はじめに

～　文書としてのソースファイル　～

読むためのソースコード

コメント規約に従ってソースファイルを記述することは、慣れないと大変面倒です。しかし,コメント規約を決めた場合と決めない場合のデバッグ効率を比較したところ,決めた方が３０%も効率アップしたという事例があります。それは,全て違う形式で書かれた文書と統一形式で書かれた文書を読んで,そして理解する差と同じです。
ソースファイルも実は,人間が読んで理解するための文書です。
その文書を複数の人が記述するためには,それなりのルールが必要です。
そこで,いろいろな開発プロジェクトで採用されている,さまざまなコメント規約について,その内容を比較検討し,良いところを抜き出し,本書を作成しました。

ドキュメント自動生成ツール【A HotDocument】

本文中では,ドキュメント自動生成ツール【A HotDocument】(以下,【A HotDocument】と略す)について触れています。本書のコメント規約に従えば,【A HotDocument】でさらに詳細なドキュメントが生成できます。
【A HotDocument】とは,下記の特長を持ったツールです。

・最高品質の納品用ドキュメント

【A HotDocument】は,ソースファイルから納品用のドキュメントを自動生成するツールです。面倒なドキュメント作成作業を無くし,確実に工数の削減を可能にしました。市販パッケージソフトのマニュアル以上の最高品質ドキュメントは,社内システムの内部資料としてもご利用いただけます。

・充実した製品ラインナップ

【A HotDocument】はVisual Basic、AccessをはじめVisual C++、VisualC#、Visual J#,Java,C#Builder,C++Builder,Excelまで対応しています。そのため,異なる言語でも統一されたドキュメント体系を得ることができます。

【A HotDocument】製品案内URL：http://www.hotdocument.net/

目次

第1章
Visual Basic/Access/Excelコメント規約

本章では,Visual Basic/Access/Excelのコメントをどのように記述するかを述べる。

1.1 ソースファイルの記述例

ソースファイルの記述例を以下に示す。
ソースファイルの履歴のコメント,プロシージャの頭書きなどは以下のように記述する。

■ 宣言部の記述例

```
' @(h) Select.vb ver 1.1 ( '07.09.15 今井 浩司 )
' @(h) Select.vb ver 1.0 ( '07.09.13 今井 浩司 )

' @(s)
'   機能選択プロシージャ
'   本プロシージャは使用する機能を選択するものである。

Option Explicit

Private Structure SYMBOLINFO        ' 宣言文字構造
     Dim SyDataType As String       ' As宣言文字列
     Dim SyDataTypeDef As String    ' 型宣言文字
End Structure

Private Type TYPE_SYMBOLINFO        ' ユーザ型宣言文字構造
     Dim TpSyDataType As String     ' As宣言文字列
     Dim TpSyDataTypeDef As String  ' 型宣言文字
End Type

Public Enum WeekDay   ' 一週間
     Sun = 1          ' 日曜日
     Mon = 2          ' 月曜日
     Tue = 3          ' 火曜日
     Wed = 4          ' 水曜日
     Thu = 5          ' 木曜日
     Fri = 6          ' 金曜日
     Sat = 7          ' 土曜日
End Enum
```



```
Public Interface IInfoBlock          ' インフォブロックのインターフェース
     Property Selected() As Boolean  ' 選択情報プロパティ
     Sub UpdateDataSet()             ' アップデートデータ
End Interface

Declare Function ExitWindow Lib "USER" () As Integer ' Windowsの終了

Public Delegate Sub FractalDoneCallback() ' フラクタル終了処理

Public Event ClickFileMenu(ByVal sender As Object, _
                           ByVal Index As Integer) ' クリックメニュー

Public gSymbolData As SYMBOLINFO     ' 宣言文字データ

Public Const APPNAME = "HotDoc"      ' アプリケーション名
Public Const LOGDIR = "C:\LOG"       ' ログ格納ディレクトリ

Public gSelectAppNo As Integer       ' 選択アプリケーション番号
Public gTMPDIR As String             ' テンポラリディレクトリ名

Private Const FRMTITLE = "Select"    ' フォームタイトル
Private Const FRMNO = 1              ' フォーム番号

Private mSelectMouduleNo As Integer  ' 選択モジュール番号
Private mInFileName As String        ' 入力ファイル名

Namespace OpenNewDatas ' 新規データオープン処理
   Public Class AddInNewMenu ' 新規メニュークラス
      Implements Extensibility.IDTExtensibility2 ' 拡張機能

      Const MEMBERMAX = 256  ' メンバーの最大数
      Const ARGSMAX = 256    ' 引き数の最大
         ・・・
   End Class
End Namespace
```

■ モジュール部の記述例

```
'
' 機能      : メッセージ文字列の取得
'
' 返り値    : メッセージ文字列
'
' 引き数    : ARG1 - ファイル番号
'             ARG2 - メッセージインデックス番号
'
' 機能説明  : メッセージ定義ファイルより指定されたメッセージ
'             を取得する
'
' 備考      : 返り値の文字列の最大文字数は80文字
'
Private Function GetMsg(FileNum As Integer, Index As Index) As String

      '' メッセージファイルのオープンをする
      OpenMsgFile(FileNum)

      If (FileNum < 0) Then
            ''' オープンエラーの場合,エラー処理をする
            …
      End If

      ' デバッグ用の処理
      ZZDebug_Print()

      '' メッセージファイルの読み込みをする
      GetMsg = ReadMsgFile(FileNum, Index)

      '' メッセージファイルのクローズをする
      CloseMsgFile(FileNum)

End Function
```


