Micrium Coding Standards

Application Note AN-2000 Rev. D

Contents

Articles

Coding Standards Introduction 1
Coding Statements 2
Comments 21
Constants 29
Data Types 35
Functions 40
Naming Conventions 66
Operators 80
Portability 93
Source Code 98
Source Files 101
Variables 116

Coding Standards Introduction

Conventions should be established early in a project. These conventions are necessary to maintain consistency throughout the project. Adopting conventions increases productivity and simplifies project maintenance.

There are many ways to code programs in C (or any other language). The style you use is just as good as any other as long as you strive to attain the following goals:

- Portability
- Consistency
- Neatness
- Easy maintenance
- Easy understanding
- Simplicity

The chosen style should be used consistently throughout all your projects. Furthermore, a single style should be adopted by all team members in a large project. Adopting a common coding style reduces code maintenance headaches and costs. Adopting a common style helps avoid code rewrites.

The conventions for Micrium software are outlined in this document. These standards should be followed for all new software modules, ports, BSPs and applications.

Basic Principles

The fundamental purpose of these standards is to promote maintainability of the code. This means that the code must be readable, understandable, testable and portable. To achieve this, a few principles should be followed:

- **Keep the spirit of the standards.** Where you have a coding decision to make and there is no direct standard, then you should always keep within the spirit of the standard.
- **Comply with ANSI C standards.** For portability, code should comply with an accepted release of the C standard. ISO/IEC 9899:TC2. Furthermore, the safety-aware spirit of the Motor Industry Software Reliability Association's Guidelines for the Use of the C Language in Vehicle-Based Software (MISRA rules) should guide code development. MISRA rules will be cited throughout this document when practices are recommended.
- **Keep the code simple.**
- **Be explicit.** Avoid implicit or obscure features of the language. Say what you mean.
- **Be consistent.** Use the same rules as much as possible.
- **Avoid complicated statements.** Statements comprising many decision points are hard to follow and test.
- **Update old code.** Whenever existing code is modified, try to update the document to abide with the conventions outlined in this document. This will ensure that old code will be upgraded over time.

Coding Statements

Simple Code Statements

In general, there should only be one code action/statement per line of code. Use of the comma operator is also highly discouraged (MISRA 2004 Rule 12.10) [see also Comma Operator].

```
DispSegTblIx =    0;
DispDigMsk   = 0x80;
```

rather than

```
DispSegTblIx = 0; DispDigMsk = 0x80;
```

```
event_done = AppEventIsDone(APP_INIT);
if (event_done == DEF_YES) {
    LED_Out(APP_INIT_LED, DEF_ON);
}
```

rather than

```
if (AppEventIsDone(APP_INIT) == DEF_YES) {
    LED_Out(APP_INIT_LED, DEF_ON);
}
```

```
 pval++;
*pval = 0;
```

rather than

```
*++pval = 0;
```

However, industry-standard usage may be acceptable:

```
*pval++ = 0;
```

rather than

```
*pval = 0;
 pval++;
```

`void` Statements

Any statement intended to ignore any effects and produce no actual code should be explicitly **`void`**'ed as shown in Listing 1. This may include **`void`**'ing function arguments or local variables that may not be used, in order to suppress compiler warnings that these arguments or variables are unused. **`void`** statements are preferred over self-assignments as shown in Listing 1 since **`void`** statements can be used to ignore and suppress warnings for both function arguments and local variables whereas self-assignments can only be used for function arguments but not local variables.

The return values of functions may also be **`void`**'ed in order to inform both the compiler and code reviewer that the return value was intentionally ignored/unused as shown in Listing 1. If a return value is ignored, an appropriate comment/note should explain why the return value can be ignored.

Listing 1 — `void` Statement Formats

```
void  SomeTask (*p_arg)
{
    CPU_ADDR  addr;

   (void)&p_arg;
   (void)&addr;
    ...
}
```

void's left-most character 'v' should be aligned to a column of a multiple of 4 **&** the address operator should immediately follow **(void)**, and immediately prefix variable.

Listing 2 — Self-Assignments Deprecated/Discouraged

```
void  SomeTask (*p_arg)
{
    CPU_ADDR  addr;

    p_arg = p_arg;
    addr  = addr;
    ...
}
```

The statement `p_arg = p_arg;` prevents the 'variable unused' compiler warning for function arguments, but may generate a 'variable used before initialized' warning for local variables.

Listing 3 — `void` Function Return Values

```
(void)OSSemPost(p_signal);16
```

In the above example, we ignore **OSSemPost()**'s return error value. **(void)** should immediately immediately prefix the function call name. **void**'s left-most character 'v' should be aligned to a column of 4.

Listing 4 — `void` Statement Examples

```
static  void  NetBuf_Discard (NET_IF_NBR      if_nbr,
                              void           *pbuf,
                              NET_STAT_POOL  *pstat_pool)
{
   (void)&if_nbr;                                                          (1)
   (void)&pbuf;

    ...

    NET_CTR_ERR_INC(Net_ErrCtrs.NetIF_ErrCtrs[if_nbr].NetIF_ErrBufLostCtr);    (2)
}

    while ((psock_event->EventType != NET_SOCK_EVENT_NONE) &&
            (sock_events_nbr_rem > 0)) {

        sock_id = (NET_SOCK_ID)psock_event->SockID;
        conn_id = (NET_CONN_ID)psock_event->ConnID;

        psignal = (OS_EVENT  *)0;
        switch (psock_event->EventType) {
            case NET_SOCK_EVENT_SOCK_RX:
            case NET_SOCK_EVENT_SOCK_ERR_RX:
                 psignal = NetOS_Sock_RxQ_SignalPtr[sock_id];
                 break;

                 ...

            case NET_SOCK_EVENT_TRANSPORT_RX:
            case NET_SOCK_EVENT_TRANSPORT_ERR_RX:
#if    (NET_CONN_CFG_FAMILY == NET_CONN_FAMILY_IP_V4_SOCK)
#ifdef  NET_TCP_MODULE_PRESENT
                 psignal = NetOS_TCP_RxQ_SignalPtr[conn_id];
#endif
#endif
```

```
                break;

            case NET_SOCK_EVENT_TRANSPORT_TX:
            case NET_SOCK_EVENT_TRANSPORT_ERR_TX:
#if    (NET_CONN_CFG_FAMILY == NET_CONN_FAMILY_IP_V4_SOCK)
#ifdef  NET_TCP_MODULE_PRESENT
                psignal = NetOS_TCP_TxQ_SignalPtr[conn_id];
#endif
#endif
                break;

            case NET_SOCK_EVENT_NONE:
            case NET_SOCK_EVENT_SOCK_TX:
            case NET_SOCK_EVENT_SOCK_ERR_TX:
            default:
                (void)&conn_id;                                          (3)
                *perr =  NET_SOCK_ERR_INVALID_EVENT;
                 return (0u);
        }
```

(1) Prevent possible 'variable unused' warning ...

(2) ...since **if_nbr** is used dynamically. The buffer is discarded by ignoring and not returning to any buffer pool.

(3) **conn_id void**'d since dynamic declaration logic would be overly complex.

`if` and `if...else` Statements

`if` and **`if...else`** statements should be formatted as shown in Listing 5. The body must be enclosed in braces (MISRA 2004 Rule 14.8).

Listing 5 — `if` and `if...else` Statement Formats

```
if (addr_host == NET_IP_ADDR_THIS_HOST) {                                (1)
    valid = DEF_NO;

} else if (addr_host == NET_IP_ADDR_BROADCAST) {
    valid = DEF_NO;                                                      (2)

} else if ((addr_host & NET_IP_ADDR_LOCAL_HOST_MASK_NET) ==              (3)
                        NET_IP_ADDR_LOCAL_HOST_NET     ) {
    valid = DEF_NO;

} else if ((addr_host & NET_IP_ADDR_LOCAL_LINK_MASK_NET) ==
                        NET_IP_ADDR_LOCAL_LINK_NET     ) {
    if ((addr_host < NET_IP_ADDR_LOCAL_LINK_HOST_MIN) ||                (4)
        (addr_host > NET_IP_ADDR_LOCAL_LINK_HOST_MAX)) {
         valid = DEF_NO;                                                 (5)
    }

    ...

} else if ((addr_host & NET_IP_ADDR_CLASS_D_MASK) ==                     (6)
                        NET_IP_ADDR_CLASS_D) {
    valid = DEF_NO;

} else {
    valid = DEF_NO;
}
```

(1) Use K&R-style braces: one space after **if** before opening parenthesis and one space after closing parenthesis before opening brace.

(2) Body indented by four spaces.

(3) **else if** on same line as closing brace of previous **if** conditional and opening brace of **else if** conditional; one space before and after **else if**; closing brace on same column as initial **if** conditional.

(4) Comparison operators on multiple lines aligned vertically

(5) Body indented extra space to align with conditionals

(6) **else** on same line as closing brace of previous **if** conditional and opening brace of **else** conditional; one space before and after **else**; closing brace on same column as initial **if** conditional.

`if` Conditionals

`if` statements should use either more restrictive or less restrictive conditionals appropriately to execute **`if`** statement code less frequently or more frequently, respectively. For example, code which should execute less frequently or optionally should use more restrictive conditions as shown in Listing 6. Whereas, code which should execute more frequently or by default should use less restrictive conditionals as shown in Listing 7.

Listing 6 — More Restrictive `if` Conditionals

```
if (opt == DEF_ENABLED) {                                                    (1)
    handle opt;
}

reg_val = RegRd();
if ((reg_val & reg_mask) == reg_val) {                                       (2)
    handle specific reg mask action;
}
```

(1) **opt** handled IFF **DEF_ENABLED**.

(2) **reg** action handled IFF exactly equal to **mask**.

Listing 7 — Less Restrictive `if` Conditionals

```
valid = IsValid(obj);
if (valid != DEF_YES) {                                                      (1)
    Rtn err;
}

block = Sock's blocking option;
if (block != DEF_NO) {                                                       (2)
    Wait for sock operation;
}
```

(1) Return error unless **valid** is **DEF_YES**

(2) Sockets **block** on operations by default

Comparisons

if statement conditionals may be composed of multiple logical or numerical comparisons, but each comparison should be as simple as possible as shown in Listing 8. However when possible, multiple comparisons should be combined into a single logical comparison prior to the **`if`** conditional test as shown in Listing 9. This makes testing/debugging the comparison logic much easier. Lastly, functions or macros should **NOT** be called from within an **`if`** statement conditional as shown in Listing 10 (see also Simple Code Statements).

Listing 8 — Simplest `if` Conditionals Preferred

```
tx_seg_push = DEF_BIT_IS_SET(pseg_hdr->TCP_Flags, NET_TCP_FLAG_TX_PUSH);               (1)
tx_nagle    = ((pconn->TxWinSizeNagleEn == DEF_DISABLED         ) ||          (2)
               (pconn->TxSeqNbrNext     == pconn->TxSeqNbrUnAckd)) ? DEF_YES : DEF_NO;
tx_th_nagle = ((tx_nagle    == DEF_YES) &&
               (tx_seg_push == DEF_YES) &&
               (tx_data_min <= pconn->TxWinSizeAvail)) ? DEF_OK : DEF_FAIL;

if (tx_th_nagle != DEF_OK) {                                                          (3)
    ...
```

(1) **DEF_BIT_IS_SET()** macro called outside **if** statement.

(2) Multiple comparisons combined prior to **if** statement.

(3) Single logical comparison preferred, when possible.

Listing 9 — Simple `if` Conditionals

```
tx_seg_push = DEF_BIT_IS_SET(pseg_hdr->TCP_Flags, NET_TCP_FLAG_TX_PUSH);               (1)
tx_nagle    = ((pconn->TxWinSizeNagleEn == DEF_DISABLED         ) ||          (2)
               (pconn->TxSeqNbrNext     == pconn->TxSeqNbrUnAckd)) ? DEF_YES : DEF_NO;

if (!((tx_nagle    == DEF_YES) &&                                                     (3)
      (tx_seg_push == DEF_YES) &&
      (tx_data_min <= pconn->TxWinSizeAvail))) {
    ...
```

(1) **DEF_BIT_IS_SET()** macro called outside **if** statement.

(2) Some comparisons combined prior to **if** statement.

(3) Multiple, simple comparisons allowed, when necessary.

Listing 10 — `if` Conditionals with Invalid Macro Call

```
if (!((((pconn->TxWinSizeNagleEn == DEF_DISABLED         )  ||
        (pconn->TxSeqNbrNext     == pconn->TxSeqNbrUnAckd)))                    &&
        (DEF_BIT_IS_SET(pseg_hdr->TCP_Flags, NET_TCP_FLAG_TX_PUSH) == DEF_YES) &&      (1)
        (tx_data_min             <= pconn->TxWinSizeAvail))) {
```

(1) **DEF_BIT_IS_SET()** macro should *not* be called in **if** statement.

Explicit Comparisons

Explicit comparisons should always be made. Listing 11 is preferred since it compares the Boolean variables explicitly to **DEF_YES**, resulting in more obvious (and demonstrably correct) behavior than Listing 12.

Listing 11 — Explicit Comparison Preferred

```
addr_conflict = ((sender_protocol_verifd == DEF_YES) &&
                 (sender_hw_verifd       != DEF_YES)) ? DEF_YES : DEF_NO;
```

Listing 12 — Implicit Comparison Discouraged

```
addr_conflict = sender_protocol_verifd && !sender_hw_verifd;
```

Range Comparisons

Many comparisons of integer values and all comparisons of floating point values should check for thresholds and ranges of values rather than a single value.

`if (x < 1) {`	rather than	`if (x == 0) {`
`if (y >= 5) {`	rather than	`if (y == 5) {`
`if (z <= 0.0) {`	rather than	`if (z == 0.0) {`

Appropriate Context

Comparisons (or assignments) of many values, especially Boolean, to **#define** constants should use the most appropriate context. Historically, many Boolean values are compared (or assigned) to **TRUE** or **FALSE**, even though the choice of true or false is usually NOT the most appropriate context. The examples below show use of appropriate Boolean constants for specific context; but these examples are guidelines for appropriate comparisons/assignments of any types, not just Boolean.

DEF_ENABLED / DEF_DISABLED

Use to configure features. For example:

```
#define  NET_CTR_CFG_ERR_EN   DEF_DISABLED

p_conn->TxWinSizeNagleEn    = DEF_ENABLED;
```

DEF_ON / DEF_OFF

Use to turn hardware on/off. For example:

```
LED_5 = DEF_OFF;

while (DEF_ON) {
```

See also Infinite Task Loops.

DEF_YES / DEF_NO

Use to answer yes/no questions. For example:

```
conn = NetSock_IsConn(sock_id);

done = DEF_NO;
while (done == DEF_NO) {
    ...
    if (some action) {
        done = DEF_YES;
    }
}
```

See also **while** Loop Conditionals.

DEF_OK / DEF_FAIL

Use to indicate a failed or successful status. For example:

```
NetARP_CfgCacheTimeout()
```

Return(s):
`DEF_OK`, ARP cache timeout successfully configured.
`DEF_FAIL`, otherwise.

`#if` and `#if...#else` Statements

Most **`if`** and **`if...else`** Rules also apply to preprocessor **`#if`** / **`#if...#else`** and **`#ifdef`** / **`#ifndef`** / **`#ifdef...#else`** / **`#ifndef...#else`** statements. All **`#if`** preprocessor conditionals should be enclosed in parentheses; but **`#ifdef`** / **`#ifndef`** conditionals should not. Most preprocessor statements should start on column 1 and use additional blank lines as needed to visually separate preprocessor statement bodies as shown in Listing 13. However, nested preprocessor statement bodies **may** be indented as shown in Listing 14 but preprocessor statements/conditionals should **NOT** be interleaved or aligned with source code as shown in Listing 15. Also note that the indentation does **NOT** have to be as consistent as for source code. In other words, preprocessor statements **may** be indented; but as Eric Julien says, "it must not be awful".

Listing 13 — `#if` and `#if...#else` Statement Non-Indented Format

```
#if (NET_SOCK_CFG_TYPE_STREAM_EN == DEF_ENABLED)
        case NET_SOCK_TYPE_STREAM:
#ifdef  NET_SECURE_MODULE_PRESENT
             secure = DEF_BIT_IS_SET(psock->Flags, NET_SOCK_FLAG_SOCK_SECURE);
             if (secure == DEF_YES) {
                 NetSecure_SockCloseNotify(psock, &err);
             }
#endif
             rtn_code = NetSock_CloseHandlerStream(sock_id, psock, perr);
             break;
#endif

#ifdef   OS_GLOBALS
#define  OS_EXT
#else
#define  OS_EXT  extern
#endif
```

All preprocessor statements starts on column 1. **OS_???** symbols on multiple lines aligned vertically.

```
#if     (OS_TICKS_PER_SEC > 0)
                                                                                                                (1)
#define  NET_OS_TIMEOUT_MAX_SEC                         (NET_OS_TIMEOUT_MAX_TICK                             / OS_TICKS_PER_SEC)
#define  NET_OS_TIMEOUT_MAX_mS                         ((NET_OS_TIMEOUT_MAX_TICK *  DEF_TIME_NBR_mS_PER_SEC) / OS_TICKS_PER_SEC)

#if     (OS_TICKS_PER_SEC >  (DEF_TIME_NBR_uS_PER_SEC / (DEF_INT_32U_MAX_VAL     /  NET_OS_TIMEOUT_MAX_TICK)))                    (2)
#define  NET_OS_TIMEOUT_MAX_uS                          (NET_OS_TIMEOUT_MAX_TICK * (DEF_TIME_NBR_uS_PER_SEC  / OS_TICKS_PER_SEC))
#else
#define  NET_OS_TIMEOUT_MAX_uS                           DEF_INT_32U_MAX_VAL
#endif

#else
#define  NET_OS_TIMEOUT_MAX_SEC                          DEF_INT_32U_MIN_VAL
#define  NET_OS_TIMEOUT_MAX_mS                           DEF_INT_32U_MIN_VAL
#define  NET_OS_TIMEOUT_MAX_uS                           DEF_INT_32U_MIN_VAL
#endif
```

(1) Blank lines visually separate nested preprocessor statement bodies

(2) Operators/symbols on multiple lines aligned vertically

Listing 14 — `#if` and `#if...#else` Statement Indented Format

```
    #if (OS_TASK_CREATE_EXT_EN == 1)
    #if (OS_STK_GROWTH == 1)
    os_err = OSTaskCreateExt((void (*)(void *)) NetOS_IF_RxTask,
                             (void          * ) 0,
                             (OS_STK        * )&NetOS_IF_RxTaskStk[NET_OS_CFG_IF_RX_TASK_STK_SIZE - 1],
                             (INT8U           ) NET_OS_CFG_IF_RX_TASK_PRIO,
                             (INT16U          ) NET_OS_CFG_IF_RX_TASK_PRIO,
                             (OS_STK        * )&NetOS_IF_RxTaskStk[0],
                             (INT32U          ) NET_OS_CFG_IF_RX_TASK_STK_SIZE,
                             (void          * ) 0,
                             (INT16U          )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));
    #else
    os_err = OSTaskCreateExt((void (*)(void *)) NetOS_IF_RxTask,
                             (void          * ) 0,
                             (OS_STK        * )&NetOS_IF_RxTaskStk[0],
                             (INT8U           ) NET_OS_CFG_IF_RX_TASK_PRIO,
                             (INT16U          ) NET_OS_CFG_IF_RX_TASK_PRIO,
                             (OS_STK        * )&NetOS_IF_RxTaskStk[NET_OS_CFG_IF_RX_TASK_STK_SIZE - 1],
                             (INT32U          ) NET_OS_CFG_IF_RX_TASK_STK_SIZE,
                             (void          * ) 0,
                             (INT16U          )(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));
    #endif
#else
    #if (OS_STK_GROWTH == 1)
    os_err = OSTaskCreate((void (*)(void *)) NetOS_IF_RxTask,
                          (void          * ) 0,
                          (OS_STK        * )&NetOS_IF_RxTaskStk[NET_OS_CFG_IF_RX_TASK_STK_SIZE - 1],
                          (INT8U           ) NET_OS_CFG_IF_RX_TASK_PRIO);
    #else
    os_err = OSTaskCreate((void (*)(void *)) NetOS_IF_RxTask,
                          (void          * ) 0,
                          (OS_STK        * )&NetOS_IF_RxTaskStk[0],
                          (INT8U           ) NET_OS_CFG_IF_RX_TASK_PRIO);
    #endif
#endif
```

Note that in the above example, the `#else` statement is indented with respect to the `#if` statement, but that the statements contained within the `#if` are *not* indented.

```
#ifndef  NET_SOCK_CFG_SEL_EN
#error  "NET_SOCK_CFG_SEL_EN                        not #define'd in 'net_cfg.h'"
#error  "                                    [MUST be  DEF_DISABLED]           "
#error  "                                    [     ||  DEF_ENABLED ]           "

#elif  ((NET_SOCK_CFG_SEL_EN != DEF_DISABLED) && \
        (NET_SOCK_CFG_SEL_EN != DEF_ENABLED ))
#error  "NET_SOCK_CFG_SEL_EN                   illegally #define'd in 'net_cfg.h'"
#error  "                                    [MUST be  DEF_DISABLED]           "
#error  "                                    [     ||  DEF_ENABLED ]           "

#elif   (NET_SOCK_CFG_SEL_EN == DEF_ENABLED)

    #ifndef  NET_SOCK_CFG_SEL_NBR_EVENTS_MAX
    #error  "NET_SOCK_CFG_SEL_NBR_EVENTS_MAX            not #define'd in 'net_cfg.h' "
    #error  "                                       [MUST be  >= NET_OS_NBR_EVENTS_MIN]"
    #error  "                                       [     &&  <= NET_OS_NBR_EVENTS_MAX]"

    #elif   (DEF_CHK_VAL(NET_SOCK_CFG_SEL_NBR_EVENTS_MAX, \
                         NET_OS_NBR_EVENTS_MIN,           \
                         NET_OS_NBR_EVENTS_MAX) != DEF_OK)
    #error  "NET_SOCK_CFG_SEL_NBR_EVENTS_MAX      illegally #define'd in 'net_cfg.h' "
    #error  "                                       [MUST be  >= NET_OS_NBR_EVENTS_MIN]"
    #error  "                                       [     &&  <= NET_OS_NBR_EVENTS_MAX]"
    #endif

#endif
```

Note that in the above example, the statements contained within the `#elif` statement are indented.

Listing 15 — Segregated Preprocessor and Source Code Statements Preferred

```
    for (prio = 0u; prio <= OS_TASK_IDLE_PRIO; prio++) {
        err = OSTaskStkChk(prio, &stk_data);
        if (err == OS_ERR_NONE) {
            ptcb = OSTCBPrioTbl[prio];
            if (ptcb != (OS_TCB *)0) {                              /* Make sure task 'ptcb' is ...   */
                if (ptcb != OS_TCB_RESERVED) {                      /* ... still valid.              */
#if OS_TASK_PROFILE_EN > 0u
    #if OS_STK_GROWTH == 1u
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom + ptcb->OSTCBStkSize;
    #else
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom - ptcb->OSTCBStkSize;
    #endif
                    ptcb->OSTCBStkUsed = stk_data.OSUsed;           /* Store number of entries used   */
#endif
                }
            }
        }
    }
```

Either of the example below are acceptable:

Listing 16 — Interleaved Preprocessor and Source Code Statements Discouraged

```
    for (prio = 0u; prio <= OS_TASK_IDLE_PRIO; prio++) {
        err = OSTaskStkChk(prio, &stk_data);
        if (err == OS_ERR_NONE) {
            ptcb = OSTCBPrioTbl[prio];
            if (ptcb != (OS_TCB *)0) {                              /* Make sure task 'ptcb' is ...   */
                if (ptcb != OS_TCB_RESERVED) {                      /* ... still valid.               */
#if OS_TASK_PROFILE_EN > 0u
                    #if OS_STK_GROWTH == 1u
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom + ptcb->OSTCBStkSize;
                    #else
                    ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom - ptcb->OSTCBStkSize;
                    #endif
                    ptcb->OSTCBStkUsed = stk_data.OSUsed;           /* Store number of entries used   */
#endif
                }
            }
        }
    }
```

```
    for (prio = 0u; prio <= OS_TASK_IDLE_PRIO; prio++) {
        err = OSTaskStkChk(prio, &stk_data);
        if (err == OS_ERR_NONE) {
            ptcb = OSTCBPrioTbl[prio];
            if (ptcb != (OS_TCB *)0) {                              /* Make sure task 'ptcb' is ...   */
                if (ptcb != OS_TCB_RESERVED) {                      /* ... still valid.               */
                    #if OS_TASK_PROFILE_EN > 0u
                        #if OS_STK_GROWTH == 1u
                            ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom + ptcb->OSTCBStkSize;
                        #else
                            ptcb->OSTCBStkBase = ptcb->OSTCBStkBottom - ptcb->OSTCBStkSize;
                        #endif
                        ptcb->OSTCBStkUsed = stk_data.OSUsed;       /* Store number of entries used   */
                    #endif
                }
            }
        }
    }
```

`switch` Statements

`switch` conditions should be formatted as shown in Listing 17. A **`switch`** statement should always have at least one case (MISRA 2004 Rule 15.5) in addition to a **`default`** case (MISRA 2004 Rule 15.3), which should be the last case in the group of **`default`** cases as shown in Listing 18, but may be empty for any optional **`switch`** statements as shown in Listing 19. Every non-empty case should be terminated by a **`break`** (MISRA 2004 Rule 15.2) or a **`return`** statement. However, complex or sequential code may be continued from non-empty case(s) to the next non-empty case(s) if appropriately commented as shown in Listing 20. **`switch`** cases should be arranged in some logical order: sequential by state, expected error codes; most-likely occurring to least-likely occurring; etc. Except for any optional **`switch`** statements, every expected **`case`** should be included in a **`switch`** statement even if grouped with the default case as shown in <xr id="fig:Expected_switch_Cases">Listing</xr>. A **`switch`** statement may be used to validate for a Boolean expression (opposes MISRA 2004 Rule 15.4) as shown in Listing 21, especially if the **`default`** case invalidates all other non-Boolean values.

Listing 17

```
    switch (pbuf_hdr->ProtocolHdrType) {                                        (1)
        case NET_PROTOCOL_TYPE_ICMP:
             ix  = pbuf_hdr->ICMP_MsgIx;
             len = pbuf_hdr->ICMP_MsgLen;
             break;
                                                                                (2)

#ifdef  NET_IGMP_MODULE_PRESENT
        case NET_PROTOCOL_TYPE_IGMP:
             ix  = pbuf_hdr->IGMP_MsgIx;                                        (3)
             len = pbuf_hdr->IGMP_MsgLen;
             break;
#endif

        case NET_PROTOCOL_TYPE_UDP:                                             (4)
        case NET_PROTOCOL_TYPE_TCP:
             ix  = pbuf_hdr->TCP_UDP_HdrIx;
             len = pbuf_hdr->TCP_UDP_HdrLen + (CPU_INT16U)pbuf_hdr->DataLen;
             break;                                                             (5)

        case NET_PROTOCOL_TYPE_NONE:
        default:                                                                (6)
            *perr = NET_ERR_INVALID_PROTOCOL;
             return;           /* Prevent 'break NOT reachable' compiler warning.  */  (7)
    }
```

(1) Use K&R-style braces: one space after **switch** before opening parenthesis and one space after closing parenthesis before opening brace.

(2) Two blank lines between non-empty cases.

(3) The **case** statement and labels are indented by four spaces, while the **case** statement body is indented by five spaces.

(4) Grouped cases should be consecutively listed with no blank lines.

(5) Each non-empty cases should end with **break**.

(6) Non-empty **default** case is required for all non-optional **switch** statements.

(7) Non-empty cases can also end with **return** in order to prevent a 'break NOT reachable' compiler warning. Such a use should be commented for clarity.

Listing 18 — Non-Terminal `default` Case Example

```
switch (win_update_code) {
    case NET_TCP_CONN_TX_WIN_REMOTE_UPDATE:
    default:                                                                    (1)
         if (pconn->TxWinSizeRemote > 0) {
             if (pconn->TxQ_ZeroWinTmr != (NET_TMR *)0) {
                 NetTmr_Free(pconn->TxQ_ZeroWinTmr);
                 pconn->TxQ_ZeroWinTmr  = (NET_TMR *)0;
             }
         } else {
             if (pconn->TxQ_ZeroWinTmr == (NET_TMR *)0) {
                 timeout_ms   = pconn->TxRTT_RTO_ms;
                 timeout_tick = pconn->TxRTT_RTO_tick;
                 tmr_update   = DEF_YES;
             }
         }
         break;

    case NET_TCP_CONN_TX_WIN_TIMEOUT:
         timeout_ms   =  (NET_TCP_TIMEOUT_MS)NetTCP_TxConnRTO_CalcBackOff(pconn, pconn->TxWinZeroWinTimeout_ms);
         timeout_tick = ((NET_TMR_TICK      )timeout_ms * NET_TMR_TIME_TICK_PER_SEC) / DEF_TIME_NBR_mS_PER_SEC;
         tmr_update   =   DEF_YES;
         break;
}
```

(1) The **default** case doesn't need to be the last **switch** statement case, but should be the last case in the group of **default** cases.

Listing 19 — Optional `switch` Statements Include Empty `default case`

```
switch (ack_code) {
    case NET_TCP_CONN_RX_ACK_VALID:
    case NET_TCP_CONN_RX_ACK_DUP:
         NetTCP_RxPktConnHandlerReTxQ(pconn, ack_code, pbuf_hdr, perr);
         if (*perr != NET_TCP_ERR_NONE) {
              return;
         }
         break;

    default:                              /* 'default' case intentionally empty. */    (1)
         break;
}
```

(1) Recommend empty **default** case with comment for optional **switch** statements.

