

# Documentando Código Flex

## MXML e ActionScript 3



DClick Desenvolvimento de Software Ltda

[dev@dclick.com.br](mailto:dev@dclick.com.br)

Rafael M. Martinelli

*v1.0 – Janeiro/2008*

## Índice

1. Pré-requisito .....	1
2. Objetivo.....	1
3. Introdução .....	1
4. ASDoc .....	1
5. Criando Comentários .....	3
6. Formatando os comentários.....	4
6.1 Usando a tag @private.....	4
6.2 Usando HTML tags.....	5
6.3 Caracteres Especiais .....	5
6.4 Escondendo texto em comentários .....	6
7. O processo de <i>Parse</i> .....	6
8. Documentando elementos ActionScript .....	7
8.1 Documentando classes.....	7
8.2 Documentando propriedades .....	7
8.3 Variáveis.....	7
8.3.1 Propriedades definidas com gets e sets implícitos.....	8
8.4 Documentando Métodos.....	9
8.5 Documentando effects, events e styles.....	10
8.6 Documento arquivos MXML.....	11
9. Usando a Tag @see .....	12
10. Executando a ferramenta ASDoc .....	13
10.1 Excluindo classes .....	13
10.2 Pastas .....	13
10.3 Exemplos de Código .....	14
11. Arquivo de exemplo final .....	18
12. Cabeçalho de arquivos .....	27
12.1 AS .....	27
12.2 MXML .....	28
13. Conclusão .....	28
14. Apêndices .....	28

## 1. Pré-requisito

Conhecimento básico de programação em ActionScript e MXML.

## 2. Objetivo

Este documento tem como objetivo demonstrar melhores práticas de documentação de classes ActionScript através do uso da ferramenta ASDoc fornecida pela Adobe.

## 3. Introdução

A primeira versão do ASDoc surgiu para documentar classes .as do Flash, quando o Flex ainda nem existia. Podemos dizer a grosso modo, que o ASDoc foi baseado no JavaDoc para documentar classes .as. Dessa forma, os desenvolvedores já acostumados com o JavaDoc, terão grande facilidade para aprender a documentar classes em ActionScript. Veremos ao longo desse documento que existem algumas particularidades com relação ao JavaDoc, algumas limitações e alguns benefícios.

Em resumo, podemos dizer que o ASDoc é uma boa ferramenta de documentação de código em ActionScript e o seu uso é altamente recomendado em qualquer tipo/tamanho de projeto.

## 4. ASDoc

O ASDoc é uma ferramenta que já vem embutido no SDK do Flex a partir da versão 2.01 hotfix 1. O ASDoc é executado a em linha de comando tanto no Windows quanto no Mac OS. No Windows pode ser encontrado em:

```
<path de instalação do Flex Builder>/flex/Flex SDK 2/bin/asdoc.exe
```