1.2 履歴のコメント

履歴のコメントには,ファイル名,バージョン,作成日/修正日,担当者名を記述する。

■ 履歴のコメントの例

```
' @(h) Select.vb ver 1.1 ( '07.09.15 今井 浩司 )
' @(h) Select.vb ver 1.0 ( '07.09.13 今井 浩司 )
```

1.3 ソースファイルの頭書きのコメント

ソースファイルの頭書きには,ソースファイル全体に対する説明を記述する。

■ ソースファイルの頭書きのコメント例

```
' @(s)
'   機能選択プロシージャ
'   本プロシージャは使用する機能を選択するものである。
```

1.4 構造体/ユーザ型/Enum/Interfaceのコメント

構造体/ユーザ型/Enum/Interfaceのコメントは,次のように記述する。

■ 構造体のコメントの例

```
Private Structure SYMBOLINFO        ' 宣言文字構造
     Dim SyDataType As String       ' As宣言文字列
     Dim SyDataTypeDef As String    ' 型宣言文字
End Structure
```

■ ユーザ型のコメントの例

```
Private Type TYPE_SYMBOLINFO        ' ユーザ型宣言文字構造
     Dim TpSyDataType As String     ' As宣言文字列
     Dim TpSyDataTypeDef As String  ' 型宣言文字
End Type
```

■ Enumのコメントの例

```
Public Enum WeekDay  ' 一週間
     Sun = 1         ' 日曜日
     Mon = 2         ' 月曜日
     Tue = 3         ' 火曜日
     Wed = 4         ' 水曜日
     Thu = 5         ' 木曜日
     Fri = 6         ' 金曜日
     Sat = 7         ' 土曜日
End Enum
```

■ Interfaceのコメントの例

```
Public Interface IInfoBlock          ' インフォブロックのインターフェース
     Property Selected() As Boolean ' 選択情報プロパティ
     Sub UpdateDataSet()            ' アップデートデータ
End Interface
```


1.5 Public/Private定数のコメント

Public/Private定数のコメントは,次のように記述する。

■ Public定数のコメントの例

```
Public Const APPNAME = "HotDoc"     ' アプリケーション名
Public Const LOGDIR = "C:¥LOG"      ' ログ格納ディレクトリ
```

■ Private定数のコメントの例

```
Private Const FRMTITLE = "Select"   ' フォームタイトル
Private Const FRMNO = 1             ' フォーム番号
```

メモ
旧バージョンの【A HotDocument】では、定数、変数などのコメントはシングルクォートを2つ(")記述する必要がありましたが、現バージョンの製品では1つでも構いません。

1.6 Public/Private変数のコメント

Public/Private変数のコメントは,次のように記述する。

■ Public変数のコメントの例

```
Public gSelectAppNo As Integer      ' 選択アプリケーション番号
Public gTMPDIR As String            ' テンポラリディレクトリ名
```

■ Private変数のコメントの例

```
Private mSelectMouduleNo As Integer ' 選択モジュール番号
Private mInFileName As String       ' 入力ファイル名
```

1.7 Declare/Delegate/Eventのコメント

Declare/Delegate/Eventのコメントは,次のように記述する。

■ Declareのコメントの例

```
Declare Function ExitWindow Lib "USER" () As Integer ' Windowsの終了
```

■ Delegateのコメントの例

```
Public Delegate Sub FractalDoneCallback() ' フラクタル終了処理
```

■ Eventのコメントの例

```
Public Event ClickFileMenu(ByVal sender As Object, _
                           ByVal Index As Integer) ' クリックメニュー
```

1.8 Namespace/Implementsのコメント

Namespace/Implementsのコメントは,次のように記述する。

■ Namespace/Implementsのコメントの例

```
Namespace OpenNewDatas                ' 新規データオープン処理
   Public Class AddInNewMenu          ' 新規メニュークラス
      Implements Extensibility.IDTExtensibility2      ' 拡張機能

      Const MEMBERMAX = 256           ' メンバーの最大数
      Const ARGSMAX = 256             ' 引き数の最大
         …
   End Class
End Namespace
```


1.9 プロシージャの頭書きのコメント

プロシージャおよびプロパティの頭書きのコメントは,次のように記述する。
(1)必要のない項目は削除してもよい。
(2)「機能」はプロシージャの説明を一行で記述する。

■プロシージャ頭書きのコメントの例

```
'
' 機能       : メッセージ文字列の取得
'
' 返り値     : メッセージ文字列
'
' 引き数     : ARG1 - ファイル番号
'              ARG2 - メッセージインデックス番号
'
' 機能説明   : メッセージ定義ファイルより指定されたメッセージ
'              を取得する
'
' 備考       : 返り値の文字列の最大文字数は80文字
'
```

VBAのイベントプロシージャ,OCXのイベントプロシージャ,カスタムコントロール,アドインなどのイベントプロシージャ頭書きの一行目のコメントは,以下のように記述する。