Listing 20 **`switch`** *Statement Continued* **`case`***s*

```
switch (bit_nbr) {
    case 3u:
         if (mdc_ref_clk >  50000u) {
             mdc_cfg = MCFG_CLK_SEL_28;
             break;
         }
         if (mdc_ref_clk >  35000u) {
             mdc_cfg = MCFG_CLK_SEL_20;
             break;
         }
         if (mdc_ref_clk >  25000u) {
             mdc_cfg = MCFG_CLK_SEL_14;
             break;
         }
         if (mdc_ref_clk >  20000u) {
             mdc_cfg = MCFG_CLK_SEL_10;
             break;
         }
                                  /* 'break' intentionally omitted.  */               (1)

    case 2u:
         if (mdc_ref_clk >  15000u) {
             mdc_cfg = MCFG_CLK_SEL_8;
             break;
         }
         if (mdc_ref_clk >  10000u) {
             mdc_cfg = MCFG_CLK_SEL_6;
             break;
         }
                                  /* 'break' intentionally omitted.  */

    case 1u:
         mdc_cfg = MCFG_CLK_SEL_4;
         break;

    default:
        *perr = NET_DEV_ERR_INVALID_CFG;
         return;
}
```

```
switch (err) {

    ...

    case NET_CONN_ERR_CONN_HALF:
         if (paddr_remote != (CPU_INT08U *)0) {
             break;
         }
                                  /* 'break' intentionally omitted;   */              (2)
                                  /* MUST execute the following case: */
                                  /* 'NET_CONN_ERR_CONN_FULL'.        */
    case NET_CONN_ERR_CONN_FULL:
         if (conn_id_srch == psock->ID_Conn) {
            *perr =  NET_SOCK_ERR_NONE;
             return (NET_SOCK_BSD_ERR_NONE);
         }
         NET_CTR_ERR_INC(Net_ErrCtrs.NetSock_ErrInvalidAddrInUseCtr);
        *perr =  NET_SOCK_ERR_ADDR_IN_USE;
         return (NET_SOCK_BSD_ERR_BIND);

    ...
```

(1) Cases *may* intentionally omit **break** to continue to the next case(s) to avoid code duplication.

(2) Comments should indicate which case(s) should immediately follow the continued case to prevent moving or inserting code between the continued cases.

```
switch (*perr) {
    case NET_IF_ERR_NONE:
         NET_CTR_STAT_INC(Net_StatCtrs.NetIP_StatTxDatagramCtr);
        *perr = NET_IP_ERR_NONE;
         break;

    case NET_ERR_TX:
    case NET_IF_ERR_LINK_DOWN:
    case NET_IF_ERR_LOOPBACK_DIS:
         NET_CTR_ERR_INC(Net_ErrCtrs.NetIP_ErrTxPktDiscardedCtr)
         return;

    case NET_IP_ERR_INVALID_ADDR_HOST:                                              (1)
    case NET_IP_ERR_INVALID_ADDR_GATEWAY:
    case NET_IP_ERR_TX_DEST_INVALID:
    case NET_UTIL_ERR_NULL_PTR:
    case NET_UTIL_ERR_NULL_SIZE:
    default:                                                                        (2)
         NetIP_TxPktDiscard(pbuf, perr);
         return;
}
```

(1) Include all expected cases (for instance, all error codes returned from a function)...

(2) ...even those grouped with the **default** case.

Listing 21 — Boolean `switch` Examples

```
switch (nagle_en) {
    case DEF_ENABLED:
    case DEF_DISABLED:
         break;

    default:
        *perr =  NET_TCP_ERR_INVALID_ARG;
         return (DEF_FAIL);
}

switch (secure) {
    case DEF_ENABLED:
         NetSecure_InitSession(psock, perr);
         if (*perr != NET_SECURE_ERR_NONE) {
              NetOS_Unlock();
              return (DEF_FAIL);
         }
         DEF_BIT_SET(psock->Flags, NET_SOCK_FLAG_SOCK_SECURE);
         break;

    case DEF_DISABLED:
         DEF_BIT_CLR(psock->Flags, NET_SOCK_FLAG_SOCK_SECURE);
         break;

    default:
         NetOS_Unlock();
        *perr =  NET_SOCK_ERR_INVALID_ARG;
         return (DEF_FAIL);
```

`for` Loops

`for` loops should be formatted as shown in Listing 22. The body must be enclosed in braces (MISRA 2004 Rule 14.8), even for an empty **`for`** loop containing no statements.

Listing 22 — `for` Loops

```
for (i = 0u; i < mask_size; i++) {                                      (1)
    if (addr_subnet_mask & mask) {
        mask_nbr_one_bits++;                                            (2)
    }
    mask <<= 1u;
}

for (t = 0; t < dly; t++) {                                             (3)
    ;                                                                   (4)
}
```

(1) Use K&R-style braces: one space after **for** before opening parenthesis and one space after closing parenthesis before opening brace.

(2) Body indented by four spaces.

(3) At least one space after each semicolon.

(4) Single semicolon indicates intentional empty-body **for** loop.

`for` Loop Usage

`for` loops should only be used for a constant or known number of iterations. For example, initializing a fixed-size array as shown in Listing 23 or handling a dynamic, but known, number of iterations as shown in Listing 24. Multiple expressions or statements using the comma operator in any part of a **`for`** loop's control expressions is highly discouraged (see also Comma Operator and Simple Code Statements) but multiple simple counters is one reasonable exception as shown in Listing 25. However, **NO** expressions or statements in any of a **`for`** loop's control expressions should be used to implement infinite loops (see also Infinite Task Loops). Lastly, most Comparison and Context Rules should also apply for **`for`** loops when applicable; but **`for`** loop control expressions and comparisons should typically use only integer types and avoid floating-point types whenever possible.

Listing 23 — Use `for` Loops for Known Number of Iterations

```
p_conn = &NetTCP_ConnTbl[0];
for (i = 0; i < NET_TCP_CFG_NBR_CONN; i++) {
    NetTCP_ConnInit(p_conn);
    p_conn++;
}
```

(1) Number of iterations constant at compile-time = **`NET_TCP_CFG_NBR_CONN`** − 0.

Listing 24 — Use `for` Loops for Known Number of Iterations

```
dig_exp = 1.0f;
for (i = 1u; i < nbr_dig; i++) {                                        (1)
    dig_exp *= 10.0f;
}

for (i = nbr_dig; i > 0; i--) {                                         (2)
    ...
}
```

(1) Number of iterations known at run-time = **`nbr_dig`** − 1

(2) Number of iterations known at run-time = **`nbr_dig`** − 0

Listing 25 — Multiple `for` Loop Control Expressions Exception

```
#define  MEM_VAL_COPY_GET_INTU_LITTLE(addr_dest, addr_src, val_size)                \
    do {                                                                            \
        CPU_SIZE_T  i;                                                              \
        CPU_SIZE_T  j;                                                              \
                                                                                    \
        for (i = 0, j = (val_size) - 1; i < (val_size); i++, j--) {                 \
            (*(((CPU_INT08U *)(addr_dest)) + i)) = (*(((CPU_INT08U *)(addr_src)) + j)); \
        }                                                                           \
    } while (0)
```

`while` and `do...while` Loops

`while` and **`do...while`** loops should be formatted as shown in Listing 26. The body must be enclosed in braces (MISRA 2004 Rule 14.8), even for an empty loop.

Listing 26 — `while` and `do...while` Loop Formats

```
while (size_rem >= 0) {                                                    (1)
   *pmem_08++  = data_val;
    size_rem  -= sizeof(CPU_INT08U);                                       (2)
}

do {
    pstr_srch_ix = (CPU_CHAR *)(pstr + srch_ix);                           (3)
    srch_cmp     =  Str_Cmp_N(pstr_srch_ix, psrch_str, srch_str_len);
    srch_done    = (srch_cmp == 0) ? DEF_YES : DEF_NO;                     (4)
    srch_ix++;
} while ((srch_done == DEF_NO) &&                                          (5)
         (srch_ix   <= srch_len));
```

(1) Use K&R-style braces: one space after **while** before opening parenthesis and one space after closing parenthesis before opening brace.

(2) Body indented by four spaces

(3) Use K&R-style braces: one space after **do** before opening parenthesis and one space after closing parenthesis before opening brace.

(4) Equal signs on multiple lines aligned vertically

(5) One space before and after **while**; no space following closing parenthesis before semicolon.

`while` and `do...while` Loop Usage

`while` and **`do...while`** loops should only be used for an unknown number of iterations. For example, searching a dynamically-sized list as shown in Listing 27. Also, most Comparison and Context Rules should also apply for **`while`** and **`do...while`** loops when applicable.

Listing 27 — Use `while` Loops for Unknown Number of Iterations

```
p_buf = NetIF_TxListHead;
found = DEF_NO;
while ((p_buf != (NET_BUF *)0) &&
       (found ==  DEF_NO)) {
    found = (p_buf->DataPtr == p_buf_data) ? DEF_YES : DEF_NO;
    if (found != DEF_YES) {
        p_buf  = p_buf->NextTxListPtr;
    }
}
```

`while` and `do...while` Loop Conditionals

`while` and **`do...while`** loops should use more restrictive conditionals to continue looping. This prevents any incorrect, erroneous, or corrupted conditions from looping indefinitely. The **`while`** loop in Listing 28 is only executed while **`done`** is **`DEF_NO`** and exits immediately if **`done`** is set to **`DEF_YES`** **OR** is ever corrupted. Whereas the **`while`** loop in Listing 29> only exits once **`done`** is set to **`DEF_YES`**. This method is discouraged because if **`done`** is corrupted to some value that is neither **`DEF_NO`** or **`DEF_YES`**, the **`while`** loop could continue

indefinitely with corrupted state.

Listing 28 — More Restrictive `while` Loop Conditionals Preferred

```
done = DEF_NO;
while (done == DEF_NO) {                                                    (1)
    ...
    if (some condition) {
        done = DEF_YES;
    }
}
```

(1) **while** loop exits immediately when **done** is set to non-**DEF_NO** value.

Listing 29 — Less Restrictive `while` Loop Conditionals Discouraged

```
done = DEF_NO;
while (done != DEF_YES) {                                                   (1)
    ...
    if (some condition) {
        done = DEF_YES;
    }
}
```

(1) **while** loop may continue indefinitely if **done** corrupted to non-**DEF_YES** values.

Infinite Task Loops

Tasks (or processes) are often implemented as infinite loops that never exit. Historically, **`for`** loops with NO control expressions have been used to implement non-exiting, infinite loops. An infinite **`for`** loop, as shown in Listing 30, uses **`NULL`** arguments whose lack of information inadequately indicates that the **`NULL`** control expression is replaced with an invariant non-zero constant which prevents the **`for`** loop from ever exiting. Whereas an always-true **`while`** loop, as shown in Listing 31, is a better and much more intuitive representation of an infinite loop.

Listing 30 — Infinite `while` Loop Preferred

```
while (DEF_ON) {                                                            (1)
    AppTask();
    OS_Dly(1);
}
```

(1) Infinite loop **while** CPU is **ON**.

Listing 31 — Infinite `for` Loop Discouraged

```
for (;;) {                                                                    (1)
    AppTask();
    OS_Dly(1);
}
```

(1) Infinite **for** loop generated since **NULL** control expression replaced with non-zero constant.

`break` and `continue`

When possible, **`break`** (outside of **`switch`** statements) and **`continue`** should be avoided (MISRA 2004 Rules 14.5 and 14.6). A **`flag`** variable, modified in the loop body and checked in the loop condition, is preferred to breaking from the loop. For example, the use of a loop **`flag`** in Listing 32 is preferred to breaking from the loop in Listing 33.

Listing 32 — Loop Flag Preferred

```
found = DEF_NO;

while (found == DEF_NO) {

    ...

    if (...) {
        found = DEF_YES;
    }
}
```

Listing 33 — Loop `break` Discouraged

```
while (DEF_TRUE) {

    ...

    if (...) {
        break;
    }
}
```

Labels and `goto`

Labels should not be used (except in switch statements) [MISRA 1998 Rule 55 (rescinded)].

`goto` should *not* be used (MISRA 2004 Rule 14.4).

Comments

Comments allow a programmer to communicate details about implementation and guide a reader through code.

Content

Make every comment count. Get to the point. Do not state what a decent programmer would read from the code. Explain what the code does from a 'high-level' standpoint.

Do NOT use …

- **'Emotions' or profanity.** For example, do NOT use comments such as "Let's make this one big happy structure!".
- **The product name.** Consistently use a generic referent for the module such as "OS" rather than "µC/OS-II" or "network protocol suite" rather than "µC/TCP-IP".

DO use …

- **Structured sentences** (as much as possible).
- **Uppercase words** to emphasize meaning.
- **Acronyms, abbreviations and mnemonics** as long as the audience will understand the meaning.
  - High-level section comment blocks or function comment block headers and customer-provided template, port, or driver files should use fewer or no acronyms, abbreviations or mnemonics.

Style

Only C-style comments should be used:

```
/* This is a C-style comment. */
```

C++-style comments should NEVER be used:

```
// This is a C++-style comment.
```

Source comments should not be nested (MISRA 2004 Rule 2.3). Multi-line comments with a single comment terminator should NOT be used:

```
/* This type of comment can lead to confusion especially when describing a
   function like ClkUpdateTime().  The function looks like actual code! */
```

Multi-line source comments should use trailing/preceding ellipses:

Listing 1 — Comment Ellipses

```
/* Multi-line comments should use ellipses ...       */                              (1)
/* ... to indicate additional comment lines.         */

/* Multi-line comments may use 2-period ellipses  .. */                              (2)
/* .. if the comment is too long for the allotted .. */
/* .. space or to align words within the comment:    */

/* If inc < max avail, ..                            */                              (3)
/* .. inc nbr ack'd octets.                          */                              (4)
```

(1) Preferably use 3-period ellipses.

(2) Use 2-period ellipses for comments that require more space. Note that trailing ellipses align one space following the longest trailing comment line.

(3) May use 2-period ellipses to align words.

(4) Note: 'inc' aligned following 2-period ellipsis.

Major source comment code block sections may be denoted by a capitalized block header, centered within dashes:

```
void  NetIF_AddrHW_Get (NET_IF_NBR   if_nbr,
                        CPU_INT08U  *paddr_hw,                                Comment sections capitalized
                        CPU_INT08U  *paddr_len,                                and centered within dashes
                        NET_ERR     *perr)                                                 |
{                                                                                          v
                                                                  /* ----------------- ACQUIRE NET LOCK ----------------- */
    NetOS_Lock(perr);                                             /* See Note #1b.                                       */
    if (*perr != NET_OS_ERR_NONE) {
         return;
    }

#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)
    if (Net_InitDone != DEF_YES) {                                /* If init NOT complete, exit (see Note #2).           */
        NetOS_Unlock();
       *perr = NET_ERR_INIT_INCOMPLETE;
        return;
    }
#endif

                                                                  /* ---------------- GET NET IF HW ADDR ---------------- */
    NetIF_AddrHW_GetHandler(if_nbr, paddr_hw, paddr_len, perr);

                                                                  /* ----------------- RELEASE NET LOCK ----------------- */
    NetOS_Unlock();                                               ^                                                     ^
}                                                                 |                                                     |
                                                              column 65                                             column 121
```

Source Comments

NEVER interleave comments between code blocks, as shown in Listing 2. This makes the code difficult to follow because the comments distract the visual scanning of the code. Instead, use trailing comments to the right of all code, declarations, definitions; all starting at and terminating at the same columns as shown in Listing 3. Any word which appears in the AAM dictionary should be abbreviated accordingly.

Listing 2 — Avoid Interleaving Comments with Code

```
void  ClkUpdateTime (void)
{                                                       /* DO NOT comment like this! */
    /* Update the seconds */
    if (ClkSec >= CLK_MAX_SEC) {
        ClkSec = 0;
        /* Update the minutes */
        if (ClkMin >= CLK_MAX_MIN) {
            ClkMin = 0;
            /* Update the hours */
            if (ClkHour >= CLK_MAX_HOURS) {
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

Listing 3 — Use Trailing Comments Whenever Possible

```
void ClkUpdateTime (void)
{
    if (ClkSec >= CLK_MAX_SEC) {                        /* Update secs.                 */
        ClkSec = 0;
        if (ClkMin >= CLK_MAX_MIN) {                    /* Update mins.                 */
            ClkMin = 0;
            if (ClkHour >= CLK_MAX_HR) {                /* Update hrs.                  */
                ClkHour = 0;
            } else {
                ClkHour++;
            }
        } else {
            ClkMin++;
        }
    } else {
        ClkSec++;
    }
}
```

Comments should start and end on columns of 4; e.g., columns 65 and 121:

```
    if (pip_addrs == (NET_IP_ADDRS *)0) {                       /* If addr NOT found, ...                                 */
        NetOS_Unlock();
       *perr =  NET_IP_ERR_INVALID_ADDR_HOST;                   /* ... rtn err.                                           */
        return (NET_IP_ADDR_NONE);
    }                                                           ^                                                         ^
                                                                |                                                         |
                                                              column 65                                                 column 121
```

Comment Blocks

Comment blocks are used before functions, data declarations, constant definitions, etc. to describe the following section. Besides for function comment block headers, most comment blocks just include a title and possible Note(s) section which should be formatted as shown in Listing 4. Since comment blocks and notes are (hopefully) read by customers, AAM abbreviations should not be used.

Listing 4 — Example Comment Block

```
/*
*********************************************************************************************************
*                                 IP HEADER TIME-TO-LIVE (TTL) DEFINES                                  (1)
*
* Note(s) : (1) RFC #1122, Section 3.2.1.7 states that :                                                (2)
*
*               (a) "A host MUST NOT send a datagram with a Time-to-Live (TTL) value of zero."
*               (b) "When a fixed TTL value is used, it MUST be configurable."
*
*           (2) (a) RFC #1112, Section 6.1 states that "if the upper-layer protocol chooses not to
*                   specify a time-to-live, it should default to 1 for all multicast IP datagrams,
*                   so that an explicit choice is required to multicast beyond a single network".
*
*               (b) RFC #1112, Appendix I, Section 'Informal Protocol Description' states that
*                   IGMP "queries ... carry an IP time-to-live of 1" & that a "report is sent ...
*                   with an IP time-to-live of 1".
*
*                   Hence, every IGMP message uses a Time-to-Live (TTL) value of 1.
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                          IP FLAG DEFINES                                              (3)
*********************************************************************************************************
*/
```

(1) Title centered, capitalized, but **NOT** AAM abbreviated, but industry-standard acronyms, abbreviations or mnemonics acceptable

(2) Optional notes section

(3) Comment block without notes

Notes

Comment block headers may include a Note(s) section that may contain an ordered, hierarchical list of notes. If a code block requires an extended explanation, the comment text should be placed into a note in its comment block header as shown in Listing 5. Since comment blocks and notes are (hopefully) read by customers, AAM abbreviations should not be used. The numbering/lettering of notes should start using numbers then using the following sequence for each note's subsections: lower-case letters, numbers, upper-case letters, numbers, etc.:

- The first level should use (1), (2), (3) …
- The second level should use (a), (b), (c) …
- The third level should use (1), (2), (3) …
- The fourth level should use (A), (B), (C) …
- The fifth level should use (1), (2), (3) …
- …and so on …

A Function Comment Block Header's Description section may include Note(s) which will start numbering at Note #1 which would then carry into the function's Note(s) section (see Function Comment Block Header Description Notes Example).

Listing 5 — Example Comment Block Notes

```
             column 17
                 |
                 v
* Note(s): (1)  (a)  If the number of digits to format ('nbr_dig') is zero; then NO formatting
*                    is performed except possible NULL-termination of the string (see Note #4).
*
*                (b)  If the number of digits to format ('nbr_dig') is less than the number of
*                     significant integer digits of the number to format ('nbr'); then an invalid
*                     string is formatted instead of truncating any significant integer digits.
*
*           (2)  The number's base MUST be between 2 & 36, inclusive.
*
*           (3)  Leading character option prepends leading characters prior to the first non-zero digit.
*
*                (a)  (1)  Leading character MUST be a printable ASCII character.
*
*                          (2)  (A) Leading character MUST NOT be a number base digit, ...
*                               (B) with the exception of '0'.
*
*                (b)  (1)  The number of leading characters is such that the total number of significant
*                          integer digits plus the number of leading characters plus possible negative
*                          sign character is equal to the requested number of integer digits to format
*                          ('nbr_dig').
*
*                (c)  (1)  If the value of the number to format is     zero  ...
*                     (2)  ... & the number of digits to format is non-zero, ...
*                     (3)  ... but NO leading character available;           ...
*                     (4)  ... then one digit of '0' value is formatted.
```

(1) Each note and sub-note should start and indent on columns of 4; typically starting on column 17 for function comment block notes and column 13 for non-function comment block notes.

(2) One blank line between notes and major subnotes

(3) Minor subnotes do NOT have to be separated by a blank line, but should include trailing and/or preceding ellipses

[Note that the hierarchical numbers/letters are enclosed in sequential parenthesis, not alternating between parentheses and square brackets.]

Code blocks may then reference notes by their hierarchical numbers/letters.

If the note referred to by a code block is in the section's comment header (e.g. function comment block header), the reference can be formatted:

```
x = 20;        /* See Note #1a3B.                                */
```

If the note is in the comment header of a different function or section, the reference should be formatted:

```
x = 20;        /* See 'Mem_Copy()  Note #1a3B'.                  */
```

If the note is in the comment header of a different file, the reference should be formatted:

```
x = 20;        /* See 'lib_mem.c  Mem_Copy()  Note #1a3B'.       */
```

If the note reference follows other comments, the reference should be encapsulated in parentheses:

```
x = 20;        /* Init x coor (see Note #1a3B).                  */
```

If the note reference follows comments with a closing parenthesis, the reference should be encapsulated in square brackets:

```
x1 = 20;       /* Init x coor(s) [see Note #1a3B].               */
x2 = 40;
```

[Note that when referencing notes (or function names), parentheses and square brackets should be alternated to avoid using multiple levels of sequential parentheses or square brackets.]

Listing 6 — Advanced Comment Block Notes Examples

```
* (b) (1) (A) "The value of ... [the threshold] SHOULD correspond to at least 100 seconds."
*
*         (B) Therefore, the minimum threshold value is calculated as follows :
*
*             (1)    [                                                              ]
*                    [ Exponential * (Exponential ^ i),  when < Maximum Exponential ]
*                    [   Scalar          Base                      Timeout Value    ]        Minimum
*      Summation     [                                                              ] >=  Retransmission
*   i = 0 --> i = N  [ Maximum Exponential                                          ]       Threshold
*                    [    Timeout Value               ,  otherwise                 ]
*                    [                                                              ]
*
*                    [                                    ]
*                    [  3 * (2 ^ i)  ,  when < 64 seconds ]
*      Summation     [                                    ]  >=  100 seconds
*   i = 0 --> i = N  [ 64 seconds    ,  otherwise         ]
*                    [                                    ]
*
*                                                         N  >=  4.11
*
*             (2)                                         N   =  5
*
*               where
*                        N                          Minimum Excessive Retransmission Threshold
*                                                       (in number of retransmissions)
*                        Exponential Scalar         Exponential Scalar = 3 (see 'TCP ROUND-TRIP
*                                                       TIME (RTT) / RE-TRANSMIT TIMEOUT (RTO)
*                                                       DEFINES  Note #3a1A1b')
*                        Exponential Base           Exponential Base   = 2 (see 'TCP ROUND-TRIP
*                                                       TIME (RTT) / RE-TRANSMIT TIMEOUT (RTO)
*                                                       DEFINES  Note #3b2')
*                        Minimum Retransmission     Minimum Excessive Retransmission Threshold
*                              Threshold                (in seconds; see Note #7b1A)
```

```
* Description : (1) Calculates the total memory segment size for the number of blocks with specific size
*                   & alignment:
*
*
*     -----                  ======================  ---
*       ^     Mem Addr  ---> |  /  /  /  /  /  /  |   ^
*       |      (see          | /  /  /  /  /  /  /|   |    Mem Align Offset
*       |     Note #1a)      |/  /  /  /  /  /  / |   |  (see Notes #1e & #2a)
*       |                    |  /  /  /  /  /  /  |   v
*       |                    ======================  ---
*       |                    |                    |   ^
*       |                    |                    |   |
*       |                    |     Mem Blk #1     |   |        Blk Size
*       |                    |                    |   |     (see Note #1c)
*       |                    |                    |   v
*       |                    ----------------------  ---
*       |                    |  /  /  /  /  /  /  |   ^
*       |                    | /  /  /  /  /  /  /|   |    Blk Align Offset
```

```
*       |                     |/  /  /  /  /  /  / |   |  (see Notes #1f & #2b)
*       |                     |  /  /  /  /  /  /  |   v
*       |                     ======================  ---
*                             |         .          |
*   Total Size                |         .          |
* (see Note #2c)              |         .          |
*                             ======================  ---
*       |                     |                    |   ^
*       |                     |                    |   |
*       |                     |   Mem Blk #N - 1   |   |        Blk Size
*       |                     |                    |   |     (see Note #1c)
*       |                     |                    |   v
*       |                     ----------------------  ---
*       |                     |  /  /  /  /  /  /  |   ^
*       |                     | /  /  /  /  /  /  /|   |    Blk Align Offset
*       |                     |/  /  /  /  /  /  / |   |  (see Notes #1f & #2b)
*       |                     |  /  /  /  /  /  /  |   v
*       |                     ======================  ---
*       |                     |                    |   ^
*       |                     |                    |   |
*       |                     |     Mem Blk #N     |   |        Blk Size
*       |                     |                    |   |     (see Note #1c)
*       v                     |                    |   v
*     -----                   ======================  ---
*
*  where
*
*  (a) Mem Addr           Memory address of the beginning of the memory block
*                         ('pmem_addr')
*
*  (b) N                  Number of memory blocks to allocate ('blk_nbr')
*
*  (c) Blk Size           Size of memory block to allocate ('blk_size')
*
*  (d) Align              Required block memory alignment ('blk_align')
*
*  (e) Mem Align Offset   Offset required to align first memory block
*
*  (f) Blk Align Offset   Offset required to align every memory block
*
*
```

Special Comments

Special comments indicate the presence of known bugs, legacy implementation and future issues to address. The comment markers shown in Listing 7 are recommended. Using consecutive question marks MUST be avoided since it signifies the start of a trigraph sequence (MISRA 2004 Rule 4.2).

Listing 7 — Special Comments

```
/* &&&& Question(s) regarding implementation or design specification. */

/* #### Technical issue not yet/satisfactorily resolved.              */

/* $$$$ Future function that needs to be implemented.                */

/* @@@@ Old code to leave as-is because ....                         */
```

Constants

Numeric Constants

Numeric constants, whether **`#define`**'d, declared within **`enum`**'s, assigned to static globals, or hard-coded within source assignments or expressions; **may** require type suffixing.

The **`u`** suffix should be used with caution. Every operation/assignation using a constant with a **`u`** suffix will be considered as unsigned. Care must be taken if such a constant has a global scope, as it could effect multiple statements.

The **`L`** suffix should NEVER be used because it's implementation is CPU/compiler dependent (equivalent to **`long`** type).

Integer Constants

Integer constants may be suffixed with either the **`u`** unsigned modifier as shown in Listing 1. The **`u`** unsigned suffix modifier is used to declare an integer constant as unsigned while the **`L`** long suffix modifier is used to declare an integer constant as long. Note that the unsigned suffix modifier is unambiguous since it can modify any integer independent of the native word size for either the CPU or compiler. However, the long suffix modifier is ambiguous since it promotes the integer to the longest word size implemented/supported by the CPU and/or compiler.

Listing 1 — Integer Constant Examples

```
10      Signed      integer '10'

10u   Unsigned      integer '10'

-1      Signed      integer '-1'

-1u   Unsigned      integer '-1'
```

Integer constants must be carefully and appropriately suffixed, since the suffixed constants will promote other constants or variables in expressions to their suffixed modifications. For example, an unsigned integer constant will promote other constants or variables in an expression to unsigned, even initially negative constants (e.g., **`-1u`** examples shown in Listing 2). This is the more dangerous case since many arithmetic expressions that should be evaluated as signed, may include unsigned integer constants and still result in the correct signed answers for many

ranges of input. But for other ranges of signed values, these expressions result in incorrect signed answers due to incorrect promotion of the expression to unsigned.

Listing 2 shows an example of a signed expression in µC/TCP-IP that was broken for over a year when the **#define**'s included in the expression were modified to unsigned integer constants. The signed expression usually calculated fairly low, positive or negative signed values. But when **NET_TCP_TX_RTT_GAIN_DEV_INIT_NUMER** was incorrectly re-**#define**'d to from **1** to **1u**, small negative values for **rtt_cur_ms_scaled** were promoted to very high unsigned values. And even after dividing by unsigned **NET_TCP_TX_RTT_GAIN_DEV_INIT_DENOM = 2u**, the very high unsigned values were cast back and assigned to **rtt_dev_ms_scaled** as very high signed values. Correctly **#define**'ing **NET_TCP_TX_RTT_GAIN_DEV_INIT_NUMER** to **1**, small negative values for **rtt_cur_ms_scaled** remain signed and result in the expected calculation of fairly low negative signed values.

Listing 2 — Example of Signed Arithmetic Expression Incorrectly using Unsigned Integer Constants in µC/TCP-IP's `NetTCP_TxConnRTT_RTO_Calc()`

```
rtt_dev_ms_scaled = (rtt_cur_ms_scaled * NET_TCP_TX_RTT_GAIN_DEV_INIT_NUMER)
                                       / NET_TCP_TX_RTT_GAIN_DEV_INIT_DENOM;
```

However, if unsigned integer constants must be used in signed expressions (most likely by using generic **#define**'d constants), then these constants **MUST** be cast to an appropriate signed data type as shown in Listing 3.

Listing 3 — Example of Signed Arithmetic Expression Correctly using Unsigned Integer Constants in µC/TCP-IP's `NetTCP_TxConnRTT_RTO_Calc()`