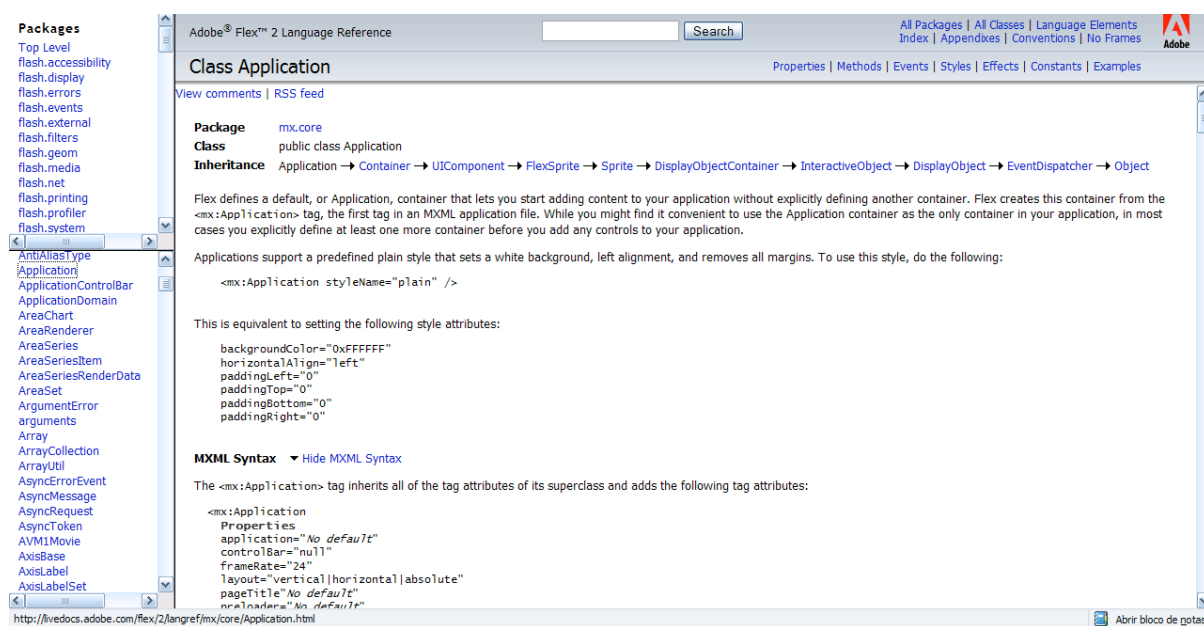
Por ser executado via linha de comando, iremos ver ao longo deste documento que a melhor maneira de se executar o ASDoc é através de um script ANT.

O ASDoc funciona fazendo o parse de classes ActionScript e arquivos MXML gerando a API de documentação para todos os métodos e propriedades `public` e `protected` e

para as metadatas [Bindable], [DefaultProperty], [Event], [Style] e [Effect].

Para usar a ferramenta, podemos especificar uma única classe, múltiplas classes, um namespace, um diretório ou a combinação deles.

O resultado final que o ASDoc irá gerar é uma documentação em HTML no estilo JavaDoc. A própria Adobe usa o ASDoc para gerar o Adobe Flex Language Reference. Abaixo podemos ver um exemplo do resultado final que teremos:



The screenshot shows the Adobe Flex 2 Language Reference documentation for the Class Application. The interface includes a search bar, navigation links (All Packages, All Classes, Language Elements, Index, Appendices, Conventions, No Frames), and a sidebar with a package tree. The main content area displays the following information:

**Class Application**  
 Properties | Methods | Events | Styles | Effects | Constants | Examples

View comments | RSS feed

**Package** mx.core  
**Class** public class Application  
**Inheritance** Application → Container → UIComponent → FlexSprite → Sprite → DisplayObjectContainer → InteractiveObject → DisplayObject → EventDispatcher → Object

Flex defines a default, or Application, container that lets you start adding content to your application without explicitly defining another container. Flex creates this container from the `<mx:Application>` tag, the first tag in an MXML application file. While you might find it convenient to use the Application container as the only container in your application, in most cases you explicitly define at least one more container before you add any controls to your application.

Applications support a predefined plain style that sets a white background, left alignment, and removes all margins. To use this style, do the following:

```
<mx:Application styleName="plain" />
```

This is equivalent to setting the following style attributes:

```
background-color="0xFFFFFFFF"
horizontal-align="left"
padding-left="0"
padding-top="0"
padding-bottom="0"
padding-right="0"
```

**MXML Syntax** Hide MXML Syntax

The `<mx:Application>` tag inherits all of the tag attributes of its superclass and adds the following tag attributes:

```
<mx:Application
  Properties
  application="No default"
  controlBar="null"
  frameRate="24"
  layout="vertical|horizontal|absolute"
  pageTitle="No default"
  styleName="No default"
```

Figura 1 – Exemplo de documentação ASDoc.