メモ
旧バージョンの【A HotDocument】では、プロシージャの前に"@(f)" を記述する必要がありましたが、現バージョンの製品では必要ありません。

■ VBAのイベントプロシージャなどの場合

```
' @(e)
'
' 機能       : メッセージ文字列の取得
'
'               …
'
```

1.10 XMLドキュメントコメント

【A HotDocument】for Visual Basic 2005以降の製品は,XMLドキュメントコメントにも対応している。
本書のコメント規約以外にも,クラス,プロシージャの直前で,<summary>,<param>などのタグで記述したコメントの内容をドキュメントに反映できる。
最新の情報は,【A HotDocument】の[ヘルプ(H)]-[コメント規約(C)]に記述している。

1.11 キーワードの変更

プロシージャの頭書きのキーワードは自由に追加,削除できる。
変更する場合には,【A HotDocument】の『ｵﾌﾟｼｮﾝ]メニュー[ﾌﾟﾛｼｰｼﾞｬｺﾒﾝﾄ]より設定をする。

■プロシージャ頭書きのコメントの例

```
'
' 機能      : 文字列の分割
'
' 返り値    : 分割後の文字列
'
' 引き数    : ARG1 - 対象となる文字列
'
' 作成日    : 2007年09月15日
'
' 作成者    : 今井 浩司
'
' 機能説明 : 文字列を分割して配列にセットする。
'             最大配列数を超えたら,エラーメッセージを出力する。
'
' 注意事項 : 最大配列数以上の,配列を作らなければいけない。
```

1.12 プロシージャ中のコメント

【A HotDocument】で生成されるプロシージャ定義書に反映させたいコメントは,シングルクォート2つ('')以降に記述する。
プロシージャ定義書の中で段下げは,シングルクォート(')の数により深くすることができる。

■プロシージャ中のコメントの例

```
'' メッセージファイルのオープンをする
OpenMsgFile(FileNum)

If (FileNum < 0) Then
      ''' オープンエラーの場合,エラー処理をする
      …
End If

' デバッグ用の処理
ZZDebug_Print()

'' メッセージファイルの読み込みをする
GetMsg = ReadMsgFile(FileNum, Index)

'' メッセージファイルのクローズをする
CloseMsgFile(FileNum)
```


第2章
C/C++ コメント規約

本章では,C/C++のコメントをどのように記述するかを述べる。

2.1 ソースファイルの記述例

ソースファイルの記述例を以下に示す。
ソースファイルの履歴のコメント,ソースファイルの頭書きなどは以下の順番で記述する

■ファイルの記述例

```
// @(h) Comment.cpp ver 2.0 ( '07.09.13 今井 浩司 )
// @(h) Comment.cpp ver 1.0 ( '07.09.01 今井 浩司 )

// @(s)
//   ファイルI/O関数群
//     本ファイルはファイル入出力に関する関数群を
//     集めたものである。
//
#define HC_MAX_PATH       128       // フルパス 最大文字列数
#define HC_MAX_EXT         3        // 拡張子 最大文字列数
#define HC_MAX_FILENAME  128        // ファイル名 最大文字列数
#define HC_MAX_DATE       9         // 日付 "YY/MM/DD"
#define NUMBER_OF(a) ¥
     ( sizeof(a) / sizeof(a[0]) )   // 配列の要素数を得る

typedef struct Memory {   // メモリ上のバッファ
  char *s;                // 文字列
  struct Memory *next;    // リスト構造ポインタ
} MEMORY;

union u_tag {   // タグ共用体
  int ival;     // 整数
  float fval;   // 実数
  char *sval;   // 文字列
} u;
```

```
enum Boolean {          // ブール型定数
  False,                // 偽
  True                  // 真
};

class Box {             // 直方体
private:
  int height;           // 縦
  int width;            // 横
  int depth;            // 高さ
public:
  Box(int,int,int);     // コンストラクタ
  int volume(void);     // 容積
  int surface(void);    // 表面積
};
```

■関数定義部の記述例

```
//
// 機能 : 文字位置の取得
//
// 返り値 : 文字位置[byte](検索されなかった場合は 0を返す)
//
// 機能説明 : 文字列の中から,指定文字列を検索し位置を返す。
//
// 備考 : pStrInStr( "a#c#d#eeee", "#" ) -> 2
//        pStrInStr( "a#c#d##eeee", "##" ) -> 6
//        pStrInStr( "abcdeabcde", "AA" ) -> 0
//
int pStrInStr(const char *tarstr,  // 検索される対象の文字列
              const char *srcstr)  // 検索する指定部分文字列
{
  int nsrc = strlen(srcstr);
  int ntar = strlen(tarstr);
  int i;

  /// 文字列の長さ分ループする
  for (i = 0; i < ntar; i++)
     //// ヒットした場合カウントアップし,それを返り値とする
     if (strncmp(tarstr+i, srcstr, nsrc) == 0)
        return i+1;
  return 0;
}
```


2.2 履歴のコメント

ソースファイルの履歴のコメントには,ファイル名,バージョン,作成日/修正日,担当者名を記述する。

■履歴のコメントの例

```
// @(h) Comment.cpp ver 2.0 ( '07.09.13 今井 浩司 )
// @(h) Comment.cpp ver 1.0 ( '07.09.01 今井 浩司 )
```