```
rtt_dev_ms_scaled = (rtt_cur_ms_scaled * (NET_TCP_TX_RTT_MS_SCALED)NET_TCP_TX_RTT_GAIN_DEV_INIT_NUMER)    (1)
                                     / (NET_TCP_TX_RTT_MS_SCALED)NET_TCP_TX_RTT_GAIN_DEV_INIT_DENOM;
```

(1) Where **NET_TCP_TX_RTT_MS_SCALED** is an appropriately-sized signed data type.

Therefore, caution must be used when **#define**'ing unsigned integer constants, especially any values that may be used to generically **#define** other integer constants as shown in Listing 4.

Listing 4 — Examples of Unsigned Values `#define`*'d as Unsigned Integer Constants*

```
#define  DEF_INT_16U_MAX_VAL                  65535u                                        (1)
#define  DEF_INT_32U_MAX_VAL              4294967295u
#define  DEF_INT_64U_MAX_VAL   18446744073709551615u
```

```
#define  DEF_TIME_NBR_HR_PER_DAY                  24uL                                      (2)
#define  DEF_TIME_NBR_MIN_PER_HR                  60uL
#define  DEF_TIME_NBR_SEC_PER_MIN                 60uL

#define  DEF_TIME_NBR_SEC_PER_HR   (DEF_TIME_NBR_SEC_PER_MIN * DEF_TIME_NBR_MIN_PER_HR)  (3)
#define  DEF_TIME_NBR_SEC_PER_DAY  (DEF_TIME_NBR_SEC_PER_HR  * DEF_TIME_NBR_HR_PER_DAY)
#define  DEF_TIME_NBR_SEC_PER_WK   (DEF_TIME_NBR_SEC_PER_DAY * DEF_TIME_NBR_DAY_PER_WK)
#define  DEF_TIME_NBR_SEC_PER_YR   (DEF_TIME_NBR_SEC_PER_DAY * DEF_TIME_NBR_DAY_PER_YR)
```

(1) These 'maximum unsigned' values are **#define**'d as unsigned integer constants but **MUST** be cast to appropriate signed data types when used in signed expressions.

(2) Time constants ideally **NOT #define**'d as unsigned integer constants since they may be used in signed expressions, either directly or indirectly through other **#define**'s; however, also **#define**'d as unsigned and **MUST** be cast to appropriate signed data types when used in signed expressions.

(3) Also, **#define**'d as long integer constants to prevent overflow of later time constant **#define**'s.

Listing 5 — Examples of Unsigned Values NOT `#define`'d as Unsigned Integer Constants

```
#define  NET_TCP_TX_RTT_GAIN_RTO                  4                                      (1)
#define  NET_TCP_TX_RTT_GAIN_RTO_NUMER            4
#define  NET_TCP_TX_RTT_GAIN_RTO_DENOM            1

#define  NET_TCP_TX_RTT_SCALE                     8
#define  NET_TCP_TX_RTT_MS_SCALE  ((NET_TCP_TX_RTT_MS_SCALED)DEF_TIME_NBR_mS_PER_SEC * NET_TCP_TX_RTT_SCALE)
```

(1) TCP RTT constants **not #define**'d as unsigned integer constants since they may be used in signed expressions, either directly or indirectly through other **#define**'s.

Also note that not all issues can be easily resolved by appropriately or correctly suffixing integer constants, especially when using or **`#define`**'ing generic constants which are expected to be portable across a wide variety of CPUs and compilers. Some compiler, toolchains, or analysis tools may suggest or require suffixing certain integer constants for some situations but not others; while other compilers or tools may suggest the exact opposite modifications. These non-trivial porting issues require careful consideration as to when to modify or not modify specific constants. Mr. Jim Elliott and Mr. Grant Killey from Validated Software have proposed minimizing the use of **`u`** whenever possible and avoiding the use of **`L`** , however acknowledging that this may or may not satisfy all compiler, toolchains, or analysis tools.

Floating-Point Constants

This section is to be completed.

`#define`'s

`#define` constants should be aligned, when possible, and suffixed, where appropriate, as shown in Listing 6. **`#define`**'s which include multiple tokens, constants, and/or operations **MUST** be enclosed in parentheses.

Listing 6 — Example `#define`'s

```
#define  DEF_BIT_00                          0x01u                                          (1)
#define  DEF_BIT_08                        0x0100u
#define  DEF_BIT_16                    0x00010000u
```

```
#define  DEF_TIME_NBR_MIN_PER_HR               60u
#define  DEF_TIME_NBR_MIN_PER_DAY             (DEF_TIME_NBR_MIN_PER_HR  * DEF_TIME_NBR_HR_PER_DAY)   (2)

#define  DEF_TIME_NBR_SEC_PER_MIN              60u
#define  DEF_TIME_NBR_SEC_PER_HR              (DEF_TIME_NBR_SEC_PER_MIN * DEF_TIME_NBR_MIN_PER_HR)   (3)
#define  DEF_TIME_NBR_SEC_PER_DAY             (DEF_TIME_NBR_SEC_PER_HR  * DEF_TIME_NBR_HR_PER_DAY)
```

```
#define  NET_IP_HDR_LEN_MASK                 0x0Fu                                          (4)
#define  NET_IP_HDR_LEN_MIN                     5
#define  NET_IP_HDR_LEN_MAX                    15
```

(1) Right-align rightmost digit of constants; column 60 is a recommended column to align rightmost digits (or columns 52, 60, 68, etc.).

(2) Align operators.

(3) Use parentheses for **#define**'s with multiple tokens / operations.

(4) Use type suffixes carefully where appropriate.

`#define` Usage

`#define`'s (or Non-volatile Constants) are preferred over hard-coded values in source code for two main reasons:

- Name indicates context (e.g., **`DEF_NBR_MIN_PER_HR`** vs. **`DEF_NBR_SEC_PER_MIN`**; **`ERR_NONE`** or **`FLAG_NONE`**, etc. vs. **`0`**)
- Typos generate compiler errors (e.g. **`DEF_NBR_DEG_IN_CIRCLF`** or **`DEF_OCTET_NBE_BITS`** typo will fail to compile vs. hard-coded **`350`** or **`6`** which will compile with incorrect constants)

For example, a novice reader of Listing 7 might not necessarily understand the CRC algorithm or its usage of constants and therefore might not discover any incorrect or mis-typed constants. However, Listing 8 provides context for most of the CRC algorithm's constants which may be useful even to readers unfamiliar with CRCs.

Listing 7 — `NetUtil_32BitCRC_Calc()` with Hard-coded Constants

```
crc  = 0xFFFFFFFFu;                                                    (1)
poly = 0xEDB88320u;

pdata_val = p_data;
for (i = 0u; i < data_len; i++) {
    crc_data_val = (CPU_INT32U)((crc ^ *pdata_val) & 0xFF);

    for (j = 0u; j < 8u; j++) {
        crc_data_val_bit_zero = crc_data_val & 1u;
        if (crc_data_val_bit_zero > 0) {
            crc_data_val = (crc_data_val >> 1u) ^ poly;
        } else {
            crc_data_val = (crc_data_val >> 1u);
        }
    }

    crc = (crc >> 7) ^ crc_data_val;                                   (2)
    pdata_val++;
}
```

(1) Hard-coded values provide no context for the values.

(2) Incorrect **7** right shifts not apparent to reader unknowledgeable or unfamiliar with CRC algorithm; 7 ***might*** seem correct as a prime number one less than number of octets.

Listing 8 — `NetUtil_32BitCRC_Calc()` with `#define`'d Constants

```
crc  = NET_UTIL_32_BIT_ONES_CPL_NEG_ZERO;                                    (1)
poly = NET_UTIL_32_BIT_CRC_POLY_REFLECT;

pdata_val = p_data;
for (i = 0u; i < data_len; i++) {
    crc_data_val = (CPU_INT32U)((crc ^ *pdata_val) & DEF_OCTET_MASK);

    for (j = 0u; j < DEF_OCTET_NBR_BITS; j++) {
        crc_data_val_bit_zero = crc_data_val & DEF_BIT_00;
        if (crc_data_val_bit_zero > 0) {
            crc_data_val = (crc_data_val >> 1u) ^ poly;
        } else {
            crc_data_val = (crc_data_val >> 1u);
        }
    }

    crc = (crc >> DEF_OCTET_NBR_BITS) ^ crc_data_val;
    pdata_val++;
}
```

(1) **#define**'d values provide some context for the values. **DEF_OCTET_MASK**, **DEF_OCTET_NBR_BITS**, **DEF_BIT_00**, and **DEF_OCTET_NBR_BITS** all indicate expected context and macro values, even for novice readers.

Non-volatile Constants

Constants that are referenced frequently, require a fixed memory location, and/or do NOT require any data modifications may be declared as non-volatile **`const`**. A compiler/linker might allocate constants in non-volatile/ROM section memory based on linker and/or optimization settings. But declaring constants with **`const`** forces the compiler/linker to allocate storage for each constant in the linker's ROM section. For constants referenced in code, this allows a single memory location to be allocated per constant, instead of possibly replicating the constant in code space for every reference to the constant. This may not save on code/ROM space since indirect loads of numeric constants from absolute memory locations may not save that much memory versus immediate loads from code space. But it does have the benefit of explicitly/strongly typing the constants.

Non-volatile Numeric Constants

Constants that are referenced frequently or require a fixed memory location may be declared as non-volatile **`const`** as shown in Listing 9.

Listing 9 — Example Non-volatile Numeric Constants

```
const  CPU_FP32    Pi_FP32   = 3.141592;
const  CPU_INT32U  Pi_Int32  =  3141592;
const  CPU_INT16U  Pi_Int16  =  31415;

static  const  CPU_INT32U  NetIPv4_AddrMaskClassA     =  0x00000000u;        (1)
static  const  CPU_INT32U  NetIPv4_AddrMaskClassB     =  0xE0000000u;
static  const  CPU_INT32U  NetIPv4_AddrMaskClassC     =  0xE0000000u;
static  const  CPU_INT32U  NetIPv4_AddrMaskClassD     =  0xE0000000u;
static  const  CPU_INT32U  NetIPv4_AddrMaskLocalhost  =  0x7F000000u;
```

(1) Use **static** to declare local, file-scope constants. Type-suffixing is not absolutely necessary since constants' data types are explicitly declared.

Non-volatile ASCII String Constants

Most compilers/linkers automatically allocate ASCII strings in code as constant strings in the ROM linker section. Also, most compilers typically declare a single instance for each unique string (i.e., no duplicate or redundant copies of identical strings). And depending on compiler/linker settings, ASCII strings hard-coded in tables or declared as global variables, as shown in Listing 10, might be allocated in either the ROM or RAM linker sections. However, using **`const`** should force most compilers/linkers to allocate strings hard-coded in tables or declared as variables into the ROM linker section as shown in Listing 11 and Listing 12, respectively.

Listing 10 — Example Global ASCII String Variables/Constants

```
       CPU_CHAR   TgtName[] = "Micrium Target";                                  (1)
       CPU_CHAR  *HostName  = "Micrium Host";
```

(1) Linkers might allocate **TgtName/HostName** strings in either ROM or RAM linker sections, depending on compiler/linker settings

Listing 11 — Example Global ASCII String Constants

```
const  CPU_CHAR   AppName[] = "Example App";                                    (1)
```

(1) **const** forces **AppName** string to be allocated in ROM linker section.

Listing 12 — Example Table ASCII String Constants

```
static  const  HTTPs_STATUS  HTTPs_Status[] = {                                 (1)
    { 200,  (const  CPU_CHAR *)"OK"         },
    { 201,  (const  CPU_CHAR *)"Created"    },
    { 202,  (const  CPU_CHAR *)"Accepted"   },                                  (2)
    { 204,  (const  CPU_CHAR *)"No Content" },
         ...
```

(1) **const** allocates **HTTPs_Status[]** in ROM linker section. See also Non-volatile Tables.

(2) **const** modifies **CPU_CHAR** to read as "pointer to constant CPU_CHAR" so pointed-to strings should be allocated in ROM linker section.

Non-volatile Tables

Depending on compiler/linker settings, tables might be allocated in either the ROM or RAM linker sections. However, using **`const`** should force most compilers/linkers to allocate tables into the ROM linker section as shown in Listing 13.

Listing 13 — Example Constant Tables

```
const  NET_PHY_CFG_ETHER  NetPhy_Cfg_Generic_0 = {
    0u,
    NET_PHY_BUS_MODE_MII,
    NET_PHY_TYPE_EXT,
    NET_PHY_SPD_AUTO,
    NET_PHY_DUPLEX_AUTO
};
```

Data Types

Portable Data Types

Standard C data types MUST be avoided because their size is not portable (MISRA 2004 Rule 6.3). Instead, the data types in Listing 1 should be declared (within a µC/CPU port) based on the target processor and compiler used.

Listing 1 — Portable (µC/CPU) Data Types

```
typedef            char        CPU_CHAR;           /*  8-bit character                */
typedef  unsigned  char        CPU_BOOLEAN;        /*  8-bit boolean or logical       */
typedef  unsigned  char        CPU_INT08U;         /*  8-bit unsigned integer         */
typedef    signed  char        CPU_INT08S;         /*  8-bit   signed integer         */
typedef  unsigned  short       CPU_INT16U;         /* 16-bit unsigned integer         */
typedef    signed  short       CPU_INT16S;         /* 16-bit   signed integer         */
typedef  unsigned  int         CPU_INT32U;         /* 32-bit unsigned integer         */
typedef    signed  int         CPU_INT32S;         /* 32-bit   signed integer         */
typedef  unsigned  long        CPU_INT64U;         /* 64-bit unsigned integer         */
typedef    signed  long        CPU_INT64S;         /* 64-bit   signed integer         */

typedef            float       CPU_FP32;           /* 32-bit floating point           */
typedef            double      CPU_FP64;           /* 64-bit floating point           */

typedef  volatile  CPU_INT08U  CPU_REG08;          /*  8-bit register                 */
typedef  volatile  CPU_INT16U  CPU_REG16;          /* 16-bit register                 */
typedef  volatile  CPU_INT32U  CPU_REG32;          /* 32-bit register                 */
typedef  volatile  CPU_INT64U  CPU_REG64;          /* 64-bit register                 */
```

Custom Data Types

All custom data types MUST be **`typedef`**'d as shown in Listing 2 and **MUST NEVER** be **`#define`**'d as shown in Listing 3.

Listing 2 — `typedef`'d Data Types Mandatory

```
typedef  CPU_INT16U  in_port_t;                                                    (1)
typedef  CPU_INT32U  in_addr_t;

typedef  OS_EVENT  *NET_SECURE_OS_LOCK;

typedef  NET_TS_MS   NET_TCP_TX_RTT_MS;                                            (2)
typedef  CPU_INT32S  NET_TCP_TX_RTT_MS_SCALED;

typedef  void  (*NET_BUF_FNCT)(NET_BUF  *pbuf);                                    (3)

typedef  void  (*CPU_FNCT_VOID)(void);                                             (4)
typedef  void  (*CPU_FNCT_PTR )(void  *p_obj);

typedef  void  (*EVENT_FNCT)(EVENT_DATA      event_data[],                         (5)
                             EVENT_DATA_QTY  nbr_data);
```

(1) Data types/qualifiers should start two spaces after **typedef**. Qualifiers should be separated by (at least) two spaces, including any pointer operators.

(2) Two spaces following longest qualifier(s) and longest data types for a group of related **typedef**d data types; all other qualifiers, data types, and names to be vertically aligned.

(3) Function names are encapsulated within parentheses, preceded by a pointer operator, and should start two spaces after (longest) return data type (including any pointer operators).

(4) Zero spaces after (longest) function name and function parameters' opening parenthesis

(5) Parameter/argument list within parentheses; one parameter per line; two spaces following longest parameter qualifier(s) and longest data types; all other parameter qualifiers, data types, and names to be vertically aligned.

Listing 3 — `#define`'d Data Types Prohibited

```
#define  FS_BUF  CPU_CHAR
#define  FS_PTR  CPU_CHAR *
```

See also Function Prototype examples

`struct` and `union` Types

All structures and unions MUST be **`typedef`**'d as shown in Listing 4. The data type name MUST be written using all uppercase characters. A tag name MUST be provided, identical to the data type name, except using all lowercase characters. The data types of each member are indented 4 spaces, and the structure member names are also aligned vertically. Structures with function pointer members MUST be **`typedef`**'d as shown in Listing 5.

[Note: Unions are not recommended and should be avoided whenever possible (MISRA 2004 Rule 18.4).]

Listing 4 — Proper Structure Data Type

```
typedef  struct  comm_buf {                                                          (1)
    CPU_CHAR     RxBuf[COMM_RX_SIZE];  /* Storage of chars rx'd                    */  (2)
    CPU_CHAR    *RxInPtr;              /* Ptr to next ix   in Rx buf to rx next char */
    CPU_CHAR    *RxOutPtr;             /* Ptr to next char in Rx buf to extract    */  (3)
    CPU_INT16U   RxCtr;                /* Nbr of chars in Rx buf                   */  (4)

    CPU_CHAR     TxBuf[COMM_TX_SIZE];  /* Storage for chars to send                */  (5)
    CPU_CHAR    *TxInPtr;              /* Ptr to next ix   in Tx Buf to queue next char */
    CPU_CHAR    *TxOutPtr;             /* Ptr to next char in Tx Buf to send       */  (6)
    CPU_INT16U   TxCtr;                /* Nbr of chars in Tx buf to send           */
} COMM_BUF;                                                                          (7)
```

(1) Two spaces on either side the **struct** tag.

(2) One space before the brace bracket.

(3) One structure member per line.

(4) Indent by four spaces.

(5) Two spaces following longest qualifier(s) and longest data types; all other member qualifiers, data types, and names to be vertically aligned.

(6) Pointer operator(s) should immediately prefix member names.

(7) One space between the brace bracket and the data type name.

Listing 5 — `struct`'s with Function Pointer Members

```
typedef  struct  net_if_api {
    void         (*Start)               (NET_IF             *pif,
                                         NET_ERR            *perr);
                                                                                    (1)
    void         (*Stop)                (NET_IF             *pif,
                                         NET_ERR            *perr);

    CPU_BOOLEAN  (*AddrHW_IsValid)      (NET_IF             *pif,                   (2)
                                         CPU_INT08U         *paddr_hw);

    void         (*AddrMulticastAdd)    (NET_IF             *pif,
                                         CPU_INT08U         *paddr_protocol,        (3)
                                         CPU_INT08U          paddr_protocol_len,
                                         NET_PROTOCOL_TYPE   addr_protocol_type,
                                         NET_ERR            *perr);

    void         (*AddrMulticastRemove)(NET_IF             *pif,                   (4)
                                         CPU_INT08U         *paddr_protocol,
                                         CPU_INT08U          paddr_protocol_len,
                                         NET_PROTOCOL_TYPE   addr_protocol_type,
                                         NET_ERR            *perr);
} NET_IF_API;
```

(1) At least one line break between function prototypes.

(2) Function names are encapsulated within parentheses, preceded by a pointer operator, and should start two spaces after longest return data type (including any pointer operators); all other function names to be vertically aligned.

(3) Parameter/argument list within parentheses; one parameter per line; two spaces following longest parameter qualifier(s) and longest data types; all other parameter qualifiers, data types, and names to be vertically aligned.

(4) Zero spaces after longest function name and function parameters' opening parenthesis; all other function prototype parameters' opening parentheses to be vertically aligned.

See also Function Prototype examples

`enum` Types

This section is to be completed.

Data Type Qualifiers

`const` and **`volatile`** data type qualifers should be used whenever applicable. **`const`** has two related functions: to declare constant data in non-modifiable memory or to declare non-modifiable function arguments. **`volatile`** is used to prevent the compiler from caching or optimizing access to special variables, registers, or memory locations. CPU and device registers should typically be **`typedef`**'d with **`volatile`** register data types as shown in Listing 6.

Listing 6 — Register Data Types Example

```
typedef  struct  net_dev {
    CPU_REG32  MACIS;                  /* Ethernet MAC Raw Interrupt Status/Acknowledge.   */
    CPU_REG32  MACIM;                  /* Ethernet MAC Interrupt Mask.                     */
    CPU_REG32  MACRCTL;                /* Ethernet MAC Receive Control.                    */
    CPU_REG32  MACTCTL;                /* Ethernet MAC Transmit Control.                   */
    CPU_REG32  MACDATA;                /* Ethernet MAC Data.                               */
    CPU_REG32  MACIA0;                 /* Ethernet MAC Individual Address 0.               */
    CPU_REG32  MACIA1;                 /* Ethernet MAC Individual Address 1.               */
    CPU_REG32  MACTHR;                 /* Ethernet MAC Threshold.                          */
    CPU_REG32  MACMCTL;                /* Ethernet MAC Management Control.                 */
    CPU_REG32  MACMDV;                 /* Ethernet MAC Management Divider.                 */
    CPU_REG32  Reserved0;              /* [Reserved].                                      */
    CPU_REG32  MACMTXD;                /* Ethernet MAC Management Transmit Data.           */
    CPU_REG32  MACMRXD;                /* Ethernet MAC Management Receive Data.            */
    CPU_REG32  MACNP;                  /* Ethernet MAC Number of Packets.                  */
    CPU_REG32  MACTR;                  /* Ethernet MAC Transmission Request.               */
    CPU_REG32  Reserved1;              /* [Reserved].                                      */
    CPU_REG32  MACLED;                 /* Ethernet MAC LED Encoding.                       */
    CPU_REG32  MACIX;                  /* Ethernet PHY MDIX.                               */
} NET_DEV;
```

Scope

If a data type is to be used only in the implementation file, then it *should* be declared in the **`LOCAL DATA TYPES`** section of the module's source code file. If the data type is global, if must be placed in the **`DATA TYPES`** section of the module's header file.

Forward Declarations

If a structure data type must be referenced prior to its actual definition, a forward declaration of the structure must be used so that each reference has pre-knowledge of the existence of the structure. For example, if a structure internally includes pointers to a structure of the same type (or several structures have interdependent links), forward declarations allow the compiler to at least have knowledge of all structures' names, if not yet their definitions. Listing 7 shows an example of a forward-declared structure, **`LIST_NODE`**, whose definition has member pointers of **`LIST_NODE`** type.

Listing 7 — Example Forward Declaration of a Structure

```
typedef  struct  list_node  LIST_NODE;  /*   Forward declaration of LIST_NODE structure .. */

struct  list_node {                     /* .. allows definition  of LIST_NODE structure .. */
    LIST_NODE  *NextPtr;                /* .. to define pointers to LIST_NODE structures.  */
    LIST_NODE  *PrevPtr;
    void       *Data;
};
```

In addition, products or modules which define data types in many header files, may need to forward declare some of its data types prior to their actual definitions in their respective header files. This may be required when a header file that includes references to structures defined in later header files. For example, a header file may include function prototypes with pointers to structures defined in later header files. Therefore, any structures data types that require references prior to their definitions should be forward declared in a product's or module's common **`types.h`** header file.

Listing 8 shows some example forward-declared structures and data types from µC/TCP-IP' **net_type.h** header file and Listing 9 shows their references in **net_buf.h** function prototypes before these data structures are defined in later included header files.

Listing 8 — Example µC/TCP-IP net_type.h Data Types

```
    /*          NETWORK BUFFER DATA TYPES          */
typedef  struct  net_buf      NET_BUF;

    /*   NETWORK INTERFACE & DEVICE DATA TYPES   */
typedef          CPU_INT08U   NET_IF_NBR;

typedef  struct  net_if       NET_IF;
typedef  struct  net_if_api   NET_IF_API;

typedef  struct  net_dev_cfg  NET_DEV_CFG;
```

Listing 9 — Example µC/TCP-IP net_buf.h Data Type References

```
    /*   NETWORK BUFFER FUNCTION PROTOTYPES   */

NET_BUF    *NetBuf_Get       (NET_IF_NBR        if_nbr,
                              NET_TRANSACTION   transaction,
                              NET_BUF_SIZE      size,
                              NET_BUF_SIZE      ix,
                              NET_BUF_SIZE     *pix_offset,
                              CPU_INT16U        flags,
                              NET_ERR          *perr);

CPU_INT08U *NetBuf_GetDataPtr(NET_IF           *pif,
                              NET_TRANSACTION   transaction,
                              NET_BUF_SIZE      size,
                              NET_BUF_SIZE      ix,
                              NET_BUF_SIZE     *pix_offset,
                              NET_BUF_SIZE     *p_data_size,
                              NET_TYPE         *ptype,
                              NET_ERR          *perr);
```

Casting

Casting should be used with extreme caution, only when necessary, since it will hide helpful compiler warnings (especially if changes are made a long time after the code using an explicit cast was written).

Functions

Prototype Declarations

Prototype declarations must be provided for all functions (MISRA 2004 Rule 8.1) as shown in Listing 1. Parameter name identifiers must be given for all function parameters/arguments (MISRA 2004 Rule 16.3). Each function's parameter types and identifiers must be identical in both its prototype and definition (MISRA 2004 Rules 8.3, 16.4). Functions with no parameters must have the parameter type **`void`** (MISRA 2004 Rule 16.5). Each function must have an explicit return type (MISRA 2004 Rule 8.2); return **`void`** if no return type is required.

Functions should be logically grouped and ordered, with the same order for both the function prototypes and definitions. One preferred ordering is to group global API functions first: **`Init()`**, **`Cfg()`**, **`Start()/Stop()`**, etc.; followed by global internal functions; followed by local module functions. Functions not referenced or called from other source code files should be declared at file-scope with **`static`** qualifier (MISRA 2004 Rules 8.10, 8.11), as shown in Listing 2.

Listing 1 — Example Global Function Prototype Declarations

```
void          Mem_Copy  (       void        *p_dest,                          (1)
                         const  void        *p_src,
                                CPU_SIZE_T   size);                           (2)

CPU_BOOLEAN   Mem_Cmp   (const  void        *p_1_mem,
                         const  void        *p_2_mem,
                                CPU_SIZE_T   size);                           (3)

CPU_SIZE_T    Str_Len   (const  CPU_CHAR    *p_str);

CPU_SIZE_T    Str_Len_N (const  CPU_CHAR    *p_str,                           (4)
                                CPU_SIZE_T   len_max);

CPU_CHAR     *Str_Copy  (       CPU_CHAR    *p_str_dest,                      (5)
                         const  CPU_CHAR    *p_str_src);

CPU_CHAR     *Str_Copy_N(       CPU_CHAR    *p_str_dest,                      (6)
                         const  CPU_CHAR    *p_str_src,
                                CPU_SIZE_T   len_max);

CPU_CHAR     *Str_Cat   (       CPU_CHAR    *p_str_dest,
                         const  CPU_CHAR    *p_str_cat);
                                                                              (7)
CPU_CHAR     *Str_Cat_N (       CPU_CHAR    *p_str_dest,
                         const  CPU_CHAR    *p_str_cat,
                                CPU_SIZE_T   len_max);
```

(1) Always declare the return type; use **void** if no return type

(2) Function names should start two spaces after longest return data type (including any pointer operators); all other function names to be vertically aligned

(3) Parameter/argument list within parentheses; one parameter per line; two spaces following longest parameter qualifier(s) and longest data types; all other parameter qualifiers, data types, and names to be vertically aligned

(4) Qualifiers should be separated by two spaces, including any pointer operators

(5) Pointer operator(s) should immediately prefix function** or parameter names (**for functions that return pointers)

(6) Zero spaces after longest function name and function parameters' opening parenthesis; all other function prototype parameters' opening parentheses to be vertically aligned

(7) At least one line break between function prototypes

Listing 2 — Example Local Function Prototype Declarations

```
static  CPU_BOOLEAN   Mem_PoolBlkIsValidAddr(MEM_POOL    *p_mem_pool,                  (1)
                                             void        *p_mem_blk);

static  CPU_SIZE_T    Mem_PoolSegCalcTotSize(void        *p_mem_addr,                  (2)
                                             CPU_SIZE_T   blk_nbr,
                                             CPU_SIZE_T   blk_size,
                                             CPU_SIZE_T   blk_align);

static  void         *Mem_PoolSegAlloc      (MEM_POOL    *p_mem_pool,
                                             CPU_SIZE_T   size,
                                             CPU_SIZE_T   align);
```

(1) Declare local functions with **static** modifier

(2) Return data types should start two spaces after **static**

To facilitate unit testing, it is possible to use a macro instead of the **`static`** modifier. That macro could then be **`#define`**d as either **`static`** or nothing, as shown in Listing 3.

Listing 3 — Example module static modifier

```
#ifdef   FS_NAND_INTERN_ACCESS_EN
    #define  FS_NAND_INTERN                                                        (1)
#else
    #define  FS_NAND_INTERN     static
#endif

FS_NAND_INTERN  void             FS_NAND_Close         (FS_DEV           *p_dev);       (2)

FS_NAND_INTERN  void             FS_NAND_Rd            (FS_DEV           *p_dev,
                                                        void             *p_dest,
                                                        FS_SEC_NBR        sec_start,
                                                        FS_SEC_QTY        sec_cnt,
                                                        FS_ERR           *p_err);
```

(1) By default, FS_NAND_INTERN is **`#define`**d to **`static`**

(2) Declare local functions with **FS_NAND_INTERN** modifier

Definitions

Whenever possible, functions should be kept short; one standard metric, 'less than one page', gives a rough quantification of 'short'. However, each function should perform one or more related action with as few side effects as possible. In some cases, these related actions may not be short and therefore may not be easily or reasonably broken into sub-handler functions.

Related or similar functions should be similar or nearly identical in implementation, commenting, stylistic layout, and appearance as shown in Listing 4.

Whenever possible, functions should be reentrant [1] by using local variables. However, locks or critical sections may be required to protect certain objects or modules from corruption when accessed by multiple tasks.

Function arguments should be checked or validated as soon as possible and exit the function for any errors (opposes MISRA 2004 Rule 14.7).

The type of the return expression shall match the function's declared return type (MISRA 2004 Rule 16.8); an explicit cast may be necessary. The return expression should be enclosed in parentheses.

Function definitions should be separated by at least two line feeds.

Listing 4 — Example Function Definition

