Micriµm, Inc.

Acronyms, Abbreviations and Mnemonics Dictionary

Application Note

AN-2001

Jean J. Labrosse
Jean.Labrosse@Micrium.com
www.Micrium.com

1.00 Introduction

When creating names for variables and functions (identifiers), it is often the practice to use acronyms (e.g. `OS`, `ISR`, `TCB` and so on), abbreviations (`buf`, `doc` etc.) and mnemonics (`clr`, `cmp`, etc.). The use of acronyms, abbreviations and mnemonics (from now on AAM) allows an identifier to be descriptive while requiring fewer characters. Unfortunately, if AAMs are not used consistently, they may add confusion. To ensure consistency, we have created a list of AAMs that we use in all our projects. The same AAM is used throughout, once it is assigned. I call this list the *Acronym, Abbreviation and Mnemonic Dictionary* and the list for Micriµm is shown in Table 1. As we need more AAMs, we will revise this document.

There might be instances where our doesn't make sense for your products/projects. For instance, if you are an engineering firm working on a project for different clients and the products that you develop are totally unrelated, then a different list for each project would be more appropriate; the vocabulary for the farming industry is not the same as the vocabulary for the defense industry. We use the rule that if all products are similar, they use the same dictionary.

A common dictionary to a project team will also increase the team's productivity. It is important that consistency be maintained throughout a project, irrespective of the individual programmer(s). Once `buf` has been agreed to mean *buffer* it should be used by all project members instead of having some individuals use *buffer* and others use `bfr`. To further this concept, you should always use `buf` even if your identifier can accommodate the full name; stick to `buf` even if you can fully write the word *buffer*.

2.00 Creating Identifiers using AAMs

#define constants, macros, variables and function names (i.e. identifiers) should make use of the file name as a prefix. This prefix makes it easy to locate identifier declarations in medium to large projects. It also makes it very easy to know where these identifiers are declared and 'belong' to. For example, all functions in a file named `KBD.C` and functions in a file named `VIDEO.C` could be declared as follows:

```
KBD.C
          KbdGetChar()
          KbdGetLine()
          KbdGetFnctKey()

VIDEO.C
          VideoGetAttrib()
          VideoPutChar()
          VideoPutStr()
          VideoSetAttrib()
```

It's not necessary to use the whole file/module name as a prefix. For example, a file called `KEYBOARD.C` could have functions starting with `Kbd` instead of `Keyboard`. It is also preferable to use upper case characters to separate words in an identifier (a.k.a. Camel Back) instead of using underscores. Underscores don't add any meaning to names and they use up character spaces.

As much as possible, use 'module-object-operation' format with AAMs. When creating identifiers, specify the name of the module (or sub-system) first, followed by the object and then the operation as shown below.

```
OSSemPost()
OSSemPend()
```

> Here, the module name is *OS* (Operating System), the object is *Sem* (Semaphore) and the operation that can be performed on the object is *Post* or *Pend*.

What's nice about this technique is that it allows you to group similar items together. Of course, it may be a bit difficult to get used to it in the beginning because it's more natural to think in terms of Action-Object such as `OSPostSem()`. We prefer the Object-Action method because it groups similar objects with their actions. For example:

```
OSSemAccept()
OSSemCreate()
OSSemDel()
OSSemPost()
OSSemPend()
OSSemQuery()
```

We have recently started to append an underscore character after the module name to separate the module from the Object-Action. Using the above function name as examples, the function names would now look as follows:

```
OS_SemAccept()
OS_SemCreate()
OS_SemDel()
OS_SemPost()
OS_SemPend()
OS_SemQuery()
```

3.00 AAM Dictionary

The table below is the current Micriμm's AAM dictionary. The list is subject to change but would most likely only change by adding names. Once an AAM is assigned, it should not be changed.