2.3 ソースファイルの頭書きのコメント

ソースファイルの頭書きには,ソースファイルに含まれる関数群の説明を記述する。

■ソースファイルの頭書きのコメント例

```
// @(s)
//   ファイルI/O関数群
//      本ファイルはファイル入出力に関する関数群を
//      集めたものである。
//
```

2.4 構造体/共用体/列挙定数のコメント

構造体/共用体/列挙定数のコメントは以下のように記述する。

■構造体/共用体/列挙定数のコメントの例

```
typedef struct Memory {   // メモリ上のバッファ
  char *s;                // 文字列
  struct Memory *next;    // リスト構造ポインタ
} MEMORY;

union u_tag {   // タグ共用体
  int ival;     // 整数
  float fval;   // 実数
  char *sval;   // 文字列
} u;

enum Boolean {    // ブール型定数
  False,          // 偽
  True            // 真
};
```

2.5 #defineのコメント

#defineディレクティブのコメントは以下のように記述する。

■#defineのコメントの例

```
#define HC_MAX_FILENAME  128         // ファイル名 最大文字列数
#define NUMBER_OF(a) ¥
      ( sizeof(a) / sizeof(a[0]) )   // 配列の要素数を得る
```

2.6 クラスのコメント

クラスに関するコメントは以下のように記述する。

■クラスのコメントの例

```
class Box {     // 直方体
private:
  int height;   // 縦
  int width;    // 横
  int depth;    // 高さ
public:
  Box(int,int,int);     // コンストラクタ
  int volume(void);     // 容積
  int surface(void);    // 表面積
};
```

2.7 関数の頭書きのコメント

関数の頭書きのコメントは次のように記述する。
(1)必要のない項目は削除してもよい。
(2)「機能」は関数の説明を一行で記述する。

■関数の頭書きのコメント例

```
//
// 機能      : 文字位置の取得
//
// 返り値    : 文字位置[byte](検索されなかった場合は 0を返す)
//
// 機能説明 : 文字列の中から,指定文字列を検索し位置を返す。
//
// 備考      : pStrInStr( "a#c#d#eeee", "#" ) -> 2
//             pStrInStr( "a#c#d##eeee", "##" ) -> 6
//             pStrInStr( "abcdeabcde", "AA" ) -> 0
//
```


2.8 関数定義のコメント

関数の定義は次のように記述する。
(1)関数の型,関数名を記述する。
(2)次の行以降,引数の後にコメントを記述する。

■関数定義のコメントの例(ANSIスタイル)

```
char *cm_getMessage(
  char *msg_filename,   // メッセージ定義ファイル名
  int msg_number)       // メッセージ番号
{
  int i;
}
```

■関数定義のコメントの例(K&Rスタイル)

```
char *cm_getMessage(msg_filename, msg_number)
    char *msg_filename; // メッセージ定義ファイル名
    int msg_number;     // メッセージ番号
{
  int i;
}
```

2.9 キーワードの変更

関数の頭書きのキーワードは自由に追加,削除できる。
変更する場合には,【A HotDocument】の「ｵﾌﾟｼｮﾝ]メニュー[関数ｺﾒﾝﾄ]より設定をする。

■関数頭書きのコメントの例

```
//
// 機能      : 文字位置の取得
//
// 返り値    : 文字位置[byte](検索されなかった場合は 0を返す)
//
// 機能説明 : 文字列の中から,指定文字列を検索し位置を返す。
//
// 作成日    : 2007年9月28日
//
// 作成者    : 今井 浩司
//
// 使用例    : pStrInStr( "a#c#d#eeee", "#" ) -> 2
//             pStrInStr( "a#c#d##eeee", "##" ) -> 6
//             pStrInStr( "abcdeabcde", "AA" ) -> 0
//
// 注意事項 : 最大配列数以上の,配列を作らなければいけない。
//
```

2.10 関数中のコメント

【A HotDocument】で生成される関数定義書に反映させたいコメントは,(/// …) ,(//// …)のように記述する。
関数定義書の中の段下げは,(/) の数により深くすることができる。

■関数中のコメントの例

```
{
  int nsrc = strlen(srcstr);
  int ntar = strlen(tarstr);
  int i;

  /// 文字列の長さ分ループする
  for (i = 0; i < ntar; i++)
     //// ヒットした場合カウントアップし,それを返り値とする
     if (strncmp(tarstr+i, srcstr, nsrc) == 0)
        return i+1;
  return 0;
}
```


2.11 C言語ソースファイルの記述例

C言語ソースファイルの記述例を以下に示す。
ソースファイルの履歴のコメント,ソースファイルの頭書きなどは以下の順番で記述する。
基本的には,C++のコメントの"//"が "/* ～ */"に変更されたのみである。
ただし,関数の処理説明の書き方が,"///"から"/*c ～ */",","////"から"/*t ～ */"に変更される。
関数定義書の中の段下げは,"t"の数により深くすることができる。