```
/*
*********************************************************************************************
*                                     Str_Char_N()
*
* Description : Search string for first occurrence of specific character, up to a maximum
*                   number of characters.
*
* Argument(s) : pstr            Pointer to string (see Note #1).                              (1)
*
*               len_max         Maximum number of characters to search (see Notes #2c & #3e).
*
*               srch_char       Search character.
*
* Return(s)   : Pointer to first occurrence of srch_char in string, if any (see Note #2b1).
*
*               Pointer to NULL, otherwise (see Note #2b2).
*
* Caller(s)   : Application.
*
* Note(s)     : (1) String buffer NOT modified.
*
*               (2) (a) IEEE Std 1003.1, 2004 Edition, Section 'strchr() : DESCRIPTION'          (2)
*                       states that :
*
*                       (1) "The strchr() function shall locate the first occurrence of
*                            'c' ('srch_char')... in the string pointed to by 's' ('pstr')."
*                       (2) "The terminating null byte is considered to be part of the string."
*
*                   (b) IEEE Std 1003.1, 2004 Edition, Section 'strchr() : RETURN VALUE'
*                       states that "upon completion, strchr() shall return" :
*
*                       (1) "a pointer to the byte," ...
*                       (2) "or a null pointer if the byte was not found."
*                           (A) #### Although NO strchr() specification states to return
*                               NULL for any other reason(s), NULL is also returned for
*                               any error(s).
*
*                   (c) Ideally, the 'len_max' argument would be the last argument in this
*                       function's argument list for consistency with all other custom string
*                       library functions. However, the 'len_max' argument is sequentially
*                       ordered as the second argument to comply with most standard library's
*                       strnchr() argument list.
```

```
*
*               (3) String search terminates when :                                             (3)
*
*                   (a) String pointer passed a NULL pointer.
*                       (1) No string search performed; NULL pointer returned.
*
*                   (b) String pointer points to NULL.
*                       (1) String overlaps with NULL address; NULL pointer returned.
*
*                   (c) String's terminating NULL character found.
*                       (1) Search character NOT found in search string; NULL pointer
*                           returned (see Note #2b2).
*                       (2) Applicable even if search character is the terminating NULL
*                           character (see Note #2a2).
*
*                   (d) Search character found.
*                       (1) Return pointer to first occurrence of search character in search
*                           string (see Note #2a1).
*
*                   (e) 'len_max' number of characters searched.
*                       (1) Search character NOT found in search string within first
*                           'len_max' number of characters; NULL pointer returned.
*                       (2) 'len_max' number of characters MAY include terminating NULL
*                           character (see Note #2a2).
*********************************************************************************************
*/                                                                                              (4)

CPU_CHAR  *Str_Char_N (const  CPU_CHAR    *pstr,                                              (5)(6)
                              CPU_SIZE_T   len_max,
                              CPU_CHAR     srch_char)                                           (7)
{
    const  CPU_CHAR    *pstr_char;                                                             (8)
           CPU_SIZE_T   len_srch;

    if (pstr == (const CPU_CHAR *)0) {                      /* Rtn NULL if srch str ptr NULL  */  (9)
        return ((CPU_CHAR *)0);                                                                 (10)
    }

    if (len_max == 0) {                                     /* Rtn NULL if srch len = 0       */  (11)
        return ((CPU_CHAR *)0);
    }

    pstr_char = pstr;
    len_srch  = 0u;
```

```
    while (( pstr_char != (const CPU_CHAR *)  0 )      &&   /* Srch str until NULL ptr        */  (12)
           (*pstr_char != (      CPU_CHAR  )'\0')      &&   /* ... or NULL char               */
           (*pstr_char != (      CPU_CHAR  )srch_char) &&   /* ... or srch char found         */
           ( len_srch  <  (      CPU_SIZE_T)len_max)) {     /* ... or max nbr chars srch'd.   */
        pstr_char++;                                        /* See Notes #3b-#3e              */  (13)
        len_srch++;
    }

    if (pstr_char == (const CPU_CHAR *)0) {                 /* Rtn NULL if NULL ptr found      */
        return ((CPU_CHAR *)0);
    }

    if (len_srch >= len_max) {                              /* Rtn NULL if srch char NOT found */
        return ((CPU_CHAR *)0);                             /* ... within max nbr of chars     */
    }

    if (*pstr_char != srch_char) {                          /* Rtn NULL if srch char NOT found */
         return ((CPU_CHAR *)0);                                                                  (14)
    }

    return ((CPU_CHAR *)pstr_char);                         /* Else rtn ptr to found srch char */ (15)
}                                                                                                 (16)
```

(1) Comment block describes function.

(2) Notes may list/reference software requirements ...

(3) ... as well as describe software implementation details.

(4) One line feed between function comment block header and function definition.

(5) Always declare the return type; use **void** if no return type.

(6) Function names should start two spaces after return data type (including any pointer operators).

(7) One space after function name and before function parameters' opening parenthesis.

(8) Parameter/argument list within parentheses; one parameter per line; two spaces following longest qualifier(s) and longest data types; all other parameter qualifiers, data types, and names to be vertically aligned.

(9) Pointer operator(s) should immediately prefix function or parameter names

(10) All local variables declared at beginning of function, followed by two blank lines.

(11) Arguments validated/checked at beginning of function.

(12) Comments to right of code.

(13) Comments may reference function note(s).

(14) Return values always enclosed in parentheses; one space following **return**, no space following closing parenthesis before semicolon.

(15) **const CPU_CHAR *pstr_char** cast and returned as non-**const CPU_CHAR ***.

(16) Two blank lines following function definition.

Comment Block Headers

Each function (or macro) must include a function comment block header, as shown in Listing 5; which includes five mandatory sections:

- **Description** A brief sentence describing what the function does but may include notes (see also **Note(s)** and Comment Notes).
- **Argument(s)** List the function arguments by name and their purpose. The argument names should match the names in both the function prototype and the definition. If an argument is valid only for a certain set of values, then those values should be listed and described. If an argument has been checked or validated by previous functions (see Argument Validation), it should be underlined and explicitly stated as checked or validated as shown in Listing 8. If there are no arguments, specify "none.".
- **Return(s)** List the return value(s) of the function. If nothing is returned, specify "none.".
- **Caller(s)** List the caller(s) of the function. If the function is used extensively from various locations, specify "various.". If the function is called only by the application, specify "Application.".
- **Note(s)** List any function notes (see also **Description** and Comment Notes). If no Note(s) are listed, specify "none.".

Listing 5 — Function Comment Block Header Template

```
/*
*********************************************************************************************
*                                     FunctionName()
*
* Description : Brief description of function.
*
* Argument(s) : Argument list        OR  "none."
*                                                                                          (1)
* Return(s)   : Return value(s)      OR  "none."
*
* Caller(s)   : Calling function(s)  OR  "various."  OR  "Application."
*
*               This function is a <ProductName> application interface (API) function &
*               MAY be called by application function(s).  [if a global API function]
*
*                   OR
*
*               This function is an INTERNAL <ProductName> function & MUST NOT be called
*               by application function(s).  [if an internal, non-API function]
*
*                   OR
*
*               This function is an <ProductName> [board-support package (BSP) |
*               device driver | interface(IF) | operating system (OS)] function & SHOULD
*               be called only by appropriate <ProductName> [BSP | device driver |
*               interface | OS | ProductName] function(s).
*
*                   OR
*
*               <No message>  [if an internal static function]
*
* Note(s)     : Detailed Note(s)  OR  "none."
*********************************************************************************************
*/
```

(1) Normally one blank line between each section

Listing 6 — Simple Function Comment Block Header Example

```
/*
*********************************************************************************************
*                                NetIP_GetAddrDfltGateway()                                   (1)
*
* Description : Get the default gateway IP address for a conListing d IP host address.
*
* Argument(s) : addr    ConListing d IP host address to get the default gateway               (2)
*                       (see Note #3).
*
*               p_err   Pointer to variable that will receive the return error code          (3)
*                       from this function :
*
*                       NET_IP_ERR_NONE                 ConListing d IP address's default    (4)
*                                                          gateway successfully returned.
*                       NET_IP_ERR_INVALID_ADDR_HOST    Invalid or not yet conListing d IP host (5)
*                                                           address.
*
*                       NET_ERR_INIT_INCOMPLETE         Network initialization NOT complete.
*
*                                                       ---- RETURNED BY NetOS_Lock() : ----    (6)
*                       NET_OS_ERR_LOCK                 Network access NOT acquired.
*
* Return(s)   : ConListing d IP host address's default gateway in host-order, if NO error(s).   (7)
*
*               NET_IP_ADDR_NONE,                                          otherwise.
*
* Caller(s)   : Application.                                                                   (8)
*
*               This function is a network protocol suite application interface (API)          (9)
*               function & MAY be called by application function(s) [see also Note #1].
*
* Note(s)     : (1) NetIP_GetAddrDfltGateway() is called by application function(s) & ... :    (10)
*
*                   (a) MUST NOT be called with the global network lock already acquired; ...
*                   (b) MUST block ALL other network protocol tasks by pending on & acquiring
*                       the global network lock (see 'net.h  Note #3').
*
*                   This is required since an application's network protocol suite API
*                   function access is asynchronous to other network protocol tasks.
*
*               (2) NetIP_GetAddrDfltGateway() blocked until network initialization completes.
*
*********************************************************************************************
*/
```

(1) Function name with trailing '()' centered.

(2) Argument descriptions should all start on same indentation column following longest argument name, preferably on column 29 or 33.

(3) However, **p_err** description, "Pointer to variable ..." should always start on column 29.

(4) Error codes should start on column 33 one blank line after **p_err**.

(5) Error code descriptions should start on column 65 or on the next indentation column following longest error code

(6) Error codes not returned directly from function but returned via sub-handler functions should be labeled "--RETURNED BY SubHanlderFnctName() : --".

(7) List all possible return values with the conditions when each is returned.

(8) Global API function only called by "Application" functions.

(9) Redundantly indicate function may be called by "Application" functions.

(10) Notes start at #1 since no detailed notes in **Description** section (see also Comment Notes).

Listing 7 — Detailed Function Comment Block Header Example

```
/*
*********************************************************************************************
*                                          NetIP_Tx()
*
* Description : (1) Prepare & transmit IP datagram packet(s) :                                (1)
*
*                   (a) Validate  transmit packet
*                   (b) Prepare & transmit packet datagram
*                   (c) Update    transmit statistics
*                                                                                             (2)
*
* Argument(s) : pbuf        Pointer to network buffer to transmit IP packet.
*
*               addr_src    Source      IP address.
*
*               addr_dest   Destination IP address.
*
*               TOS         Specific TOS to transmit IP packet (see Note #2a).
*
*               TTL         Specific TTL to transmit IP packet (see Note #2b) :
*
*                           NET_IP_TTL_MIN              Minimum TTL transmit value   (1)          (3)
*                           NET_IP_TTL_MAX              Maximum TTL transmit value (255)
*                           NET_IP_TTL_DFLT             Default TTL transmit value (128)
*                           NET_IP_TTL_NONE             Replace with default TTL
*
*               flags       Flags to select transmit options; bit-field flags logically OR'd :
*
*                           NET_IP_FLAG_NONE            No  IP transmit flags selected.
*                           NET_IP_FLAG_TX_DONT_FRAG    Set IP 'Don't Frag' flag.
*
*               popts       Pointer to one or more IP options configuration data structures
*                           (see Note #2c) :
*
*                           NULL                        NO IP transmit options configuration.
*                           NET_IP_OPT_CFG_ROUTE_TS     Route &/or Internet Timestamp options
*                                                           configuration.
*                           NET_IP_OPT_CFG_SECURITY     Security options configuration
*                                                           (see 'net_ip.c  Note #1d').
*
```

```
*               perr        Pointer to variable that will receive the return error code from
*                           this function :
*
*                           NET_IP_ERR_NONE             IP datagram(s) successfully prepared &
*                                                           transmitted to network interface
*                                                           layer.
*                           NET_IP_ERR_TX_PKT           IP datagram(s) NOT successfully
*                                                           prepared or transmitted to network
*                                                           interface layer.
*
*                                                       --- RETURNED BY NetIP_TxPktDiscard() : ----
*                           NET_ERR_TX                  Transmit error; packet discarded.
*
*                                                       ------- RETURNED BY NetIP_TxPkt() : -------
*                           NET_IF_ERR_LOOPBACK_DIS     Loopback interface disabled.
*                           NET_IF_ERR_LINK_DOWN        Network  interface link state down (i.e.
*                                                           NOT available for receive or transmit).
*
* Return(s)   : none.
*
* Caller(s)   : NetICMP_TxMsgErr(),                                                           (4)
*               NetICMP_TxMsgReq(),
*               NetICMP_TxMsgReply(),
*               NetUDP_Tx(),
*               NetTCP_TxPkt().
*
*               This function is an INTERNAL network protocol suite function & SHOULD NOT be      (5)
*               called by application function(s).
*
* Note(s)     : (2) (a) RFC #1122, Section 3.2.1.6 states that :                              (6)
*
*                       (1) "The IP layer MUST provide a means ... to set the TOS field of
*                            every datagram that is sent;" ...
*                       (2) "the default is all zero bits."
*
*                       See also 'net_ip.h  IP HEADER TYPE OF SERVICE (TOS) DEFINES'.
*
*                   (b) RFC #1122, Section 3.2.1.7 states that :
*
*                       (1) "The IP layer MUST provide a means ... to set the TTL field of
*                            every datagram that is sent."
*
*                       (2) "A host MUST NOT send a datagram with a Time-to-Live (TTL) value
*                           of zero."
*
```

```
*                       (3) "When a fixed TTL value is used, it MUST be configurable."
*
*                       See also 'net_ip.h  IP HEADER TIME-TO-LIVE (TTL) DEFINES'.
*
*                   (c) RFC #1122, Section 3.2.1.8 states that "there MUST be a means ... to
*                       to specify IP options to be included in transmitted IP datagrams".
*
*                       See also 'net_ip.h  IP HEADER OPTION CONFIGURATION DATA TYPES'.
*
*               (3) Network buffer already freed by lower layer; only increment error counter.
*********************************************************************************************
*/
```

(1) Detailed **Description** starts with Note #1 (see also Comment Notes).

(2) Two blank line after detailed **Description** section note(s).

(3) Arguments listing a valid set of values should list any value **#define**'s starting on column 33 one blank line after the argument, with each value's descriptions starting on column 65 or on the next indentation column following longest value.

(4) List all callers when called only by a few internal functions.

(5) Indicate function may be only called by internal functions

(6) Notes start at #2 since **Description** section included Note #1 (see also Comment Notes).

Listing 8 — Function Comment Block Header with Argument Validation Example

```
/*
*********************************************************************************************
*                                    NetIP_RxPktDemuxDatagram()
*
* Description : Demultiplex IP datagram to appropriate protocol layer.                          (1)
*
* Argument(s) : pbuf        Pointer to network buffer that received IP datagram.
*               ----        Argument checked   in NetIP_Rx().                                   (2)
*
*               pbuf_hdr    Pointer to network buffer header.                                   (3)
*               --------    Argument validated in NetIP_Rx().
*
*               perr        Pointer to variable that will receive the return error code from
*                           this function :
*
*                           NET_ERR_INVALID_PROTOCOL    Invalid/unknown protocol type.
*                           NET_ERR_RX                  Receive error; packet discarded.
*
*                                                       ------ RETURNED BY NetICMP_Rx() : ------
*                           NET_ICMP_ERR_NONE           ICMP message successfully demultiplexed.
*
*                                                       ------ RETURNED BY NetIGMP_Rx() : ------
*                           NET_IGMP_ERR_NONE           IGMP message successfully demultiplexed.
*
*                                                       ------ RETURNED BY NetUDP_Rx() : -------
*                           NET_UDP_ERR_NONE            UDP datagram successfully demultiplexed.
*
*                                                       ------ RETURNED BY NetTCP_Rx() : -------
*                           NET_TCP_ERR_NONE            TCP segment  successfully demultiplexed.
*
* Return(s)   : none.
*
* Caller(s)   : NetIP_Rx().
*
* Note(s)     : (1) When network buffer is demultiplexed to higher-layer protocol receive,
*                   buffer's reference counter is NOT incremented since the IP layer does NOT
*                   maintain a reference to the buffer.
*
*               (2) Default case already invalidated in NetIP_RxPktValidate().  However, the
*                   default case is included as an extra precaution in case 'ProtocolHdrType'
*                   is incorrectly modified.
*********************************************************************************************
*/
```

(1) Arguments checked/validated in previous functions should be underlined and listed as checked or validated.

(2) **pbuf** previously and explicitly checked in **NetIP_Rx()**.

(3) **pbuf_hdr** previously validated in **NetIP_Rx()** since it was initialized from explicitly checked **pbuf**.

Arguments

Functions must **NOT** be defined with a variable number of parameters/arguments (MISRA 2004 Rule 16.1).

Since arguments in C are passed by value, a function's argument values should not be changed within the function whenever possible. Instead, if any argument values must be modified, assign and modify them in local variables as shown in Listing 9. Maintaining all argument's original passed values aids debugging and testing of each function.

Listing 9 — Modify Function Arguments in Local Variables

```
CPU_SIZE_T  Str_Len_N (const  CPU_CHAR    *p_str,
                              CPU_SIZE_T   len_max)
{
    const  CPU_CHAR    *p_str_len;
           CPU_SIZE_T   len;

    p_str_len = p_str;                                                          (1)
    len       = 0u;
    while (( p_str_len != (const CPU_CHAR *)  0 ) &&
           (*p_str_len != (      CPU_CHAR  )'\0') &&
           ( len       <  (      CPU_SIZE_T)len_max)) {                         (2)
        p_str_len++;
        len++;
    }

    return (len);
}
```

(1) **p_str** assigned to **p_str_len** so **p_str** remains pointing to the start of the string, while **p_str_len** advances through **p_str** below.

(2) Instead of decrementing **len_max** until **0**, **len** incremented and compared to **len_max**.

Note that this does NOT mean that any addresses, data, objects, or memory pointed to by pointers cannot be modified. Just each passed pointer value itself should not be modified. The minor exception to this rule is when a pointer value is used to optionally return values whenever the passed pointer is non-**`NULL`**. Mr. Jim Elliott from Validated Software suggested (in email entitled "RE: from VSC: null pointers invoking default behavior", dated 2010/12/02 0939h) to overwrite any **`NULL`** optional pointers to point them to an unused, dummy local variable as shown in Listing 10, versus having to validate an optional pointer before each access as shown in Listing 11.

Listing 10 — Modifying `NULL` Pointer Argument Exception

```
 if (p_octets_reqd == (CPU_SIZE_T *) 0) {                /* If NOT avail, ...                          */ (1)
     p_octets_reqd  = (CPU_SIZE_T *)&octets_reqd_unused; /* ... re-cfg NULL rtn ptr to unused local var. */
    (void)&octets_reqd_unused;                           /* Prevent possible 'variable unused' warning.  */
 }
*poctets_reqd = 0u;                                      /* Init octets req'd for err.                 */ (2)

 ...

*poctets_reqd = size;                                    /* Rtn add'l heap size needed.                */ (3)
```

(1) If optional return pointer **p_octets_reqd == NULL**, overwrite with pointer to an unused, dummy local variable ...

(2) ... in order to assign values to pointed-to address ...

(3) ... without checking if pointer is **NULL** every access.

versus

Listing 11 — Optional `NULL` Pointer Argument

```
if (poctets_reqd != (CPU_SIZE_T *)0) {                                          (1)
   *poctets_reqd  = (CPU_SIZE_T  )0u;                  /* Init octets req'd for err.  */
}

...

if (poctets_reqd != (CPU_SIZE_T *)0) {                                          (2)
   *poctets_reqd  = (CPU_SIZE_T  )size;                /* Rtn add'l heap size needed. */
}
```

(1) Without overwriting possibly **NULL** optional pointer ...

(2) ... **MUST** check if pointer is non-**NULL** before every access

`const` Arguments

Any pointer argument whose pointed-to data, object, or memory is not modified by the function *should* be qualified as **`const`** (MISRA 2004 Rule 16.7) as shown in Listing 12. However, since it might not be convenient or possible to maintain every argument's **const** property, a fuction may need to cast **const** arguments to non-**const**.

Listing 12 — `const` Pointer Argument Examples

```
CPU_SIZE_T   Str_Len (const  CPU_CHAR  *p_str);

CPU_CHAR    *Str_Copy(       CPU_CHAR  *p_str_dest,                             (1)
                     const  CPU_CHAR  *p_str_src);

CPU_CHAR    *Str_Cat (       CPU_CHAR  *p_str_dest,                             (2)
                     const  CPU_CHAR  *p_str_cat);

CPU_INT16S   Str_Cmp (const  CPU_CHAR  *p_1_str,
                     const  CPU_CHAR  *p_2_str);
```

(1) **p_str_dest** is non-**const** because the pointed-to destination string is modified by these functions.

(2) All other strings are **const** since they are **NOT** modified by these functions.

Listing 13 — `const` Pointer Argument Function Call Examples

```
       CPU_CHAR    AppName[100]    = "Example App";
       CPU_CHAR    AppName_Demo[]  = "Demo";
const  CPU_CHAR    AppName_TCPIP[] = "TCP-IP";
       CPU_SIZE_T  size;

size = Str_Len(&AppName[0]);                                                    (1)
size = Str_Len(&AppName_TCPIP[0]);

(void)Str_Cat(&AppName[0], &AppName_Demo[0]);
(void)Str_Cat(&AppName[0], &AppName_TCPIP[0]);

(void)Str_Copy(&AppName_TCPIP[0], &AppName[0]);                                 (2)
(void)Str_Cat (&AppName_TCPIP[0], &AppName[0]);
```

(1) Note that either **const** or non-**const** strings may be passed into **const CPU_CHAR** arguments.

(2) But **const** strings may **NOT** be passed into non-**const CPU_CHAR** arguments.

Listing 14 — `const` Argument Handling

```
CPU_CHAR  *Str_Char_N (const  CPU_CHAR    *p_str,                               (1)
                              CPU_SIZE_T   len_max,
                              CPU_CHAR     srch_char)
{
    const  CPU_CHAR    *p_str_char;
           CPU_SIZE_T   len_srch;

    ...

    p_str_char = p_str;                                                         (2)
    len_srch   = 0u;
    while (( p_str_char != (const CPU_CHAR *)  0 )      &&                      (3)
           (*p_str_char != (      CPU_CHAR  )'\0')      &&
           (*p_str_char != (      CPU_CHAR  )srch_char) &&
           ( len_srch   <  (      CPU_SIZE_T)len_max)) {
        p_str_char++;
        len_srch++;
    }

    if (*p_str_char != srch_char) {
         return ((CPU_CHAR *)0);
    }

    return ((CPU_CHAR *)p_str_char);                                            (4)
}
```

(1) **const** argument **p_str**...

(2) ... assigned to **const** local variable **p_str_char**, ...

(3) ... compared to **const NULL** pointer, ...

(4) ... but cast to non-**const CPU_CHAR** return value.

`p_err` Arguments

If a function returns error information via a **`p_err`** argument, it **MUST** return an appropriate error code before every possible **`return`** statement and end of function. Even when a final sub-handler is called immediately before the end of the function, any sub-handler **`p_err`** errors *should* be handled and **`return`**'d from; **BUT** the function should also set a final **`p_err`** of no error before the function's end as shown in Listing 15. This allows a function to handle each sub-handler functions' errors in a consistent and correct manner. It also allows sub-handler functions and their subsequent error handling to be moved as one code block.

Listing 15 — Function with Final `p_err` Handler

```
void  SomeFnct (ERR  *p_err)
{
    ...

    SomeHandler(p_err);
    if (*p_err != ERR_NONE) {
         handle err here, if necessary;
         return;
    }

    LastHandler(p_err);
    if (*p_err != ERR_NONE) {                                          (1)
         handle err here, if necessary;
         return;
    }

   *p_err = ERR_NONE;                                                  (2)
}
```

(1) Even though **p_err == ERR_NONE** if no errors from **LastHandler()** ...

(2) ... should still set **p_err** to **ERR_NONE** for consistency and correctness.

An exception to this rule may be acceptable when no error handling is performed after multiple paths as shown in Listing 16.

Listing 16 — Functions with Multiple Paths without Final `p_err` Handlers

```
void  SomeFnct (ERR  *p_err)
{
    ...

    if ( ... ) {
        SomeHandler(p_err);
    } else if ( ... ) {
        PenultimateHandler(p_err);
    } else if ( ... ) {
        LastHandler(p_err);
    } else if ( ... ) {
       *p_err = SOME_ERR;
    } else {
       *p_err = ANOTHER_ERR;
    }
                                                                                (1)

}

void  AnotherFnct (ERR  *p_err)
{
    ...

    switch ( ... ) {
        case THIS:
             PenultimateHandler(p_err);
             break;

        case THAT:
             LastHandler(p_err);
             break;

        default:
            *p_err = SOME_ERR;
             return;
    }
                                                                                (2)

}
```

(1) Final **p_err = ERR_NONE; NOT** needed; **p_err** set in each final case above.

(2) Final **p_err = ERR_NONE; NOT** needed; **p_err** set in each final case above.

Listing 17 — Functions with Multiple Paths and `p_err` Handlers

```
void  SomeFnct (ERR  *p_err)
{
    ...

    if ( ... ) {
        SomeHandler(p_err);
        if (*p_err != ERR_NONE) {
             handle err here, if necessary;
             return;
        }

    } else if ( ... ) {
        PenultimateHandler(p_err);
        if (*p_err != ERR_NONE) {
             handle err here, if necessary;                                    (1)
             return;
        }

    } else if ( ... ) {
        LastHandler(p_err);
        if (*p_err != ERR_NONE) {
             handle err here, if necessary;
             return;
        }

    } else if ( ... ) {
       *p_err = SOME_ERR;
        return;

    } else {
       *p_err = ANOTHER_ERR;
        return;
    }

   *p_err = ERR_NONE;
}
```

```
void  SomeFnct (ERR  *p_err)
{
    ...

    switch ( ... ) {
        case THIS:
             PenultimateHandler(p_err);
             break;

        case THAT:
             LastHandler(p_err);
             break;

        default:
            *p_err = SOME_ERR;
             return;
    }

    if (*p_err != ERR_NONE) {
         handle err here, if necessary;                                    (2)
         return;
    }

   *p_err = ERR_NONE;
}
```

(1) **p_err** errors handled individually in each case.

(2) **p_err** errors handled commonly for all cases.

Argument Validation

Function arguments may need to be validated and/or checked before used by the function. Arguments should be checked as close to the beginning of a function and exited early with error for any invalid argument conditions or state. Each argument's validation might be mandatory or optional, depending on whether an invalid argument results in critical faults or is simply ignored. Also, each product or module might include developer-conListing d macros that dynamically validate/check certain arguments only when enabled. Arguments might require validating the state of an argument or derivative argument object. A function's arguments might be validated explicity or directly within the function but might also be validated in sub-handler functions. However, once an argument is validated, it should **NOT** be re-validated or checked in subsequent sub-handler functions but should be explicitly indicated in the Function Comment Block Header Argument(s) section.

Listing 18 — Mandatory Argument Validation Example

```
void  NetIF_Start (NET_IF_NBR   if_nbr,
                   NET_ERR     *perr)
{
    ...

                                                    /* ------------ GET NET IF ------------ */
    pif = NetIF_Get(if_nbr, perr);                                                          (1)
    if (*perr != NET_IF_ERR_NONE) {
         NetOS_Unlock();
         return;
    }

    if (pif->En != DEF_DISABLED) {                  /* If net IF already started, ...       */ (2)
        NetOS_Unlock();
       *perr = NET_IF_ERR_INVALID_STATE;            /* ... rtn err.                         */
        return;
    }

    ...
```

(1) **if_nbr** validated in **NetIF_Get**().

(2) **if_nbr** interface's state validated as **DISABLED**.

Listing 19 — Configurable Argument Validation Example

```
NET_SOCK_ID  NetSock_Accept (NET_SOCK_ID         sock_id,
                             NET_SOCK_ADDR      *paddr_remote,
                             NET_SOCK_ADDR_LEN  *paddr_len,
                             NET_ERR            *perr)
{
    ...