## 5. Criando Comentários

Um comentário consiste no que se escreve entre os caracteres `/**` e `*/`. É uma boa prática colocar um espaço depois do asterisco de comentário para torná-lo mais legível. Tanto o tab antes do asterisco, o asterisco e os espaços posteriores são descartados do documento ASDoc gerado. Um exemplo simples de comentário seguindo as sugestões seria:

```
/**
 * Meu primeiro comentário
 *
 * @tag Texto da tag
 */
```

Para que os comentários ASDoc sejam reconhecidos pela ferramenta, devemos colocá-los antes da declaração de uma classe, interface, construtor, método, propriedade ou metadata que queremos documentar.

A ferramenta ASDoc irá ignorar comentários colocados no corpo de métodos e irá reconhecer apenas um comentário por declaração. Sabendo disso, um erro comum é colocar o comentário de uma classe antes de um import, ou seja, o comentário estará vinculado ao import e não à classe.

ASDoc que não funcionam:

```
public function meuMetodo():void
{
    /**
     * Este comentário nunca irá aparecer na documentação
     */
}
```

```
/**
 * Este comentário irá documentar o import e não a classe
 */
import mc.controls.*;

public class minhaClasse
{
}
```

## 6. Formatando os comentários

Comentários ASDoc são feitos de frases e tags como já vimos. Os comentários até o início das tags são usados para descrever o a classe, método, evento, efeito etc. fazem. Este comentário será colocado na seção de resumo de cada elemento da documentação. A sessão de tags começa com a declaração da primeira tag, que é feita através do sinal de @. A descrição inicial não pode continuar depois da seção de tag. Podemos ter várias linhas de tags. Algumas podem ser repetidas como @param e @see e outras não podem. Veja o apêndice para encontrar a URL da lista completa de tags.

### 6.1 Usando a tag @private

Por default, a ferramenta ASDoc gera documentação para todos os elementos public e protected. Para fazer com que a ferramenta ignore estes elementos, insira um comentário com a tag @private.

```
/**
 * Este método e seu comentário não aparecerão na documentação.
 * @private
 */
public function meuMetodo():void
{
    /**
     * Este comentário nunca irá aparecer na documentação
     */
}
```

Outro ponto importante que devemos saber é que a ferramenta gera *output* para todas as classes da lista de *input*. Desta forma, se quisermos dar uma pasta como parâmetro para a ferramenta e não quisermos que algumas classes apareçam na documentação, devemos colocar a tag `@private` antes da definição da classe

```
/**
 * Esta classe seu comentário não aparecerão na documentação
 * @private
 */
import mc.controls.*;

public class minhaClasse
{
}
```

É importante lembrarmos que mesmo que não desejamos que alguns elementos não apareçam na documentação, isto não quer dizer que não devem ser documentados.

**Documentar todos os elementos é sempre uma boa pratica.**

## 6.2 Usando HTML tags

Necessariamente, devemos escrever nossos comentários em *XHTML-compliant HTML*. Podemos usar diversos recursos de HTML para formatarmos nossos comentários e é muito interessante que os usemos para melhorar a leitura dos comentários. A lista das tags HTML suportados pode ser vista na URL encontrada no apêndice.

## 6.3 Caracteres Especiais

Caso exista a necessidade de se usar caracteres especiais como ">", "<", "@" etc, devemos usar o código HTML equivalente para que não ocorra erro no *parse*. Por exemplo, para colocar o caractere ">" usamos `&gt;`, para "<" usamos `&lt;`, para "@" usamos `&64` etc.

## 6.4 Escondendo texto em comentários

O ASDoc já vem com um *style sheet* para escondermos textos em comentários, o *style sheet* **hide**. O exemplo abaixo ilustra o seu uso:

```
/**
 * Comentando um método ou classe
 * Descrevendo suas funcionalidades
 *
 * <span classe="hide">Este texto não irá aparecer na documentação
 * final.</span>
 *
 * @eventType mx.events.FlexEvent.BUTTON_DOWN
 */
```

## 7. O processo de *Parse*

O processo de *parse* da ferramenta segue as seguintes regras:

- Se um comentário ASDoc vem antes de um elemento ActionScript, o ASDoc copia o elemento e o comentário para o arquivo final;
- Se um elemento ActioScript não é precedido por um comentário, a ferramenta copia o elemento para o arquivo final sem comentário;
- Se um comentário contem a tag @private o elemento ActionScript e o comentário são ignorados;
- O texto do comentário deve sempre vir antes de qualquer @ tag. Caso isso não ocorra, o comentário será interpretado como sendo um comentário da tag;
- Comentário AcrionScript com os símbolos // ou /\* são ignorados;
- Todo HTML deve seguir a convenção de XML, ou seja toda tag aberta deve ser fechada. Por exemplo <li> e </li>. Caso quisermos forçar uma quebra de linha devemos usar <br />.

