|  |
| --- |
|  |
| コーディング基準 |
| 第1.1版  最終更新日：2013年2月15日 |
|  |
| 伊藤忠テクノソリューションズ株式会社 |

更新履歴

|  |  |  |  |
| --- | --- | --- | --- |
| 版数 | 更新日 | 更新者 | 更新内容 |
| 1.0 | 2011年12月16日 | CTC | 初版作成 |
| 1.1 | 2013年2月15日 | CTC | 項「3.5.2. コードブロックに対応する波括弧（必須）」を新規追加 |
|  |  |  |  |

本文書は著作権法上の保護を受けています。本文書の著作権、並びに著作者人格権は、一部の引用を除き、伊藤忠テクノソリューションズ株式会社が保有します。

伊藤忠テクノソリューションズ株式会社による承諾を得ずに無断で複写、複製することは禁止されています。

本文書は伊藤忠テクノソリューションズ株式会社の重要文書です。秘密保持契約が締結されていない協力会社への配布、並びに、本文書を取引対象とした著作権の譲渡は禁止されています。

目次

[1. はじめに 1](#_Toc350339164)

[1.1. 文書の目的 1](#_Toc350339165)

[1.2. 文書の対象者 1](#_Toc350339166)

[1.3. 文書を利用するに際しての前提条件/前提知識 1](#_Toc350339167)

[2. 共通 2](#_Toc350339168)

[2.1. 命名規則 2](#_Toc350339169)

[2.1.1. キャメルケースの優先的利用（必須） 2](#_Toc350339170)

[2.1.2. 共通認識のない略語の利用（禁止） 2](#_Toc350339171)

[2.2. エンコーディング 3](#_Toc350339172)

[2.2.1. UTF-8の選択（必須） 3](#_Toc350339173)

[3. ActionScript 4](#_Toc350339174)

[3.1. ソースファイル 4](#_Toc350339176)

[3.1.1. インポート可能なAPIと同じファイル名（必須） 4](#_Toc350339177)

[3.2. ブロックとインデント 5](#_Toc350339178)

[3.2.1. 半角空白によるインデント（必須）[FlexFormatter対応] 5](#_Toc350339179)

[3.2.2. 行末でのブロック開始（必須）[FlexFormatter対応] 5](#_Toc350339180)

[3.2.3. 行頭でのブロック終了（必須）[FlexFormatter対応] 5](#_Toc350339181)

[3.2.4. 最大行幅の統一（必須）[FlexFormatter対応] 5](#_Toc350339182)

[3.2.5. 継続行のインデント（必須）[FlexFormatter対応] 6](#_Toc350339183)

[3.2.6. 行折り返し時の行頭の演算子（必須） 6](#_Toc350339184)

[3.2.7. 空白行の挿入（必須）[FlexFormatter対応] 6](#_Toc350339185)

[3.3. 文と式 7](#_Toc350339186)

[3.3.1. セミコロンによる明示的な文の終了（必須） 7](#_Toc350339187)

[3.4. 演算子 7](#_Toc350339188)

[3.4.1. 後置演算子前の空白（禁止）[FlexFormatter対応] 7](#_Toc350339189)

[3.4.2. 単項演算子後の空白（禁止）[FlexFormatter対応] 7](#_Toc350339190)

[3.4.3. 二項演算子前後の空白（必須）[FlexFormatter対応] 7](#_Toc350339191)

[3.4.4. 三項演算子前後の空白（必須）[FlexFormatter対応] 8](#_Toc350339192)

[3.5. 制御構文 8](#_Toc350339193)

[3.5.1. 制御指示子に対応する丸括弧前後の空白（必須）[FlexFormatter対応] 8](#_Toc350339194)

[3.5.2. コードブロックに対応する波括弧（必須） 9](#_Toc350339195)

[3.6. 型注釈 10](#_Toc350339196)

[3.6.1. 型注釈の利用（必須） 10](#_Toc350339197)

[3.6.2. 空白を含まない型注釈のコロン（必須） 10](#_Toc350339198)

[3.7. パッケージ 10](#_Toc350339199)

[3.7.1. 英小文字で開始するパッケージ名（必須）[FlexPMD対応] 10](#_Toc350339200)

[3.8. インポート 11](#_Toc350339201)

[3.8.1. オンデマンドインポートの利用（禁止） 11](#_Toc350339202)

[3.8.2. 未使用クラスのインポート（禁止） 11](#_Toc350339203)

[3.8.3. 順序あるインポートの宣言（必須）[FlexFormatter対応] 11](#_Toc350339204)

[3.9. 名前空間 12](#_Toc350339205)

[3.9.1. スネークケースによる名前空間名（必須） 12](#_Toc350339206)

[3.9.2. 名前空間参照における名前修飾子の優先的な利用（必須） 12](#_Toc350339207)

[3.10. 修飾子 12](#_Toc350339208)

[3.10.1. 順序ある修飾子の宣言（必須）[FlexFormatter対応] 12](#_Toc350339209)

[3.11. メタデータタグ 13](#_Toc350339210)

[3.11.1. 順序あるクラスメタデータタグの宣言（必須）[FlexFormatter対応] 13](#_Toc350339211)

[3.11.2. 角括弧および丸括弧前後の空白（禁止）[FlexFormatter対応] 13](#_Toc350339212)

[3.11.3. AsDocの下のメタデータタグ 13](#_Toc350339213)

[3.12. 関数 14](#_Toc350339214)

[3.12.1. ローワーキャメルケースの適用（必須） 14](#_Toc350339215)

[3.12.2. 動詞で始まる関数名（必須） 14](#_Toc350339216)

[3.12.3. 丸括弧前後の空白（禁止）[FlexFormatter対応] 15](#_Toc350339217)

[3.13. クラス・インタフェース 15](#_Toc350339218)

[3.13.1. アッパーキャメルケースの適用（必須） 15](#_Toc350339219)

[3.13.2. 動詞で始まるクラス名（禁止） 15](#_Toc350339220)

[3.13.3. インタフェース名への”I”プレフィックス追加（必須）[FlexPMD対応] 15](#_Toc350339221)

[3.13.4. 順序ある継承と実装の宣言（必須）[FlexFormatter対応] 16](#_Toc350339222)

[3.13.5. flash.events.Eventを継承したクラスの名称（必須） 16](#_Toc350339223)

[3.13.6. Errorを継承したクラスの名称（必須） 16](#_Toc350339224)

[3.13.7. mx.formatters.Formatterを継承したクラスの名称（必須） 16](#_Toc350339225)

[3.13.8. mx.validators.Validatorを継承したクラスの名称（必須） 17](#_Toc350339226)

[3.13.9. 動的クラスの作成と利用（禁止）[FlexPMD対応] 17](#_Toc350339227)

[3.13.10. prototypeプロパティの利用（禁止） 17](#_Toc350339228)

[3.14. プロパティ 18](#_Toc350339229)

[3.14.1. 可変プロパティ名のローワーキャメルケースの適用（必須） 18](#_Toc350339230)

[3.14.2. 不変プロパティ名のアッパースネークケースの適用（必須） 18](#_Toc350339231)

[3.14.3. 動詞で始まるプロパティ名（禁止） 19](#_Toc350339232)

[3.14.4. ストレージ変数に対する”\_”プレフィックスの適用（必須） 19](#_Toc350339233)

[3.14.5. setterの引数名（必須） 19](#_Toc350339234)

[3.14.6. 不要なgetterとsetterの定義（禁止） 19](#_Toc350339235)

[3.14.7. 型制約としてのArrayElementTypeメタデータタグの信頼（禁止） 20](#_Toc350339236)

[3.15. メソッド 21](#_Toc350339237)

[3.15.1. ローワーキャメルケースの適用（必須） 21](#_Toc350339238)

[3.15.2. 動詞で始まるメソッド名（必須） 21](#_Toc350339239)

