**Revision History**

|  |  |  |
| --- | --- | --- |
| **Author** | **Date Modified** | **Reason** |
| Naveen Raj Selvaraj | March 16, 2016 | Initial creation |
| Naveen Raj Selvaraj | May 6, 2016 | Edited Datatypes section (portability) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[1. Creating a new module 2](#_Toc446077334)

[1.1. src direcotry 2](#_Toc446077335)

[1.2. inc direcotry 2](#_Toc446077336)

[1.3. obj and exe directories 2](#_Toc446077337)

[1.4. Makefile 2](#_Toc446077338)

[2. Coding guidelines 3](#_Toc446077339)

[2.1. Macros 3](#_Toc446077340)

[2.2. Typedefinitions 3](#_Toc446077341)

[2.3. Variables 3](#_Toc446077342)

[2.3.1. Datatypes 3](#_Toc446077343)

[2.3.2. Declaration 3](#_Toc446077344)

[2.3.3. Variable Names – Do's and Dont's 4](#_Toc446077345)

[2.4. Functions 5](#_Toc446077346)

[2.5. Punctuations 5](#_Toc446077347)

1. Files and Directories

A new module can be created by creating a directory with <module\_name> and having the following files and directories inside the directory <module\_name>,

1. src/

2. inc/

3. obj/

4. exe/

5. make.h

6. Makefile

The module should be named in all lowercase without any space or special characters. There is no hard limit on the name length. But it should be as short as possible.

* 1. src direcotry

This directory contains all the source files (.c) for the module. The files should be named as <module\_name><file\_name>.c. For example, the module name is node, the files can be named as nodemain.c (file that contains main function), nodeutil.c (file that contains the utility functions used by main), nodeapi.c (file that provides API support to other modules).

* 1. inc direcotry

This directory contains the header files related to this module. This directory should have the following header files.

<module\_name>def.h - contains all the MACRO definitions for the module

<module\_name>glob.h - contains all the global variables used by the module

<module\_name>proto.h - contains the function declarations

<module\_name>tdfs.h - contains the type definitions (structure, enum definitions)

<module\_name>inc.h - This file includes all the above files and other standard include files as required by the module. The idea is to make any source file in this module compilable, simply by including this single header file.

* 1. obj and exe directories

These directories would be used to keep the object (.o) files and the executable files respectively after compilation.

* 1. Makefile

The make.h contains the variables used for the compilation. Makefile includes make.h and contains the compilation rules.

1. Coding guidelines
   1. Macros

Macros should be defined in the '<module\_name>defs.h' file. The macros should be named in all UPPER CASE and only '\_' can be used as separator.

(e.g.) #define SUCCESS 0

#define FAILURE -1

#define MAX\_BLOCK\_SIZE 4096

#define MAX\_STR\_LEN 256

#define MAX(x, y) (x > y ? x : y)

* 1. Typedefinitions

Structures, unions and enums should be type defined in the '<module\_name>tdfs.h' file.

|  |  |
| --- | --- |
| **Struct / Union** | **Enum** |
| typedef struct/union ConnData\_t  {  ...  } tConnData; | typedef enum FileTypes\_e  {  FILE\_TYPE\_FILE, // should be UPPERCASE  FILE\_TYPE\_DIR,  FILE\_TYPE\_INVALID  }tFileType; |

* 1. Variables
     1. Datatypes

Avoid using the primitive types (char, int, long). Use the data types defined in cmntdfs.h (CHAR, INT1, UINT1, INT2, UINT2, INT4, UINT4, INT8, UINT8, VOID). This enables portability. For example, ‘long’ takes 4 bytes or 8 bytes depending on the machine architecture. So, never use the primitive types.