                                                                        /* ------------ VALIDATE ADDR PTRS ------------ */  (1)
#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)
    if (paddr_len == (NET_SOCK_ADDR_LEN *)0) {                          /* See Note #4a1.                              */  (2)
        NET_CTR_ERR_INC(Net_ErrCtrs.NetSock_ErrNullPtrCtr);
       *perr =  NET_SOCK_ERR_NULL_PTR;
        return (NET_SOCK_BSD_ERR_ACCEPT);
    }

     addr_len = (NET_CONN_ADDR_LEN)*paddr_len;
#endif
   *paddr_len = (NET_SOCK_ADDR_LEN) 0;                                  /* Cfg dflt addr len for err (see Note #4b).    */

#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)
    if (addr_len < (NET_CONN_ADDR_LEN)NET_SOCK_ADDR_SIZE) {             /* Validate initial addr len (see Note #4a).    */
        NET_CTR_ERR_INC(Net_ErrCtrs.NetSock_ErrInvalidAddrLenCtr);
       *perr =  NET_SOCK_ERR_INVALID_ADDR_LEN;
        return (NET_SOCK_BSD_ERR_ACCEPT);
    }

    if (paddr_remote == (NET_SOCK_ADDR *)0) {
        NET_CTR_ERR_INC(Net_ErrCtrs.NetSock_ErrNullPtrCtr);
       *perr =  NET_SOCK_ERR_NULL_PTR;
        return (NET_SOCK_BSD_ERR_ACCEPT);
    }
#endif

    ...
                                                                                                                          (3)
#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)                         /* -------- VALIDATE LISTEN SOCK USED --------- */
   (void)NetSock_IsUsed(sock_id, perr);                                                                                   (4)
    if (*perr != NET_SOCK_ERR_NONE) {
         NetOS_Unlock();
         return (NET_SOCK_BSD_ERR_ACCEPT);
    }
#endif

    ...
```

(1) Comment block section **NOT** on same line as configurable validation since address pointer validation extends over several sections.

(2) Address pointers explicitly checked/validated.

(3) Comment block section on same line as configurable validation since section **ONLY** validates the listen socket.

(4) **sock_id** validated in **NetSock_IsUsed()**.

Local Variables

All local variables should be declared at the beginning of the function, one variable per line, followed by two blank lines as shown in Listing 20. Whenever possible, declare local variables in some logical grouping/ordering(s):

1. dynamically-included (via preprocessor)
2. pointers
3. largest-to-smallest size
4. order used
5. grouped by name/usage

`static` local variables are discouraged since they require pre-**`main()`** initialization whereas Global or Local Global Variables can be explicitly initialized in product or module initialization code.

Listing 20 — Local Variables Examples

```
CPU_CHAR  *Str_Cat_N (       CPU_CHAR    *pstr_dest,
                      const  CPU_CHAR    *pstr_cat,
                             CPU_SIZE_T   len_max)
{
           CPU_CHAR    *pstr_cat_dest;                                          (1)
    const  CPU_CHAR    *pstr_cat_src;
           CPU_SIZE_T   len_cat;

NET_SOCK_ID  NetSock_Accept (NET_SOCK_ID         sock_id,
                             NET_SOCK_ADDR      *paddr_remote,
                             NET_SOCK_ADDR_LEN  *paddr_len,
                             NET_ERR            *perr)
{
#if (NET_SOCK_CFG_FAMILY == NET_SOCK_FAMILY_IP_V4)
    NET_SOCK_ADDR_IP   *paddr_ip;                                             (2)
#endif
#ifdef  NET_SECURE_MODULE_PRESENT
    CPU_BOOLEAN         secure;
#endif
    CPU_INT08U          addr_remote[NET_SOCK_CFG_ADDR_LEN];
    NET_CONN_ADDR_LEN   addr_len;
    NET_SOCK           *psock_listen;
    NET_SOCK           *psock_accept;
    NET_SOCK_ID         sock_id_accept;
    NET_CONN_ID         conn_id_accept;
    NET_CONN_ID         conn_id_transport;
    CPU_BOOLEAN         block;
    NET_ERR             err;

static  void  NetSock_ConnHandlerAddrRemoteValidate (NET_SOCK       *psock,
                                                     NET_SOCK_ADDR  *paddr_remote,
                                                     NET_ERR        *perr)
{
```

```
#if (NET_SOCK_CFG_FAMILY == NET_SOCK_FAMILY_IP_V4)
    NET_SOCK_ADDR_IP      *paddr_ip;
#endif
    CPU_INT08U             addr_remote[NET_SOCK_CFG_ADDR_LEN];
    CPU_INT08U             addr_local[NET_SOCK_CFG_ADDR_LEN];
    NET_CONN_ADDR_LEN      addr_local_len;
    NET_CONN_ID            conn_id;
    NET_CONN_ID            conn_id_srch;
    NET_CONN_FAMILY        conn_family;
    NET_CONN_PROTOCOL_IX   conn_protocol_ix;
    NET_ERR                err;

static  void  NetTCP_TxConnTxQ (NET_TCP_CONN        *pconn,
                                NET_BUF_HDR         *pbuf_hdr,
                                NET_TCP_ACK_CODE     tx_ack_code,
                                CPU_BOOLEAN          tx_q_timeout,
                                NET_TCP_CLOSE_CODE   close_code,
                                NET_ERR             *perr)
{
    NET_BUF           *pseg;
    NET_BUF           *pseg_next;
    NET_BUF           *pbuf_q_tail;
    NET_BUF_HDR       *pseg_hdr;
    NET_BUF_HDR       *pseg_next_hdr;
    NET_BUF_HDR       *pbuf_q_tail_hdr;
    NET_CONN_ID        conn_id;
    NET_IF_NBR         if_nbr;
    NET_IP_TOS         TOS;
    NET_IP_TTL         TTL;
    NET_IP_ADDR        src_addr;
    NET_IP_ADDR        dest_addr;
    NET_TCP_PORT_NBR   src_port;
    NET_TCP_PORT_NBR   dest_port;
    NET_TCP_SEQ_NBR    seq_nbr;
    NET_TCP_SEQ_NBR    ack_nbr;
    NET_TCP_WIN_SIZE   win_size;
    NET_TCP_WIN_SIZE   tx_data_qd;
    NET_TCP_WIN_SIZE   tx_data_min;
    NET_TCP_WIN_SIZE   tx_th_q_min;
    NET_TMR_TICK       timeout_tick;
    CPU_INT16U         flags_tcp;
    CPU_INT16U         flags_ip;
    CPU_BOOLEAN        tx_ack;
    CPU_BOOLEAN        tx_seg;
    CPU_BOOLEAN        tx_seg_push;
    CPU_BOOLEAN        tx_seg_close;
    CPU_BOOLEAN        tx_segs_txd;
```

```
    CPU_BOOLEAN        tx_segs;
    CPU_BOOLEAN        tx_done;
    CPU_BOOLEAN        tx_nagle;
    CPU_BOOLEAN        tx_th_seg;
    CPU_BOOLEAN        tx_th_mss;
    CPU_BOOLEAN        tx_th_nagle;
    CPU_BOOLEAN        tx_th_silly_win;
    CPU_BOOLEAN        tx_tmr_free;
    CPU_BOOLEAN        net_rx_avail;
    NET_CTR            net_rx_nbr;
    NET_CTR            tx_seg_nbr;
    NET_ERR            err;
    NET_ERR            err_rtn;
```

(1) One local variable per line; two spaces following longest variable qualifier(s) and data types; all other variable qualifiers, data types, and names to be vertically aligned.

(2) Dynamically included variables declared first.

(3) Largest variables declared before smaller variables.

(4) Dynamically included variables declared first.

(5) Largest variables declared before smaller variables.

(6) Similar variables grouped together (by name/utility/type).

(7) Pointer variables declared before other variables.

(8) Pointer operator(s) should immediately prefix variable names.

(9) Largest variables declared before smaller variables.

(10) Similar variables grouped together (by type).

(11) Similar variables grouped together (by name).

(12) Similar variables grouped together (by utility).

Listing 21 — Example of Dynamic Local Variable Ordering Exception

```
void  NetIF_ISR_Handler (NET_IF_NBR         if_nbr,
                         NET_DEV_ISR_TYPE   type,
                         NET_ERR           *perr)
{
    NET_IF       *pif;
    NET_IF_API   *pif_api;
#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)                                 (1)
    CPU_BOOLEAN   done;
    CPU_SR_ALLOC();
#endif
```

(1) Dynamically-included local variables atypically declared after all other local variables to group **done** with **CPU_SR_ALLOC()** which **MUST** be declared last [see **`CPU_SR_ALLOC()`**].

`cpu_sr` Local Variable

µC/CPU critical section macros, **`CPU_CRITICAL_ENTER()`** and **`CPU_CRITICAL_EXIT()`**, (conditionally) require the CPU's specific status register to be declared as a local variable, **`cpu_sr`** (see also Critical Sections). If µC/CPU's 'status register' allocation macro **`CPU_SR_ALLOC()`** is used to declare **`cpu_sr`**, it **MUST** be declared following all other local variables as shown in Listing 22.

Listing 22 — Allocating `cpu_sr` with `CPU_SR_ALLOC()` Example

```
CPU_BOOLEAN  NetIP_CfgAddrAdd (NET_IF_NBR    if_nbr,
                               NET_IP_ADDR   addr_host,
                               NET_IP_ADDR   addr_subnet_mask,
                               NET_IP_ADDR   addr_dflt_gateway,
                               NET_ERR      *perr)
{
    CPU_BOOLEAN     addr_cfgd;
    NET_IP_IF_CFG  *pip_if_cfg;
    NET_IP_ADDRS   *pip_addrs;
    CPU_SR_ALLOC();                                                          (1)

    ...
}
```

(1) **CPU_CRITICAL_ENTER() MUST** be declared following ALL other local variables.

Error Handling

The following is the recommended error handling sequence, especially for non-end-of-function returns due to intermediate fault conditions, as shown in Listing 23:

1. Resource deallocation, if any
2. Debug output, if any
3. **`p_err`** assignment, if any
4. **`return`** statement

Listing 23 — Recommended Error Handling Sequence Example

```
if (some_fault) {
    ObjChipSelDis(p_obj);                                                    (1)
    ObjFree(p_obj);
    TRACE("Object condition failed.\n\r");                                   (2)
   *p_err = OBJ_FAULT;                                                       (3)
    return;                                                                  (4)
}
```

(1) Resource deallocation

(2) Debug output

(3) **p_err** assignment

(4) **return;**

Calls

In a function invocation, there should be no spaces between a function name and its open parenthesis. There should be no space between the open and close parentheses when a function without arguments is invoked:

```
DispInit();
```

At least one space is needed after each comma to separate function arguments:

```
DispStr(s, x, y);
```

Functions with many arguments should be separated argument one per line and aligned:

```
NetIP_TxPkt(p_buf,                                                          (1)
            p_buf_hdr,
            addr_src,
            addr_dest,
            TOS,
            TTL,
            flags,
            p_opts,
            p_err);

NetASCII_Str_to_MAC(p_dev_cfg->HW_AddrStr,                                  (2)
                    &hw_addr[0],
                     p_err);

Clk_DateTime_Make(&date_time,                                               (3)
                  2010,
                    10,
                    23,                                                     (4)
                     6,
                     2,
                    21,
                  -18000);
```

(1) Align left-most alphanumeric characters on same column immediately following opening parenthesis if the first argument is NOT offset by prefixing operator(s).

(2) Align left-most alphanumeric characters on same column immediately following opening parenthesis even if other argument(s) are offset by prefixing operator(s).

(3) Align left-most alphanumeric characters on same column offset by prefixing operators, when necessary.

(4) Align consecutive, numerical constants' one's digit on same column.

The type of the parameters passed to a function should be compatible with the expected types in the function prototype (MISRA 2004 Rule 10.2). Function calls which require any argument(s) to be cast should be separated one argument per line and aligned:

```
                             Width of all casts is width of longest
                             cast expression, including any pointer
                             operators, to align closing parenthesis
                             for all cast expressions on same column
                                             |
                                             v
                                      ------------
rx_len = (CPU_INT16S)NetSock_RxDataFrom(              sock_id,
                                        (void       *)rx_data,
                                        (CPU_INT16U  )sizeof(rx_data),
                                                      0,
                                                     &addr_remote,
                                                     &addr_len,
                                        (void       *)0,
                                                      0u,
                                        (CPU_INT08U *)0,
                                                     &err);
                                        ^             ^
                                        |             |
                      Opening parenthesis for casts   |
                       starts one column following    |
                      function's opening parenthesis  |
                                                      |
                               Align arguments' left-most alphanumeric characters
                               on same column following casts' closing parenthesis
                               [as long as no cast is followed by an argument with
                                         any prefixing operator(s)]

rx_len = (CPU_INT16S)NetSock_RxData(             sock_id,
                                    (void     *)&rx_data[0],
                                    (CPU_INT16U) sizeof(rx_data),
                                                 0,
                                                &err);
                                                 ^
                                                 |
                            Align arguments' left-most alphanumeric characters
                             on same column following the longest sequence of
                              operators following casts' closing parenthesis
```

```
FnctWithFirstArgCast((CPU_INT16U)int32u,                                    (1)
                                 &err);

AnotherFnct(            p_file,
            (CPU_INT16U)int32u,
                        p_err);

FnctWithManyArgs((void    *)p_array,                                        (2)
                            int32u,
                            int16u,
                            addr,
                 (ADDR_SIZE)addr_size,
                            flags,
                            p_err);

AnotherFnctWithManyArgs((void      *) p_array,                              (3)
                                      int32u,
                                      int16u,
                                      addr,
                        (ADDR_SIZE *)&addr_size,
                                      flags,
                                     &err);

FnctWithManyArgs(            p_mem,
                             int32u,
                 (CPU_INT16U)int32u,
                             addr,
                 (ADDR_SIZE )addr_size,
                             flags,
                             p_err);

AnotherFnctWithManyArgs(              p_mem,
                                     *p_int32u,                             (4)
                        (CPU_INT16U )*p_int32u,                             (5)
                                      addr,
                        (ADDR_SIZE *) 0,                                    (6)
                                      flags,
                                     &err);
```

(1) **int32u** cast explicitly cast to **CPU_INT16U**

(2) non-void **p_array** MUST be cast to **void ***

(3) non-void **p_array** MUST be cast to **void ***

(4) ***p_int32u** does NOT need cast for **CPU_INT32U** argument

(5) ***p_int32u** does need cast for **CPU_INT16U** argument

(6) **NULL** pointers SHOULD be explicitly cast for clarity

Return Values

Ideally, functions' return values should be checked and/or handled (MISRA 2004 Rule 16.10) as shown in Listing 24.

Listing 24 — Example Function Return Value Handled

```
addr_ip_host = NetIP_GetAddrHostCfgdHandler(addr_ip_remote);
if (addr_ip_host != NET_IP_ADDR_NONE) {
    paddr_local = (NET_SOCK_ADDR *)&addr_ip_local;
} else {
    paddr_local = (NET_SOCK_ADDR *)0;
}
```

Listing 25 — Example Function Return Value Ignored

```
(void)OSSemPendAbort(psignal,                /* Ignore OSSemPendAbort()'s return count. */
                     OS_PEND_OPT_BROADCAST,
                    &os_err);
```

References

[1] http://en.wikipedia.org/wiki/Reentrant_(subroutine)

Naming Conventions

Table 1 — Summary of Naming Conventions

Category	Format	Examples
Appropriateness	Use appropriate, precise, but concise names	
Uniqueness	All file-scope and global identifiers must be unique	
Abbreviation	Use appropriate AAM abbreviations	
Length	No more than 31 significant characters	
Hierarchy	Use consistent hierarchical naming schemes	Module-Object-Operation Module-Object-State Module-Operation-Object
Units	Suffixed with a preceding underscore ('_') All lowercase Singular whenever possible **ms**, **us**, and **ns** suffixed to camel-case identifiers **mS**, **uS**, and **nS** suffixed to all-uppercase identifiers	**NetOS_TCP_RxQ_TimeoutTbl_tick** **CPU_IntDisMeasStop_cnts** **TxRTT_RTO_sec** **NetOS_TimeoutMin_us** **NET_APP_TIME_DLY_MAX_mS**
Files	Prefixed with the module identifier followed by an underscore ('_') All lowercase Words/tokens separated by an underscore ('_') Assembly filenames should be suffixed with **_a** Configuration filenames should be suffixed with **_cfg** OS port filenames should be suffixed with **_os** BSP port filenames should be suffixed with **_bsp**	**lib_str.c** **os_task.c** **net_if_loopback.h** **cpu_a.asm** **ftp-s_cfg.h** **dhcp-c_os.c** **net_bsp.c**

#define constants, **#define** macros, **enum** Data Types	Prefixed with the module identifier followed by an underscore ('_') All uppercase Words/tokens separated by an underscore ('_')	**DEF_OCTET_NBR_BITS** **DEF_BIT()** **NET_ARP_CACHE_STATE_RESOLVED**
typedef'd Data Types	Prefixed with the module identifier followed by an underscore ('_') All uppercase Words/tokens separated by an underscore ('_')	**CPU_INT16S** **NET_ARP_CACHE**
struct or **union** members	Camelback with each individual AAM word/token starting with an initial capital Acronyms followed by an underscore ('_') Pointers should be suffixed with **Ptr**	**CLK_DATE_TIME.DayOfYr** **NET_ARP_CACHE.HW_Addr** **NET_ARP_CACHE.TmrPtr**
Functions	Prefixed with the module identifier followed by an underscore ('_') Camelback with each individual AAM word/token starting with an initial capital Acronyms also followed by an underscore ('_')	**Clk_IsLeapYr()** **NetUtil_TS_Get()**
File-scope variables, Global variables	Prefixed with the module identifier followed by an underscore ('_') Camelback with each individual AAM word/token starting with an initial capital Acronyms also followed by an underscore ('_') Pointers should be suffixed with **Ptr** Tables should be suffixed with **Tbl**	**HTTPs_Secure** **Clk_TZ_sec** **NetOS_IF_RxQ_SignalPtr** **DHCPc_MsgTbl[]**
Local (function-scope) variables	All lowercase Words/tokens separated by an underscore ('_') Pointers should be prefixed by **p_**	**sock_id** **size_tot** **p_tcb**

Appropriateness

For all categories of identifiers, appropriate English-language names should be chosen for their given usage and context. Frivolous names; such as of pets, children, favorite sports teams, etc.; should never be used. Overly short or undescriptive names like **Tbl[]**, **Task()**, **MODULE_ERR**, or **p** should be avoided. Instead, specific and precise but concise names like **NetOS_TCP_RxQ_TimeoutTbl_tick[]**, **FTPs_CtrlTask()**, **CLK_OS_ERR_SIGNAL**, or **p_seg_hdr** should be used whenever possible. Exceptions for short names may include simple local variable loop counters, like **i** and **j**.

Uniqueness

All module names should be unique, and all file-scopes or global identifiers should be unique (MISRA 2004 Rules 5.5, 5.6, and 5.7). Local (function-scope) variables should not use the same name as (or 'hide') a file-scope or global identifier. Exceptions to this might include maintaining consist names in a product's device drivers as shown in Listing 1.

Also, identifiers used in standard librairies should NOT be re-used (MISRA 2004 Rules 20.1 and 20.2)

Listing 1 — µC/TCP-IP's Device Driver Template

```
/*
*********************************************************************************************************
*                                          LOCAL DATA TYPES
*
* Note(s) : (1) Device driver data types may be arbitrarily named.  However, it is recommended that device
*               driver data types be named using the names provided below.
*********************************************************************************************************
*/

typedef  struct  net_dev_data {
    MEM_POOL     RxDescPool;
    MEM_POOL     TxDescPool;
    DEV_DESC    *RxBufDescPtrStart;
    DEV_DESC    *RxBufDescPtrCur;
    DEV_DESC    *RxBufDescPtrEnd;
    DEV_DESC    *TxBufDescPtrStart;
    DEV_DESC    *TxBufDescPtrCur;
    DEV_DESC    *TxBufDescCompPtr;
    DEV_DESC    *TxBufDescPtrEnd;
    CPU_INT16U   RxNRdyCtr;
} NET_DEV_DATA;

typedef  struct  net_dev {
    CPU_REG32    CTRL;
    CPU_REG32    DMA_RX_START_ADDR;
    CPU_REG32    DMA_TX_START_ADDR;
    CPU_REG32    IAH;
    CPU_REG32    IAL;
    CPU_REG32    RESERVED1;
    CPU_REG32    RESERVED2;
    CPU_REG32    IER;
    CPU_REG32    ISR;
    CPU_REG32    RESERVED3;
} NET_DEV;
```

```
/*
*********************************************************************************************************
*                                     LOCAL FUNCTION PROTOTYPES
*
* Note(s) : (1) Device driver functions may be arbitrarily named.  However, it is recommended that device
*               driver functions be named using the names provided below.  All driver function prototypes
*               should be located within the driver C source file ('net_dev_&&&.c') & be declared as
*               static functions to prevent name clashes with other network protocol suite device drivers.
*********************************************************************************************************
*/

static  void  NetDev_Init               (NET_IF  *pif,
static  void  NetDev_Start              (NET_IF  *pif,
static  void  NetDev_Stop               (NET_IF  *pif,

static  void  NetDev_Rx                 (NET_IF  *pif,
static  void  NetDev_Tx                 (NET_IF  *pif,

static  void  NetDev_AddrMulticastAdd   (NET_IF  *pif,
static  void  NetDev_AddrMulticastRemove(NET_IF  *pif,

static  void  NetDev_ISR_Handler        (NET_IF  *pif,
static  void  NetDev_IO_Ctrl            (NET_IF  *pif,
```

Abbreviation

When appropriate, all name/identifier words and tokens should be abbreviated using the AAM dictionary.

Length

Names/identifiers should not have more than 31 non-unique significant characters (MISRA 2004 Rules 1.4 and 5.1). This does not mean that names cannot be longer than 31 characters, just that any two or more identifiers must NOT be identical in the first 31 characters. Any names/identifiers that exceed 31 significant characters should be abbreviated as shown in Listing 2 with an apppropriate note indicating for the abbreviation and reason.

Listing 2 — Examples of Identifiers Abbreviated to Maintain 31 Significant Character Uniqueness

```
/*
*********************************************************************************************************
*                                     LOCAL FUNCTION PROTOTYPES
*
* Note(s) : (1) NetTCP_RxPktConnHandlerState&&&() abbreviated to NetTCP_RxPktConnHandler&&&() to enforce
*               ANSI-compliance of 31-character symbol length uniqueness.
*********************************************************************************************************
*/

                                                                        /* See Note #1.                 */
static  void  NetTCP_RxPktConnHandlerListen   (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerSyncRxd  (NET_TCP_CONN  *pconn,                                     (1)
static  void  NetTCP_RxPktConnHandlerSyncTxd  (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerConn     (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerFinWait1 (NET_TCP_CONN  *pconn,                                     (2)
static  void  NetTCP_RxPktConnHandlerFinWait2 (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerClosing  (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerTimeWait (NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerCloseWait(NET_TCP_CONN  *pconn,
static  void  NetTCP_RxPktConnHandlerLastAck  (NET_TCP_CONN  *pconn,
```

(1) **NetTCP_RxPktConnHandlerSync???()** only significant within first 31 characters without **State** token.

(2) **NetTCP_RxPktConnHandlerFinWait?()** only significant within first 31 characters without **State** token.

Listing 3 — Examples of Identifiers Abbreviated to Maintain 31 Significant Character Uniqueness

```
/*
*********************************************************************************************************
*                                   TCP CONGESTION CONTROL DEFINES
*
* Note(s) : (2) (c) 'SLOW_START_TH' (i.e. 'Slow Start Threshold') abbreviated to 'SST' for some TCP control
*                   constants to enforce ANSI-compliance of 31-character symbol length uniqueness.
*********************************************************************************************************
*/

#define  NET_TCP_SST_UNACKD_DATA_NUMER                    1                                             (1)
#define  NET_TCP_SST_UNACKD_DATA_DENOM                    2

#define  NET_TCP_SST_MSS_SCALAR                           2
```

(1) **NET_TCP_SST_UNACKD_DATA_?????** are only 29 characters long but each unique within first 31 characters only if **SLOW_START_TH** abbreviated to **SST**.

Listing 4 — Examples of Identifiers Abbreviated to Maintain 31 Significant Character Uniqueness

```
/*
*********************************************************************************************************
*                                     NETWORK DEBUG STATUS DEFINES
*
* Note(s) : (2) 'STATUS_FAULT' abbreviated to 'SF' for some network debug status fault codes to enforce
*               ANSI-compliance of 31-character symbol length uniqueness.
*********************************************************************************************************
*/

#define  NET_DBG_SF_RSRC_LOST                 DEF_BIT_00    /* Net              rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_BUF             DEF_BIT_01    /* Net buf          rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_BUF_RX_LARGE    DEF_BIT_02    /* Net buf large rx rsrc(s) lost.            */ (1)
#define  NET_DBG_SF_RSRC_LOST_BUF_TX_LARGE    DEF_BIT_03    /* Net buf large tx rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_BUF_TX_SMALL    DEF_BIT_04    /* Net buf small tx rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_TMR             DEF_BIT_05    /* Net tmr          rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_CONN            DEF_BIT_06    /* Net conn         rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_ARP_CACHE       DEF_BIT_08    /* ARP cache        rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_TCP_CONN        DEF_BIT_10    /* TCP conn         rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_SOCK            DEF_BIT_11    /* Sock             rsrc(s) lost.            */
#define  NET_DBG_SF_RSRC_LOST_RESERVED        NET_DBG_SF_LOCK   /* Reserved; MUST NOT use (see Note #3a).*/
```

```
#define  NET_DBG_SF_RSRC_LO                   DEF_BIT_00    /* Net              rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_BUF               DEF_BIT_01    /* Net buf          rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_BUF_RX_LARGE      DEF_BIT_02    /* Net buf large rx rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_BUF_TX_LARGE      DEF_BIT_03    /* Net buf large tx rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_BUF_TX_SMALL      DEF_BIT_04    /* Net buf small tx rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_TMR               DEF_BIT_05    /* Net tmr          rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_CONN              DEF_BIT_06    /* Net conn         rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_ARP_CACHE         DEF_BIT_08    /* ARP cache        rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_TCP_CONN          DEF_BIT_10    /* TCP conn         rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_SOCK              DEF_BIT_11    /* Sock             rsrc(s) lo.              */
#define  NET_DBG_SF_RSRC_LO_RESERVED          NET_DBG_SF_LOCK   /* Reserved; MUST NOT use (see Note #3a).*/
```

(1) **NET_DBG_SF_RSRC_LOST_BUF_?X_?????** are 33 characters long but each unique within first 31 characters

Hierarchy

Whenever possible, names/identifiers should be created using hierarchical organization schemes as shown in Listing 5. As much as possible, use a consistent scheme when naming similar or related identifiers:

- Module-Object
- Module-Object-Operation
- Module-Object-Status
- Module-Object-Handler
- Module-Object-State-Handler
- Module-Status
- Module-Status-Object
- Module-Question-Object
- Module-Operation-Object

Listing 5 — Hierarchical Naming Examples

```
DHCPc_EXT  DHCPc_MSG    DHCPc_MsgTbl[DHCPc_NBR_MSG_BUF];
DHCPc_EXT  DHCPc_MSG   *DHCPc_MsgPoolPtr;
DHCPc_EXT  DHCPc_MSG   *DHCPc_MsgListHead;

DHCPc_EXT  DHCPc_COMM   DHCPc_CommTbl[DHCPc_NBR_COMM];                          (1)
DHCPc_EXT  DHCPc_COMM  *DHCPc_CommPoolPtr;
DHCPc_EXT  DHCPc_COMM  *DHCPc_CommListHead;

DHCPc_EXT  DHCPc_TMR    DHCPc_TmrTbl[DHCPc_NBR_TMR];
DHCPc_EXT  DHCPc_TMR   *DHCPc_TmrPoolPtr;
DHCPc_EXT  DHCPc_TMR   *DHCPc_TmrListHead;

static  void         DHCPc_MsgInit     (DHCPc_ERR       *perr);               (2)
static  void         DHCPc_MsgRxHandler(DHCPc_COMM      *pcomm);
static  DHCPc_MSG   *DHCPc_MsgGet      (DHCPc_ERR       *perr);
static  CPU_INT08U  *DHCPc_MsgGetOpt   (DHCPc_OPT_CODE   opt_code,
static  void         DHCPc_MsgFree     (DHCPc_MSG       *pmsg);
static  void         DHCPc_MsgClr      (DHCPc_MSG       *pmsg);

static  void         DHCPc_CommInit    (DHCPc_ERR       *perr);
static  DHCPc_COMM  *DHCPc_CommGet     (NET_IF_NBR       if_nbr,
static  void         DHCPc_CommFree    (DHCPc_COMM      *pcomm);
static  void         DHCPc_CommClr     (DHCPc_COMM      *pcomm);

static  void         DHCPc_TmrInit     (DHCPc_ERR       *perr);
static  void         DHCPc_TmrCfg      (DHCPc_IF_INFO   *pif_info,
static  DHCPc_TMR   *DHCPc_TmrGet      (void             *pobj,
static  void         DHCPc_TmrFree     (DHCPc_TMR       *ptmr);
static  void         DHCPc_TmrClr      (DHCPc_TMR       *ptmr);
```

```
CPU_BOOLEAN  NetIP_GetAddrHost          (NET_IF_NBR   if_nbr,
NET_IP_ADDR  NetIP_GetAddrHostCfgd      (NET_IP_ADDR  addr_remote);             (3)
NET_IP_ADDR  NetIP_GetAddrSubnetMask    (NET_IP_ADDR  addr,
NET_IP_ADDR  NetIP_GetAddrDfltGateway   (NET_IP_ADDR  addr,
NET_IF_NBR   NetIP_GetAddrHostIF_Nbr    (NET_IP_ADDR  addr);
NET_IF_NBR   NetIP_GetAddrHostCfgdIF_Nbr(NET_IP_ADDR  addr);

CPU_BOOLEAN  NetIP_IsAddrClassA         (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrClassB         (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrClassC         (NET_IP_ADDR  addr);                    (4)
CPU_BOOLEAN  NetIP_IsAddrClassD         (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrThisHost       (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrLocalHost      (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrLocalLink      (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrBroadcast      (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrMulticast      (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrHost           (NET_IP_ADDR  addr);                    (5)
CPU_BOOLEAN  NetIP_IsAddrHostCfgd       (NET_IP_ADDR  addr);
CPU_BOOLEAN  NetIP_IsAddrsCfgdOnIF      (NET_IF_NBR   if_nbr,

CPU_BOOLEAN  NetIP_IsValidAddrHost      (NET_IP_ADDR  addr_host);
CPU_BOOLEAN  NetIP_IsValidAddrHostCfgd  (NET_IP_ADDR  addr_host,
CPU_BOOLEAN  NetIP_IsValidAddrSubnetMask(NET_IP_ADDR  addr_subnet_mask);
```