[3.15.3. 丸括弧前後の空白（禁止）[FlexFormatter対応] 21](#_Toc350339240)

[3.16. 変数 22](#_Toc350339241)

[3.16.1. 変数名のローワーキャメルケースの適用（必須） 22](#_Toc350339242)

[3.17. イベント 22](#_Toc350339243)

[3.17.1. カスタムイベントクラスにおけるcloneメソッドのオーバライド（必須）[FlexPMD対応] 22](#_Toc350339244)

[3.17.2. カスタムイベントクラスによる変更可能なプロパティの公開（禁止）[FlexPMD対応] 23](#_Toc350339245)

[3.17.3. イベント名のハードコード（禁止）[FlexPMD対応] 23](#_Toc350339246)

[3.17.4. イベントハンドラ名へのHandlerサフィックスの追加（必須） 23](#_Toc350339247)

[3.18. ドキュメンテーションコメント 24](#_Toc350339248)

[3.18.1. 行頭のアスタリスク（必須） 24](#_Toc350339249)

[3.18.2. public要素およびprotected要素へのドキュメンテーションコメント（必須）[FlexPMD対応] 24](#_Toc350339250)

[3.18.3. 関数やメソッドのドキュメンテーションコメントタグ（必須） 24](#_Toc350339251)

[3.18.4. getterおよびsetterとドキュメンテーションコメント（必須） 25](#_Toc350339252)

[3.18.5. Eventのドキュメンテーションコメント（必須） 26](#_Toc350339253)

[3.18.6. デフォルト値の明示（必須） 27](#_Toc350339254)

[3.19. 外部クラスの取り込み 28](#_Toc350339255)

[3.19.1. スクリプトタグ内部でのincludeの使用（禁止） 28](#_Toc350339256)

[4. MXML 29](#_Toc350339257)

[4.1. ソースファイル 29](#_Toc350339259)

[4.1.1. クラス名としてのファイル名（必須） 29](#_Toc350339260)

[4.2. ブロックとインデント 29](#_Toc350339261)

[4.2.1. 半角空白によるインデント（必須）[FlexFormatter対応] 29](#_Toc350339262)

[4.2.2. 最大行幅の統一（必須）[FlexFormatter対応] 29](#_Toc350339263)

[4.2.3. 継続行のインデント（必須）[FlexFormatter対応] 29](#_Toc350339264)

[4.2.4. 1行1タグ（必須）[FlexFormatter対応] 29](#_Toc350339265)

[4.2.5. 順序あるMXMLタグの宣言（必須） 30](#_Toc350339266)

[4.2.6. 属性のカテゴライズと順序付け（必須）[FlexFormatter対応] 31](#_Toc350339267)

[4.3. ID属性 31](#_Toc350339268)

[4.3.1. 最初の属性としてのID属性（必須）[FlexFormatter対応] 31](#_Toc350339269)

[4.3.2. ローワーキャメルケースの適用（必須） 31](#_Toc350339270)

[4.4. スクリプトタグ 32](#_Toc350339271)

[4.4.1. 長すぎるスクリプトタグ（禁止）[FlexPMD対応] 32](#_Toc350339272)

[4.5. イベントハンドラ属性 32](#_Toc350339273)

[4.5.1. イベントハンドラ内での複文の実行（禁止） 32](#_Toc350339274)

[4.6. 外部クラスの取り込み 33](#_Toc350339275)

[4.6.1. スクリプトタグ内部でのincludeの使用（禁止） 33](#_Toc350339276)

[4.6.2. MXML記述部での<mx:Script source=”” />の使用（禁止） 33](#_Toc350339277)

[5. CSS 34](#_Toc350339278)

[5.1. ソースファイル 34](#_Toc350339280)

[5.1.1. ファイル名へのローワーキャメルケースの適用（必須） 34](#_Toc350339281)

[5.1.2. スタイル名へのローワーキャメルケースの適用（必須） 34](#_Toc350339282)

[5.2. ブロックとインデント 34](#_Toc350339283)

[5.2.1. 半角空白によるインデント（必須） 34](#_Toc350339284)

[5.2.2. 行末でのブロック開始（必須） 34](#_Toc350339285)

[5.2.3. 行頭でのブロック終了（必須） 35](#_Toc350339286)

[5.2.4. スタイル名とスタイル値の間の空白（必須） 35](#_Toc350339287)

[5.2.5. 最大行幅の統一（必須） 35](#_Toc350339288)

[5.2.6. 1行1スタイル（必須） 35](#_Toc350339289)

[5.3. スタイルクラス 36](#_Toc350339290)

[5.3.1. スタイルクラス名へのローワーキャメルケースの適用（必須） 36](#_Toc350339291)

参考資料

Adobe Systems Inc. “Flex SDK コード記述に関する規則とベストプラクティス.” Adobe Open Source. 2008年5月19日. http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions-ja [アクセス日: 2010年8月11日].

MooreSean. “Flexのベストプラクティス - パート2.” Adobe Developer Connection. 2008年12月2日. http://www.adobe.com/jp/devnet/flex/articles/best\_practices\_pt2.html [アクセス日: 2010年8月18日].

—. “Flexのベストプラクティス–パート1.” Adobe Developper Connection. 2008年7月7日. http://www.adobe.com/jp/devnet/flex/articles/best\_practices\_pt1.html [アクセス日: 2010年8月18日].

# はじめに

## 文書の目的

本文書は、Flexアプリケーション開発のためのコーディングを規格化することによるアプリケーション品質の向上を目的としています。

## 文書の対象者

本文書はFlexアプリケーション開発に携わる全てのプログラマを対象としています。

## 文書を利用するに際しての前提条件/前提知識

本文書の内容を理解し適切に利用するには、Flexを利用したアプリケーション開発経験が1年程度必要です。

なお、本文書はActionScript 3.0を対象として記述されています。Flex SDKのバージョンは、3.5以降に対応していますが、4.1を念頭に記述しています。バージョン固有の基準を提示する場合にはその旨を併せて記述しています。

# 共通

## 命名規則

### キャメルケース[[1]](#footnote-1)の優先的利用（必須）

**説明:** ファイル名、パッケージ名、インタフェース名、クラス名、メソッド名、関数名、変数名、プロパティ名、イベント名などFlexアプリケーション構成要素に名称をつける場合には、ASCII文字で構成し、キャメルケースを優先的に利用してください。

**理由:** Flexアプリケーション開発の一般的な慣習だからです。[Adobe Systems Inc 2008] に従います。

**注意:**「3.9.1 スネークケースによる名前空間名（必須）」や「3.14.2 不変プロパティ名のアッパースネークケースの適用（必須）」のようにこの規則が該当しない場合あります。

**補足:** アッパーキャメルケースかローワーキャメルケースかなどの詳細については、アプリケーション構成要素別に記述していますので、そちらを参照してください。

### 共通認識のない略語の利用（禁止）

**説明:** 用語集にない略語の利用は禁止します。

**理由:** シンボル（変数名や関数名など）の意味が明確になるからです。ソースコードの可読性が向上し、ソースコードの統一感も高まります。

**補足:** 公式のドキュメント[Adobe Systems Inc 2008] によると以下の略語についてはFlex SDK内部で標準化されている（共通認識がある）とのことです。

* + - acc: accessibility（アクセシビリティ）
    - auto: automatic（自動）
    - eval: evaluate（評価）
    - impl: implementation（実装）
    - info: information（情報）
    - num: number（数値）
    - min: minimum（最小）
    - max: maximum（最大）
    - nav: navigation（ナビゲーション）
    - regexp: regular expression（正規表現）
    - util: utility（ユーティリティ）

## エンコーディング

### UTF-8の選択（必須）

**説明:** ソースファイルのエンコーディングにはUTF-8を選択してください。なお、BOMの付加可否については、BOMなしとしてください。

**理由:** Flash Playerのデフォルトエンコーディングだからです。UTF-8であれば、特別な対処なく非ASCII文字を文字化けせずに取り扱うことができます。また、FlexコンパイラはソースファイルのエンコーディングをBOMから判断することができますが、これは明示的でないためコンパイラ引数の利用を推奨します。よって、ソースファイル毎にBOMは必要ありません。

