifetime have been set.
AP responds to PsPoll after Sifs with Ack or data. Basis for choice of response is unspecified.
'respond with data?'
PsPoll_ _Sifs
import(mBssId), import(mBssId),)
else
(=0)
(false)
fsdu!sqf:= seqnum,
seqnum:= if seqnum=4095 then 0 else seqnum+1 fi, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
Tifs
Pdu_ Request (fsdu)
*
(false)
tpdu:= setSeq(tpdu, fsdu!sqf)
rspdu:= mkCtl(ack,O2, addr2(rpdu))
addr1 (fsdu!pdus(1))
addr2 (rpdu)=
PduRe_ quest(fsdu) to self
(true)
send_ frag
TxRequest (rspdu,rxrate)
Sifs_Data_ _Response
-
'txrate:= selected tx data rate'
See 9.6 for rules about selecting transmit data rate.
tx_ wait
Tifs
*
tpdu:= setDurId (tpdu,
(aSifsTime + (calcDur(txrate,stuff(aMpdu_ DurationFactor,sAckCtsLng)) + aPlcp_ HeaderLength + aPreambleLength) + if (fsdu!fTot = (fsdu!fCur+1)) then 0 else ((2*aSifsTime) + (calcDur(txrate,stuff (aMpduDurationFactor, sAckCtsLng)) + aPlcpHeaderLength + aPreambleLength) + stuff(aMpduDurationFactor,((length (fsdu!pdus(fsdu!fCur+1)) + sCrcLng)*8)) + aPlcpHeaderLength+aPreambleLength))))
txc_ req
tpdu:=set_ PwrMgt(tpdu, import(mPsm))
See corresponding page of station version for comments on use with FH & IR PHYs.
next_ frag
((length (tpdu) +
sCrcLng) > import(aRtsThreshold)) and (not fsdu!grpa) and ((fsdu!fCur=0) or retry(tpdu) or fsdu!resume) and (not import(mCfp))
fsdu!eol < now
(false)
(true)
(true)
(false)
rtsdu:= mkctl(rts,
stuff(aMpduDurationFactor, ((length(tpdu)+sCrcLng)*8)) + aPlcpHeaderLength + aPreambleLength + (3*aSifsTime) + (2*calcDur (txrate, stuff(aMpduDuration_ Factor,sAckCtsLng)) + aPlcp_ HeaderLength+aPreambleLength))
send_ mpdu
fsdu!fCur:= fsdu!fCur+1
PduConfirm (fsdu, txLife)
send_ rts
send_ frag
TxC_Idle


Process Tx_Coordination_AP
ap_tx_dcf_3c(7)
send_ rts
Wait_Rts_ _Backoff
not import (mBkIP)
*
mFxIP:=true, cTfrg:= inc(cTfrg)
export (mFxIP, cTfrg)
TxRequest (rtsdu,txrate)
Wait_Rts_ _Sent
fsdu!lrc:=0, fsdu!src:=0
TxConfirm
*
set(now+dUsec (aSifsTime + calcDur(txrate,
stuff(aMpduD_ urationFactor, sAckCtsLng))+ aPlcpHeaderL_ ength+aPream_ bleLength+aSl_ otTime), Trsp)
Wait_Cts
Trsp
*
cNcts:= inc(cNcts)
export(cNcts)
cts_ fail
send_ mpdu
Wait_Mpdu_ _Backoff
*
end_ fx
mFxIP:=false
export (mFxIP)
ccw:= aCWmin
slrc:=0, ssrc:=0,
fsdu!fTot= fsdu!fCur+1
(false)
(true)
next_ frag
Cts (endRx, txrate)
reset (Trsp)
ssrc:=0, fsdu!src:=0
rx_ cts
not import (mBkIP)
TxRequest (tpdu,txrate)
(true)
fsdu!grpa
(false)
Wait_Pdu_ _Sent
TxConfirm
set(now+dUsec (aSifsTime + calcDur(txrate,
Wait_Ack
*
PduConfirm (fsdu, success)
import (mCfp)
(true)
(false)
TxC_Idle
export(cCts)
set(endRx +dSifsDelay, Tifs)
tpdu:= setDurId (tpdu,
Wait_Cts_ _Sifs
Tifs
*
stuff(aMpduDur_ ationFactor,sAck CtsLng))+aPlcp_ HeaderLength+ aPreambleLength +aSlotTime), Trsp)
Ack (endRx, txrate)
end_ fx
Backoff (ccw,-1)
TxC_Backoff
rx_ cts
cCts:= inc(cCts)
(aSifsTime + (calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng))+aPlcpHeader_ Length + aPreambleLength) + if (fsdu!fTot = (fsdu! fCur+1)) then 0 else ((2*aSifsTime)+(calcDur (txrate,stuff(aMpdu_ DurationFactor,sAck_ CtsLng)) + aPlcpHeader_ Length + aPreambleLen_ gth)+stuff(aMpduDuration_ Factor,((length(fsdu!pdus (fsdu!fCur+1))+sCrcLng) *8)) + aPlcpHeaderLength + aPreambleLength))))
*
Ack signal is generated when Ack, CfAck, or Data+CfAck received.
Trsp
cNack:= inc(cNack)
export(cNack)
ack_ fail