* + 1. Declaration

|  |  |  |
| --- | --- | --- |
| **Type** | **Naming convention** | **Example** |
| Character, Structures, Unions, Enums | Variable name should start with uppercase | CHAR Var1;  CHAR UserInput;  struct ext3\_inode Inode;  tEnum Colors;  tConnData ConnSocket; |
| Integer | <i/u><SizeInBytes><VariableName> i or u – signed or unsigned, depending on the data type  size – 1, 2, 4 or 8, depending on the data type | UINT1 u1Value;  INT1 i1IsValid;  UINT2 u2Index1;  INT2 i2NewValue;  UINT4 u4Index2;  INT4 i4RetVal;  UINT8 u8Offset;  INT8 i8Result; |
| Array | Similar to Integer and prefixed with 'a' to indicate that the variable is an array | UINT1 au1Inodes[MAX\_INODES]; INT4 ai4ConnStat[MAX\_CONN]; |
| Pointers | Pointers should be named similar to CHAR variables and prefixed with 'p' to indicate that the variable is a pointer.  \*p<VariableName> | struct ext3\_dir \*pDirEntry;  INT4 \*pVal; |
| Global Variables | If the variable is declared as a global variable, a 'g' should be prefixed and the naming convention for it's type should be followed. | UINT1 gu1Flag;  UINT1 gau1Array[MAX];  INT1 \*gpBlock;  INT4 gi4ConnSocket; |

* + 1. Variable Names – Do's and Dont's
* Do not use any special characters in the variable name. (e.g) i4Ret\_val or i4Ret-val. Instead name the variable as i4RetVal.
* Do not use i, j, etc for index variables. Use u4Index1, u4Index2 as index variable names.
* Declare and Initialize separately. Declare all the variables needed for a function block and then initialize all the variables. Use memcpy and memset for initializing structures, arrays.

(e.g) Don't

/\* Declaration and Initialization \*/

INT4 i4RetVal = 0;

CHAR au1Buffer[MAX\_BUFF\_SIZE];

memset(au1Buffer, 0, MAX\_BUFF\_SIZE);

CHAR \*pBuff = Buffer;

Do

/\* Declaration \*/

INT4 i4RetVal;

CHAR au1Buffer[MAX\_BUFF\_SIZE];

CHAR \*pBuff;

/\* Initialization \*/

i4RetVal = 0;

memset(au1Buffer, 0, MAX\_BUFF\_SIZE);

pBuff = Buffer;

* A variable must be initialized before it’s usage.
* Use meaningful variable names and keep the names as short as possible.
* Never pass a structure/array variable as an argument to a function. Instead pass the address of the structure/array variable as the argument.

(e.g) /\* Don't do this \*/

void Function(struct ext3\_dir\_entry DirEntry)

/\* Recommended way \*/

void Function(struct ext3\_dir\_entry \*pDirEntry)

* 1. Functions
* Functions should be named as

<FileName><FuncName>( )

(e.g) InodeUtilPrintInode() - function is in inodeutil.c

NodeLogDumpPacket() - funtion is in nodelog.c

* There should be “NO SPACE” between the function name and the parentheses

(e.g) definition: VOID FunctionName () should be VOID FunctionName()

calling statement: FunctionName ()should be FunctionName()

* Functions should return SUCCESS/FAILURE (integers) wherever possible. This return values should be verified by the caller function.
* When printing error/debug messages, \_\_func\_\_, \_\_LINE\_\_ should be used to identify the function and line no. that printed the message.

(e.g) printf("ERROR:read error %s:%d\n", \_\_func\_\_, \_\_LINE\_\_);

* The maximumum character limit per line is 80.
  1. Punctuations
* Control statements (if/for/while/switch) and parentheses should be separated by a SPACE

(e.g) if (statement)

{

}

for (;;)

{

}

do

{

}while (statment);

* There should NOT be a SPACE after the opening parenthesis ( and before the closing parenthesis ).

(e.g) while ( TRUE ) /\* Don't \*/

while (TRUE) /\* Recommended way – No space after ( and before ) \*/

* A single SPACE should be used after a comma (,)/semicolon(;) and not before.

(e.g) /\* Don't - space before and after ; \*/

for (i4Index = 0 ; i4Index < 10 ; i4Index++)

/\* Do - space only after ; \*/

for (i4Index = 0; i4Index < 10; i4Index++)

/\* Don't - space before and after , \*/

Function(Arg1 , Arg2)

/\* Do - space only after , \*/

Function(Arg1, Arg2)

* All Operators and Operands should be separated by a SPACE,except for unary operators.

(e.g) INT4 i4Result = i4Val1 + i4Val2;

if (u1Flag1 && u1Flag2)

while ((i4Val1 >= i4Val2) && (u1Flag != FALSE))

a > b ? a : b

u4Index++ (++ is a unary operator)

~Bits (~ is a unary operator)

FunctionName(i4Index – 1);

u4GroupNo = (u4InodeNo - 1) / sb.s\_inodes\_per\_group;