# ActionScript



## ソースファイル

### インポート可能なAPIと同じファイル名（必須）

**説明:** ActionScriptソースファイルにインポート可能なAPIが含まれている場合、ファイル名は必ずそのAPIと同じ名称としてください。例えば、下記のコードスニペットが格納されるファイル名は「publicFunction.as」でなければなりません。

**package** jp.co.ctc\_g.cf.dp1.examples {

**public** **function** publicFunction():**void** {

}

}

コード 3‑1 公開関数を含むソースファイル

さらに、下記のコードスニペットが格納されるファイル名は「PublicClass.as」です。

**package** jp.co.ctc\_g.cf.dp1.examples {

**public** **class** PublicClass {

**public** **function** PublicClass() {

}

}

}

コード 3‑2 公開クラスを含むソースファイル

**理由:** ソースファイル名の一般的な慣習だからです。この規則に従っていない場合、開発環境によってはビルドエラーとなります。

**補足:** 関数名やクラス名の命名規則については、「3.12 関数」や「3.13 クラス・インタフェース」を参照してください。

## ブロックとインデント

### 半角空白によるインデント（必須）[FlexFormatter対応]

**説明:** インデントはタブではなく半角空白を利用してください。また、半角空白数は2とします。

**理由:** タブによるインデントはエディタにより見た目が変わってしまうだけでなく、空白と混在することでインデントが揃わなくなるからです。また、インデントに利用する半角空白数を2としているのは、ActionScriptはパッケージブロックがあるためにネストが深くなるからです。

### 行末でのブロック開始（必須）[FlexFormatter対応]

**説明:** パッケージブロック、クラスブロック、メソッドブロック、ifブロックやforブロックなどの波括弧で囲まれる領域は、行末で開始してください。その場合、ブロック開始前には半角空白を置いてください。なお、Flex BuilderやFlash Builderは行頭でブロックを開始します。

**public** **function** foo():**void** {

**if** (bar) {

doStuff();

}

}

コード 3‑3 行末でのブロック開始例

**理由:** サーバサイドの開発言語との統一性を図るためです。例えば、サーバサイドにJava言語を利用するのであれば、行末でブロックを開始する記述方法が一般的です。

**注意:** 行末でブロックを開始するように自動フォーマットを設定する機能は、提供されていません[[2]](#footnote-2)。そこで、サードパーティ製プラグインである「Flex Formatter[[3]](#footnote-3)」を推奨します。設定方法については『CFP-DP1 開発環境構築手順』を参照してください。

**補足:** オブジェクトリテラルはブロックではないため、この基準に従う必要はありません。

### 行頭でのブロック終了（必須）[FlexFormatter対応]

**説明:** パッケージブロック、クラスブロック、メソッドブロック、ifブロックやforブロックなどの波括弧で囲まれる領域は、行頭で終了してください。

**理由:** ブロックを波括弧で表現する言語では、一般的な作法だからです。

### 最大行幅の統一（必須）[FlexFormatter対応]

**説明:** ソースコードの最大行幅を統一してください。最大行幅は80としてください。

**理由:** 利用しているエディタや参照しているソースにかかわらず、見た目を統一するためです。

### 継続行のインデント（必須）[FlexFormatter対応]

**説明:** 継続行のインデントは、通常のインデントの2倍確保、つまり半角空白4としてください。

**var** s:String = **"Lorem ipsum dolor sit amet, "**

+ **"consectetur adipisicing elit,"**

+ **"sed do eiusmod tempor incididunt ut "**

+ **"labore et dolore magna aliqua. "**;

コード 3‑4 継続行のインデントの例

**理由:** 継続行であることが明確になるからです。

### 行折り返し時の行頭の演算子（必須）

**説明:** 演算子を含む文を折り返す場合には、演算子を行頭に配置しつつ折り返してください。例えば、if文の条件節が複数行にわたる場合には以下のように記述します。

**if** (firstCondition.isSatisfied

&& secondCondition.isSatisfied

&& thirdCondition.isSatisfied) {

doStuff();

}

コード 3‑5 複数行にまたがる条件節の例

**理由:** 折り返し行が式の途中であるということが明確になるからです。

### 空白行の挿入（必須）[FlexFormatter対応]

**説明:** ソースコード内の以下の区切りにおいて空白行を加えてください。

* + - package宣言とimport宣言の間の区切り
    - import宣言とclass宣言の間の区切り
    - class宣言と変数宣言やメソッド宣言の間の区切り
    - import宣言と関数宣言の間の区切り

**理由:** ソースコードを視覚的に適切に区切ることで可読性が向上するからです。

## 文と式

### セミコロンによる明示的な文の終了（必須）

**説明:** 文は必ずセミコロンで終了させてください。

**理由:** コンパイラに意図しない解釈をされることを防ぐためです。文末が明示されることでの可読性向上も期待できます。これは[Adobe Systems Inc 2008] に従います。

## 演算子

### 後置演算子前の空白（禁止）[FlexFormatter対応]

**説明:** 後置演算子の前には空白を含めないでください。

variable++ *// OK*

variable ++ *// NG: 空白が含まれています*

コード 3‑6 後置演算子前の空白例

**理由:** オペランドの関係を明確にするためです。

### 単項演算子後の空白（禁止）[FlexFormatter対応]

**説明:** 単項演算子の後には空白を含めないでください。

++variable *// OK*

~variable *// OK*

++ variable *// NG: 空白が含まれています*

~ variable *// NG: 空白が含まれています*

コード 3‑7 単項演算子後の空白例

**理由:** オペランドの関係を明確にするためです。

### 二項演算子前後の空白（必須）[FlexFormatter対応]

**説明:** 二項演算子の前後には空白を含めてください。

variable = “variable” *// OK*

variable < 10 *// OK*

variable=”variable” *// NG: 空白が含まれていません*

variable< 10 *// NG: 空白が含まれていません*

コード 3‑8 二項演算子前後の空白例

**理由:** 可読性を向上させるためです。

### 三項演算子前後の空白（必須）[FlexFormatter対応]

**説明:** 三項演算子の前後には空白を含めてください。

var b:int = variable ? 0 : -1; *// OK*

var b:int = variable?0:-1; *// NG: 空白が含まれていません*

コード 3‑9 三項演算子前後の空白例

**理由:** 可読性を向上させるためです。

## 制御構文

### 制御指示子に対応する丸括弧前後の空白（必須）[FlexFormatter対応]

**説明:** if、for、while、do-while、switch-case、try-catchの制御指示子と丸括弧の間に空白を含めてください。さらに、丸括弧に継続して式やブロックが指定される場合にはそれらとの間にも空白を含めてください。

*// OK*

**if** (condition) {

}

*// OK*

**for** (var i:int = 1; I < length; i++) {

}

*// NG: 空白が含まれていません*

**if**(condition){

}

*// NG: 空白が含まれていません*

**for**(var i:int = 1; I < length; i++){

}

コード 3‑10 制御指示子に対応する丸括弧前後の空白例

**理由:** 可読性を向上させるためです。メソッドや関数の呼び出しと区別しやすくなります。

### コードブロックに対応する波括弧（必須）

**説明:** if、for、while、do-while、のコードブロックは、常に中括弧で囲ってください。

*// OK*

**if** (condition) {

do();