Absolute	**Abs**
Action	**Act**
Active	Active
Add	Add
Address	**Addr**
Address Resolution Protocol	**ARP**
Adjust	**Adj**
Alarm	**Alm**
Alarming	**Alarming**
Application	**App**
Analog Input	**AI**
Analog Output	**AO**
Analog to Digital Converter	**ADC**
Assign	Assign
Assignment	**Assign**
Attributes	**Attrib**
Automatic	**Auto**
Auxiliary	**Aux**
Average	**Avg**
Background	**Bgnd**
Bank	Bank
Binary	**Bin**
Bit	Bit
Buffer	**Buf**
Cache	Cache
Calculate	**Calc**
Calculation(s)	**Calc**
Calibration	**Cal**
Change	**Chng**
Channel	**Ch**
Character	**Chr**
Check	**Chk**
Clear	**Clr**
Clock	**Clk**
Column	**Col**
Command	**Cmd**
Communications	**Comm**
Compare	**Cmp**
Complement	**Cpl**
Computation	**Comp**
Compute	**Comp**
Condition	**Cond**
Configuration	**Cfg**
Configure	**Cfg**
Consecutive	**Consec**
Constant	**Const**
Control	**Ctrl**

Control Byte	**CB**
Control Word	**CW**
Convention	**Conv**
Conversion	**Conv**
Count(s)	**Cnt(s)**
Counter	**Ctr**
Create	Create
Creation	**Create**
Current	**Cur**
Date	Date
Day	Day
Debug	**Dbg**
Decrement	**Dec**
Default	**Dflt**
Defined	**Def**
Definition	**Def**
Delay	**Dly**
Delete	**Del**
Denominator	**Denom**
Depth	Depth
Destination	**Dest**
Detect	Detect
Development	**Dev**
Device	**Dev**
Digit	**Dig**
Digital	**Dig**
Digital to Analog Converter	**DAC**
Direction	**Dir**
Directory	**Dir**
Disable	**Dis or DISABLE**
Disabled	**Dis or DISABLED**
Discharge	**Disc**
Disk	Disk
Display	**Disp**
Divider	**Div**
Divisor	**Div**
Driver	**Drv**
Down	Down
Duration	**Dur**
Dynamic	**Dyn**
Edge	Edge
Efficiency	**Eff**
Elapsed	Elapsed
Enable	**En or ENABLE**
Enabled	**En or ENABLED**
Engineering Units	**EU**
Entry	Entry
Error	**Err**
Ethernet	**Ether**
Exception	**Except**
Extension	**Ext**
Extern	**Ext**
External	**Ext**
Family	**Fam**

Fault	**Flt**
Find	Find
Flag	Flag
Flow	Flow
Flush	Flush
Flywheel	**Fly**
Force	Force
Forced	Forced
Foreground	**Fgnd**
Fraction	**Fract**
Frame	Frame
Frequency	**Freq**
Function	**Fnct**
Ground	**Gnd**
Group	**Grp**
Hardware	**Hard**
Header	**Hdr**
Hour(s)	**Hr**
I.D. #	**ID**
Identifier	**ID**
Inactive	**Inact**
Increment	**Inc**
Index	**Ix**
Initialize	**Init**
Input	**In**
Input/Output	**IO**
Interface	**IF**
Internet Control Message Protocol	**ICMP**
Internet Protocol	**IP**
Interrupt	**Int**
Interrupt Service Routine	**ISR**
Jump	Jump
Key	Key
Keyboard	**Kbd**
Lamp	Lamp
Last	Last
Latch	Latch
Liquid Crystal Display	**LCD**
Light Emitting Diode	**LED**
Left	Left
Length	**Len**
Level	Level
Limit	**Lim**
Linear	Line
Linear	**Lin**
Link	Link
List	List
Local	Local
Location	**Local**
Manual	**Man**
Map	Map
Mask	**Msk**
Master	Master
Maximum	**Max**

