

# Editing checkoff list

This paper provides a checklist of editing tasks. Completion of these tasks by IEEE editors can improve Sponsor ballot review efficiencies while reducing pre- and post-production delays. Supportive material is attached and cross-referenced, as needed to clarify sometimes-subtle item specifications.

## Acronyms (see 4.1.3):

- Capitalization: Only proper nouns are capitalized in the acronym listing.
- Clarity: Uncommon acronyms are spelled out on first use within each clause.

## Cross-references (see 4.6):

- Concise: Subclause cross-references have the form “4.3.3”, not “subclause 4.3.3”.
- Linked: Cross-references are hyperlinked to their referenced text, for navigational convenience.
- Marked: Cross-references are marked (not manually typed), for robustness after updates. (These may be found by searching for “[0-9].[0-9]”, “Clause [0-9]”, and “Annex [0-9]” text.)
- Sections: Number portions of clauses are called subclauses, not sections.

## Figures:

- Callouts: ALLCAPS and lower-case callouts are not intermixed.
- Fonts: The font within figures is 8-point Arial, unless a smaller (6-point minimum) font is required.

## General:

- Automatic: Each cross-reference hyperlinks to the referenced location.
- Capitalization: Only proper nouns and first word of titles/headings are capitalized (see 4.1).
- Nonbreaking: Field names and table/figure/equation titles do not break across lines.
- Numbers: Drafts have approximate line numbers, to simplify comment submittals (see right, 1-54).
- Setup: Update the draft number and date; delete any extraneous blank pages.

## Naming (special fields and registers):

- Explicit: No implicit definitions (e.g., paragraphs starting with “This value...”).
- Hyphens: Hyphens are not included (no “reset-start” registers, see 4.3.1).
- Nonbreaking: Special names do not break across lines.
- Spaces: Spaces are not included within the name (no “reset start” registers, see 4.3.1).
- Uniform: Names have no abbreviation or alias (see 4.3.3), except for Figure/legend (see 3.4.2).

## References:

- Normative: Every reference is associated with a may/should/shall statement.
- Present: Every reference is cross-referenced elsewhere in the document.

## Spelling (see 4.4):

- Checked: A spelling checker has been run, with all identified errors corrected.
- Ellipsis: The special ‘...’ character has been used, rather than three periods.
- Spacing: Single spaces (not double spaces) are used between words and sentences.

## Styles (see 4.5):

- Baseline: Editor-specific styles have been removed from the document.
- Overrides: Paragraph and character style overrides (e.g, bold and italics) are not present.
- Reassert: Baseline styles have been imported, to eliminate unintended changes and overrides.

## Tables:

- Continued: Cross-page tables have “continued” in their following title lines.
- Lined: Cross-page tables have a very thin (not blank) line underneath the before-break row.
- Lining: Tables have very thin lines between rows/column, thin lines around header and body.

## Terminology (terms, definitions, and notation):

- Acronyms: Acronyms within definitions are included in the acronyms clause (see 4.1.3).
- Conformance: The terms “shall”, “should”, and “may” are defined (see 3.1).
- Defined. Nonsubscripted numbers within code should be valid (0b10011 is an invalid C constant).
- Repeats: The defined term is not reused within its definition (see 4.2.2).
- Subscripts: Subscripts for non-decimal numbers: 19 is  $13_{16}$  is  $10011_2$  (see 3.3).

### 3. Terms, definitions, and notation

NOTE—Clause 3 contains examples of specifications that may be included in an IEEE Standard. The clause numbers '3' has been manually set to the clause number expected in your draft.

#### 3.1 Conformance levels

Several key words are used to differentiate between different levels of requirements and options, as follows:

**3.1.1 expected:** Describes the behavior of the hardware or software in the design models assumed by this standard; other hardware and software design models may also be implemented.

NOTE—The preceding “expected” conformance statement has been found to be useful in some standards. The following “may, shall, should” conformance definitions should be provided by IEEE standards; if provided, their definitions shall be as follows.

**3.1.2 may:** Indicates a course of action permissible within the limits of the standard with no implied preference (“may” means “is permitted to”).