■ ファイルの記述例

```
/* @(h) Comment.c ver 2.0 ( '07.09.13 今井 浩司 ) */
/* @(h) Comment.c ver 1.0 ( '07.09.01 今井 浩司 ) */

/* @(s)
 *   ファイルI/O関数群
 *     本ファイルはファイル入出力に関する関数群を
 *     集めたものである。
 */
#define HC_MAX_PATH       128      /* フルパス 最大文字列数 */
#define HC_MAX_EXT         3       /* 拡張子 最大文字列数 */
#define HC_MAX_FILENAME  128       /* ファイル名 最大文字列数 */
#define HC_MAX_DATE       9        /* 日付 "YY/MM/DD" */
#define NUMBER_OF(a) ¥
      ( sizeof(a) / sizeof(a[0]) ) /* 配列の要素数を得る */

typedef struct Memory {  /* メモリ上のバッファ */
  char *s;               /* 文字列 */
  struct Memory *next;   /* リスト構造ポインタ */
} MEMORY;

union u_tag {  /* タグ共用体 */
  int ival;    /* 整数 */
  float fval;  /* 実数 */
  char *sval;  /* 文字列 */
} u;

enum Boolean {  /* ブール型定数 */
  False,        /* 偽 */
  True          /* 真 */
};
```

■ 関数定義部の記述例

```
/*
 * 機能 : 文字位置の取得
 *
 * 返り値 : 文字位置[byte](検索されなかった場合は 0を返す)
 *
 * 機能説明 : 文字列の中から,指定文字列を検索し位置を返す。
 *
 * 備考 : pStrInStr( "a#c#d#eeee", "#" ) -> 2
 *        pStrInStr( "a#c#d##eeee", "##" ) -> 6
 *        pStrInStr( "abcdeabcde", "AA" ) -> 0
 */
int pStrInStr(tarstr, srcstr)
  const char *tarstr; /* 検索される対象の文字列 */
  const char *srcstr; /* 検索する指定部分文字列 */
{
  int nsrc = strlen(srcstr);
  int ntar = strlen(tarstr);
  int i;

  /*c 文字列の長さ分ループする */
  for (i = 0; i < ntar; i++)
  {
     /*t ヒットした場合カウントアップし,それを返り値とする */
     if (strncmp(tarstr+i, srcstr, nsrc) == 0)
        return (i+1);
     /*tt 更にインデントがある場合 */
  }
  return (0);
}
```


第3章
C#コメント規約

本章では,C#のコメントをどのように記述するかを述べる。

3.1 ソースファイルの記述例

ソースファイルの記述例を以下に示す。
ソースファイルの履歴のコメント,ソースファイルの頭書きなどは以下の順番で記述する。

■ ファイルの記述例

```
// @(h) Form1.cs ver 1.0 ( '07.09.15 今井 浩司 )
// @(h) Form1.cs ver 1.1 ( '07.09.20 今井 浩司 )

// @(s)
//   Form1クラスのサンプルソース
//     全ての画面の処理、および画面に関係する
//     数々の定数の宣言をこのファイルに記述する。
//

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

```
namespace Sample // Form1の名前空間
{
/// <summary>
/// Form1クラスは、全ての画面メソッドを定義しています。
/// ボタンをクリックされた時の処理、フォームが終了した時の
/// 処理はこのクラス内で全て定義しています。
/// </summary>
public class Form1 : System.Windows.Forms.Form
{
  public const int LOOP_COUNT = 2; // ループの数
  public static int nSize = 10;    // ボタンのサイズ

  public interface InterSample // interfaceコメント
  {
    event MyDelegate MyEvent1; // MyEvent1コメント
    event MyDelegate MyEvent2; // MyEvent2コメント
  }

  public struct PERSON // structコメント
  {
    string Name;       // 名前
    int Age;           // 年齢
  }

  public enum WeekDay  // enumコメント
  {
    Sun = 1,           // 日曜日
    Mon = 2,           // 月曜日
    Tue = 3,           // 火曜日
    Wed = 4,           // 水曜日
    Thu = 5,           // 木曜日
    Fri = 6,           // 金曜日
    Sat = 7,           // 土曜日
  }

  public delegate void MyDelegate(int i); // delegateコメント
  public event MyDelegate MyEvent1;       // eventコメント

  public event MyDelegate Event2          // eventコメント
  {
    add
    {
      …
    }
  }
  [DllImport("user32.dll", EntryPoint="GetDC")]
  extern private static IntPtr GetDC(IntPtr hDC); // DllImportコメント
```



```
  //
  // 機能       : ボタンをクリックされた時の処理
  //
  // 機能説明   : ボタンをクリックされた時の処理を行う。
  //              クラス定数LOOP_COUNT回、ダイアログを表示する。
  //
  // 引き数     : ARG1 - 発生ｲﾍﾞﾝﾄｵﾌﾞｼﾞｪｸﾄ
  //              ARG2 - イベント引き数
  //
  // 返り値     : なし
  //
  // 備考       : 簡単なサンプルです。
  //
  private void button1_Click (Object sender, System.EventArgs e)
  {
    int i;
    String Buf;

    /// 固定文字列を代入する
    Buf = "回目のボタンの文字列";

    /// 以下の処理を、ループの回数分行う
    for (i = 0; i < LOOP_COUNT; i++)
    {
        String MessageString;
        //// 文字列を連結してダイアログに表示する
        MessageString = i + Buf;
        MessageBox.Show(MessageString);
    }
    }
  }
}
```

3.2 履歴のコメント

ソースファイルの履歴のコメントには,ファイル名,バージョン,作成日/修正日,担当者名を記述する。

■ 履歴のコメント例