}

*// NG: 中括弧で囲われていません*

**if**(condition) do();

コード 3‑11 コードブロックに対応する波括弧の例

**理由:** 変更に強いコードとするためです。例えば中括弧のない条件ステートメントがステートメントに追加された場合に予期せぬ動作を引き起こします。

## 型注釈

### 型注釈の利用（必須）

**説明:** 変数の宣言には型注釈を含めてください。同様に、関数の戻り値にも型注釈を付加してください。

**理由:** コンパイル時に最適化されるからです。これによりメモリ消費量が減少し、パフォーマンスが向上する可能性があります。さらに、Flex BuilderやFlash Builderによる入力支援が有効になります。

**補足:** アスタリスク（＊）による型注釈も型注釈です。

### 空白を含まない型注釈のコロン（必須）

**説明:** 型注釈のコロンの前や後には空白を含めないでください。変数の型だけでなく、関数の戻り値型も同様です。

*// OK*

**public** **var** foo:Array = [];

*// NG: コロンの後に空白が含まれています*

**public** **var** bar: Array = [];

コード 3‑11 空白を含まない型注釈の例

**理由:** 一貫性のあるコーディングにより可読性を高めるためです。

## パッケージ

### 英小文字で開始するパッケージ名（必須）[FlexPMD対応]

**説明:** パッケージ名は小文字で開始してください。これはサブパッケージにもあてはまります。

**package** jp.co.ctc\_g.application // OK

**package** jp.co.ctc\_g.Application // NG

コード 3‑12 小文字と大文字のパッケージ名の例

**理由:** パッケージとクラスがより容易に区別できるようになるからです。また、これはActionScriptだけでなくJavaにおいても一般的な慣習です。

## インポート

### オンデマンドインポートの利用（禁止）

**説明:** パッケージをインポートする際には、\*を利用したオンデマンドインポートを利用しないでください。

**import** flash.events.\*; *// NG: オンデマンドインポートです*

コード 3‑13 オンデマンドインポートの例

**理由:** オンデマンドインポートを利用すると、当該ソースコードが依存している他のクラスや関数を把握し難くなるからです。

### 未使用クラスのインポート（禁止）

**説明:** 利用していないクラスや関数はインポートしないでください。

**理由:** 保守担当者に不要な誤解を与えてしまうからです。保守性が低下します。

### 順序あるインポートの宣言（必須）[FlexFormatter対応]

**説明:** インポート文の順序にルールを設けてください。例えば、以下の順序でインポート文を記述してください。また、それぞれのパッケージにおける記述順序は、自然順序付け（アルファベット昇順）とするのがよいでしょう。

* + - flashパッケージ
    - flashxパッケージ
    - mxパッケージ
    - sparkパッケージ
    - airパッケージ
    - com.adobeパッケージ
    - org.osmfパッケージ
    - サードパーティライブラリのパッケージ
    - 当該アプリケーションの他のパッケージ

**理由:** インポート文が順序づけられることで依存関係の一覧性が向上するからです。

**捕捉:** 自動でこの順序に並べ替える場合は、Flex Formatterが便利です。

## 名前空間

### スネークケースによる名前空間名（必須）

**説明:** 名前空間名は、全てアルファベット小文字のスネークケースで命名してください。

**public** **namespace** ctc\_flex = **"http://www.ctc-g.co.jp/cf"**; *// OK*

**public** **namespace** ctcFlex = **"http://www.ctc-g.co.jp/cf"**; *// NG: キャメルケースです*

**理由:** Flexの名前空間での一般的な慣習だからです。[Adobe Systems Inc 2008] に従います。

### 名前空間参照における名前修飾子の優先的な利用（必須）

**説明:** 名前空間を利用する場合には、use namespace文を利用してブロックごと名前空間で修飾するのではなく、メソッド呼び出し時やプロパティアクセス時に名前空間で修飾してください。

**public** **class** CustomNumericStepper **extends** NumericStepper {

**public** **function** **set** readOnlyText(value:Boolean):**void** {

mx\_internal::inputField.editable = !value;

}

}

コード 3‑14 プロパティアクセス時に名前空間で修飾する例

**理由:** 名前空間に属するメソッドやプロパティであることが明確になるからです。use文を利用すると、名前空間で保護されているメソッドやプロパティが不明確になります。これは[Adobe Systems Inc 2008] に従います。

**補足:** 利用している名前空間はインポート文にて確認できるため、依存している名前空間が不明確になることはありません。

## 修飾子

### 順序ある修飾子の宣言（必須）[FlexFormatter対応]

**説明:** クラスやインタフェース、関数、メソッド、プロパティを宣言する際に付加する修飾子は、以下の順序で宣言してください。

* + - override
    - public | protected | internal | private | カスタム名前空間
    - static | final
    - class | interface | function | const | var | namespace
    - get | set

**理由:** 修飾子が順序づけられることで、可読性が向上するからです。[Adobe Systems Inc 2008] に従います。

## メタデータタグ

### 順序あるクラスメタデータタグの宣言（必須）[FlexFormatter対応]

**説明:** クラスメタデータタグは、以下の順序で宣言してください。

* + - Event
    - Style
    - Effect
    - Exclude
    - その他（アルファベットの昇順）

**理由:** クラスメタデータタグが順序づけられることで、可読性が向上するからです。[Adobe Systems Inc 2008] に従います。

### 角括弧および丸括弧前後の空白（禁止）[FlexFormatter対応]

**説明:** メタデータタグを宣言する際、角括弧および丸括弧の前後には空白を含めないでください。ただし、メタデータタグプロパティを区切るカンマの後には半角空白を記述してください。

*// OK*

[**Event**(name=**"messageChange"**, type=**"flash.events.Event"**)]

*// NG: 空白が含まれていません*

[**Event**(name=**"messageChange"**,type=**"flash.events.Event"**)]

*// NG: 空白が含まれています*

[**Event** ( name = **"messageChange"**, type = **"flash.events.Event"** )]

コード 3‑15 メタデータタグに対する半角空白の適切な指定例

**理由:** 表記方法を統一し、可読性を向上させるためです。[Adobe Systems Inc 2008] に従います。

### AsDocの下のメタデータタグ

**説明:** メタデータタグを宣言する際、AsDocの下部に記載して下さい。

*// OK*

/\*\*

\* asdoc

\*/

[Bindable]

*// NG1: AsDocの上部にメタデータタグが記載されています*

[Bindable]

/\*\*

\* asdoc

\*/

*// NG2: AsDocの上部にメタデータタグが記載されています*

[Bindable]

/\*\*

\* asdoc

\*/

コード 3‑16 メタデータタグAsDocの下のメタデータタグ

**理由:** 表記方法を統一し、可読性を向上させるためです。また、Javaのコーディング規約とも整合性を確保しています。

## 関数

### ローワーキャメルケースの適用（必須）

**説明:** 関数名はローワーキャメルケースを利用してください。つまり、関数名を構成する最初の単語の先頭文字をアルファベット小文字で記述してください。

**理由:** 表記方法を統一し、可読性を向上させるためです。コンストラクタやコピーコンストラクタとの区別も容易になります。

### 動詞で始まる関数名（必須）

**説明:** 関数名は動詞あるいは動詞+名詞を基本としてください。ただし、イベントハンドラに関してはこの基準に従う必要はありません。詳細は「3.17.4 イベントハンドラ名へのHandlerサフィックスの追加（必須）」を参照してください。

**public** **function** find(id:String):Foo {

}

**public** **function** showMessage(message:Message):**void** {

}

コード 3‑16 動詞で始まる関数の例

**理由:** 関数であることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### 丸括弧前後の空白（禁止）[FlexFormatter対応]

**説明:** 関数を呼び出す際の丸括弧の前後に空白を含めないでください。

find(**"htoEa"**); *// OK*

find (**"htoEa"**); *// NG: 空白が含まれています*

find( **"htoEa"** ); *// NG: 空白が含まれています*

コード 3‑17 関数を呼び出す際の適切な空白例

**理由:** 可読性を向上させるためです。制御指示子との区別もつきやすくなります。

## クラス・インタフェース

### アッパーキャメルケースの適用（必須）

**説明:** クラス名やインタフェース名はアッパーキャメルケースを利用してください。つまり、これらの名前を構成する最初の単語の先頭文字をアルファベット大文字で記述してください。

**理由:** 表記方法を統一し、可読性を向上させるためです。関数やメソッド、プロパティとの区別も容易になります。

### 動詞で始まるクラス名（禁止）

**説明:** クラス名やインタフェース名は動詞で開始しないでください。

*// NG: 動詞で開始されています*

**public** **class** ShowMessage {

}

コード 3‑18 動詞で始まらないクラス名の例

**理由:** クラスやインタフェースであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### インタフェース名への”I”プレフィックス追加（必須）[FlexPMD対応]

**説明:** インタフェース名への”I”プレフィックス追加をしてください。

**public** **interface** IExample {

}

コード 3‑19 インタフェースに"I"プレフィックスを付加した例

**理由:** インタフェース名に”I”プレフィックスを付加することはFlex標準ライブラリのコードでは一般的ですが、他のライブラリや言語でもそうであるとは言えないからです。サーバサイドの開発言語と整合性を図ることも考慮して、インタフェース名への"I"プレフィックス追加は任意とします。