Process Tx_Coordination_AP
ap_retry_4c(7)
cts_ fail
ack_ fail
mFxIP:=false
mFxIP:=false
export (mFxIP)
export (mFxIP)
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
ccw:= if ccw = aCWmax
then aCWmax else (2*ccw)+1 fi
Backoff (ccw, -1)
fsdu!pdus (fsdu!fCur):= setRetry (fsdu!pdus (fsdu!fCur),1)
tpdu:= setRetry (tpdu,1),
Backoff (ccw, -1)
((length (tpdu) +
sCrcLng) > import(aRts_ Threshold))
(true)
(false)
fsdu!lrc:= fsdu!lrc+1
slrc:= slrc+1,
ssrc:= ssrc+1,
fsdu!src:= fsdu!src+1
import(aLong_ RetryLimit)
slrc =
(true)
(false)
ssrc =
(false)
(true)
import(aShort_ RetryLimit)
ccw:= aCwMin
ccw:= aCwMin
import(aLong_ RetryLimit)
fsdu!lrc =
(false)
(true)
fsdu!src =
(false)
(true)
import(aShort_ RetryLimit)
PduConfirm (fsdu, retryLimit)
cFail:= inc(cFail), cont:= false
cont:= true
This shows the case where the same pdu is retried after the backoff. It is also allowable to return this fsdu to PM_Filter with status=partial, and to go to TxC_Backoff state with cont=false. This will allow a different pdu (if available) to be sent as the next transmission.
export(cFail)
TxC_Backoff


Process Tx_Coordination_AP
ap_dwell_5c(7)
TxC_Backoff
Done
(bstat)
*
cont
(false)
(true)
TxC_Idle
cont:= false
send_
frag
*
SwChnl
(chan,
doBkoff)
ChangeNav
(0,cswitch)
'channel
change is
Phy-specific'
'PlmeSet._
request
(chan stuff)'
Wait_
_Channel
'PlmeSet.
confirm
(status stuff)'
*
SwDone
doBkoff
(false)
(true)
Backoff
(ccw,-1)
SwChnl_
_Backoff
Done
(bstat)
*
TxC_Idle


Process Tx_Coordination_AP
ap_pcf_6a(7)
TxC_Cfp
Transitions on this page are only present for point coordinator.
Attach CfPoll and/or generate CfPoll without data based on polling list if mCfPoll=true.
Pdu_ Request (fsdu)
not import (mCfp)
TxCfAck (endRx)
*
pack:= ftype(tpdu)
tpdu mkFrame (cfend,
import(mBssId), import(mBssId),)
rtype:= cfAck
tx_ sifs
tpdu:= fdsu!pdus (fsdu!fCur)
tpdu:= mkFrame(rtype,
import(mBssId), import(mBssId),)
fsdu!eol
else
(=0)
tx_ sifs
fsdu!sqf:= seqnum,
seqnum:= seqnum+1, fsdu!eol:= now + import (aMaxTransmit_ MsduLifetime)
Wait_Cfp_ Sifs
tpdu:= setSeq(tpdu, fsdu!sqf)
Trsp
*
TxCfAck (,)
Change data to data+cfAck if appropriate.
tpdu:= setFtype (tpdu,
ftype(tpdu) or pack)
TxRequest (tpdu, txrate)
tpdu:= setFtype (tpdu,data_ack)
See 9.6 for rules about selecting transmit data rate.
'txrate:= selected tx data rate'
cTfrg:= inc(cTfrg),
cTmcfrg:= if fsdu!grpa then inc(cTmcfrg) else cTmcfrg fi
-
Wait_Cfp_ Sifs
export (cTfrg, cTmcfrg)
Wait_Cfp_ TxDone
TxConfirm
*
set(now+ aSifsTime, Trsp)
Wait_CfAck