```
// @(h) Form1.cs ver 1.0 ( '07.09.15 今井 浩司 )
// @(h) Form1.cs ver 1.1 ( '07.09.20 今井 浩司 )
```

3.3 ソースファイルの頭書きのコメント

ソースファイルの頭書きには,ソースファイルに含まれるメソッドの説明を記述する。

■ ソースファイルの頭書きのコメント例

```
// @(s)
//   Form1クラスのサンプルソース
//      全ての画面の処理、および画面に関係する
//      数々の定数の宣言をこのファイルに記述する。
//
```

3.4 クラスのコメント

クラスに関するコメントは以下のように記述する。
クラス一覧の説明は,コメントの一行目だけ出力される。

■ クラスのコメント例

```
/// <summary>
/// Form1クラスは、全ての画面メソッドを定義しています。
/// ボタンをクリックされた時の処理、フォームが終了した時の
/// 処理はこのクラス内で全て定義しています。
/// </summary>
public class Form1 : System.Windows.Forms.Form
```


3.5 メソッドの頭書きのコメント

メソッドの頭書きのコメントは次のように記述する。
(1)必要のない項目は削除してもよい。
(2)「機能」はメソッドの説明を一行で記述する。

■ メソッドの頭書きのコメント例

```
//
// 機能      : ボタンをクリックされた時の処理
//
// 機能説明  : ボタンをクリックされた時の処理を行う。
//             クラス定数LOOP_COUNT回、ダイアログを表示する。
//
// 引き数    : ARG1 - 発生ｲﾍﾞﾝﾄｵﾌﾞｼﾞｪｸﾄ
//             ARG2 - イベント引き数
//
// 返り値    : なし
//
// 備考      : 簡単なサンプルです。
//
```

3.6 定数、変数のコメント

定数、変数に関するコメントは,以下のように記述する。
宣言の後にコメントを記述すると,定数,変数一覧の説明に出力される。

■定数、変数のコメント例

```
public const int LOOP_COUNT = 2;  // ループの数
public static int nSize = 10;     // ボタンのサイズ
```

3.7 Namespace/Interface/構造体/Enumのコメント

Namespace/Interface/構造体/Enumに関するコメントは,以下のように記述する。
宣言の後にコメントを記述すると,各ドキュメントの説明に出力される。

■ Namespaceのコメント例

```
namespace Sample // Form1の名前空間
{
  …
}
```

■ Interfaceのコメント例

```
public interface InterSample   // interfaceコメント
{
  event MyDelegate MyEvent1;   // MyEvent1コメント
  event MyDelegate MyEvent2;   // MyEvent2コメント
}
```

■構造体のコメント例

```
public struct PERSON   // structコメント
{
  string Name;         // 名前
  int Age;             // 年齢
}
```

■Enumのコメント例

```
public enum WeekDay // enumコメント
{
  Sun = 1,           // 日曜日
  Mon = 2,           // 月曜日
  Tue = 3,           // 火曜日
  Wed = 4,           // 水曜日
  Thu = 5,           // 木曜日
  Fri = 6,           // 金曜日
  Sat = 7,           // 土曜日
}
```


3.8 Delegate/Event/DllImportのコメント

Delegate/Event/DllImportに関するコメントは,以下のように記述する。
宣言の後にコメントを記述すると,各ドキュメントの説明に出力される。

■ Delegateのコメント例

```
public delegate void MyDelegate(int i); // delegateコメント
```

■ Eventのコメント例

```
public event MyDelegate MyEvent1;    // eventコメント

public event MyDelegate Event2 // eventコメント
{
  add
  {
    …
  }
}
```

■ DllImportのコメント例

```
[DllImport("user32.dll", EntryPoint="GetDC")]
extern private static IntPtr GetDC(IntPtr hDC); // DllImportコメント
```

3.9 XMLドキュメントコメント

XMLドキュメントコメントからコメントを抜き出すために,下記のようなコメント形式にも対応している。
メソッドのXMLドキュメントコメントは,<summary>,<param>などの全てのタグに対応している。
XMLの構文エラーの場合は,ドキュメントに出力されない。
また,最新の情報は,【A HotDocument】の[ﾍﾙﾌﾟ(H)]-[ｺﾒﾝﾄ規約(C)]に記述している。

■ XMLドキュメントコメントの例

```
/// <summary>
///  XMLドキュメントコメントの説明を出力します。
///  以下のタグはコメントを抽出できます。
/// </summary>
/// <remarks>
///  機能説明はこのように記述します。
/// </remarks>
/// <param name="Coumment">出力コメント文字列</param>
/// <param name="Size">出力コメントサイズ</param>
/// <returns>
///  返り値はこのように記述します。
/// </returns>
/// <exception>
///  例外処理はこのように記述します。
/// </exception>
private void WebPageBuildComment(string Coumment, int Size)
{
}

/// <summary>
/// ループの数
/// </summary>
public const int LOOP_COUNT = 2;

/// <summary>
/// ボタンのサイズ
/// </summary>
public static int nSize = 10;
```



```
/// <summary>
/// interfaceコメント
/// </summary>
public interface InterSample
{
  event MyDelegate MyEvent1; // MyEvent1コメント
  event MyDelegate MyEvent2; // MyEvent2コメント
}
/// <summary>
/// structコメント
/// </summary>
public struct PERSON
{
  string Name;   // 名前
  int Age;       // 年齢
}