**3.1.3 shall:** Indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (“shall” means “is required to”).

**3.1.4 should:** An indication that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (“should” means “is recommended to”).

#### 3.2 Terms and definitions

NOTE—The document’s terms and definitions are included here.

#### 3.3 Numerical values

NOTE—The following notation text, or its equivalent, are required to explain your notation. The subscript definition is preferred within English text, although other notations are used within code.

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2, ... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number “26” may also be represented as “1A<sub>16</sub>” or “11010<sub>2</sub>”.

These notational conventions have one exception: MAC addresses and OUI/EUI values are represented as strings of 8-bit hexadecimal numbers separated by hyphens and without a subscript. Examples of MAC and OUI/EUI notation include 01-80-C2-00-00-15 and AA-55-11, respectively.

## 3.4 Field notation

### 3.4.1 Field names

NOTE—All documents should describe their naming conventions, with text similar to the following.

This document describes values that are located in frame and register locations. For clarity, distinct capitalization conventions are used when naming different components, as illustrated in Table 3.1.

**Table 3.1—Names of command, status, and CSR values**

Name	Row	Description
MAX_VALUE	1	A defined constant value.
StateMachineName	2	A formal state machine name
parameter_value	3	A Service Primitive parameter
<i>RoutineName()</i>	4	A subroutine name, when referenced within text
<i>runCommand</i>	5	A referenced control register.
<i>startCode</i>	6	The <i>startCode</i> field.
<i>start</i>	7	The <i>start</i> bit.
<i>runCommand.startCode</i>	8	The <i>startCode</i> field within the <i>runCommand</i> register.
<i>runCommand.start</i>	9	The <i>start</i> bit within the <i>runCommand</i> register.

NOTE—If ever applied to a variable name, the italics style should always applied to that variable names, whether contained in tables, figures, or headings (but not C-code, where plain Courier is always used).

NOTE—When lengthy descriptions are necessary, row numbers and following clarifications should supplement the descriptive column, as illustrated within Table 3.1.

**Table 3.1-1:** Constant values are spelled with capital letters; an underscore separates run-together words.

**Table 3.1-2:** Formal state machine names start with a capital letter and contain no blank spaces.

**Table 3.1-3:** Service primitive parameters include underscores, for consistency with the past and to differentiate these parameters from defined field values.

**Table 3.1-4:** Subroutine names (within normal text) are distinguished by italics and first-capital-letter. The italics formatting convention applies to the name, not associated special symbols, such as ‘(’ and ‘)’.

**Table 3.1-5, Table 3.1-6, Table 3.1-7:** Register names, fields, and bit names start with a lower-case letter. Each run-together word starts with a capital letter. Run-together names like *runCommand* are preferred because they are more compact than underscore-separated names (like *run\_command*).

**Table 3.1-8, Table 3.1-9:** When the register location is unclear or ambiguous, the name of a field includes the name of the register where that field is located.

### 3.4.2 Fields within figures

NOTE—Numbering of bits within registers causes problems, due to distinct bit numbering conventions adopted by little-endian and big-endian designers. Bit ordering arguments are not easily resolved by using the bit-transmission order, which can be PHY dependent or ambiguous. Arguments and confusions are best resolved by avoiding the numbering altogether, which also simplifies each illustration.

The location of fields within registers is specified by the cumulative widths of fields within the register, as illustrated in Figure 3.1. The width of each field (in bits) is implied by bottom-line tick marks; the field name is normally contained within its bounding rectangle.

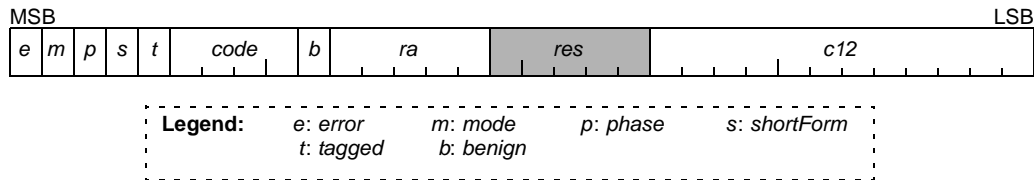


Figure 3.1—Expanded bit-field descriptions

NOTE—Abbreviations within figures are the only time that special names are spelled differently. Within such figures, the association with the abbreviated and standard name must be specified in the legend.