Process Tx_Coordination_AP
ap_cf_retry_7a(7)
Wait_CfAck
Ack
(endRx,)
Trsp
*
TxC_Cfp
cNack:=
inc(cNack),
export
(cNack)
fsdu!pdus
(fsdu!fCur):=
setRetry
(fsdu!pdus
(fsdu!fCur),1)
tpdu:=
setRetry
(tpdu,1),
fsdu!src:=
fsdu!src+1
fsdu!src =
import(aLong_
RetryLimit))
(true)
(false)
PduConfirm
(fsdu,
retryLimit)
PduConfirm
(fsdu,
partial)
This returns the fsdu
to the queue. At the
next cf-poll, either
the same fsdu or a
different fsdu may
be selected for
transmission.
cFail:=
inc(cFail)
cRtry:=
inc(cRtry)
export(cFail)
export(cRtry)
set(now+
aSifsTime,
Trsp)
TxC_Wait_
Pifs
Trsp
*
TxC_Cfp

Annex D

(normative)

ASN.1 encoding of the MAC and PHY MIB

Publisher's note:

It has come to our attention that the definition of the Management Information Base (MIB) in the approved draft standard contains inconsistencies between the definitions in Clause 11, Clause 13, and Annex D. Because the definitions in Annex D are not correct, Annex D is not being published in this first edition.

The Working Group is planning to submit a PAR for the revision of the standard to make Annex D consistent with Clauses 11 and 13. They are also writing an interpretation of the Annex D material, which will be made available in December 1997 at no cost to all purchasers of the published standard. This information will also be posted on our web site at **standards.ieee.org/reading/index.html**.

Annex E

(informative)

Bibliography

E.1 General

[B1] ANSI Z136.1-1993, American National Standard for the Safe Use of Lasers.

[B2] IEC 60825-1 (1993), Safety of laser products—Part 1: Equipment classification, requirements and user's guide.

[B3] IEEE Std 802.10-1992, IEEE Standards for Local and Metropolitan Area Networks: Interoperable LAN/MAN Security (SILS) (ANSI).

[B4] Schneier, Bruce, "Applied Cryptography, Protocols, Algorithms and Source Code in C," New York: Wiley: 1994.

E.2 Specification and description language (SDL) documentation

[B5] Belina, Ferenc, Dieter Hogrefe, and Amardeo Sarma*, SDL with Applications from Protocol Specification*. Prentice Hall Europe, Hertfordshire, UK, 1991.

An introductory text on SDL, also useful as a language reference (for SDL-88).

[B6] Ellsberger, Jan, Dieter Hogrefe, and Amardeo Sarma*, SDL, Formal Object-Oriented Language for Communicating Systems*; (Prentice Hall Europe, Hertfordshire, UK, 1997.

A recently published book, which appears to be the most comprehensive single-volume introduction and reference for SDL-92, including its object-oriented extensions.

[B7] Faergemand, Ove and Anders Olsen, *New Features in SDL-92*; SDL Newsletter (ISSN 1023-7151), no. 16 (May, 1993), pp. 10–29. Also available online at http://www.tdr.dk/public/SDL/SDL.html.

This provides a summary of the changes from SDL-88 to SDL-92.

[B8] Olsen, Anders, Ove Faergemand, Birger Moller-Pedersen, Rick Reed, and T. R. W. Smith, *Systems Engineering Using SDL-92*. Elsevier Science B.V., Amsterdam, the Netherlands, 1994.

A detailed guide to using SDL-92, including a thorough explanation of abstract data type mechanism and SDL combined with ASN.1 (Z.105).