### 順序ある継承と実装の宣言（必須）[FlexFormatter対応]

**説明:** クラス宣言時の継承と実装は、以下の順序で宣言してください。なお、複数のインタフェースを実装する際には、アルファベットの昇順に記述してください。

* + - extends
    - implements

**public** **class** Foo **extends** Bar **implements** IBaz, IQux {

}

コード 3‑20 順序ある継承と実装の宣言例

**理由:** 継承と実装の宣言が順序づけられることで、可読性が向上するからです。

### flash.events.Eventを継承したクラスの名称（必須）

**説明:** Eventクラスのサブクラスには、クラス名にサフィックスとしてEventを付加してください。

**public** **class** FooEvent **extends** Event {

}

コード 3‑21 EventクラスのサブクラスへのEventサフィックスの付加

**理由:** イベントクラスであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### Errorを継承したクラスの名称（必須）

**説明:** Errorクラスのサブクラスには、クラス名にサフィックスとしてErrorを付加してください。

**public** **class** FooError **extends** Error {

}

コード 3‑22 ErrorクラスのサブクラスへのErrorサフィックスの付加

**理由:** 例外クラスであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### mx.formatters.Formatterを継承したクラスの名称（必須）

**説明:** Formatterクラスのサブクラスには、クラス名にサフィックスとしてFormatterを付加してください。

**public** **class** FooFormatter **extends** Formatter {

}

コード 3‑23 FormatterクラスのサブクラスへのFormatterサフィックスの付加

**理由:** フォーマッタクラスであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### mx.validators.Validatorを継承したクラスの名称（必須）

**説明:** Validatorクラスのサブクラスには、クラス名にサフィックスとしてValidatorを付加してください。

**public** **class** FooValidator **extends** Validator {

}

コード 3‑24 ValidatorクラスのサブクラスへのVlidatorサフィックスの付加

**理由:** 入力値検証クラスであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### 動的クラスの作成と利用（禁止）[FlexPMD対応]

**説明:** 動的クラス[[4]](#footnote-4)は基本的に使用しないでください。作成あるいは利用する場合は、充分に妥当な理由がある場合に限定してください。

**public** **dynamic** **class** DynamicExample {

**public** **function** DynamicExample() {

}

}

コード 3‑25 動的クラスの例

**理由:** プログラムが複雑になり、保守性が低下するからです。動的クラスのインスタンスはメモリ効率が悪く、プロパティやメソッドへのアクセスパフォーマンスも相対的に芳しくありません。

**補足:** 標準ライブラリのAPIやサードパーティライブラリのAPIが動的クラスのインスタンスを要求している場合は、この限りではありません。また、テンポラリに値を保持するような小さな動的クラスのインスタンスのパフォーマンスへの影響は限定的であり、神経質になるほどではありません。

### prototypeプロパティの利用（禁止）

**説明:** prototypeプロパティは利用しないでください。一般的に、プロパティやメソッドの動的な追加・削除・オーバライドは通常の業務アプリケーションの範疇ではありません。よって、システム共通処理を担当するアーキテクチャチームがprototypeの利用を判断し、その利用を管理してください。

**public** **dynamic** **class** Prototype {

prototype.foo = **"foo"**;

prototype.bar = **function**():**void** {

}

}

コード 3‑26 prototypeプロパティを利用した例

**理由:** prototypeオブジェクトとtraitオブジェクトが組み合わされることによって、予期せぬバグの温床となるからです。また、動的クラスでのみprototypeプロパティは実質的な効果があるという点も、prototypeオブジェクトを推奨しない理由のひとつです[[5]](#footnote-5)。動的クラスが非推奨である理由については「3.13.9 動的クラスの作成と利用（禁止）」を参照してください。

**補足:** prototypeを利用してもActionScript 3.0以降のtraitによる型システムとの型互換性は確保されます。

## プロパティ

### 可変プロパティ名のローワーキャメルケースの適用（必須）

**説明:** 可変プロパティ名にはローワーキャメルケースを利用してください。つまり、プロパティ名を構成する最初の単語の先頭文字をアルファベット小文字で記述してください。

**public var** variable:uint;

コード 3‑27 可変プロパティ宣言例

**理由:** 表記方法を統一し、可読性を向上させるためです。クラスやインタフェース、コンストラクタやコピーコンストラクタとの区別も容易になります。

### 不変プロパティ名のアッパースネークケースの適用（必須）

**説明:** 不変プロパティ名にはアッパースネークケースを利用してください。つまり、プロパティ名を構成する全ての単語をアルファベット大文字で記述し、単語間をアンダースコアで結んでください。

**public** **const** CONSTANT\_VARIABLE:String = **"constant"**;

コード 3‑28 不変プロパティ宣言例

**理由:** 表記方法を統一し、可読性を向上させるためです。クラスやインタフェース、コンストラクタやコピーコンストラクタとの区別も容易になります。

### 動詞で始まるプロパティ名（禁止）

**説明:** 動詞で始まるプロパティ名は禁止します。ただし、Function型のプロパティの場合には動詞で開始しても構いません。

**public** **var** compare:Function = **function**(lhs:Foo, rhs:Foo):int {

**return** lhs.value.compare(rhs.value);

};

コード 3‑29 動詞で始まるプロパティの例

**理由:** 動詞で開始するとプロパティと関数やメソッドの区別がつきにくくなるからです。

**補足:** 上記コードの通り、ActionScriptはFunction型の変数を宣言する際、戻り値の型や引数の型についての情報を付帯させることができませんので、利用には注意してください。

### ストレージ変数に対する”\_”プレフィックスの適用（必須）

**説明:** プロパティと直接対応するストレージ変数の名前は、プロパティ名に”\_”をプレフィックスとして付加したものとします。

**private** **var** \_variable:uint;

**public** **function** **get** variable():uint {

**return** \_variable;

}

コード 3‑30 ストレージ変数へのアンダースコア付加例

**理由:** ActionScriptコーディングの慣習だからです。[Adobe Systems Inc 2008] に従います。

### setterの引数名（必須）

**説明:** setterの引数名には、そのプロパティの意味や型にかかわらず、valueを指定してください。

**public** **function** **set** variable(value:uint):**void** {

\_variable = value;

}

コード 3‑31 setterの引数名にvalueを指定する例

**理由:** ActionScriptの慣習だからです。[Adobe Systems Inc 2008] に従います。

### 不要なgetterとsetterの定義（禁止）

**説明:** 定義する必要のないgetterとsetterを定義しないでください。例えば、あるプロパティをpublicと宣言しても問題ない[[6]](#footnote-6)にもかかわらず、過度なカプセル化への意識からストレージ変数とgetterとsetterを定義することは避けてください。

**private** **var** \_verbose:Boolean;

**public** **function** **get** verbose():Boolean {

**return** \_verbose;

}

**public** **function** **set** verbose(value:Boolean):**void** {

\_verbose = value;