/// <summary>
/// enumコメント
/// </summary>
public enum WeekDay
{
  Sun = 1,  // 日曜日
  Mon = 2,  // 月曜日
  Tue = 3,  // 火曜日
  Wed = 4,  // 水曜日
  Thu = 5,  // 木曜日
  Fri = 6,  // 金曜日
  Sat = 7,  // 土曜日
}

/// <summary>
/// delegateコメント
/// </summary>
public delegate void MyDelegate(int i);
```

3.10 キーワードの変更

メソッドの頭書きのキーワードは自由に追加,削除できる。
変更する場合には,【A HotDocument】の,[ｵﾌﾟｼｮﾝ]メニュー[ﾒｿｯﾄﾞｺﾒﾝﾄ]より設定をする。
また,XMLドキュメントコメントのコメントタグも自由に追加できる。

■ メソッド頭書きのコメント例

```
//
// 機能     : 文字位置の取得
//
// 返り値   : 文字位置[byte](検索されなかった場合は 0を返す)
//
// 機能説明 : 文字列の中から,指定文字列を検索し位置を返す。
//
// 作成日   : 2007年09月15日
//
// 作成者   : 今井 浩司
//
// 使用例   : pStrInStr( "a#c#d#eeee", "#" ) -> 2
//            pStrInStr( "a#c#d##eeee", "##" ) -> 6
//            pStrInStr( "abcdeabcde", "AA" ) -> 0
//
// 注意事項 : 最大配列数以上の,配列を作らなければいけない。
//
```

■XMLドキュメントコメント形式のメソッド頭書きのコメント例

```
///<summary>
///  XMLドキュメントコメントの説明を出力します。
///  以下のタグはコメントを抽出できます。
///</summary>
///<newtag>
///  新規タグはこのように記述します。
///</newtag>
///<value>
///  値はこのように記述します。
///</value>
///<returns>
///  返り値はこのように記述します。
///</returns>
```


3.11 メソッド中のコメント

【A HotDocument】で生成されるメソッド定義書に反映させたいコメントは,(/// …) ,(//// …)
のように記述する。
メソッド定義書の中の段下げは,(/) の数により深くすることができる。

■ メソッド中のコメント例

```
{
    int i;
    String Buf;

    /// 固定文字列を代入する
    Buf = "回目のボタンの文字列";

    /// 以下の処理を、LOOP_COUNT回行う
    for (i = 0; i < LOOP_COUNT; i++)
    {
      String MessageString;

      //// 文字列を連結してダイアログに表示する
      MessageString = i + Buf;
      MessageBox.Show(MessageString);
    }
}
```

第4章
Javaコメント規約

本章では,Javaのコメントをどのように記述するかを述べる。

4.1 ソースファイルの記述例

ソースファイルの記述例を以下に示す。
ソースファイルの履歴のコメント,ソースファイルの頭書きなどは以下の順番で記述する。

■ ファイルの記述例

```
// @(h) Form1.java        ver 1.0 ( '07.09.15 今井 浩司 )
// @(h) Form1.java        ver 1.1 ( '07.09.20 今井 浩司 )

// @(s)
//    Form1クラスのサンプルソース
//    ボタンクリック処理を含む全ての画面の処理を
//    このファイルに記述する。
//

package Sample;

import System.Drawing.*;
import System.Collections.*;
import System.ComponentModel.*;
import System.Windows.Forms.*;
import System.Data.*;

public class Form1 extends System.Windows.Forms.Form//Form1の概要
{
  private System.Windows.Forms.Button button1;
  private System.ComponentModel.Container components = null;

  public static final int LOOP_COUNT = 2;
  public static int nSize = 0;
```



```
  //
  // 機能       : ボタンをクリックされた時の処理
  //
  // 機能説明   : ボタンをクリックされた時の処理を行う。
  //              クラス定数LOOP_COUNT回、ダイアログを表示する。
  //
  // 返り値     : なし
  //
  // 備考       : 簡単なサンプルです。
  //
  private void button1_Click (Object sender,      // 発生ｲﾍﾞﾝﾄｵﾌﾞｼﾞｪｸﾄ
                              System.EventArgs e)  // イベント引き数
  {
    int i;
    String Buf;

    /// 固定文字列を代入する
    Buf = "回目のボタンの文字列";

    /// 以下の処理を、LOOP_COUNT回行う
    for (i = 0; i < LOOP_COUNT; i++)
    {
      String MessageString;

      //// 文字列を連結してダイアログに表示する
      MessageString = i + Buf;
      MessageBox.Show(MessageString);
    }
  }

  /**
    * JavaDocの説明でも、これらのタグはコメントを抽出できます。
    * 複数行にまたがった説明も記述できます。
    *
    * @param 引き数はこのように記述します。
    * @return 返り値はこのように記述します。
    * @author 作成者はこのように記述します。
    * @version バージョンはこのように記述します。
    * @since 導入ﾊﾞｰｼﾞｮﾝはこのように記述します。
    * @see 関連項目はこのように記述します。
    * @exception 例外はこのように記述します。
    * @deprecated 説明はこのように記述します。
    */
  private void JavaDoc()
  {
    /// メソッド内のコメントはこのように記述します。
  }
}
```

4.2 履歴のコメント

ソースファイルの履歴のコメントには,ファイル名,バージョン,作成日/修正日,担当者名を記述する。

■ 履歴のコメントの例

