補

P A F E E

Pragmatic Application Framework for Embedded Environment

コーディング規約

Rev 1.0

１．一般規則

・パスカルケース（先頭を大文字）にする（〇：GetLength　X：getLength）。
・限定的にハンガリアン記法を使用する（12章参照）。
・長すぎる単語以外は基本的に省略しない（〇：MemoryManager　X：MemMng）。
・名前付けは全て英語にする（カナは使用禁止）。
・アスキーコードのみで記述する（漢字などの2バイトコードは使用しない）。
・doxygen によるドキュメント化ができるようにする。

２．ファイル

・命名規則は一般規則に従う。
・基本的に1クラスにつき、1個づつヘッダファイル(.h)とソースファイル(.cpp)を作成する。
・クラス内部にクラスを宣言する場合はまとめてもよい。
・ファイル名はクラス名と同じにする。
・ヘッダファイルには多重インクルードの問題を回避するように、下記のような定義を追加する。

例） MemoryManager.h の場合

```
#ifndef _MEMORY_MANAGER_H_INCLUDED_
#define _MEMORY_MANAGER_H_INCLUDED_
クラス宣言
#endif
```

３．名前空間

・名前空間として Pafee を定義する。
・本プログラムの全てのクラスはこのネームスペース内に宣言、定義する。

４．クラス

・命名規則は一般規則に従うが、クラスを示すプレフィックスは用いない（〇：MemoryManager　X：CMemoryManager）

５．構造体型宣言

・構造体を typede する場合には全て大文字とし、単語の区切りにはアンダーバーを使用する(〇：LIST_ELEMENT　X：ListElement)
・構造体を示すプレフィックス、サフィックスは付加しない（〇：LIST_ELEMENT　×：LIST_ELEMENT_T）。

６． 定数

・全て大文字とし、単語の区切りにはアンダーバーを使用する（〇：MAX_SIZE　X：MaxSize）。

・定数を示すプレフィックス、サフィックスは付加しない。

・define ではなく static const を用いてクラスメンバとして定義する（データ型が明確になるため）。

（static const の場合は基本的はグローバル変数扱いだが、コンパイラの最適化により定数と同じ扱いになる場合が多い→要検証）

・グローバルな定数を定義する場合は先頭に PAFEE_を付加する

例：PAFEE_ENDIAN_BIG

７． 列挙型

・全て大文字とし、単語の区切りにはアンダーバーを使用する（〇：STATE_READY　X：StateReady）。

・データの種類を示す単語を先頭に記述する（〇：STATE_READY　X：READY_STATE）。

・基本的にクラス内で定義する。グローバルな列挙型を定義する必要がある場合は定数と同様に先頭に PAFEE_を付加する。

・　状態やモードなど範囲が決まっているものはできるだけ enum を使用する。

８． ユーザ定義データ型

・typedef でデータ型をユーザ定義する場合は、全て大文字とし、単語の区切りにはアンダーバーを使用する。

・基本的に以下に定義したユーザ定義データ型を使用する（int ,char, short,long についてはそのまま使用しても良い）。

型	説明
BOOL	unsigned int
CHAR	signed char
UCHAR	unsigned char
SHORT	signed short
USHORT	unsigned short
LONG	signed long
ULONG	unsigned long

※　これらは Pafee 名前空間で定義される

９．関数名

・命名規則は一般規則に従う。

・以下のように動詞、名詞の順番で記述する（例：GetLength()）。

VerbNoun

・非 static のグローバルなCの関数を宣言する場合は下記のように先頭に小文字で pafee を追加する。

pafeeVerbNoun

１０．変数名

・命名規則は一般規則に従う。

・先頭は小文字で始まる（例：length）。

・プレフィックスを付加する場合もある（例：pLength）。これについては１２．ハンガリアン記法について“を参照。

name

prefixName

・メンバ変数の場合は先頭に m_を付加する(public/protected/private の違いは表現しない)

m_name

m_prefixName

・static なグローバル変数の場合は先頭に g_を付加する

g_name

g_prefixName

・非 static なグローバル変数の場合は先頭に pafee_を付加する

pafee_name

pafee_prefixName

※グローバル変数の利用は極力避けること

１１．ソースコードの記述

１１．１　ファイルヘッダ

doxygen でドキュメント化できるように以下のフォーマットでファイルの先頭にファイルヘッダを付ける。

```
//*****************************************************
//
//          project         PAFEE
//!          @file:          MemoryManager.cpp
//!          @author:     Your Name
//!          @brief:        Memory management class
//
//*****************************************************
```

著作権やライセンスに関する説明をファイルヘッダの直後に配置する

１１．２　関数ヘッダ

doxygen でドキュメント化できるように以下のフォーマットで関数の上に関数ヘッダを付ける。

コメントには入出力がわかるように先頭に[in][out][in/out]を付ける。

```
//-------------------------------------------------------------------------
//!   Get user name
//!   @return: Result (TRUE:Success FALSE:failure)
//-------------------------------------------------------------------------
BOOL    UserManager::GetUserName(
                                        ULONG       userID,             ///< [in] User identifier
                                        char*       szUserName,     ///< [out] User name buffer
                                        ULONG       bufferSize          ///< [in] User name buffer size
                                        )
{
      XXXX
}
```

１１．３　インデント・タブ

・インデントには 1 個のタブを使用する（スペースではなく）。

・インデント以外にはタブを使用しない

１１．４　コメント

コメントは全て英語で記述する。

１１．５　括弧

・波括弧は独立した行とする。

〇:

```
if(a == 10)
{
}
```

X:

```
if(a == 10){
}
```

１１．６　条件式

・条件式には優先順位に関わらず意味単位で括弧でくくる

〇:

```
if((a == 1) || (a == 2))
```

X:

```
if(a == 1 || a == 2)
```

・条件式内のスペースについては特に規定を設けないが、下記のようなフォーマットが望ましい

```
if((a == 1) || (a == 2))
```

・switch 文は下記のようなフォーマットで記述する

```
switch(state)
{
case STATE_IDLE
    xxxx
    break;
case STATE_READY
    {
        int a;
        xxx
    }
    break;
default:
    break;
}
```

※default は必ず付けること

１１．７　エンディアン

・エンディアン依存コードは#ifdef で分ける。

・エンディアンを示す定数シンボルは以下となる

　ビッグエンディアン：PAFEE_ENDIAN_BIG

　リトルエンディアン：PAFEE_ENDIAN_LITTLE

・デフォルトはリトルエンディアンなるようにコーディングする（ビッグエンディアンの場合はコンパイル時に指定する）

　例）

```
#ifdef PAFEE_ENDIAN_BIG
        xxx
#else
        xxxx
#endif
```

１１．８　例外

例外の利用は最低限にとどめる

API レベルのメソッドについては throw()を付けて自分の送出する例外を明示する。

１１．９　その他の留意点

・変更しない変数、関数については、できるだけ const を付ける。

・アラインに配慮する。最低4バイトアラインにデータを配置すること。

・

・

・

１２．ハンガリアン記法について

データ型に対してプレフィックスをつけるシステムハンガリアン記法は原則として使わないが以下の４点については、コードの可読性の向上に役立つと考えられるため例外的に採用する。

例外的に利用するシステムハンガリアン記法

プレフィックス	意味	例
p	ポインタ	pBuffer,pLength
pp	ポインタのポインタ	ppBuffer, ppLength
is/has/can	ブーリアン	isReady, hasData
sz	文字列	szLog

使用してはいけないシステムハンガリアン記法の例

プレフィックス	意味	例
ul	ULONG	ulLength
dw	ULONG	dwLength
b	ブーリアン	bReady