}

コード 3‑32 冗長なプロパティ宣言の例

**理由:** 冗長な記述によりソースコードの見通しが悪くなるからです。

**補足:** Bindableメタデータタグの利用により、バイトコードレベルでgetterとsetterが自動生成される場合がありますが、本基準はあくまでもソースコードレベルでの指針を提示しています。

### 型制約としてのArrayElementTypeメタデータタグの信頼（禁止）

**説明:** 配列（Array）型変数宣言時にArrayElementTypeメタデータタグを利用しても構いませんが、Array内要素の型制約としては不充分ですので信頼しないでください。ArrayElementTypeメタデータタグを付加していたとしても、必要な場合にはArrayに含まれている要素型を検証してください。

**理由:** このメタデータタグは、MXMLにてタグを利用して当該変数に値を代入する場合に限りコンパイルエラーとなるのであり、それ以外の全てのケースにおいてエラーとならないからです。ArrayElementTypeメタデータタグはC++のテンプレートやJavaにおけるジェネリクスとは異なり、型の安全性を保証するものではありません。

**注意:** 本基準は、ArrayElementTypeメタデータタグの利用を制限するものではありません。MXMLにてコンパイルエラーが検出されるようになる点は非常に優れています。

**補足:** このメタデータタグは、コンパイル後のバイトコードには含まれません。よって、バージョン2以降のAVM[[7]](#footnote-7)を利用していたとしても実行時型違反の検出対象とはなりません。

## メソッド

### ローワーキャメルケースの適用（必須）

**説明:** メソッド名はローワーキャメルケースを利用してください。つまり、メソッド名を構成する最初の単語の先頭文字をアルファベット小文字で記述してください。

**理由:** 表記方法を統一し、可読性を向上させるためです。コンストラクタやコピーコンストラクタとの区別も容易になります。

### 動詞で始まるメソッド名（必須）

**説明:** メソッド名は動詞あるいは動詞+名詞を基本としてください。ただし、イベントハンドラに関してはこの基準に従う必要はありません。詳細は「3.17.4 イベントハンドラ名へのHandlerサフィックスの追加（必須）」を参照してください。

**public** **class** Foo {

**public** **function** find(id:String):Foo {

}

**public** **function** showMessage(message:Message):**void** {

}

}

コード 3‑33 動詞で始まるメソッドの例

**理由:** メソッドであることが明確になるからです。[Adobe Systems Inc 2008] に従います。

### 丸括弧前後の空白（禁止）[FlexFormatter対応]

**説明:** メソッドを呼び出す際の丸括弧の前後に空白を含めないでください。

foo.find(**"htoEa"**); *// OK*

foo.find (**"htoEa"**); *// NG: 空白が含まれています*

foo.find( **"htoEa"** ); *// NG: 空白が含まれています*

コード 3‑34 メソッドを呼び出す際の適切な空白例

**理由:** 可読性を向上させるためです。制御指示子との区別もつきやすくなります。

## 変数

### 変数名のローワーキャメルケースの適用（必須）

**説明:** 変数名にはローワーキャメルケースを利用してください。つまり、変数名を構成する最初の単語の先頭文字をアルファベット小文字で記述してください。

**public var** variable:uint;

コード 3‑35 可変プロパティ宣言例

**理由:** 表記方法を統一し、可読性を向上させるためです。

## イベント

### カスタムイベントクラスにおけるcloneメソッドのオーバライド（必須）[FlexPMD対応]

**説明:** カスタムイベントクラスを作成した場合は、必ずcloneメソッドをオーバライドしてください。

**public** **class** CustomEvent **extends** Event {

**public** **var** \_data:FooProperty;

**public** **function** **get** data():FooProperty {

**return** \_data;

}

**public** **function** CustomEvent(

type:String,

data:FooProperty,

bubbles:Boolean = **false**,

cancelable:Boolean = **false**) {

**super**(type, bubbles, cancelable);

}

**override** **public** **function** clone():Event {

**var** avatar:CustomEvent

= **new** CustomEvent(type, \_data, bubbles, cancelable);

*// Eventからプロパティをコピー(省略)して返却*

**return** avatar;

}

}

コード 3‑36 カスタムイベントでのcloneメソッドオーバライド例

**理由:** メソッドをオーバライドしないと、サブクラスの属性値がコピーされないからです。また、オーバライドしないと呼び出し側はクローンされたインスタンスをCustomEvent型として利用できないという問題があります。

### カスタムイベントクラスによる変更可能なプロパティの公開（禁止）[FlexPMD対応]

**説明:** カスタムイベントクラスが変更可能なプロパティを公開することは禁止です。カスタムイベントに必要な追加のプロパティ値はコンストラクタ引数で受け取り、インスタンス生成後の外部からの変更を許可しないでください。

**理由:** カスタムイベントが変更可能なプロパティを公開すると、イベントシーケンスにおいてデータが任意に変更可能になりコードの保守が難しくなるからです。例えば、あるイベントリスナによるイベントプロパティの変更は、そのイベントをリッスンしていた全てのリスナーに影響を与える可能性があります。

**注意:** 例えばArrayクラスやDictionaryクラス、Objectクラスなどのコンテナとして機能するクラスのインスタンスを公開する場合、読取専用プロパティとすることでそのインスタンスの参照は変更できなくなりますが、そのインスタンスの内容は変更できます。

### イベント名のハードコード（禁止）[FlexPMD対応]

**説明:** イベントを生成してディスパッチする場合やリスナーをアタッチする場合は、イベント定数を利用してください。定数の代わりに文字列リテラルをハードコードすることを禁止します。

**理由:** タイプミスによる不要なバグの混入を防ぐためです。また、マジックストリングはリファクタリング容易性やコード保守性を犠牲にします。

### イベントハンドラ名へのHandlerサフィックスの追加（必須）

**説明:** イベントハンドラ名には、イベントの種類を示す語句に”Handler”を結合した名称としてください。もし、イベントハンドラがサブコンポーネントからディスパッチされるイベントを処理する場合には、ハンドラ名の前にコンポーネント名（MXMLの場合はid値）を付け、アンダースコアで接続するようにしてください。

**public** **function** creationCompleteHandler(event:FlexEvent):**void** {

}

**public** **function** button\_mouseClickHandler(event:MouseEvent):**void** {

}

3‑37 イベントハンドラ名の例

**理由:** Flash Builder のイベントハンドラ自動生成機能を利用した場合、上記形式でメソッドが生成されるからです。これは [Adobe Systems Inc 2008] に準じています。

**注意:** 本基準のレベルはFlash Builderとの連携を考慮して、推奨としました。この点を重視しないのであれば、本基準のレベルを任意として扱って構いません（イベントハンドラ名の一貫性は保証してください）。

## ドキュメンテーションコメント

### 行頭のアスタリスク（必須）

**説明:** ドキュメンテーションコメントは”/\*\*”の次の行から開始して、各行の先頭にはアスタリスク（＊）を付加してください。終了は単独行に”\*/”を記述してください。

/\*\*

\* このクラスはドキュメンテーションコメントの

\* サンプルを提示するためのクラスです。

\*/

**public** **class** DocumentationComment {

}

コード 3‑38 ドキュメンテーションコメントの単純な例

**理由:** ドキュメンテーションコメントの一般的な慣習だからです。行頭にアスタリスクを付加するのは、それがドキュメンテーションコメント行であることを明示するためです。

### public要素およびprotected要素へのドキュメンテーションコメント（必須）[FlexPMD対応]

**説明:** クラス、インタフェース、名前空間、コンストラクタ、関数、メソッド、プロパティ、メタデータタグなどのプログラム要素がpublic あるいはprotectedとして定義されている場合には、ドキュメンテーションコメントを記述してください。