```
#if (NET_SOCK_CFG_FAMILY == NET_SOCK_FAMILY_IP_V4)
    NET_SOCK_ADDR_IP      *paddr_ip;
    NET_IP_ADDR            addr_ip_host;                                        (6)
    NET_IP_ADDR            addr_ip_net;
    NET_IP_ADDR            addr_ip_tbl[NET_IP_CFG_IF_MAX_NBR_ADDR];
    NET_IP_ADDRS_QTY       addr_ip_tbl_qty;
#endif
    CPU_BOOLEAN            valid;
    CPU_BOOLEAN            conn_avail;
    CPU_BOOLEAN            addr_over_wr;
    CPU_BOOLEAN            port_random_reqd;
    NET_PORT_NBR           port_nbr_host;
    NET_PORT_NBR           port_nbr_net;
    CPU_INT08U             addr_local[NET_SOCK_CFG_ADDR_LEN];                   (7)
    CPU_INT08U             addr_remote[NET_SOCK_CFG_ADDR_LEN];
    NET_CONN_ADDR_LEN      addr_remote_len;
    CPU_INT08U            *paddr_remote;
    NET_SOCK              *psock;
    NET_SOCK_STATE         sock_state;
    NET_CONN_FAMILY        conn_family;
    NET_CONN_PROTOCOL_IX   conn_protocol_ix;
    NET_CONN_ID            conn_id;
    NET_CONN_ID            conn_id_srch;
```

(1) µC/DHCPc's module-object naming scheme ...

(2) ... parallels its module-object-operation function naming scheme

(3) µC/TCP-IP's IP address functions organized as **NetIP_GetAddr()** and ...

(4) ... **NetIP_IsAddr()** functions since

(5) ... other **NetIP_Is???()** functions available:

(6) **addr_ip** vs. **ip_addr** to parallel ...

(7) ... other **addr** variables

File-scope and global identifiers should be prefixed by their module identifier first. This prefix makes it easy to know which modules identifiers belong to, i.e. where they are declared and defined in. For example, the following could be potential **`keyboard.c`** and **`video.c`** identifiers:

Listing 6 — Example **`keyboard.c`** *Identifiers*

```
Kbd_GetChar()
Kbd_GetLine()
Kbd_GetFnctKey()

static  CPU_CHAR  Kbd_CharBuf[];
static  CPU_CHAR  Kbd_CharLast;

KBD_CHAR_BUF_SIZE
KBD_DEBOUNCE_DLY_MS
```

Listing 7 — Example `video.c` Identifiers

```
Vid_PutChar()
Vid_PutStr()
Vid_AttribGet()
Vid_AttribSet()

typedef  struct  vid_pixel  VID_PIXEL;

VID_EXT  VID_PIXEL  Vid_Tbl[VID_CFG_X_SIZE]
                           [VID_CFG_Y_SIZE];
```

Note that a consistent scheme in one module or section of a module might not apply to another module or section. On the other hand, a naming scheme or set of schemes could possibly be consistent across many modules. One of µC/TCP-IP's naming hierarchy divides each protocol layer into Receive ('**`Rx`**') and Transmit ('**`Tx`**') functions. From this base naming scheme, related functions, error codes, error and statistics counters, and objects, in each protocol layer were similarly named as shown in Listing 8.

Listing 8 — µC/TCP-IP's `Rx/Tx` Naming Scheme

IP	UDP	TCP
NetIP_Rx()	NetUDP_Rx()	NetTCP_Rx()
NetIP_RxPktValidateBuf()	NetUDP_RxPktValidateBuf()	NetTCP_RxPktValidateBuf()
NetIP_RxPktValidate()	NetUDP_RxPktValidate()	NetTCP_RxPktValidate()
NetIP_RxPktValidateOpt()		NetTCP_RxPktValidateOpt()
	NetUDP_RxPktFree()	NetTCP_RxPktFree()
NetIP_RxPktDiscard()	NetUDP_RxPktDiscard()	NetTCP_RxPktDiscard()
NetIP_Tx()	NetUDP_Tx()	
NetIP_TxPkt()		NetTCP_TxPkt()
NetIP_TxPktValidate()	NetUDP_TxPktValidate()	NetTCP_TxPktValidate()
NetIP_TxPktValidateOpt()		NetTCP_TxPktValidateOpt()
NetIP_TxPktPrepareHdr()	NetUDP_TxPktPrepareHdr()	NetTCP_TxPktPrepareHdr()
	NetUDP_TxPktFree()	NetTCP_TxPktFree()
NetIP_TxPktDiscard()	NetUDP_TxPktDiscard()	NetTCP_TxPktDiscard()
NET_IP_ERR_RX_???	NET_UDP_ERR_RX_???	NET_TCP_ERR_RX_???
NET_IP_ERR_TX_???	NET_UDP_ERR_TX_???	NET_TCP_ERR_TX_???
NetIP_ErrRxInvalidProtocolCtr	NetUDP_ErrRxInvalidProtocolCtr	NetTCP_ErrRxInvalidProtocolCtr
NetIP_ErrRxInvalidBufIxCtr	NetUDP_ErrRxInvalidBufIxCtr	NetTCP_ErrRxInvalidBufIxCtr
NetIP_ErrRxHdrTotLenCtr	NetUDP_ErrRxHdrDatagramLenCtr	NetTCP_ErrRxHdrSegLenCtr
NetIP_ErrRxPktDiscardedCtr	NetUDP_ErrRxPktDiscardedCtr	NetTCP_ErrRxPktDiscardedCtr
NetIP_ErrTxInvalidProtocolCtr	NetUDP_ErrTxInvalidProtocolCtr	NetTCP_ErrTxInvalidProtocolCtr
NetIP_ErrTxInvalidBufIxCtr	NetUDP_ErrTxInvalidBufIxCtr	NetTCP_ErrTxInvalidBufIxCtr
NetIP_ErrTxHdrDataLenCtr	NetUDP_ErrTxHdrDataLenCtr	NetTCP_ErrRxHdrDataLenCtr
NetIP_ErrTxPktDiscardedCtr	NetUDP_ErrTxPktDiscardedCtr	NetTCP_ErrTxPktDiscardedCtr
NetIP_StatRxPktCtr	NetUDP_StatRxPktCtr	NetTCP_StatRxPktCtr
NetIP_StatRxDatagramProcessedCtr	NetUDP_StatRxDatagramProcessedCtr	NetTCP_StatRxSegProcessedCtr
NetIP_StatTxDatagramCtr	NetUDP_StatTxDatagramCtr	NetTCP_StatTxSegCtr
NetIP_IF_CfgTbl[]		NetTCP_ConnTbl[]
		NetTCP_ConnPoolPtr
		NetTCP_ConnPoolStat

Units

Identifiers may be suffixed with appropriate units as shown in Listing 9.

Listing 9 — Examples of Identifier Units

```
NetOS_TCP_RxQ_TimeoutGet_ms()
NetOS_TCP_RxQ_TimeoutTbl_tick[]

TxRTT_RTO_sec
TxRTT_RTO_Max_sec

CPU_IntDisMeasStop_cnts
CPU_IntDisMeasStart_cnts

NetTCP_CfgRxWinSize_octet
NetTCP_CfgTxWinSize_octet

NetARP_CacheFoundCtr_hi
NetARP_CacheFoundCtr_lo

NetDbg_RsrcBufThLo_pct
NetDbg_RsrcBufThLoHyst_pct
```

Listing 10 — Uppercase Units Exception

```
NET_APP_TIME_DLY_MIN_SEC
NET_APP_TIME_DLY_MAX_SEC
```

versus

```
NET_APP_TIME_DLY_MIN_mS
NET_APP_TIME_DLY_MAX_mS
```

Listing 11 — `ms`, `us`, and `ns` Units

```
NetOS_TimeoutMin_ms
NetOS_TimeoutMax_ms
NetOS_TimeoutMin_us
NetOS_TimeoutMax_us

NET_APP_TIME_DLY_MIN_mS
NET_APP_TIME_DLY_MAX_mS

DEF_TIME_NBR_mS_PER_SEC
DEF_TIME_NBR_uS_PER_SEC
DEF_TIME_NBR_nS_PER_SEC
```

Identifiers

Files

Filenames should be prefixed with their module identifier followed by an underscore ('_'). All characters should be lowercase, with each individual word/token separated by an underscore ('_'). Filenames should be AAM abbreviated but may use non-abbreviated words/tokens, when appropriate. Assembly files' non-extension filename should be suffixed with **`_a`**; configuration filenames should be suffixed with **`_cfg`**; OS port filenames should be suffixed with **`_os`**; BSP port filenames should be suffixed with **`_bsp`**.

Listing 12 — Filename Examples

```
lib_str.c

os_task.c

net_if_loopback.h

cpu_a.asm

ftp-s_cfg.h

dhcp-c_os.c

net_bsp.c
```

`#define`'s and `enum` Data Types

`#define`'s and **`enum`** data types should be prefixed with their module identifier followed by an underscore ('_'). All characters should be uppercase, with each individual word/token separated by an underscore ('_').

Listing 13 — `#define` and `enum` Examples

```
#define  DEF_OCTET_NBR_BITS                 8

#define  DEF_BIT(bit)             (1uL << (bit))

typedef  enum  net_arp_cache_state {
    NET_ARP_CACHE_STATE_NONE,
    NET_ARP_CACHE_STATE_FREE,
    NET_ARP_CACHE_STATE_PEND,
    NET_ARP_CACHE_STATE_RESOLVED
} NET_ARP_CACHE_STATE;
```

`typedef`'d Data Types

`typedef`'d data types should be prefixed with their module identifier followed by an underscore ('_'). All characters should be uppercase, with each individual word/token separated by an underscore ('_').

Listing 14 — `typedef` Examples

```
typedef  signed  short  CPU_INT16S;

typedef  struct  net_arp_cache  NET_ARP_CACHE;
```

`struct` or `union` Members

`struct` and **`union`** members should be camelback-case with each individual AAM word/token starting with an initial capital. However, any AAM word/token (except the identifier's last word/token) that is all-caps (e.g., acronyms) should be followed by an underscore ('_'). If a member variable is a pointer to another variable or memory location, it should be suffixed with **`Ptr`**.

Listing 15 — `struct` Examples

```
typedef  struct  clk_date_time {
    CLK_YR      Yr;                                         /* Yr since epoch start yr      (see Note #1).      */
    CLK_MONTH   Month;                                      /* Month     [     1 to    12], (Jan to Dec).       */
    CLK_DAY     Day;                                        /* Day       [     1 to    31].                     */
    CLK_DAY     DayOfWk;                                    /* Day of wk [     1 to     7], (Sun to Sat).       */
    CLK_DAY     DayOfYr;                                    /* Day of yr [     1 to   366].                     */
    CLK_HR      Hr;                                         /* Hr        [     0 to    23].                     */
    CLK_MIN     Min;                                        /* Min       [     0 to    59].                     */
    CLK_SEC     Sec;                                        /* Sec       [     0 to    60], (see Note #2).      */
    CLK_TZ_SEC  TZ_sec;                                     /* TZ        [-43200 to 43200], (see Note #3).      */
} CLK_DATE_TIME;


typedef  struct  net_arp_cache {
    NET_TYPE              Type;                                     /* Type cfg'd @ init : NET_ARP_TYPE_CACHE.  */
    NET_ARP_CACHE        *PrevPtr;                                  /* Ptr to PREV ARP cache.                   */
    NET_ARP_CACHE        *NextPtr;                                  /* Ptr to NEXT ARP cache.                   */
    NET_BUF              *BufQ_Head;                                /* Ptr to head of cache's buf Q.            */
    NET_BUF              *BufQ_Tail;                                /* Ptr to tail of cache's buf Q.            */
    NET_TMR              *TmrPtr;                                   /* Ptr to cache TMR.                        */
    NET_IF_NBR            IF_Nbr;                                   /* IF nbr of ARP cache addr.                */
    CPU_INT08U            HW_Addr[NET_ARP_CFG_HW_ADDR_LEN];         /* Remote hw       addr.                    */
    CPU_INT08U            ProtocolAddr[NET_ARP_CFG_PROTOCOL_ADDR_LEN];  /* Remote protocol addr.                */
    NET_ARP_CACHE_STATE   State;                                    /* ARP cache state.                         */
    CPU_INT16U            Flags;                                    /* ARP cache flags.                         */
} NET_ARP_CACHE;
```

Functions

Local and global functions should be prefixed with their module identifier followed by an underscore ('_') and should be camelback-case with each individual AAM word/token starting with an initial capital. However, any AAM word/token (except the identifier's last word/token) that is all-caps (e.g., acronyms) should be followed by an underscore ('_').

Listing 16 — Function Examples

```
static  CPU_BOOLEAN  Clk_IsLeapYr (CLK_YR  yr)

NET_TS  NetUtil_TS_Get (void)
```

File-scope and Global Variables

File-scope variables and global variables should be prefixed with their module identifier followed by an underscore ('_') and should be camelback-case with each individual AAM word/token starting with an initial capital. However, any AAM word/token (except the identifier's last word/token) that is all-caps (e.g., acronyms) should be followed by an underscore ('_'). If a variable is a pointer to another variable or memory location, it should be suffixed with **`Ptr`**. If a variable is a table, it should be suffixed with **`Tbl`**.

Listing 17 — File-scope and Global Variables Examples

```
static  CPU_BOOLEAN   HTTPs_Secure;

static  CLK_TZ_SEC  Clk_TZ_sec;

NET_OS_EXT  OS_EVENT  *NetOS_IF_RxQ_SignalPtr;

DHCPc_EXT  DHCPc_MSG  DHCPc_MsgTbl[DHCPc_NBR_MSG_BUF];
```

Local (Function-Scope) Variables

Local (function-scope) variables should be all lowercase, with each individual word/token separated by an underscore ('_'). If a variable is a pointer to another variable or a memory location, it should be prefixed by a **`p_`**. Industry-standard loop counter variables (**i**, **j**, **k**, etc.) may be used.

Listing 18 — Local Variables Examples

```
NET_SOCK_ID  sock_id;

OS_TCB  *p_tcb;

CPU_SIZE_T  size_tot;
CPU_SIZE_T  i;
```

Operators

Unary Operators

Unary operators should be formatted as shown in Listing 1. Bit-wise negation operator, **~**, operands **should** be cast to desired data size (MISRA 2004 Rule 10.5), whenever possible. It is important to note that this MISRA rule has detractors, because it could lead to hiding useful compiler warnings.

Listing 1 — Unary Operators Format

```
const  CPU_INT16S  MinVal  =   -1;                                          (1)
const  CPU_INT16S  MaxVal  = +100;                                         (2)

neg_val = -val;                                                            (3)

#define  DEF_ABS(a)        (((a) < 0) ? (-(a)) : (a))                      (4)
```

```
toggle = !toggle;                                                          (5)

if ((flags_tcp & ~flag_mask) != NET_TCP_FLAG_NONE) {

flags_rdy = (OS_FLAGS)(~p_grp->Flags & flags);                             (6)

#if (!( (DEF_TIME_NBR_mS_PER_SEC >= OS_TICKS_PER_SEC) &&    \
       ((DEF_TIME_NBR_mS_PER_SEC %  OS_TICKS_PER_SEC) == 0)))

                                                                           (7)
*pmem_08++  = data_val;                                                    (8)
(*pctr)++;                                                                 (9)

 *pstr_fmt-- = (CPU_CHAR)(dig_val + '0');                                  (10)

for (i = nbr_dig; i > 0; i--) {                                            (11)
   retry_cnt++;                                                            (12)
}
```

```
do {
    *(++p_mem) = 0;                                                        (13)
} while (--i > 0);
                                                                           (14)
p_mem++;
while (i-- > 0) {
   *p_mem  = 0;
    p_mem++;
}
```

```
if ((addr_host             & (NET_IP_ADDR)~addr_subnet_mask) ==            (15)
    (NET_IP_ADDR_BROADCAST & (NET_IP_ADDR)~addr_subnet_mask)) {            (16)
```

(1) Prefix unary operators should immediately prefix ...

(2) ... constant,

(3) ... variable,

(4) ... or expression

(5) At least one space before each prefix unary operator ...

(6) ... unless preceded by a parenthesis

(7) Postfix unary operators should immediately follow ...

(8) ... variable

(9) ... or expression

(10) At least one space after each postfix unary operator ...

(11) ... unless followed by a parenthesis

(12) ... or semicolon

(13) Pre-increment operators discouraged :

(14) Use post-increment operators whenever possible :

(15) ~ operands SHOULD be cast to desired data size

(16) Align operators on multiple lines whenever possible

Shift Operators

Bit-wise shift operators should be formatted as shown in Listing 2. Bit-wise shift operands **should** be cast to desired data size (MISRA 2004 Rule 10.5), whenever possible. It is important to note that this MISRA rule has detractors, because it could lead to hiding useful compiler warnings.

Listing 2 — Shift Operators Format

```
while (sum >> 16u) {                                                                                    (1)
    sum = (sum & 0x0000FFFFu) + (sum >> 16u);
}
```

```
#define  NET_UTIL_NET_TO_HOST_32(val)  (((CPU_INT32U)(((CPU_INT32U)(val)) & 0xFF000000uL) >> 24u) | \   (2)
                                         ((CPU_INT32U)(((CPU_INT32U)(val)) & 0x00FF0000uL) >>  8u) | \   (3)
                                         ((CPU_INT32U)(((CPU_INT32U)(val)) & 0x0000FF00uL) <<  8u) | \
                                         ((CPU_INT32U)(((CPU_INT32U)(val)) & 0x000000FFuL) << 24u))

#define  NET_IP_ADDR_LOCAL_HOST_ADDR   (((NET_IP_ADDR)127uL << (3u * DEF_OCTET_NBR_BITS)) | \           (4)
                                         ((NET_IP_ADDR)  0uL << (2u * DEF_OCTET_NBR_BITS)) | \
                                         ((NET_IP_ADDR)  0uL << (1u * DEF_OCTET_NBR_BITS)) | \
                                         ((NET_IP_ADDR)  1uL << (0u * DEF_OCTET_NBR_BITS)))
```

(1) At least one space before and after each operator.

(2) **<</>>** operands SHOULD be cast to desired data size.

(3) Align operators on multiple lines whenever possible.

(4) **<<** operands SHOULD be cast to desired data size.

Arithmetic Operators

Arithmetic operators should be formatted as shown in Listing 3. To improve clarity, arithmetic expressions may include additional parentheses, spacing, and/or be formatted on multiple lines. When formatted on multiple lines, arithmetic operators should consistently prefix or follow each continued line in an arithmetic expression or group of related expressions. Some operations need to be ordered to evaluate in a specific sequence to ensure no loss or data through overflow or underflow. For example, integer multiplication should be performed before division as shown in Listing 4. This example assumes that the multiplication does not or cannot overflow. If the multiplication could overflow, then a larger intermediate data type or a another rearrangement of the equation may be required.

Listing 3 — Arithmetic Operators Format

```
rand_nbr = (((RAND_NBR)RAND_LCG_PARAM_A * seed) + (RAND_NBR)RAND_LCG_PARAM_B)  %  ((RAND_NBR)RAND_LCG_PARAM_M + 1u);      (1)
```

```
ts_ms_delta_num      = (NET_TS_MS)(os_tick_delta   * DEF_TIME_NBR_mS_PER_SEC);                                          (2)
ts_ms_delta          = (NET_TS_MS)(ts_ms_delta_num / OS_TICKS_PER_SEC);
ts_ms_tot           += (NET_TS_MS) ts_ms_delta;
ts_ms_delta_rem      = (NET_TS_MS)(ts_ms_delta_num % OS_TICKS_PER_SEC);
ts_ms_delta_rem_tot +=  ts_ms_delta_rem;
ts_ms_delta_rem_ovf  =  ts_ms_delta_rem_tot / OS_TICKS_PER_SEC;
ts_ms_delta_rem_tot -=  ts_ms_delta_rem_ovf * OS_TICKS_PER_SEC;
ts_ms_tot           +=  ts_ms_delta_rem_ovf;

CPU_IntDisMeasOvrhd_cnts  = (time_meas_tot_cnts + (CPU_CFG_INT_DIS_MEAS_OVRHD_NBR / 2))
                                              /  CPU_CFG_INT_DIS_MEAS_OVRHD_NBR;


timeout_ms_sec =   timeout_sec_tot * DEF_TIME_NBR_mS_PER_SEC;
timeout_ms_us  = ((timeout_us      % DEF_TIME_NBR_uS_PER_SEC) + ((DEF_TIME_NBR_uS_PER_SEC / DEF_TIME_NBR_mS_PER_SEC) - 1))
                                                             /  (DEF_TIME_NBR_uS_PER_SEC / DEF_TIME_NBR_mS_PER_SEC);
timeout_ms_tot =   timeout_ms_sec  + timeout_ms_us;


const  CPU_INT32U  NetTmr_DataSize   =  0u
                                     +  sizeof(NetTmr_Tbl)
                                     +  sizeof(NetTmr_PoolPtr)
                                     +  sizeof(NetTmr_PoolStat)
                                     +  sizeof(NetTmr_TaskListHead)
                                     +  sizeof(NetTmr_TaskListPtr)
                                     ;



nbr_bufs_tot = pdev_cfg->RxBufLargeNbr +
               pdev_cfg->TxBufLargeNbr +                                                                               (3)
               pdev_cfg->TxBufSmallNbr;

#define  NET_OS_TIME_DLY_MAX_SEC    ((DEF_INT_08U_MAX_VAL           * DEF_TIME_NBR_SEC_PER_HR ) + \
                                    ((DEF_TIME_NBR_MIN_PER_HR  - 1) * DEF_TIME_NBR_SEC_PER_MIN) + \
                                     (DEF_TIME_NBR_SEC_PER_MIN - 1))
```

```
if (hex_lower_case != DEF_YES) {                                                                                       (4)
   *pchar++ = (CPU_CHAR)((addr_octet_dig_val - 10u) + 'A');
} else {
   *pchar++ = (CPU_CHAR)((addr_octet_dig_val - 10u) + 'a');
}
```

(1) At least one space before and after each operator

(2) Align operators on multiple lines whenever possible.

(3) Arithmetic operations on multiple lines should have operators consistently prefix continued lines ...

(4) ... or consistently follow each continued line.

(5) Some analysis tools warn against mixed arithmetic; use parentheses to more obviously indicate order of operator evaluation.

Listing 4 — Multiplication and Division Order of Evaluation Example

```
pct = (CPU_INT08U)((pct_numer_lo * 100) / pct_denom_lo);
```

Assignment Operators

Assignment operators should be formatted as shown in Listing 5. The equal signs in a group of consecutive, related assignments should be vertically aligned, and may also be aligned with any related comparison operators. The equal signs for consecutive but unrelated groups of assignments should be vertically aligned, whenever possible, but are not required to be, as shown in Listing 6. However, whether aligned or not, different groups of unrelated assignments(, operators, or statements) should be separated by one or more blank lines.

Listing 5 — Assignment Operators Format

```
ix_align  = data_ix    + ix_offset;                                          (1)
ix_align %= data_size;
ix_align += data_size  - addr_offset;                                        (2)
ix_align %= data_size;
```

```
ip_ver       = NET_IP_HDR_VER;                                               (3)
ip_ver     <<= NET_IP_HDR_VER_SHIFT;
ip_hdr_len   = pbuf_hdr->IP_HdrLen / NET_IP_HDR_LEN_WORD_SIZE;
ip_hdr_len  &= NET_IP_HDR_LEN_MASK;
```

```
while (size_rem >=  sizeof(CPU_INT16U)) {                                    (4)
    sum_val_32   = (CPU_INT32U)*phdr_08++;
    sum_val_32 <<=  DEF_OCTET_NBR_BITS;
    sum_val_32  += (CPU_INT32U)*phdr_08++;
    sum_32      += (CPU_INT32U) sum_val_32;
    size_rem    -= (CPU_INT16U) sizeof(CPU_INT16U);
}
```

```
nbr     *= nbr_base;
nbr     += parse_dig;
nbr_fmt /= nbr_base;
```

```
if (pct_numer_hi &   NET_CTR_BIT_LO) {
    pct_numer_lo |=  NET_CTR_BIT_HI;
} else {
    pct_numer_lo &= ~NET_CTR_BIT_HI;                                         (5)
}

for (j = 0; j < DEF_OCTET_NBR_BITS; j++) {
    bit_nbr    = (j * 6) + i;
    octet_nbr  =  bit_nbr / DEF_OCTET_NBR_BITS;
    octet      =  paddr_hw[octet_nbr];
    bit        =  octet & (1 << (bit_nbr % DEF_OCTET_NBR_BITS));             (6)
    bit_val   ^= (bit > 0) ? 1 : 0;
}
```

(1) At least one space before and after each operator.

(2) Align operators on multiple lines whenever possible.

(3) Whenever possible for operators on multiple lines, vertically align right-side equal signs.

(4) Align comparison operators with related assignment operators whenever possible.

(5) Right-hand side offset by unary ~ operator.

(6) Use parentheses to more obviously indicate order of operator evaluation.

Listing 6 — Consecutive Groups of Assignments

```
seq_nbr   = pconn->TxSeqNbrNext;                                                (1)
ack_nbr   = pconn->RxSeqNbrNext - ack_nbr_delta;

flags_tcp = NET_TCP_FLAG_NONE  |                                                (2)
            NET_TCP_FLAG_TX_ACK;

win_size  = pconn->RxWinSizeActual;

TOS       = pconn->TxIP_TOS;                                                    (3)
TTL       = pconn->TxIP_TTL;
flags_ip  = pconn->TxIP_Flags;


                                                                                (4)
pseg_ack_hdr                           = &pseg_ack->Hdr;
pseg_ack_hdr->DataIx                   = (CPU_INT16U  )data_ix;
pseg_ack_hdr->DataLen                  = (NET_BUF_SIZE)data_len;
pseg_ack_hdr->TotLen                   = (NET_BUF_SIZE)pseg_ack_hdr->DataLen;
pseg_ack_hdr->ProtocolHdrType          =  NET_PROTOCOL_TYPE_TCP;
pseg_ack_hdr->ProtocolHdrTypeTransport =  NET_PROTOCOL_TYPE_TCP;
                                                                                (5)
pseg_ack_hdr->TCP_SegSync              = (CPU_BOOLEAN )DEF_NO;
pseg_ack_hdr->TCP_SegClose             = (CPU_BOOLEAN )DEF_NO;
pseg_ack_hdr->TCP_SegAck               = (CPU_BOOLEAN )DEF_YES;
pseg_ack_hdr->TCP_SegReset             = (CPU_BOOLEAN )DEF_NO;

pseg_ack_hdr->TCP_Flags                =  flags_tcp;
```

(1) Assignments in a related group should **NOT** be separated by any blank lines.

(2) Similar, related groups of consecutive assignments should be separated by (at least) one blank line.

(3) The upper four groups of assignments' equal signs are aligned since their left-hand side expressions are very similar in length.

(4) Unrelated groups of consecutive assignments should be separated by (at least) two blank lines.

(5) The equal signs in the lower groups are NOT aligned with the upper groups of assignments because the left-hand side expressions are much longer.

Comparison Operators

Comparison, or equality and relational, operators should be formatted as shown in Listing 7. The equal signs in a group of related comparisons should be vertically aligned, whenever possible; and may also be aligned with any related assignment operators as shown in Listing 8. However, if a related group includes only single-character comparison and assignment operators, the single character operators should be aligned vertically as shown in Listing 9, whenever possible.

Listing 7 — Comparison Operators Format

```
if ((time_dly_sec < 1) &&                                                    (1)
    (time_dly_us  < 1)) {
```

```
if (((reg_id1 == 0x0000u) && (reg_id2 == 0x0000u)) ||                        (2)
    ((reg_id1 == 0x3FFFu) && (reg_id2 == 0x0000u)) ||
    ((reg_id1 == 0x0000u) && (reg_id2 == 0x3FFFu)) ||
    ((reg_id1 == 0x3FFFu) && (reg_id2 == 0x3FFFu)) ||
    ((reg_id1 == 0xFFFFu) && (reg_id2 == 0x0000u)) ||
    ((reg_id1 == 0x0000u) && (reg_id2 == 0xFFFFu)) ||
    ((reg_id1 == 0x3FFFu) && (reg_id2 == 0xFFFFu)) ||
    ((reg_id1 == 0xFFFFu) && (reg_id2 == 0xFFFFu))) {

ack_dup = ((pbuf_hdr->TCP_SeqNbr  == pconn->TxWinRxdLastSeqNbr ) &&
           (pbuf_hdr->TCP_AckNbr  == pconn->TxWinRxdLastAckNbr ) &&
           (pbuf_hdr->TCP_WinSize == pconn->TxWinRxdLastWinSize) &&
           (pbuf_hdr->TCP_SegLen  == 0                         )) ? DEF_YES : DEF_NO;

while ((rx_len_tot <  rx_th_actual) &&                                       (3)
       (retry_cnt  <= retry_max)    &&
       (done       == DEF_NO)) {

while (( pstr_copy_dest != (      CPU_CHAR *)  0 ) &&
       ( pstr_copy_src  != (const CPU_CHAR *)  0 ) &&
       (*pstr_copy_src  != (      CPU_CHAR  )'\0') &&
       ( len_copy       <  (      CPU_SIZE_T)len_max)) {
```

(1) At least one space before and after each operator

(2) Whenever possible for operators on multiple lines, vertically align right-side equal signs.

(3) Align comparison operators without equal signs to the left, as if they included an equal sign.

Listing 8 — Comparison/Assignment Operators Alignment

```
if (pconn->RxWinSizeCfgdActual >  win_size_avail) {                          (1)
    pconn->RxWinSizeCfgdActual -= win_size_avail;
} else {
    pconn->RxWinSizeCfgdActual  = 0u;
}
```

```
if (data_size >= data_size_rem) {                                            (2)
    data_size -= data_size_rem;
}
```

(1) Align comparison operators without equal signs to the left, as if they included an equal sign.

(2) **>** or **>=** both valid but one might be more efficient than the other to decrement size.

Listing 9 — Comparison/Assignment Operators Alignment Exception

```
if (pconn->RxWinSizeCalcd              < pconn->RxWinSizeCfgd) {             (1)
    pconn->RxWinSizeCalcd              = pconn->RxWinSizeCfgd;
    pconn->RxWinSizeCfgdActual         = pconn->RxWinSizeCfgd;
    pconn->RxWinSizeCfgdActualUpdateRem = 0u;
}
```

(1) Align single-character comparison and assignment operators vertically.

Logical Operators

Logical operators should be formatted as shown in Listing 10. When formatted on multiple lines, logical operators should follow each continued line in an logical expression.

Listing 10 — Logical Operators Format

```
if ((addr_octet_dig_nbr >  0) ||                                                                                    (1)
    (base_10_val        == 1) ||
    (lead_zeros == DEF_YES)) {
```