If this were a defined *control* register, for example, terms like the *control.error* bit or the *control.code* field are used to describe its values.

### 3.5 Field notations

#### 3.5.1 Use of italics

All field names or variable names (such as *level* or *myMacAddress*), and sub-fields within variables (such as *thisState.level*) are italicized within text, figures and tables, to avoid confusion between such names and similarly spelled words without special meanings. A variable or field name that is used in a subclause heading or a figure or table caption is also italicized. Variable or field names are not italicized within C code, however, since their special meaning is implied by their context. Names used as nouns (e.g., subclassA0) are also not italicized.

#### 3.5.2 Field conventions

This document describes values that are packetized or MAC-resident, such as those illustrated in Table 3.2.

Table 3.2—Names of fields and sub-fields

Name	Description
<i>newCRC</i>	Field within a register or frame
<i>thisState.level</i>	Sub-field within field <i>thisState</i>
<i>thatState.rateC[n].c</i>	Sub-field within array element <i>rateC[n]</i>

Run-together names (e.g., *thisState*) are used for fields because of their compactness when compared to equivalent underscore-separated names (e.g., *this\_state*). The use of multiword names with spaces (e.g., “This State”) is avoided, to avoid confusion between commonly used capitalized key words and the capitalized word used at the start of each sentence.

A sub-field of a field is referenced by suffixing the field name with the sub-field name, separated by a period. For example, *thisState.level* refers to the sub-field *level* of the field *thisState*. This notation can be continued in order to represent sub-fields of sub-fields (e.g., *thisState.level.next* is interpreted to mean the sub-field *next* of the sub-field *level* of the field *thisState*).

Unless specifically specified otherwise, reserved fields are reserved for the purpose of allowing extended features to be defined in future revisions of this standard. For devices conforming to this version of this standard, nonzero reserved fields are not generated; values within reserved fields (whether zero or nonzero) are to be ignored.

### 3.5.3 Field value conventions

This document describes values of fields. For clarity, names can be associated with each of these defined values, as illustrated in Table 3.3. A symbolic name, consisting of upper case letters with underscore separators, allows other portions of this document to reference the value by its symbolic name, rather than a numerical value.

**Table 3.3—wrap field values**

Value	Name	Description
0	WRAP_AVOID	Frame is discarded at the wrap point
1	WRAP_ALLOW	Frame passes through wrap points
2,3	—	Reserved

Unless otherwise specified, **reserved** values are reserved for the purpose of allowing extended features to be defined in future revisions of this standard. Devices conforming to this version of this standard do not generate reserved values for fields, and process fields containing reserved values as though the field values were not supported. The intent is to ensure default behaviors for future-specified features.

A field value of TRUE shall always be interpreted as being equivalent to a numeric value of 1 (one), unless otherwise indicated. A field value of FALSE shall always be interpreted as being equivalent to a numeric value of 0 (zero), unless otherwise indicated.

## 4. General writing guidelines

### 4.1 Capitalization

Engineers seem to Capitalize Anything that could possibly be important, including Acronyms, Application-Specific Terms, and Variable names. The intent is to distinguish between normal English and words with specialized meaning. The effect is that excessive (oftentimes inconsistent) use of capitalization complicates parsing of the sentence, leading to loss of clarity and unnecessary technical ambiguities.

#### 4.1.1 When to capitalize

In general, capitalization is only necessary for proper nouns and the first word of headings (see IEEE Style Manual, 13.8 Capitalization). Other uses are strongly discouraged. Alternative techniques to can delineate words with standard-specific meanings.

- a) Variables can be distinguished from English words by any of the following naming conventions:
  - 1) A *command register* or *startTest* field can be distinguished from English words by using an italics style, with a capital at the start of each following run-together word.
  - 2) An *ExecutionRoutine()* or group of *CommonControl* registers can be distinguished by capitalizing the first run-together word.
  - 3) A *source\_address* is distinguished by the underscore that separates run together words. Most importantly, a variable or field name **should not** be split into separate words separated by spaces and tied together by the common use of Capital Letters ☹.
- b) Constants can be easily distinguished from normal English words by using all capitals within the name. Such conventions apply to a RESET constant or an enumerated START\_FAST value.
- c) Names with special meaning, such as ringlet or port, should be defined in an early “Terms and definitions” clause, and therefore need not be capitalized when used within the text.