**理由:** 並行開発時や保守時に有用な情報だからです。

### 関数やメソッドのドキュメンテーションコメントタグ（必須）

**説明:** 関数やメソッドにAsDocを記述する場合、以下のAsDocタグを過不足なく記述してください。

* + - @param
    - @return
    - @throws
    - @see

/\*\*

\* このメソッドは指定された検索条件から何かを探します。

\* 検索条件に一致する何かが見つからない場合、

\* このメソッドは長さのない配列を返却します。

\* @param query 検索条件

\* @return 検索条件に一致する何かの配列

\* @throws BadQueryFomatError 検索条件の書式が不正な場合

\* @see QueryParser

\*/

**public** **function** findSomething(query:String):Array {

doStuff();

}

コード 3‑39 関数およびメソッドのドキュメンテーションコメントの例

**理由:** 並行開発時や保守時に有用な情報だからです。

### getterおよびsetterとドキュメンテーションコメント（必須）

**説明:** ストレージ変数とそれに対応するgetterおよびsetterが定義されている場合には、ストレージ変数とsetterにprivateタグを付加して、getterにドキュメンテーションコメントを記述してください。getterあるいはsetterのどちらかしかない場合には、ストレージ変数にprivateタグを付加してメソッドにドキュメンテーションコメントを記述してください。ストレージ変数がなく、getterとsetterの両方がある場合、getterにドキュメンテーションコメントを記述して、setterにはprivateを付加してください。以上をまとめると以下の表の通りです。

表 3‑1 getterおよびsetterとドキュメンテーションコメントの関連

|  |  |  |  |
| --- | --- | --- | --- |
| ストレージ変数 | getter | setter | ドキュメンテーションコメント |
| あり | なし | なし | ストレージ変数 |
| あり | あり | あり | getter |
| あり | あり | なし | getter |
| あり | なし | あり | setter |
| なし | あり | あり | getter |
| なし | あり | なし | getter |
| なし | なし | あり | setter |

**理由:** ドキュメンテーションコメントの記述方法に一貫性を持たせるためです。

### Eventのドキュメンテーションコメント（必須）

**説明:** EventメタデータタグにはeventTypeタグにてそのイベントの種類を指定してください。このeventTypeタグによって、当該イベント発生元クラスのAsDocからイベント定数へとリンクが挿入されます。また、イベント定数にeventTypeタグを付けることにより、定数に付加されているAsDocがイベント発生元クラスのAsDocへコピーされます。

/\*\*

\* このイベントはコメントが変更された場合にトリガされます。

\* @eventType jp.co.ctc\_g.cf.dp1.event.CommentEvent.CHANGE\_COMMENT

\*/

[**Event**(name=**"changeComment"**, type=**"jp.co.ctc\_g.cf.dp1.event.CommentEvent"**)]

/\*\*

\* このクラスはドキュメンテーションコメントの

\* サンプルを提示するためのクラスです。

\*/

**public** **class** DocumentationComment {

}

コード 3‑40 イベント発生元クラスのAsDoc例

/\*\*

\* このイベントはドキュメンテーションコメントの

\* サンプルを提示するためのイベントクラスです。

\*/

**public** **class** CommentEvent **extends** Event {

/\*\*

\* CHANGE\_COMMENT 定数により、

\* イベントオブジェクトの type プロパティ値が

\* changeComment イベント用に定義されます。

\* @eventType changeComment

\*/

**public** **static** **const** CHANGE\_COMMENT:String = **"changeComment"**;

**public** **function** CommentEvent(type:String) {

**super**(type);

}

}

コード 3‑41 イベント定数値のAsDoc例

**理由:** イベントをプログラムする際に必要な情報だからです。イベント定数を容易に参照することができます。

### デフォルト値の明示（必須）

**説明:** プロパティやスタイルにデフォルト値があれば、それを明示してください。

/\*\*

\* デフォルトタグの使用を例示するためのプロパティです。

\* @default "example"

\*/

**public** **var** defaultTagExample:String = **"example"**;

}

コード 3‑42 デフォルト値を明示する例

**理由:** デフォルト値を参照するのにソースコードを参照する必要がなくなるからです。なお、イベント定数の場合は、defaultタグではなくeventTypeタグを利用してください。

**注意:** デフォルト値が計算結果である場合や複雑な記述が必要な場合には、ドキュメンテーションコメントでデフォルト値を明示してください。

**補足:** 例はプロパティで解説しましたが、Styleメタデータでもデフォルト値を明示するようにしてください。

## 外部クラスの取り込み

### スクリプトタグ内部でのincludeの使用（禁止）

**説明:** include文による外部ファイルの取り込みは禁止します。

# MXML



## ソースファイル

### クラス名としてのファイル名（必須）

**説明:** ファイル名はActionScriptクラス名の命名基準「3.13 クラス・インタフェース」に従ってください。

**理由:** MXMLは、トップレベルに定義されたタグに対応するクラスを継承した、アクションスクリプトクラスだからです。

## ブロックとインデント

### 半角空白によるインデント（必須）[FlexFormatter対応]

**説明:** インデントはタブではなく半角空白を利用してください。また、半角空白数は2とします。

**理由:** タブによるインデントはエディタにより見た目が変わってしまうだけでなく、空白と混在することでインデントが揃わなくなるからです。また、インデントに利用する半角空白数を2としているのは、MXMLはタグのネストが多く発生するためネストが深くなるからです。

### 最大行幅の統一（必須）[FlexFormatter対応]

**説明:** ソースコードの最大行幅を統一してください。最大行幅は80としてください。

**理由:** 利用しているエディタや参照しているソースにかかわらず、見た目を統一するためです。

### 継続行のインデント（必須）[FlexFormatter対応]

**説明:** 継続行のインデントは、最初の属性と同じ位置とすることを推奨します。

<s:TextInput id="text" focusIn="text\_focusInHandler(event)"

focusOut="text\_focusOutHandler(event)" />

コード 4‑1 継続行のインデント例

**理由:** 継続行であることが明確になるからです。属性位置を揃えることで、どのタグに当該属性が定義されているかを明示することもできます。

### 1行1タグ（必須）[FlexFormatter対応]

**説明:** タグは1行に1タグを記述してください。開始タグと終了タグの中に別のタグが含まれる場合には、開始タグと終了タグを同一行に記述しないでください。

<mx:Form>

<mx:FormItem label="テキスト">

<s:TextInput id="text" />

</mx:FormItem>

<mx:FormItem>

<s:Button id="button" click="button\_clickHandler(event)" />

</mx:FormItem>

</mx:Form>

コード 4‑2 1行1タグの例

<fx:Declarations>

<s:ArrayCollection>

<fx:String>First</fx:String>

<fx:String>Second</fx:String>

<fx:String>Third</fx:String>

</s:ArrayCollection>

</fx:Declarations>

コード 4‑3 非タグ要素を含む場合の例

**理由:** 一貫したMXMLの記述により、可読性が向上するからです。

### 順序あるMXMLタグの宣言（必須）

**説明:** タグは、以下の順序で宣言してください。また、これらのグループとグループの間は少なくとも1行あけてください。

* + - XML宣言
    - ルートXMLエレメント（名前空間指定を含む）
    - メタデータ
    - スタイル
    - スクリプト
    - データモデル（XMLやモデル）
    - サービス
    - バリデータ
    - フォーマッタ
    - ステート
    - トランジション
    - エフェクト
    - バインディング
    - 画面コンポーネントやコンテナ