```
// @(h) Form1.java        ver 1.0 ( '07.09.15 今井 浩司 )
// @(h) Form1.java        ver 1.1 ( '07.09.20 今井 浩司 )
```

4.3 ソースファイルの頭書きのコメント

ソースファイルの頭書きには,ソースファイルに含まれるメソッドの説明を記述する。

■ ソースファイルの頭書きのコメント例

```
// @(s)
//    Form1クラスのサンプルソース
//    ボタンクリック処理を含む全ての画面の処理を
//    このファイルに記述する。
//
```

4.4 クラスのコメント

クラスに関するコメントは以下のように記述する。

■クラスのコメントの例

```
public class Form1 extends Form  //フォームの表示は本クラスを使用
{
  …
}
public class Class1 // 本クラスは数学関連のメソッド群
{
  …
}
```


4.5 メソッドの頭書きのコメント

メソッドの頭書きのコメントは次のように記述する。
(1)必要のない項目は削除してもよい。
(2)「機能」はメソッドの説明を一行で記述する。

■ メソッドの頭書きのコメント例

```
//
// 機能        : ボタンをクリックされた時の処理
//
// 機能説明    : ボタンをクリックされた時の処理を行う。
//               クラス定数LOOP_COUNT回､ダイアログを表示する。
//
// 返り値      : なし
//
// 備考        : 簡単なサンプルです。
//
```

4.6 メソッド定義のコメント

メソッドの定義は次のように記述する。
(1)メソッドの型,メソッド名を記述する。
(2)次の行以降,引数の後にコメントを記述する。

■ メソッド定義のコメントの例

```
private void button1_Click (Object sender,       // 発生イベントオブジェクト
                            System.EventArgs e)  // イベント引き数
```

4.7 JavaDocのコメント

JavaDocのコメント規約と互換性を保つため,ﾒｿｯﾄﾞに関するほとんどのキーワードは対応している。
ただし,<B>などのタグは削除されドキュメント化される。
キーワードの追加変更も可能になっている。
また,最新の情報は,【A HotDocument】の[ﾍﾙﾌﾟ(H)]-[ｺﾒﾝﾄ規約(C)]に記述している。

■ JavaDocのコメントの例

```
/**
  * JavaDocの説明でも、これらのタグはコメントを抽出できます。
  * 複数行にまたがった説明も記述できます。
  *
  * @param 引き数はこのように記述します。
  * @return 返り値はこのように記述します。
  * @author 作成者はこのように記述します。
  * @version バージョンはこのように記述します。
  * @since 導入ﾊﾞｰｼﾞｮﾝはこのように記述します。
  * @see 関連項目はこのように記述します。
  * @exception 例外はこのように記述します。
  * @deprecated 説明はこのように記述します。
  */
private void JavaDoc()
{
  /// メソッド内のコメントはこのように記述します。
}
```


4.8 キーワードの変更

ﾒｿｯﾄﾞの頭書きのキーワードは自由に追加,削除できる。
変更する場合には,【A HotDocument】の,[ｵﾌﾟｼｮﾝ]ﾒﾆｭｰ[ﾒｿｯﾄﾞｺﾒﾝﾄ]より設定をする。

■ メソッド頭書きのコメントの例

```
//
// 機能      : 文字位置の取得
//
// 返り値    : 文字位置[byte](検索されなかった場合は 0を返す)
//
// 機能説明  : 文字列の中から,指定文字列を検索し位置を返す。
//
// 作成日    : 2007年09月15日
//
// 作成者    : 今井 浩司
//
// 使用例    : pStrInStr( "a#c#d#eeee", "#" ) -> 2
//             pStrInStr( "a#c#d##eeee", "##" ) -> 6
//             pStrInStr( "abcdeabcde", "AA" ) -> 0
//
// 注意事項 : 最大配列数以上の,配列を作らなければいけない。
//
```

■メソッド頭書きのコメントの例(JavaDoc)

```
/**
  * JavaDocの説明でも、これらのタグはコメントを抽出できます。
  * 複数行にまたがった説明も記述できます。
  *
  * @newKey 新規タグはこのように記述します。
  * @returns 返り値はこのように記述します。
  * @attention 注意はこのように記述します。
  */
```

4.9 メソッド中のコメント

【A HotDocument】で生成されるメソッド定義書に反映させたいコメントは,(/// …) ,(//// …) のように記述する。
メソッド定義書の中の段下げは,(/) の数により深くすることができる。

■メソッド中のコメントの例

```
{
   int i;
   String Buf;

   /// 固定文字列を代入する
   Buf = "回目のボタンの文字列";

   /// 以下の処理を、LOOP_COUNT回行う
   for (i = 0; i < LOOP_COUNT; i++)
   {
     String MessageString;

     //// 文字列を連結してダイアログに表示する
     MessageString = i + Buf;
     MessageBox.Show(MessageString);
   }
}
```


この規約によって、少しでもシステム開発の標準化ができたら幸いです。
【A HotDocument】開発者一同