```
#if ((NET_TCP_MAX_SEG_SIZE_MAX < DEF_INT_08U_MAX_VAL)                                                          || \  (2)
    ((NET_TCP_MAX_SEG_SIZE_MAX < DEF_INT_16U_MAX_VAL) && (NET_TCP_MAX_SEG_SIZE_MAX > DEF_INT_08U_MAX_VAL)) || \
    ((NET_TCP_MAX_SEG_SIZE_MAX < DEF_INT_32U_MAX_VAL) && (NET_TCP_MAX_SEG_SIZE_MAX > DEF_INT_16U_MAX_VAL)))

while (( pstr_copy_dest != (      CPU_CHAR *)  0 ) &&                                                           (3)
       ( pstr_copy_src  != (const CPU_CHAR *)  0 ) &&
       (*pstr_copy_src  != (      CPU_CHAR  )'\0') &&
       ( len_copy       <  (      CPU_SIZE_T)len_max)) {

flags_tcp = NET_TCP_FLAG_NONE    |
            NET_TCP_FLAG_TX_ACK  |
            NET_TCP_FLAG_TX_CLOSE;



if ((pbuf_hdr->IP_AddrSrc  & NET_IP_ADDR_LOCAL_HOST_MASK_HOST) ==
    (NET_IP_ADDR_THIS_HOST & NET_IP_ADDR_LOCAL_HOST_MASK_HOST)) {                                              (4)

pkt_aligned_16 = (((modulo_16 == 0u) && (prev_octet_valid == DEF_NO )) ||
                  ((modulo_16 != 0u) && (prev_octet_valid == DEF_YES))) ? DEF_YES : DEF_NO;

#define  DEF_CHK_VAL_MIN(val, val_min)   ((((!((val)     >= 0) && ((val_min) < 0))) && \
                                             ((((val_min) >= 0) && ((val)     < 0))  || \
                                               ((val) < (val_min)))) ? DEF_FAIL : DEF_OK)
```

(1) At least one space before and after each operator.

(2) Align operators on multiple lines whenever possible.

(3) Logical operations on multiple lines should have operators follow each continued line; ...

(4) ... but logical operators may be used elsewhere in expressions, not just at the end of each line.

`!` and `~` (Negation) Operators

See Unary Operators.

`,` (Comma) Operator

Use of the comma operator is highly discouraged (MISRA 2004 Rule 12.10) [see also Simple Code Statements]. However, multiple simple control expressions in **`for`** loops may be acceptable.

`?` (Conditional) Operator

Although anecdotally some compilers may generate poor code for the **`?`** conditional operator, it provides a compact form, as shown in Listing 11, for evaluating a simple **`if...else`** statement as shown in Listing 12. This can be convenient and improve clarity especially when grouped with many other assignment statements as shown in Listing 13.

? operator statements should be formatted as shown in Listing 14. The conditional expression must always be enclosed in parentheses but resultant expressions need only be enclosed in parentheses to improve clarity for complex expressions. Lastly, most Comparison and Context Rules should also apply for **?** operator statements when applicable.

Listing 11 — Simple **?** *Example*

```
mem_align_offset = (align_offset != 0) ? blk_align - align_offset : 0;
```

versus

Listing 12 — Simple **if...else** *Example*

```
if (align_offset != 0) {
    mem_align_offset = blk_align - align_offset;
} else {
    mem_align_offset = 0;
}
```

Listing 13 — Multiple Assignment **?** *Example*

```
p_entry_next = (NET_ICMP_TX_SRC_QUENCH *)p_tx_src_quench->NextPtr;
found        = (p_buf_hdr->IP_AddrSrc == p_tx_src_quench->Addr) ? DEF_YES : DEF_NO;
```

Listing 14 — **?** *Operator Format*

```
rtn_status = (rtn_code == NET_SOCK_BSD_ERR_NONE) ? DEF_OK : DEF_FAIL;                          (1)

frag_size_min = (ip_flag_frags_more == DEF_YES) ? NET_IP_FRAG_SIZE_MIN_FRAG_MORE               (2)
                                                : NET_IP_FRAG_SIZE_MIN_FRAG_LAST;

p_conn_tx_sync->MaxSegSizeRemote = (p_buf_hdr->TCP_MaxSegSize != NET_TCP_MAX_SEG_SIZE_NONE)   (3)
                                 ?  p_buf_hdr->TCP_MaxSegSize
                                 :  NET_TCP_MAX_SEG_SIZE_DFLT_TX;
```

```
rto_ms_backoff_calcd = (rto_ms < pconn->TxRTT_RTO_Max_ms)                                      (4)
                     ? (rto_ms * NET_TCP_TX_RTO_TIMEOUT_BACKOFF_SCALAR)                        (5)
                     :  pconn->TxRTT_RTO_Max_ms;
```

(1) One space before and after **?** and **:** operators.

(2) Long **?** statements can split onto multiple lines with **?** and **:** operators aligned vertically.

(3) or **=**, **?** and **:** operators aligned vertically

(4) Always enclose conditional expressions in parentheses.

(5) Enclose complex resultant expressions in parentheses.

* (Pointer) Operator

The * pointer/pointer-dereference, or indirection, operator has two related functions: to declare or cast a symbol, type, or expression as a pointer of a certain data type **AND** to dereference, or access, the pointed-to value of a pointer of a certain data type. * operator statements should be formatted as shown in Listing 15.

Listing 15 — * Operator Format

```
static  NET_CONN  *NetConn_ListSrch (NET_CONN_FAMILY    family,                          (1)
                                    NET_CONN         **pconn_list,                    (2)
                                    CPU_INT08U        *paddr_local)
{
    NET_CONN  *pconn_chain;                                                           (3)
    NET_CONN  *pconn_chain_next;
    NET_CONN  *pconn_next;                                                            (4)

    ...
void  (*AddrMulticastAdd)(NET_IF             *pif,                                    (5)
                          CPU_INT08U         *paddr_protocol,
                          CPU_INT08U          paddr_protocol_len,
                          NET_PROTOCOL_TYPE   addr_protocol_type,
                          NET_ERR            *perr);

if (pif_api->AddrMulticastAdd == (void (*)(NET_IF          *,                         (6)
                                           CPU_INT08U      *,
                                           CPU_INT08U       ,
                                           NET_PROTOCOL_TYPE,
                                           NET_ERR         *))0) {
```

```
p_data_buf = (CPU_INT08U *)p_data;                                                    (7)

if (p_buf == (NET_BUF *)0) {                                                          (8)
```

```
*p_addr_cur = addr_octet_val;                                                         (9)
*p_err      = NET_ASCII_ERR_NONE;

 sum_val_32 = (CPU_INT32U)(*((CPU_INT08U *)pdata_16));                                (10)

 opt_len = *(popts + 1);                                                              (11)

 addr_len = (NET_SOCK_ADDR_LEN)*paddr_len;                                            (12)

 DEF_BIT_SET(*psum_err, NET_UTIL_16_BIT_SUM_ERR_LAST_OCTET);

 OSEventNameSet((OS_EVENT *)*psignal,
```

(1) ***** declaration should immediately prefix function, ...

(2) ...argument,

(3) ...or variable names.

(4) At least two spaces before each ***** declaration operator.

(5) ***** operator should immediately prefix function name.

(6) ***** operator enclosed in parentheses, **(*)**, when used as function pointer cast.

(7) ***** operator and data type cast should be enclosed in parentheses.

(8) At least one space before each ***** cast operator.

(9) ***** dereference should immediately prefix symbol...

(10) ...or expression

(11) At least one space before each ***** dereference operator...

(12) ...unless preceded by a parenthesis

Listing 16 — Pointer Examples

```
void  NetSecure_OS_Lock (NET_SECURE_OS_LOCK  *p_lock,                                   (1)
                         NET_ERR             *p_err)
{
    INT8U  err_os;

#if (NET_ERR_CFG_ARG_CHK_EXT_EN == DEF_ENABLED)
    if (p_lock == (NET_SECURE_OS_LOCK *)0) {                                            (2)
       *p_err   =  NET_SECURE_ERR_NULL_PTR;
        return;
    }
#endif

    OSMutexPend((OS_EVENT *)*p_lock,                                                    (3)
                             0u,
                            &err_os);

...
```

```
while (p_conn  != (NET_CONN *)0) {                                                      (4)
    p_conn->ConnList = p_conn_list;
    p_conn_next      = p_conn->NextConnPtr;                                             (5)
    p_conn           = p_conn_next;
}
```

(1) Even though **NET_SECURE_OS_LOCK** is **typedef**'d as an '**OS_EVENT ***'...

(2) ...**p_lock** is still validated as non-**NULL** pointer via check with '(**NET_SECURE_OS_LOCK *0**)'...

(3) ...but **p_lock** is dereferenced as a pointer-to-a-pointer.

(4) Cast NULL pointer to expected type.

(5) Intermediate **p_conn_next** intentionally used to get current **p_conn**'s **NextConnPtr** before overwriting back to **p_conn**.

`&` (Address-of) Operator

The `&` address-of operator returns the address of any symbol that has storage: variables, tables, and functions. `&` operator cannot be used to obtain the address of expressions or constants, even if they refer to addresses or pointers to addresses. However, the `&` operator can obtain the address of a dereferenced pointer, as shown in Listing 17, but this is highly discouraged since the dereferenced indirection is **NOT** immediately intuitive or obvious.

`&` operator statements should be formatted as shown in Listing 18. Addresses of arrays should explicitly indicate the index offset even when offset by `0` (zero) as shown in Listing 19. All addresses of functions should explicitly be prefixed with the `&` operator (MISRA 2004 Rule 16.9) as shown in Listing 20.

Listing 17 — & Operator Format

```
NetStat_PoolEntryUsedDec(&NetConn_PoolStat, &err);                              (1)

err_msg_ptr = (DNSc_ERR_STRUCT *)&DNSc_Err_Msg[0];                              (2)

p_buf_hdr->UnlinkFnctPtr = (NET_BUF_FNCT)&NetARP_CacheUnlinkBuf;                (3)
```

```
p_sock = &NetSock_Tbl[0];                                                       (4)

NetOS_Lock(&err);                                                               (5)
(void)&p_data;
```

(1) **&** should immediately prefix variable...

(2) ...table,

(3) ...or function names.

(4) At least one space before each **&** operator...

(5) ...unless preceded by a parenthesis.

Listing 18 — Address-of Pointer Expressions Discouraged

```
CPU_INT32U   val32    =  32;
CPU_INT32U  *p_val32  = &val32;                                                 (1)
CPU_INT32U  *p_addr32 = &(*p_val32);                                            (2)
```

(1) **p_val32** points to address of **val32**.

(2) **p_addr32** also points to what **p_val32** points to, but is address-of expression **NOT** intuitive or obvious.

Listing 19 — Array Addresses Examples

```
p_tbl  = &OSRdyTbl[y];                                                          (1)
p_conn = &NetConn_Tbl[0];
```

(1) Always include index(es) when getting address-of an array even if getting the base address of the array at index 0.

```
p_signal = &NetOS_TCP_RxQ_SignalPtr[0];

Mem_Copy((void     *) p_addr_hw,
         (void     *)&p_cache->HW_Addr[0],
         (CPU_SIZE_T) addr_hw_len);
```

instead of

```
p_signal = NetOS_TCP_RxQ_SignalPtr;
```

instead of

```
Mem_Copy((void     *)p_addr_hw,
         (void     *)p_cache->HW_Addr,
         (CPU_SIZE_T)addr_hw_len);
```

Listing 20 — Function Addresses Examples

```
timeout_fnct = &NetTCP_ConnCloseTimeout;
```

instead of

```
timeout_fnct = NetTCP_ConnCloseTimeout;
```

```
NetTmr_Set(p_cache->TmrPtr,
           &NetARP_CacheTimeout,
            timeout_tick,
            p_err);
```

instead of

```
NetTmr_Set(p_cache->TmrPtr,
            NetARP_CacheTimeout,
            timeout_tick,
            p_err);
```

```
const  NET_IF_API  NetIF_API_Loopback = {                                       /* Loopback IF API fnct ptrs :      */
                                         &NetIF_Loopback_IF_Add,                /*   Init/add                       */
                                         &NetIF_Loopback_IF_Start,              /*   Start                          */
                                         &NetIF_Loopback_IF_Stop,               /*   Stop                           */
                                          0,                                    /*   Rx                             */
                                          0,                                    /*   Tx                             */
                                         &NetIF_Loopback_AddrHW_Get,            /*   Hw        addr get             */
                                         &NetIF_Loopback_AddrHW_Set,            /*   Hw        addr set             */
                                         &NetIF_Loopback_AddrHW_IsValid,        /*   Hw        addr valid           */
                                         &NetIF_Loopback_AddrMulticastAdd,      /*   Multicast addr add             */
                                         &NetIF_Loopback_AddrMulticastRemove,   /*   Multicast addr remove          */
                                         &NetIF_Loopback_AddrMulticastProtocolToHW,/*   Multicast addr protocol-to-hw  */
                                         &NetIF_Loopback_MTU_Set,               /*   MTU set                        */
                                         &NetIF_Loopback_GetPktSizeHdr,         /*   Get pkt hdr size               */
                                         &NetIF_Loopback_GetPktSizeMin,         /*   Get pkt min size               */
                                         &NetIF_Loopback_ISR_Handler,           /*   ISR handler                    */
                                         &NetIF_Loopback_IO_CtrlHandler         /*   I/O ctrl                       */
                                        };
```

`sizeof` Operator

The **`sizeof`** operator returns the number of bytes for data types, variables, object, or strings (as shown in Listing 21). Note that ISO/IEC 9899:TC2, Section 5.2.4.2.1.(1) defines a byte to have a *minimum* of 8 bits. However, (most) Micrium software has been developed assuming that bytes are always 8-bit (octets) and expects the **`sizeof`** operator to return sizes in terms of 8-bit bytes. (See also Portable **`sizeof`** Calculations.) The **`sizeof`** operator returns sizes via the **`size_t`** data type and should be cast to appropriate data types, wherever applicable. The **`sizeof`** operator **CANNOT** be used in preprocessor conditionals, but may be used when declaring arrays as shown in Listing 22. The **`sizeof`** operator can return the size of expressions, as shown in Listing 23, but this is highly discouraged since the usefulness of doing so is **NOT** immediately intuitive or obvious.

`sizeof` operator statements should be formatted similar to function or macro calls as shown in Listing 24. Although function and macro calls should **NOT** be called from within conditional statements (see Simple Code Statements), the **`sizeof`** operator may be used within conditional statements as shown in Listing 25.

Listing 21 — `sizeof` Operator Format

```
while (size_rem >= sizeof(CPU_INT16U)) {                                        (1)
```

```
addr_hw_len = sizeof(addr_hw);                                                  (2)

CPU_SIZE_T  Micrium_Ca_Cert_Pem_Len = sizeof(Micrium_Ca_Cert_Pem) / sizeof(CPU_CHAR);       (3)

str_len = sizeof("Some string.");                                               (4)
```

```
size_rem      -= (CPU_INT16U)sizeof(CPU_INT16U);                                (5)

#define  FD_ARRAY_SIZE    (((FD_SETSIZE - 1) / (sizeof(CPU_DATA) * DEF_OCTET_NBR_BITS)) + 1)

size_tot_ptrs  =  Mem_PoolSegCalcTotSize((void     *)pmem_addr_ptrs,
                                         (CPU_SIZE_T)blk_nbr,
                                         (CPU_SIZE_T)sizeof(void *),
                                         (CPU_SIZE_T)sizeof(void *));
```

(1) **sizeof()** should immediately prefix data type...

(2) ...variable,

(3) ...object,

(4) ...or string names.

(5) At least one space before each **sizeof** operator, unless preceded by a parenthesis.

Listing 22 — `sizeof` within Conditionals

```
for (i = 0u; i < sizeof(CPU_ALIGN); i++) {
```

```
while (size_rem >= sizeof(CPU_INT08U)) {
```

Listing 23 — `sizeof` Array Declarations

```
CPU_INT08U  addr_protocol[sizeof(NET_IP_ADDR)];
```

Listing 24 — `sizeof` Expressions Discouraged

```
CPU_SIZE_T  len;
CPU_INT08U  val;

val = 21;
len = (CPU_SIZE_T)sizeof(4 + 2);
len = (CPU_SIZE_T)sizeof(val * 3 + 2);
```

(1) **len = sizeof(int** data type), even if other data types are explicitly involved in the expression

Listing 25 — String `sizeof` Examples

```
CPU_CHAR    *p_str;
CPU_CHAR     str[50];
CPU_SIZE_T   len;

len   = (CPU_SIZE_T)Str_Len("Hello world.");                                    (1)
len   = (CPU_SIZE_T)sizeof ("Hello world.");                                    (2)

p_str = "Hello world.";
len   = (CPU_SIZE_T)sizeof(p_str);                                             (3)

Str_Copy(&str[0], p_str);
len   = (CPU_SIZE_T)sizeof (str);                                              (4)
len   = (CPU_SIZE_T)sizeof(&str[0]);                                           (5)
```

(1) Contrary to **strlen()/Str_Len()** behavior where **len** = 12 bytes.

(2) **sizeof()** includes terminating **NULL** character, **len** = 13 bytes.

(3) **len** = **sizeof**(Pointer data type) = 4 bytes (on 32-bit CPU).

(4) **len** = **sizeof**(String buffer array) = 50 bytes.

(5) **len** = **sizeof**(Pointer data type) = 4 bytes (on 32-bit CPU).

Portability

Data Types

See Portable Data Types.

Data Alignment

Many CPUs and devices require aligned data access [1] to occur on address boundaries that are even multiples of the size of the data being accessed as shown in Listing 1. For example, a 32-bit word would require access from an address that is an even multiple of 32 bits. On some CPUs, accessing data on unaligned addresses might lead to poor performance; on other CPUs, it might generate a data alignment exception. Therefore, portable code must fully respect and implement data alignment. Since compilers (typically) generate code that respects data alignment, accessing data via pointers is usually the only opportunity to cause data alignment exceptions as shown in Listing 2. Note that simply assigning an unaligned address to a pointer typically does not generate a data alignment exception; but accessing an unaligned address typically does.

Overlaying data structures (or unions) and accessing internal data members can also cause data alignment exceptions. One common mistake for network applications is declaring a generic **`sockaddr`** structure but manipulating it as a **`sockaddr_in`** structure as shown in Listing 3. Since the **`sockaddr`** structure has an unformatted character array, a compiler can instantiate it at any CPU address. However, when an application formats the **`sockaddr`** structure's array with **`sockaddr_in`** structure data types, the accesses might or might not be aligned to appropriate word addresses.

µC/LIB's **`lib_mem.h`** includes several memory copy macros that handle both data alignment and endianness as shown in Listing 5. (See also Memory Macro Performance.)

Listing 1 — Data Alignment Examples

```
 CPU_INT32U  *p_val32;
 CPU_INT16U  *p_val16;
 CPU_INT16U   val32;
 CPU_INT16U   val16;

 p_val32 = &val32;                                                    (1)
*p_val32 =  9;
 p_val16 = &val16;                                                    (2)
*p_val16 =  6;

 p_val16 = &val32;                                                    (3)


*p_val16 =  3;
```

(1) 32-bit data pointer aligned to 32-bit data

(2) 16-bit data pointer aligned to 16-bit data

(3) 16-bit data pointer aligned to 32-bit data since a 32-bit aligned address is also a 16-bit aligned address because it's an even multiple of 16 bits

- val32 = 3 (0x03000000) on little-endian CPUs
- val32 = 196617 (0x00030009) on big- endian CPUs

Listing 2 — Examples of Data Alignment Exceptions

```
 CPU_INT32U  *p_val32;
 CPU_INT16U   val16_a;
 CPU_INT16U   val16_b;

 p_val32 = &val16_a;
*p_val32 =  0;                                                        (1)
 p_val32 = &val16_b;
*p_val32 =  0;                                                        (2)

 p_val32 = 0x2000FF06;
*p_val32 = 0;                                                         (3)
```

(1) **MIGHT** cause 32-bit data alignment exception

(2) **SHOULD** cause 32-bit data alignment exception

(3) **WILL** cause 32-bit data alignment exception

Listing 3 — `sockaddr` Data Alignment Exception Example

```
sockaddr      sock_addr;                                              (1)
sockaddr_in  *p_addr;

p_addr           = &sock_addr;
p_addr->sin_addr = INADDR_ANY;                                        (2)
```

(1) Depending on what CPU address **sock_addr** is instantiated on ...

(2) ... **p_addr->sin_addr MIGHT** cause data alignment exception

Endianness

CPUs and devices typically access data words in one of two major octet orderings, big-endian or little-endian [2], as shown in Listing 4. Many data structures or data streams must be formatted in specific CPU, device, or protocol endianness. For example, network protocols and data must be formatted and communicated in big-endian order, regardless of the endianness of any CPUs or devices. µC/LIB's **`lib_mem.h`** includes several memory copy macros that handle both endianness and data alignment as shown in Listing 5. (See also Memory Macro Performance.)

Listing 4 — Endianness Examples

```
  Memory      Data
 Addresses   Values
----------   ------
0x20200040    0x11     32-bit big-   endian value @ 0x20200040 = 0x1122AABB =  287484603
0x20200041    0x22     32-bit little-endian value @ 0x20200040 = 0xBBAA2211 = 3148489233
0x20200042    0xAA     16-bit big-   endian value @ 0x20200042 = 0xAABB     =      43707
0x20200043    0xBB     16-bit little-endian value @ 0x20200042 = 0xBBAA     =      48042
```

Listing 5 — µC/LIB Memory Copy Macros handle Endianness and Alignment

```
MEM_VAL_COPY_GET_INT16U_BIG(&pbuf_hdr->IP_TotLen,  &pip_hdr->TotLen);                (1)

MEM_VAL_COPY_GET_INT32U_BIG(&pbuf_hdr->IP_AddrSrc, &pip_hdr->AddrSrc);              (2)

MEM_VAL_COPY_GET_INT16U(&addr_family, &paddr_ip->AddrFamily);                       (3)
```

(1) Copies 16-bit big- endian value from **pip_hdr->TotLen** into 16-bit **pbuf_hdr->IP_TotLen** with CPU's endianness.

(2) Copies 32-bit big- endian value from **pip_hdr->AddrSrc** into 16-bit **pbuf_hdr->IP_AddrSrc** with CPU's endianness.

(3) Copies 16-bit host-endian value from **paddr_ip->AddrFamily** into 16-bit **addr_family** with CPU's endianness.

Auto-generated Compiler Calculations

Whenever possible, allow the compiler to automatically calculate any compile-time arithmetic. This reduces possible mistakes from manual calculations and allows the compiler to consistently resolve any boundary cases in its own calculations. The following examples show calculations performed outside of function bodies, but these examples are guidelines for all source code, not just tables or global constants.

Listing 6 shows an example SSL certificate (provided by CyaSSL) whose size was manually calculated, whereas Listing 7 shows the same SSL certificate but whose size is calculated by the compiler. Having the compiler auto-calculate the certificate table sizes showed that the manually-calculated certificate sizes forgot to include the string's terminating **`NULL`** character (e.g., **`Micrium_Ca_Cert_Pem_Len`** should calculate a certificate size of '995' instead of the manually-calculated value of '994'), which allowed for the possibility of string overruns.

Listing 6 — Auto-Calculated Certificate Table Size Preferred

```
const  CPU_CHAR  Micrium_Ca_Cert_Pem[] =
"-----BEGIN CERTIFICATE-----\r\n"
"MIICpTCCAg4CCQDNdHgFKaYRWDANBgkqhkiG9w0BAQUFADCBljELMAkGA1UEBhMC\r\n"
"Q0ExDzANBgNVBAgMBlF1ZWJlYzERMA8GA1UEBwwITW9udHJlYWwxFTATBgNVBAoM\r\n"
   ...
"1YBKNbTzIJNjwTajkUPz38BjXb5sqLyPK8wRbjadm2pOlw1f7bIFunpbHpV+1XA1\r\n"
"tk3W32BqKfzy\r\n"
"-----END CERTIFICATE-----\r\n";

const  CPU_SIZE_T  Micrium_Ca_Cert_Pem_Len = sizeof(Micrium_Ca_Cert_Pem) / sizeof(CPU_CHAR);   (1)
```

(1) **Micrium_Ca_Cert_Pem[]** size auto-calculated by compiler

Listing 7 — Manually-Calculated Certificate Table Size Discouraged

```
const  CPU_SIZE_T  Micrium_Ca_Cert_Pem_Len = 994;                                            (1)
const  CPU_CHAR    Micrium_Ca_Cert_Pem[] =
"-----BEGIN CERTIFICATE-----\r\n"
"MIICpTCCAg4CCQDNdHgFKaYRWDANBgkqhkiG9w0BAQUFADCBljELMAkGA1UEBhMC\r\n"
"Q0ExDzANBgNVBAgMBlF1ZWJlYzERMA8GA1UEBwwITW9udHJlYWwxFTATBgNVBAoM\r\n"
   ...
"1YBKNbTzIJNjwTajkUPz38BjXb5sqLyPK8wRbjadm2pOlw1f7bIFunpbHpV+1XA1\r\n"
"tk3W32BqKfzy\r\n"
"-----END CERTIFICATE-----\r\n";                                                             (2)
```

(1) **Micrium_Ca_Cert_Pem[]** size manually calculated...

(2) ... incorrectly excluded terminating **NULL** character.

Listing 8 shows a table whose values were originally calculated in Excel and manually copied into the source file table. A review/test of these values would require verifying each value since any one of them could potentially have mistakes in copying or in its calculation (e.g. values floating-point-rounded instead of integer-rounded). However, Listing 9 shows the table with the arithmetic expression for each value that the compiler will calculate. Using this method, a review of the values only need confirm the correct arithmetic values in each expression, which is much easier since the expression itself is now explicitly and obviously related to each line's number base.

Listing 8 — Auto-Calculated Table Values Preferred

```
static  const  CPU_INT32U  Str_MultOvfThTbl_Int32U[] = {
   (CPU_INT32U) DEF_INT_32U_MAX_VAL,            /*                Invalid base  0.  */
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL /  1u),     /*                Invalid base  1.  */
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL /  2u),     /* 32-bit mult ovf th for base  2.  */          (1)
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL /  3u),     /* 32-bit mult ovf th for base  3.  */
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL /  4u),     /* 32-bit mult ovf th for base  4.  */          (2)
       ...
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL / 35u),     /* 32-bit mult ovf th for base 35.  */
   (CPU_INT32U)(DEF_INT_32U_MAX_VAL / 36u)      /* 32-bit mult ovf th for base 36.  */
};
```

(1) Allows compiler to correctly calculate values.

(2) Calculated equation is explicit and obvious.

Listing 9 — Manually-Calculated Table Values Discouraged

```
static  const  CPU_INT32U  Str_MultOvfThTbl_Int32U[] = {
    4294967295,                                 /*                Invalid base  0.  */
    4294967295,                                 /*                Invalid base  1.  */          (1)
    2147483647,                                 /* 32-bit mult ovf th for base  2.  */
    1431655765,                                 /* 32-bit mult ovf th for base  3.  */
    1073741823,                                 /* 32-bit mult ovf th for base  4.  */          (2)
        ...
     122713351,                                 /* 32-bit mult ovf th for base 35.  */
     119304647                                  /* 32-bit mult ovf th for base 36.  */
};
```

(1) Manual calculated values not obviously correct; requires manual review/testing.

(2) Calculated equation is **NOT** obvious.

`sizeof` Operator

Whenever possible, the **`sizeof`** operator should be used to calculate the size of declared variables/objects as shown in Listing 10, not data types as shown in Listing 11. This reduces any discrepancy errors between calculating the **`sizeof`** of a data type and using that size with a declared variable of another data type. However, there are instances when **`sizeof`** must be used on a data type, such as calculating the size or number of objects in a table as shown in Listing 6.

Also note that ISO/IEC 9899:TC2, Section 6.5.3.4.(2) states that "the **`sizeof`** operator yields the size (in bytes) of its operand", where Section 5.2.4.2.1.(1) defines that a byte has a minimum of 8 bits. However, (most) Micrium software has been developed assuming that bytes are always 8-bit (octets) and expects the **`sizeof`** operator to return sizes in terms of 8-bit bytes.

Listing 10 — Using `sizeof` on Declared Variables/Objects Preferred

```
NET_SOCK_ADDR_IP   addr;                                                                    (1)
NET_SOCK_ADDR_LEN  addr_len;

addr_len = (NET_SOCK_ADDR_LEN)sizeof(addr);                                                 (2)
```

(1) **NET_SOCK_ADDR_IP** data type

(2) **sizeof(addr)** correctly calculates **addr_len** regardless of **addr**'s data type.

Listing 11 — Using `sizeof` on Data Types Discouraged

```
NET_SOCK_ADDR_IP   addr;                                                                    (1)
NET_SOCK_ADDR_LEN  addr_len;

addr_len = (NET_SOCK_ADDR_LEN)sizeof(NET_SOCK_ADDR);                                        (2)
```

(1) **NET_SOCK_ADDR_IP** data type.

(2) **sizeof(addr)** calculates **addr_len** of **NET_SOCK_ADDR** data type instead of **NET_SOCK_ADDR_IP** data type.

References

[1] http://en.wikipedia.org/wiki/Data_structure_alignment
[2] http://en.wikipedia.org/wiki/Endianness

Source Code

This section is to be completed.

Tab Character

Tab characters (ASCII character 0x09) must not be used. Indentation must be done using the space characters only (ASCII character 0x20). Tab characters expand differently of different computers and printers. Avoiding them ensures that the intended spacing is maintained.

Indentation

Indentation of code will consist of 4 spaces, except statements under a switch/case statement, which are indented by 5 spaces. Always try to indent to columns that are multiples of 4 spaces (columns 1, 5, 9, 13, 17 …). This indentation might also be refered to as "columns of 4". See Listings 2.1, 2.2 and 2.3.

Line Width

You should NOT limit the width of C source code to 80 characters just because 1980's monitors only allowed you to display 80 characters wide. The width of a line could be based on how many characters can be printed (if you need to print the code) on an 8.5" by 11" page using a reasonable font size. Using 7-pt Arial, 132 characters (in portrait mode) can be accommodated while leaving enough room on the left side of the page for holes for insertion in a three ring binder.

A line-width of 132 characters prevents needing to interleave source code with comments. If more characters are needed to make the code clearer then you should not be limited to 132 characters. In fact, you could have code that contains initialized structures (placed in Read-Only-Memory, ROM) that are over 300 characters wide.

Blank Lines

For enhanced clarity, code blocks can be separated with blank lines. A long function is often logically divided into a set of major blocks, each of which is divided in smaller sub-blocks that may be further subdivided. From the smallest blocks to the largest, an increasing quantity of blank lines should be used to convey the nature of the hierarchy. Neighboring or nested control statements may also benefit from blank lines inserted between statements. Listing 1 demonstrates this concept.

Listing 1 — Line Feed Example