Measurement	**Meas**
Memory	**Mem**
Message	**Msg**
Minimum	**Min**
Minute	**Min**
Mode	Mode
Module	Module
Modulo	**Mod**
Multiplier	**Mult**
Negative	**Neg**
Network	**Net**
Next	Next
Nominal	**Nom**
Number	**Nbr**
Numerator	**Numer**
Offset	Offset
Operating System	**OS**
Operation	**Oper**
Optimize	**Opt**
Optimizer	**Opt**
Option	**Opt**
Optional	**Opt**
Ordinal	**Ord**
Origin	**Org**
Output	**Out**
Over	**Ovr**
Overflow	**Ovrf**
Packet	**Pkt**
Parameter	**Param**
Password	**Pswd**
Pattern	Pattern
Point	**Pt**
Pointer	**Ptr**
Port (i.e. I/O)	Port
Positive	**Pos**
Point to Point Protocol	**PPP**
Power	**Pwr**
Previous	**Prev**
Priority	**Prio**
Procedure	**Proc**
Process	Process
Product	**Prod**
Protocol	Protocol
Protect	**Prot**
Public	**Pub**
Pulse Width	**PW**
Pulse Width Modulation	**PWM**
Qualified	**Qual**
Qualifier	**Qual**
Qualify	**Qual**
Quantity	**Qty**
Query	Query
Queue	**Q**
Quick	Quick

Range	Range
Reverse Address Resolution Protocol	**RARP**
Ratio	Ratio
Read	**Rd**
Ready	**Rdy**
Recall	**Rcl**
Receive	**Rx**
Receiver	**Rx**
Register	**Reg**
Relative	**Rel**
Request	**Req**
Reserved	**Rsvd**
Reset	Reset
Respond	**Resp**
Response	**Resp**
Restart	Restart
ReSynchronize	**ReSync**
Retard	Retard
Retries	Retries
Return	**Rtn**
Right	Right
Row	Row
Run	Run
Scale	Scale
Scaling	Scaling
Schedule	**Sched**
Scheduler	**Sched**
Screen	**Scr**
Second	**Sec**
Select	**Sel**
Semaphore	**Sem**
Send	Send
Sender	Sender
Sense	Sense
Sensitivity	**Sens**
Sensor	Sensor
Serial Line Interface Protocol	**SLIP**
Serial Number	**SN**
Service	**Serv**
Set	Set
Setpoint	**Stp**
Shutdown	**Sd**
Size	Size
Slave	Slave
Slope	Slope
Socket	**Sock**
Software	**Soft**
Source	**Src**
Specification	**Spec**
Speed	**Spd**
Stack	**Stk**
Standard	**Std**
Start	Start
State	State

State Machine	**SM**
Static	Static
Statistic(s)	**Stat**
Status	Status
Stop	Stop
Storage	**Sto**
Store	**Sto**
Symbol	Symbol
Synchronize	**Sync**
System	**Sys**
System Network Management Protocol	**SNMP**
Table	**Tbl**
Target	Target
Task	Task
Task Control Block	**TCB**
Test	Test
Text	Text
Time	Time
Time Base	**TB**
Timeout	Timeout or TO
Timer	**Tmr**
Time To Live	**TTL**
Timing	Timing
Transmit	**Tx**
Transmission Control Protocol	**TCP**
Underflow	**Undf**
User Datagram Protocol	**UDP**
Units	Units
Update	Update
Upload	Upload
User	User
Valid	Valid
Value(s)	**Val**
Variable(s)	**Var**
Vector	Vector
Verify	Verify
Version	**Ver**
Volume	**Vol**
Width	Width
Write	**Wr**

References

μC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

Contacts

Micriμm, Inc.
949 Crestview Circle
Weston, FL 33327
954-217-2036
954-217-2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: www.Micrium.com

R&D Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
(785) 841-1631
(785) 841-2624 (FAX)
WEB: http://www.rdbooks.com
e-mail: rdorders@rdbooks.com