Capitalized words have previously been used in an attempt to delineate the boundaries of a special variable name, such as a Negotiation Control Register. However, the following questions must be addressed for the sentence to be correctly parsed, or the document correctly searched for alternate descriptions:

- a) Does this refer to a specific *negotiationControl* instance of a register?
- b) Does this refer to a specific *negotiationControlRegister* location?
- c) Does this refer to generic negotiation-control registers?
- d) Does this refer to a negotiation register called *control*?
- e) Does this refer to a control register called *negotiation*?

While these distinctions may be moot to the editor or familiar readers, they severely restrict document comprehension by unfamiliar engineers and/or non-native English speakers. *Do not do this!*

**4.1.2 Heading capitalization**

The first word of a clause, subclause, table, table-heading, figure, or figure insert is normally capitalized, such as:

**4.1.2.1 Shared control and status registers (CSRs)**

An exception is when the heading starts with a special variable name, such as:

**4.1.2.2 *command register***

Only proper nouns, such as California, are capitalized within the other words of a clause, subclause, table, table-heading, figure, or figure insert. Adjacent words are not necessarily capitalized, as in:

**4.1.2.3 Habits of California wildlife****4.1.3 Noncapitalized definitions**

Terms within a glossary or definitions of terms are not capitalized, unless these are proper nouns. As examples:

**3.2.29 bridge:** A functional unit interconnecting two or more networks on ...

**3.2.30 congestion domain:** The set of contiguous links associated ...

**3.2.31 cyclic redundancy check (CRC):** A computed value...

**3.2.32 loop round trip time (LRTT):** The time that it takes for a control frame...

**3.2.33 medium access control (MAC) sublayer:** The portion of the data link layer...

Acronyms introduced within these definitions (i.e., LRTT and MAC) should also be listed in the acronyms clause.

**4.1.4 Noncapitalized acronyms**

Words within acronyms are not capitalized, unless these are proper nouns. As examples:

CRC	cyclic redundancy check
FEC	forward error correction
LRTT	loop round trip time
LSB	least significant bit
MAC	medium access control (sublayer)
MSB	most significant bit
SF	signal fail
SDH	synchronous digital hierarchy
USA	United States of America
WAN	wide area network

**NOTE**—Within other clauses, uncommon acronyms should be spelled out on first usage:

Correct: the cyclic redundancy check (CRC) is computed...

Incorrect: the CRC (cyclic redundancy check) is computed...

## 4.1.5 Improper nouns

The term improper noun describes a noun that is not a proper noun, but is (typically for historical or confusion reasons) capitalized like a proper noun. To reduce improper noun usage, consider the following when classifying proper nouns.

- a) Is this a particular person, place, or thing? The *United States of America* and *California* are particular places or names; the number of *counties* in California is not.
- b) Does this avoid confusion that cannot be solved by the appropriate standard-specific definition?
- c) Is this an acronym? By itself, this is insufficient justification for capitalization (see 4.1.4). The capitalization of a *cyclic redundancy check* is unaffected by its CRC acronym abbreviation.
- d) Is this a title? Only the first word in a title is normally capitalized, so this is insufficient (see 4.1.3).
- e) Is this a definition? A defined term is not necessarily a proper noun (see 4.1.2).

Note that many standards have generated improper nouns due to improper interpretations of (c,d,e) usage. You may sometimes have to cope with such conventions, but don't propagate them.

## 4.2 Definitions

### 4.2.1 Definition clauses

From the style manual: *Acronyms shall not be defined in the definitions clause.* Instead, a following clause should be used. However, standards commonly place related content within a definitions clause, expanding the header text as follows:

#### 3. Terms, definitions, and notation

### 4.2.2 Definition text

The terms should not be used in its own definition.