```
 if (data_size < 1) {                                /* ------------ HANDLE NULL-SIZE DATA PKT ------------ */
    *psum_err = NET_UTIL_16_BIT_SUM_ERR_NULL_SIZE;

     if (prev_octet_valid != DEF_NO) {              /* If null size & last octet from prev pkt buf avail ..*/
                                                                                                              (1)
         if (last_pkt_buf != DEF_NO) {              /* ...   & on last pkt buf,              ...           */
             sum_val_32   = (CPU_INT32U)*poctet_prev;/* ...   cast prev pkt buf's last octet, ...           */
             sum_val_32 <<=  DEF_OCTET_NBR_BITS;    /* ... pad odd-len pkt len (see Note #5) ...           */
             sum_32       =  sum_val_32;            /* ...  & rtn prev pkt buf's last octet as last sum.   */

         } else {                                   /* ... & NOT on last pkt buf, ...                      */
            *poctet_last = *poctet_prev;            /* ... rtn last octet from prev pkt buf as last octet. */
             DEF_BIT_SET(*psum_err, NET_UTIL_16_BIT_SUM_ERR_LAST_OCTET);

         }

     } else {
         ;                                          /* If null size & NO prev octet, NO action(s) req'd.   */
     }

     return (sum_32);                               /* Rtn 16-bit sum (see Note #5c1).                     */
 }
                                                                                                            (2)


                                                    /* ------------ HANDLE NON-NULL DATA PKT ------------ */
 size_rem       =  data_size;
*psum_err       =  NET_UTIL_16_BIT_SUM_ERR_NONE;
```

```
                                                    /* See Notes #3 & #4.                                  */
 modulo_16      = (CPU_INT08U)((CPU_ADDR)p_data % sizeof(CPU_INT16U));
 pkt_aligned_16 = (((modulo_16 == 0) && (prev_octet_valid == DEF_NO )) ||
                   ((modulo_16 != 0) && (prev_octet_valid == DEF_YES))) ? DEF_YES : DEF_NO;
                                                                                                            (3)

 pdata_08 = (CPU_INT08U *)p_data;
 if (prev_octet_valid == DEF_YES) {                 /* If last octet from prev pkt buf avail,   ...        */
     sum_val_32   = (CPU_INT32U)*poctet_prev;
     sum_val_32 <<=  DEF_OCTET_NBR_BITS;            /* ... prepend last octet from prev pkt buf ...        */
                                                                                                            (4)
     sum_val_32  += (CPU_INT32U)*pdata_08++;
     sum_32      += (CPU_INT32U) sum_val_32;        /* ... to first octet in cur pkt buf.                  */
                                                                                                            (5)
     size_rem    -=  sizeof(CPU_INT08U);
 }

 if (pkt_aligned_16 == DEF_YES) {                   /* If pkt data aligned on 16-bit boundary, ..          */
                                                    /* .. calc sum with 16- & 32-bit data words.           */
        pdata_16  = (CPU_INT16U *)pdata_08;

.
.
.
```

(1) Enhance separation between clauses of *if* & *else* statements.

(2) Three blank lines between these major blocks.

(3) Two blank lines between these blocks.

(4) Separate steps in calculation.

(5) Separate steps in calculation.

Vertical Alignment

Equal signs for a code block must vertically align two columns after the longest left-hand side expression for the code block; i.e., the longest left-hand side expression should be followed by a single space and the equals sign.

Expressions and identifiers should be vertically aligned whenever possible, especially if alignment only requires 1-2 spaces of adjustment.

Expressions should vertically align their left-most alphanumeric characters on an indentation column of 4 where any preceding characters (asterisks, ampersands, parentheses) will align in the column(s) immediately preceding the indentation column. However, horizontal alignment may also consider other factors like identical portions of names.

Listings 2 demonstrate these concepts.

Listing 2 — Code Alignment Examples

```
= datum_temp & DEF_BIT_00;
```

```
 hamming    <<= 1;

 max_seg_size   = *popt;
 max_seg_size <<=  DEF_OCTET_NBR_BITS;
*perr           =  NET_TCP_ERR_NONE;

  addr_tbl_qty = *paddr_tbl_qty;
*paddr_tbl_qty =  0;
```

The least-significant portion of integers and floating-point numbers should be aligned. Listing 3 demonstrates this concept.

Listing 3 — Numeric Alignment Example

```
DispSegTblIx =    0;
DispDigMsk   = 0x80;
DispScale    =    1.25;

p_conn    = &NetConn_Tbl[0]
  conn_id =  p_conn->ID;
```

Spacing

The unary operators are written with no space between the operator and the operand:

```
!value
~bits
++i
j--
(CPU_INT32U)x
*ptr
&x
sizeof(x)
```

The binary operators (and the ternary operator) are written with at least one space between the operator and operands:

```
c1 = c2;
x + y
i += 2;
n > 0 ? n : -n
a < b
c >= 2
```

At least one space is needed after each semicolon:

```
for (i = 0; i < 10; i++)
```

The keywords if, else, while, for, switch and return are followed by one space:

```
if (a > b)
while (x > 0)
for (i = 0; i < 10; i++)
} else {
switch (x)
return (y)
```

Expressions within parentheses are written with no space after the opening parenthesis and no space before the closing parenthesis:

```
x = (a + b) * c;
```

Source Files

Use the template files to get started. You can copy the body of the template files at the bottom of this page.

File Heading

A comment block must be placed at the beginning of each source code file (both code and header files) containing the module name and description, copyright, copyright or distribution terms, file description, file name, module version, programmer initial(s) and note(s). This information should be structured as shown in Listing 1. The note(s) section should be omitted if no notes are given.

Listing 1 — File Heading Example

```
/*
*********************************************************************************************
*                                          uC/PRODUCT
*                                      The Embedded Product
*
*                       (c) Copyright 2004-2011; Micrium, Inc.; Weston, FL
*
*           All rights reserved.  Protected by international copyright laws.
*
*           uC/PRODUCT is provided in source form to registered licensees ONLY.  It is
*           illegal to distribute this source code to any third party unless you receive
*           written permission by an authorized Micrium representative.  Knowledge of
*           the source code may NOT be used to develop a similar product.
*
*           Please help us continue to provide the Embedded community with the finest
*           software available.  Your honesty is greatly appreciated.
*
*           You can contact us at www.micrium.com.
*********************************************************************************************
*/

/*
*********************************************************************************************
*
*                                        FILE DESCRIPTION
*
* Filename      : lib_mem.c
* Version       : V1.34.00
* Programmer(s) : ITJ
*                 FGK
*********************************************************************************************
* Note(s)       : (1) NO compiler-supplied standard library functions are used in library
*                     or product software.
*********************************************************************************************
*/
```

(1) Module Name

(2) Module Description

(3) Copyright

(4) Legal terms

(5) Max. width: 106 columns

(6) File Description

(7) Filename

(8) Module version: Vx.yy.zz

(9) Programmer(s)' Initials

(10) Note(s) optional; if NOT available, omit Note(s) and extra line of asterisks.

Application Example File Heading

For application examples, (or other examples/templates for that matter), we have a special file heading without copyright information.

```
/*
*********************************************************************************************************
*                                       EXAMPLE CODE
*
*               This file is provided as an example on how to use Micrium products.
*
*               Please feel free to use any application code labeled as 'EXAMPLE CODE' in
*               your application products.  Example code may be used as is, in whole or in
*               part, or may be used as a reference only. This file can be modified as
*               required to meet the end-product requirements.
*
*               Please help us continue to provide the Embedded community with the finest
*               software available.  Your honesty is greatly appreciated.
*
*               You can contact us at www.micrium.com.
*********************************************************************************************************
*/
```

Implementation (Source Code) File Layout

After the file header, every code file must contain the following sections in the following order:

- **INCLUDE FILES**
- **MODULE**
- **EXTERNAL C LANGUAGE LINKAGE**
- **LOCAL DEFINES**
- **LOCAL CONSTANTS**
- **LOCAL DATA TYPES**
- **LOCAL TABLES**
- **LOCAL GLOBAL VARIABLES**
- **LOCAL FUNCTION PROTOTYPES**
- **LOCAL CONFIGURATION ERRORS**
- **MODULE END**

However, some source files might not require all of the above sections or possibly in a different ordering. For example, device drivers with local functions that **MUST** be prototyped before use in a function API table might re-order **LOCAL TABLES** after **LOCAL FUNCTION PROTOTYPES**.

Each section must be preceded by a comment block; see Listing 2. The Section Name should be centered following the formula: start_column = floor((105 - name_size) / 2). The contents of a section must be followed by two blank lines. Even if a section is empty, its comment block must be included in the file and followed by two blank lines. These sections may have additional sub-sections.

Listing 2 — Comment Block Format

```
/*
*********************************************************************************************************
*                                              SECTION NAME
*********************************************************************************************************
*/
```

Functions definitions must follow these sections in the following order:

- Global functions
- Local functions

INCLUDE FILES Section

The **INCLUDE FILES** section should include all necessary header files. If the code file has a matching header file, it may be most convenient to include that header file, as is done in Listing 3; the matching header file will include the header files for external modules or other internal header files, as appropriate.

Listing 3 — INCLUDE FILES Section

```
/*
*********************************************************************************************************
*                                              INCLUDE FILES
*********************************************************************************************************
*/

#define    MICRIUM_SOURCE
#define    LIB_MEM_MODULE
#include  <lib_mem.h>
```

SOURCE CODE #define

The **MICRIUM_SOURCE** symbol MUST be #define'd at the beginning of every source file (BEFORE including any file). This #define is used by third parties (like ADI) to differentiate Micrium source code with the application code.

MODULE Section

The (optional) **MODULE** section determines whether the module is dynamically included in the build based on specific product configurations. This section should include notes for which configurations include the module as shown in Listing X.

EXTERNAL C LANGUAGE LINKAGE Section

Source files shall be wrapped (excluding the inclusion of additional headers) in extern "C" to be compliant with C++ compilers. The **EXTERNAL C LANGUAGE LINKAGE** section performs the test (#ifdef __cplusplus) and linkage specifier (extern "C"). The matching brace for the test is located in the **EXTERNAL C LANGUAGE LINKAGE END** section. Since **EXTERNAL C LANGUAGE LINKAGE** sections are usually required only to provide correct linkage information for global API symbols, a source file requires a **EXTERNAL C LANGUAGE LINKAGE** section only if its local function prototypes or data type definitions are accessed via a global API. Thus, **EXTERNAL C LANGUAGE LINKAGE** sections are only typically only required in header files.

Listing 4 — EXTERNAL C LANGUAGE LINKAGE and EXTERNAL C LANGUAGE LINKAGE END Sections

```
/*
*********************************************************************************************************
*                                     EXTERNAL C LANGUAGE LINKAGE
*
* Note(s) : (1) C++ compilers MUST 'extern'ally declare ALL C function prototypes & variable/object
*               declarations for correct C language linkage.
*********************************************************************************************************
*/

#ifdef __cplusplus
extern  "C" {                                   /* See Note #1.                                         */
#endif

            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .

/*
*********************************************************************************************************
*                                   EXTERNAL C LANGUAGE LINKAGE END
*********************************************************************************************************
*/

#ifdef __cplusplus
}                                               /* End of 'extern'al C lang linkage.                    */
#endif
```

Functions

After the configuration errors in the LOCAL CONFIGURATION ERRORS section, function definitions must be given. All definitions of global functions must come first, followed by definitions of local functions. The local functions shall be separated from the global functions using the comment block shown in Listing 1-4.

Listing 5 — LOCAL FUNCTIONS Comment Block Format

```
/*
*********************************************************************************************************
*********************************************************************************************************
*                                            LOCAL FUNCTIONS
*********************************************************************************************************
*********************************************************************************************************
*/
```

Within the global functions section and the local functions section, functions shall be ordered in the order the prototypes are given (in the LOCAL FUNCTION PROTOTYPES section of the code file and the FUNCTION PROTOTYPES section of the header file, respectively). The order of the function prototypes/definitions should be in logical, functional order as opposed to alphabetical order.

Header File Layout

After the file header, every header file must contain the following sections in the following order:

- MODULE
- INCLUDE FILES
- EXTERNAL C LANGUAGE LINKAGE
- EXTERNS (if a matching code file exists)
- DEFAULT CONFIGURATION (if necessary)
- DEFINES
- DATA TYPES
- GLOBAL VARIABLES
- MACROS
- FUNCTION PROTOTYPES
- CONFIGURATION ERRORS
- MODULE END

Each section must be preceded by a comment block; see Listing 2. The contents of a section should be followed by two blank lines. Even if a section is empty, its comment block must be included in the file and followed by two blank lines. These sections may have additional sub-sections.

MODULE and MODULE END Sections

Header files shall be guarded from duplicate inclusion by testing for the definition of a value. The MODULE performs the test (#ifndef) and definition (#define). The matching #endif for the test is located in the MODULE END section.

Listing 6 — MODULE and MODULE END Sections

```
/*
*********************************************************************************************************
*                                                MODULE
*********************************************************************************************************
*/

#ifndef  LIB_MEM_MODULE_PRESENT
#define  LIB_MEM_MODULE_PRESENT


            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .


/*
*********************************************************************************************************
*                                             MODULE END
*********************************************************************************************************
*/

#endif                                                  /* End of lib mem module include.               */
```

INCLUDE FILES Section

The header file should include all necessary base files, internal files or external module includes files. These includes should be placed in the INCLUDE FILES section.

Listing 7 — INCLUDE FILES Section

```
/*
*********************************************************************************************************
*                                            INCLUDE FILES
*********************************************************************************************************
*/

#include  <cpu.h>
#include  <lib_def.h>
#include  <app_cfg.h>
```

EXTERNAL C LANGUAGE LINKAGE Section

Header files shall be wrapped (excluding the inclusion of additional headers) in extern "C" to be compliant with C++ compilers. The **EXTERNAL C LANGUAGE LINKAGE** performs the test (#ifdef __cplusplus) and linkage specifier (extern "C"). The matching brace for the test is located in the **EXTERNAL C LANGUAGE LINKAGE END** section. Ideally, each **EXTERNAL C LANGUAGE LINKAGE** section should only encompass each module's function prototypes and variable/data type declarations.

Listing 8 — EXTERNAL C LANGUAGE LINKAGE and EXTERNAL C LANGUAGE LINKAGE END Sections

```
/*
*********************************************************************************************************
*                                      EXTERNAL C LANGUAGE LINKAGE
*
* Note(s) : (1) C++ compilers MUST 'extern'ally declare ALL C function prototypes & variable/object
*               declarations for correct C language linkage.
*********************************************************************************************************
*/

#ifdef __cplusplus
extern  "C" {                                   /* See Note #1.                                         */
#endif


            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .
            .      .      .      .      .      .      .      .      .      .      .


/*
*********************************************************************************************************
*                                    EXTERNAL C LANGUAGE LINKAGE END
*********************************************************************************************************
*/

#ifdef __cplusplus
}                                               /* End of 'extern'al C lang linkage.                    */
#endif
```

EXTERNS and GLOBAL VARIABLES Sections

A global variable needs to be allocated storage space in RAM and must be referenced in other modules using the C keyword extern. Consequently, declarations must be placed in both the code (*.c) and header (*.h) files, possibly leading to mistakes. The EXTERNS sections eliminates this source of error, so that declarations need only be done in the header (*.h) file. The code, as shown in Listing 1-9, conditionally defines a xxxx_EXT constant (in this example, LIB_MEM_EXT) that must prefix the declaration of global variables, as shown in Listing 1-10. The matching code (*.c) file should contain the #define of xxxx_MODULE (in this example, LIB_MEM_MODULE); see Listing 9.

Listing 9 — EXTERNS Section

```
/*
*********************************************************************************************************
*                                                EXTERNS
*********************************************************************************************************
*/

#ifdef   LIB_MEM_MODULE
#define  LIB_MEM_EXT
#else
#define  LIB_MEM_EXT  extern
#endif
```

Listing 10 — GLOBAL VARIABLES Section

```
/*
*********************************************************************************************************
*                                          GLOBAL VARIABLES
*********************************************************************************************************
*/

LIB_MEM_EXT  CPU_INT08U  Mem_ExampleGlobal;
```

CONFIGURATION ERRORS Section

#error should be used to flag missing #define constant or macros and to check for invalid values. The #error directive will cause the compiler to display the message with the double quotes when the condition is not met.

Listing 11 — CONFIGURATION ERRORS Section

```
/*
*********************************************************************************************************
*                                       CONFIGURATION ERRORS
*********************************************************************************************************
*/

#ifndef  LIB_MEM_CFG_ARG_CHK_EXT_EN
#error  "LIB_MEM_CFG_ARG_CHK_EXT_EN        not #define'd in 'app_cfg.h'"
#error  "                             [MUST be  DEF_DISABLED]           "
#error  "                             [     ||  DEF_ENABLED ]           "

#elif  ((LIB_MEM_CFG_ARG_CHK_EXT_EN != DEF_DISABLED) && \
        (LIB_MEM_CFG_ARG_CHK_EXT_EN != DEF_ENABLED ))
#error  "LIB_MEM_CFG_ARG_CHK_EXT_EN  illegally #define'd in 'app_cfg.h'"
#error  "                             [MUST be  DEF_DISABLED]           "
#error  "                             [     ||  DEF_ENABLED ]           "
#endif

#ifndef  LIB_MEM_CFG_POOL_EN
#error  "LIB_MEM_CFG_POOL_EN               not #define'd in 'app_cfg.h'"
#error  "                             [MUST be  DEF_DISABLED]           "
#error  "                             [     ||  DEF_ENABLED ]           "

#elif  ((LIB_MEM_CFG_POOL_EN != DEF_DISABLED) && \
        (LIB_MEM_CFG_POOL_EN != DEF_ENABLED ))
#error  "LIB_MEM_CFG_POOL_EN         illegally #define'd in 'app_cfg.h'"
#error  "                             [MUST be  DEF_DISABLED]           "
#error  "                             [     ||  DEF_ENABLED ]           "

#elif   (LIB_MEM_CFG_POOL_EN == DEF_ENABLED)

#ifndef  LIB_MEM_CFG_HEAP_SIZE
#error  "LIB_MEM_CFG_HEAP_SIZE             not #define'd in 'app_cfg.h'"
#error  "                             [MUST be  > 0]                    "

#elif   (LIB_MEM_CFG_HEAP_SIZE < 1)

#error  "LIB_MEM_CFG_HEAP_SIZE       illegally #define'd in 'app_cfg.h'"
#error  "                             [MUST be  > 0]                    "
#endif

#endif
```

Templates

Listing 12 — Source file template

```
/*
*********************************************************************************************************
*                                             uC/PRODUCT
*                                          The Embedded Product
*
*                          (c) Copyright 2004-2012; Micrium, Inc.; Weston, FL
*
*               All rights reserved.  Protected by international copyright laws.
*
*               uC/PRODUCT is provided in source form to registered licensees ONLY.  It is
*               illegal to distribute this source code to any third party unless you receive
*               written permission by an authorized Micrium representative.  Knowledge of
*               the source code may NOT be used to develop a similar product.
*
*               Please help us continue to provide the Embedded community with the finest
*               software available.  Your honesty is greatly appreciated.
*
*               You can contact us at www.micrium.com.
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*
*                                           FILE DESCRIPTION
*
* Filename      :
* Version       :
* Programmer(s) :
*********************************************************************************************************
* Note(s)       :
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                            INCLUDE FILES
*********************************************************************************************************
*/

#define    MICRIUM_SOURCE
#define    TEMPLATE_MODULE
#include  <template.h>
```

```
/*
*********************************************************************************************************
*                                     EXTERNAL C LANGUAGE LINKAGE
*
* Note(s) : (1) C++ compilers MUST 'extern'ally declare ALL C function prototypes & variable/object
*               declarations for correct C language linkage.
*********************************************************************************************************
*/

#ifdef __cplusplus
extern  "C" {                                   /* See Note #1.                                         */
#endif

/*
*********************************************************************************************************
*                                            LOCAL DEFINES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                           LOCAL CONSTANTS
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                          LOCAL DATA TYPES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                            LOCAL TABLES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                       LOCAL GLOBAL VARIABLES
*********************************************************************************************************
*/
```

```
/*
************************************************************************************************************
*                                        LOCAL FUNCTION PROTOTYPES
************************************************************************************************************
*/

/*
************************************************************************************************************
*                                       LOCAL CONFIGURATION ERRORS
************************************************************************************************************
*/

/*
************************************************************************************************************
************************************************************************************************************
*                                            GLOBAL FUNCTIONS
************************************************************************************************************
************************************************************************************************************
*/

/*
************************************************************************************************************
************************************************************************************************************
*                                            LOCAL FUNCTIONS
************************************************************************************************************
************************************************************************************************************
*/

/*
************************************************************************************************************
*                                     EXTERNAL C LANGUAGE LINKAGE END
************************************************************************************************************
*/

#ifdef __cplusplus
}                                               /* End of 'extern'al C lang linkage.                    */
#endif
```

Listing 13 — Header file template

```
/*
*********************************************************************************************************
*                                             uC/PRODUCT
*                                        The Embedded Product
*
*                          (c) Copyright 2004-2012; Micrium, Inc.; Weston, FL
*
*               All rights reserved.  Protected by international copyright laws.
*
*               uC/PRODUCT is provided in source form to registered licensees ONLY.  It is
*               illegal to distribute this source code to any third party unless you receive
*               written permission by an authorized Micrium representative.  Knowledge of
*               the source code may NOT be used to develop a similar product.
*
*               Please help us continue to provide the Embedded community with the finest
*               software available.  Your honesty is greatly appreciated.
*
*               You can contact us at www.micrium.com.
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*
*                                          FILE DESCRIPTION
*
* Filename      :
* Version       :
* Programmer(s) :
*********************************************************************************************************
* Note(s)       :
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                               MODULE
*********************************************************************************************************
*/

#ifndef  TEMPLATE_MODULE_PRESENT
#define  TEMPLATE_MODULE_PRESENT
```

```
/*
*********************************************************************************************************
*                                             INCLUDE FILES
*
* Note(s) : (1) The following common software files are located in the following directories :
*
*               (a) \<Custom Library Directory>\lib*.*
*
*               (b) (1) \<CPU-Compiler Directory>\cpu_def.h
*
*                   (2) \<CPU-Compiler Directory>\<cpu>\<compiler>\cpu*.*
*
*                   where
*                   <Custom Library Directory>      directory path for custom   library      software
*                   <CPU-Compiler Directory>        directory path for common   CPU-compiler software
*                   <cpu>                           directory name for specific processor (CPU)
*                   <compiler>                      directory name for specific compiler
*
*           (2) Compiler MUST be configured to include the '\<Custom Library Directory>\uC-LIB\',
*               '\<CPU-Compiler Directory>\' directory, & the specific CPU-compiler directory as
*               additional include path directories.
*
*           (3) NO compiler-supplied standard library functions SHOULD be used.
*********************************************************************************************************
*/

#include  <cpu.h>
#include  <lib_def.h>
#include  <app_cfg.h>

/*
*********************************************************************************************************
*                                       EXTERNAL C LANGUAGE LINKAGE
*
* Note(s) : (1) C++ compilers MUST 'extern'ally declare ALL C function prototypes & variable/object
*               declarations for correct C language linkage.
*********************************************************************************************************
*/

#ifdef __cplusplus
extern  "C" {                                   /* See Note #1.                                         */
#endif
```

```
/*
*********************************************************************************************************
*                                                EXTERNS
*********************************************************************************************************
*/

#ifdef   TEMPLATE_MODULE
#define  TEMPLATE_EXT
#else
#define  TEMPLATE_EXT  extern
#endif

/*
*********************************************************************************************************
*                                           DEFAULT CONFIGURATION
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                                DEFINES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                            DATA TYPES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                         GLOBAL VARIABLES
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                              MACROS
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                         FUNCTION PROTOTYPES
*********************************************************************************************************
*/
```

```
/*
*********************************************************************************************************
*                                     EXTERNAL C LANGUAGE LINKAGE END
*********************************************************************************************************
*/

#ifdef __cplusplus
}                                                       /* End of 'extern'al C lang linkage.                    */
#endif

/*
*********************************************************************************************************
*                                          CONFIGURATION ERRORS
*********************************************************************************************************
*/

/*
*********************************************************************************************************
*                                              MODULE END
*********************************************************************************************************
*/

#endif                                                                  /* End of template module include.      */
```

Variables

Global Variables

Global variables are declared in product/module header files as shown in Listing 1 and Listing 2. Global variables are often discouraged in order to more appropriately hide and control each module's/class's data. However, a product that references all its modules' variables from a debug or statistics module might require global vaiables as shown in Listing 3, while each module could still hide or encapsulate the majority of its global vaiable data within the module.

Listing 1 — Global Variable Declarations in Header File

```
#ifdef   MODULE                                                                         (1)
#define  MODULE_EXT                                                                     (2)
#else
#define  MODULE_EXT  extern                                                             (3)
#endif

  ...

MODULE_EXT  DATA_TYPE  VariableName;        /* Brief variable comment explaining use. */  (4)
```

(1) Each module header file dynamically defines its module **MODULE_EXT**:

(2) When module is compiled, **MODULE** is **#define**'d & **MODULE_EXT** is defined to null, thereby declaring this module's global variables later in the file.

(3) When another module is compiled, **MODULE** is not **#define**'d & **MODULE_EXT** is defined to **extern**, thereby informing all other modules about this this module's global variables.

(4) Declare header file global variables with dynamically-defined **MODULE_EXT**.

Listing 2 — Global Variable Declaration Example

```
NET_TMR_EXT  NET_TMR         NetTmr_Tbl[NET_TMR_CFG_NBR_TMR];                                    (1)
NET_TMR_EXT  NET_TMR        *NetTmr_PoolPtr;      /* Ptr to pool of free net tmrs.           */  (2)
NET_TMR_EXT  NET_STAT_POOL   NetTmr_PoolStat;                                                   (3)
NET_TMR_EXT  NET_TMR        *NetTmr_TaskListHead; /* Ptr to head of Tmr Task List.           */
NET_TMR_EXT  NET_TMR        *NetTmr_TaskListPtr;  /* Ptr to cur Tmr Task List tmr to update. */  (4)
```

(1) Declare variables with **MODULE_EXT**

(2) One variable per line

(3) Two spaces following longest qualifier(s) and longest data types; all other variable qualifiers, data types, and names to be vertically aligned.

(4) Pointer operator(s) should immediately prefix variable names.

Listing 3 — Example µC/TCP-IP `net_dbg.h` *Global Variable Data Size Statistics*

```
                                                        /* ------- NET TMR  MODULE -------- */
const  CPU_INT32U  NetTmr_DataSize   =  0u
                                    +  sizeof(NetTmr_Tbl)
                                    +  sizeof(NetTmr_PoolPtr)
                                    +  sizeof(NetTmr_PoolStat)
                                    +  sizeof(NetTmr_TaskListHead)
                                    +  sizeof(NetTmr_TaskListPtr)
                                    ;

                                                        /* ------- NET BUF  MODULE -------- */
const  CPU_INT32U  NetBuf_DataSize   =  0u
                                    +  sizeof(NetBuf_PoolsTbl)
                                    +  sizeof(NetBuf_ID_Ctr)
                                    ;

#ifdef  NET_CONN_MODULE_PRESENT                         /* ------- NET CONN MODULE -------- */
const  CPU_INT32U  NetConn_DataSize  =  0u
                                    +  sizeof(NetConn_Tbl)
                                    +  sizeof(NetConn_PoolPtr)
                                    +  sizeof(NetConn_PoolStat)
                                    +  sizeof(NetConn_ConnListHead)
                                    +  sizeof(NetConn_ConnListChainPtr)
                                    +  sizeof(NetConn_ConnListConnPtr)
                                    +  sizeof(NetConn_ConnListNextChainPtr)
                                    +  sizeof(NetConn_ConnListNextConnPtr)
                                    +  sizeof(NetConn_AccessedTh_nbr)
                                    +  sizeof(NetConn_AddrWildCardAvail)
                                    +  sizeof(NetConn_AddrWildCard)
                                    ;
```

File-scope Global Variables

File-scope global variables, also known as local global variables (or external variables), are declared in product/module source files as shown in Listing 4 and Listing 5. File-scope global variables more appropriately encapsulate a module's data but may or may not appropriately hide and control a module's data.

Listing 4 — Local Global Variable Declarations in Source File

```
static  DATA_TYPE  VariableName;  /* Brief variable comment explaining use. */          (1)
```

(1) Declare source file global variables as **static**

Listing 5 — Local Global Variable Declaration Example

```
static  DNSc_CACHE_STRUCT  DNSc_HostName[DNSc_MAX_CACHED_HOSTNAMES];                  (1)
static  CPU_INT32U         DNSc_CurOrder;                                            (2)
static  NET_IP_ADDR        DNSc_Server;                                              (3)
static  CPU_INT16U         DNSc_QueryID;
static  CPU_INT32U         DNSc_LastErr;                                             (4)
```

(1) Declare variables as **static**

(2) One variable per line

(3) Two spaces following longest qualifier(s) and longest data types; all other variable qualifiers, data types, and names to be vertically aligned.

(4) Pointer operator(s) should immediately prefix variable names.

Function-scope Local Variables

See Function Local Variables.

Contact

Micriμm, Inc.
949 Crestview Circle
Weston, FL 33327
954-217-2036
954-217-2037 (FAX)

WEB: www.Micrium.com