**理由:** 一貫したMXMLの構成により可読性が向上するからです。

**補足:** Flex SDK 4以降を利用する場合にはfx:Declarationsタグを利用してください。

### 属性のカテゴライズと順序付け（必須）[FlexFormatter対応]

**説明:** MXMLタグの属性は、カテゴライズして順序付けて記述してください。例えば以下のようなカテゴリに分割し、以下の通りの順序で記述します。

* + - 名前空間
    - プロパティ
    - イベント
    - エフェクト
    - スタイル

**理由:** 一貫したMXMLの構成により可読性が向上するからです。

## ID属性

### 最初の属性としてのID属性（必須）[FlexFormatter対応]

**説明:** あるタグにID属性を付与する場合、必ず最初の属性として記述してください。

<s:Button id="button"

text="実行"

click="button\_clickHandler(event)" />

コード 4‑4 ID属性を最初の属性として記述する例

**理由:** タグのID属性は、スクリプトなどの外部要素から参照されることが相対的に多いからです。属性の中で最も目立つ位置に宣言しておくことが可読性を考慮すると望ましいでしょう。

### ローワーキャメルケースの適用（必須）

**説明:** ID属性値（つまりID名）にはローワーキャメルケースを利用してください。

<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark">

<s:Label id="errorMessage" />

</s:Application>

コード 4‑5 ローワーキャメルケースを利用したID名の例

**理由:** ID属性は、それが付加されたタグを含むMXMLのpublicプロパティ名として扱われるからです。例えば、上記コード例は以下のようにActionScriptにコンパイルされます[[8]](#footnote-8)。

**public** **class** MXMLTest **extends** spark.components.Application {

[**Bindable**]

**public** **var** errorMessage:spark.components.Label;

*// 以下省略*

}

コード 4‑6 コンパイルされたMXMLにおけるID名の取り扱い例

## スクリプトタグ

### 長すぎるスクリプトタグ（禁止）[FlexPMD対応]

**説明:** スクリプトタグ内に記述するスクリプトは長くとも50行としてください。

**理由:** スクリプトが長くなると、MXMLの可読性が低下し、保守が難しくなるからです。MXML内に記述するスクリプトが多くなるのであれば、そのコードの責務に応じてActionScriptファイルを利用してください。

## イベントハンドラ属性

### イベントハンドラ内での複文の実行（禁止）

**説明:** イベントハンドラ属性に指定する実行可能コードは単文に留めてください。複文は禁止します。

<!-- OK -->

<s:Button id="button1" click="button1\_clickHandler(event)"/>

<!-- OK -->

<s:Button id="button2" click="currentState = STATE\_LIST"/>

<!-- NG: 複文を実行しています -->

<s:Button id="button3" click="doStuff(event);doMore(event)"/>

**理由:** イベントハンドラ属性に直接指定する文字列を極力シンプルに保つためです。MXMLタグとActionScriptの混在は保守性を低下させます。

## 外部クラスの取り込み

### スクリプトタグ内部でのincludeの使用（禁止）

**説明:** include文による外部ファイルの取り込みは禁止します。

### MXML記述部での<mx:Script source=”” />の使用（禁止）

**説明:** MXML記述部において、<mx:Script source=”” /> タグを用いた外部ファイルの取り込みは禁止します。

# CSS



## ソースファイル

### ファイル名へのローワーキャメルケースの適用（必須）

**説明:** ファイル名にはローワーキャメルケースを利用してください。

**理由:** クラスファイルやMXMLファイルと区別するためです。

### スタイル名へのローワーキャメルケースの適用（必須）

**説明:** スタイル名にはローワーキャメルケースを利用してください。

**理由:** ローワーキャメルケースでCSSファイルにスタイル名を定義しようとするとFlex BuilderやFlash Builderのコンテントアシストが無効になるからです。これらのIDEは、通常のCSSと同じ”-”記号による単語間の接続でのみ、コンテントアシストが有効になります。つまり、MXMLでの定義と同様にローワーキャメルケースのスタイル名をCSSで利用すると、記述の一貫性が保たれるものの、IDEによる支援を受けられない状況になります。よって、本基準を任意としています。

**注意:** 本基準は任意ですが、プロジェクトではどちらかに統一することを推奨します。

## ブロックとインデント

### 半角空白によるインデント（必須）

**説明:** インデントはタブではなく半角空白を利用してください。また、半角空白数は2とします。

**理由:** タブによるインデントはエディタにより見た目が変わってしまうだけでなく、空白と混在することでインデントが揃わなくなるからです。また、インデントに利用する半角空白数を2としているのは、MXMLやActionScriptと整合をとるためです。

### 行末でのブロック開始（必須）

**説明:** セレクタを記述した行の末尾からブロックを開始することを推奨します。ブロック開始前には必ず半角空白を置いてください。

TextInput.placeHolder { /\* 行末でブロック開始 \*/

color: #999;

}

コード 5‑1 行末でのブロック開始例

**理由:** 一貫したCSSの記述により、可読性が向上するからです。

### 行頭でのブロック終了（必須）

**説明:** 行頭でブロックを終了してください。

TextInput.placeHolder {

color: #999;

}　/\* 行頭でブロック終了 \*/

コード 5‑2 行頭でのブロック終了例

**理由:** 一貫したCSSの記述により、可読性が向上するからです。

### スタイル名とスタイル値の間の空白（必須）

**説明:** スタイル名とスタイル値のセミコロンの後に半角空白を置いてください。

HGroup {

padding-top: 5;

padding-botton: 5;

}

コード 5‑3 セミコロンの後に空白を配置する例

**理由:** 一貫したCSSの記述により、可読性が向上するからです。

### 最大行幅の統一（必須）

**説明:** ソースコードの最大行幅を統一してください。最大行幅は80としてください。

**理由:** 利用しているエディタや参照しているソースにかかわらず、見た目を統一するためです。

### 1行1スタイル（必須）

**説明:** 1行に1スタイルのみを定義してください。

Application {

padding-left: 5;

padding-right: 5;

}

コード 5‑4 1行に1スタイルを定義する例

**理由:** 一貫性のある記述により可読性が向上するからです。スタイルの見落としも少なくなります。

## スタイルクラス[[9]](#footnote-9)

### スタイルクラス名へのローワーキャメルケースの適用（必須）

**説明:** スタイルクラス名にはローワーキャメルケースを利用してください。

.accountingInfo {

background-image: Embed("assets/images/doller.png");

background-attachment: fixed;

font-size: 14;

}

コード 5‑5 ローワーキャメルケースによるスタイルクラス名の指定例

**理由:** 一貫性のある名前付けにより可読性が向上するからです。[Adobe Systems Inc 2008] に従います。

1. キャメルケースとは、単語の区切りを大文字で表現する記法です。FlexApplicationやFlexCodingConvensionsはキャメルケースです。競合する記法はスネークケースで、これは単語の区切りをアンダースコアで表現する記法です。例えば、flex\_applicationやflex\_coding\_convensionsはスネークケースです。 [↑](#footnote-ref-1)
2. 2010年8月11日現在、Flash Builder 4で確認しました。 [↑](#footnote-ref-2)
3. Flex Formatter（http://sourceforge.net/projects/flexformatter/） [↑](#footnote-ref-3)
4. クラス宣言がdynamic修飾子により修飾されたクラスを指します。 [↑](#footnote-ref-4)
5. コンパイラをstrictモードで実行し、かつActionScript3.0を利用している場合です。 [↑](#footnote-ref-5)
6. 問題ない場合とは、getterで対応するストレージ変数を返却するだけ、setterで対応するストレージ変数に引数の値を格納しているだけ、という場合を指します。 [↑](#footnote-ref-6)
7. バージョン2以降のAVMでは、実行時型情報（RTTI）による型違反を検出する機構が組み込まれています。 [↑](#footnote-ref-7)
8. コンパイラにより自動生成されたコードから改行位置の修正やコメントの削除など、一部変更を加えてあります。 [↑](#footnote-ref-8)
9. スタイルクラスとは、mx.core.UIComponentのstyleNameに代入できる文字列のことです。正式には”クラス”ですが、ActionScriptのクラスと区別がつかなくなるため、本ドキュメントではスタイルクラスと呼びます。なお、公式のドキュメントにおいても散見されますが、スタイルクラス名のことをクラスセレクタと呼ぶことは誤りです。これはActionScriptのタイプ（型）をタイプセレクタと呼ぶのと同じくらいおかしなことです。 [↑](#footnote-ref-9)