*Correct usage* is as follows:

**4.1.2 doublet:** Two bytes.

*Incorrect usage* is as follows:

**4.1.2 doublet:** A doublet is two bytes of data.

## 4.3 Variable names

### 4.3.1 Spaceless variable names

Using run-together spaceless variable names has the advantage that descriptive code can be easily used to specify formal definitions, as illustrated by Equation 4.1.

```
errorCount += (protocolErrors + incomingCount.crcBad + clientCount.typeBad);      (4.1)
```

This is clear and concise, when compared to the pseudo-defined pseudo code of Equation 4.2. The lack of extra spaces also reduces the equation lengths, so more of the equations will be able to fit on one line. The use of C-type conventions, to name subfields by their function (as opposed to position) adds to clarity, while simplifying text maintenance by defining bit-field positions in only one place.

```
(Error Count) ← ((Protocol Errors) + (Incoming Count)[15:14] + (Client Count)[12:11])      (4.2)
```

### 4.3.2 Hyphenless variable names

Using run-together hyphenless variable names also has the advantage that descriptive code can be easily used to specify formal definitions, as illustrated by Equation 4.3.

```
errorCount += (protocolErrors + incomingCount.crcBad + clientCount.typeBad); (4.3)
```

For example, a hyphenated name could have the subtraction-operator ambiguity associated with Equation 4.4.

```
errorCount += (protocol-errors + incomingCount.crcBad + clientCount.typeBad); (4.4)
```

### 4.3.3 Aliased variable names

Never reference the same variable using two separate names, except within the context of a figure legend (see 3.4.2). As an example, the interchangeable use of “DA” and “Destination Address” terms raises the following concerns:

- a) The term DA could refer to either “destination address” (a generic term) or “Destination Address” (a specific field within frames).
- b) If the “Destination Address” is placed in the transmitted frame, and the “DA” is checked in the received frame, the reader will be confused ☹.
- c) An acronym-based abbreviation could easily be confused with a constant, such as PASS or FAIL.

## 4.4 Spelling checker

Always run a spelling checker on your text **before** submitting to Sponsor Ballot. To simplify spelling-checker and/or character-pattern searches, the following guidelines have been found to be useful:

- a) Ellipsis. The special ellipsis character ‘...’ should be used, not three consecutive periods. This eliminates spurious duplicate-period spell-checker reports.
- b) Spaces. Only one space should be used between words and sentences. This allows a search to readily find duplicate-space errors.
- c) Styles. Special character styles are appropriate for special names, variable names, and field names. For example, styles used within this paper have the following properties:
  - 1) The NamePlain style is applied to special names (e.g., StateMachineName) with the effect of inhibiting end-of-line hyphenation and spurious spelling-checker reports.
  - 2) The NameItalic style is applied to field and variable names (e.g., *thisValue.thatField*) to force an italic style, while inhibiting end-of-line hyphenation and spurious spelling-checker reports.

## 4.5 Paragraph and character styles

Consistent printing relies on the definitions of consistent paragraph and character styles. The following practices are therefore encouraged:

- a) Re-import. The baseline styles should be re-imported to avoid unintended changes.
- b) Retain. Editors should be discouraged from application of style overrides, often lost in (a).
- c) Purge. Editors should purge extraneous style definitions that are inherited from contributions. Each extended definitions can conflict with others, despite multiple re-import (a) attempts.

## 4.6 Cross references

Cross references should be clear and concise, with the following guidelines:

- a) The term subclause, not section, is used to describe distinctively numbered portions of a clause.
- b) A proper cross reference, such as “(see 4.4)”, does not include the word “subclause” or “section”.
- c) The simple “(see 4.4)” text is usually preferred to using multiple or complex alternatives.
  - “, as considered in 4.4.”
  - “, as constrained by 4.4.”
  - “, as described in 4.4.”
  - “, as discussed in 4.4.”
  - “, as illustrated in 4.4.”
  - “, as shown in 4.4.”
  - “, as specified in 4.4.”
  - “, as standardized in 4.4.”
  - “, as mandated by 4.4.”
- d) Never type the numbers within “(see 4.4)”. Explicit cross-reference markers are preferred, because:
  - 1) Numbers are automatically updated as subclauses are inserted, deleted, or removed.
  - 2) Word and FrameMaker can generate valuable hyperlinks from marked values.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54