## 8. Documentando elementos ActionScript

### 8.1 Documentando classes

Coloque seus comentários de classe bem antes da declaração da classe como mostra o exemplo:

```
/**
 * Documentação de classe ActionScript
 */
public class MinhaClasse
{
}
```

Caso você queira que a classe não apareça na documentação final, basta colocar a tag `@private`. **Lembre-se isso não significa que você não deve documentar a sua classe.**

### 8.2 Documentando propriedades

A ferramenta ASDoc automaticamente inclui propriedades `public` e `protected` no documento final. Podemos documentar propriedades que são definidas como variáveis ou como *gets* e *sets* implícitos.

### 8.3 Variáveis

Para documentarmos propriedades definidas como variáveis, basta colocarmos os comentários antes da declaração da propriedade como mostra o exemplo:

```
/**
 * Documentando minha propriedade
 * @default null
 */
public var minhaPropriedade:String;
```

Uma boa prática ao se documentar propriedades é o uso da tag `@default`, que define o valor inicial da nossa propriedade.

Quando definimos uma propriedade como `[Bindable]`, a ferramenta ASDoc automaticamente adiciona o texto abaixo no documento final:

*This property can be used as the source for data binding.*

### 8.3.1 Propriedades definidas com gets e sets implícitos

Em métodos `get` e `set` implícitos, insira um comentário antes do método `get` e marque o método `set` como `@private`. **A própria Adobe recomenda esta prática, pois geralmente o método `get` vem antes do `set`:**

```
/**
 * Indica se o componente está habilitado ou não
 */
public function get enabled():Boolean
{
}

/**
 * @private
 */
public function set enabled(b:Boolean):void
{
}
```

Algumas considerações do parse de métodos `get` e `set`:

- Se colocarmos comentário antes de um `set` ou `get` implícito o comentário será incluído no documento final;
- Se definirmos os métodos `set` e `get`, colocamos um único comentário antes do `get` e colocamos o método setter como `@ private`;
- Não precisamos definir o método `get` e `set` em nenhuma ordem específica e eles também podem estar em qualquer ordem no código. **É claro que é uma boa prática declararmos o `get` antes do `set` e os dois estarem declarados em**

**seqüência;**

- Se definirmos apenas o método *get*, a propriedade será declarada apenas como leitura (read-only);
- Se definirmos apenas o método *set*, a propriedade será declarada como apenas escrita (write-only);
- Se declararmos os métodos *get* e *set* e não quisermos que os mesmos apareçam na documentação, temos declarar ambos com o a tag `@private`;

## 8.4 Documentando Métodos

A ferramenta ASDoc automaticamente inclui métodos `public` e `protected` no documento final. Coloque seus comentários antes da declaração do método.

```
/**
 * Documentando <code>meuMetodo()</code> que existe apenas
 * para exemplificarmos a documentação
 *
 * <p>Aqui poderíamos escrever o segundo parágrafo da documentação
 * deste método</p>
 *
 * @param parametro1 Descrição do parametro2
 * @param parametro2 Descrição do parametro2
 *
 * @return String
 *
 */
public function meuMetodo(parametro1:String, parametro2:Number):String
{
}
```

Como no exemplo acima, se um método possui parâmetros, devemos descrevê-los usando a tag `@param`. A ordem que as tags `@param` aparecem deve ser a mesma ordem dos parâmetros no método. Também vemos que se um método retorna algum tipo de valor, temos que colocar a tag `@return` com a descrição do tipo de valor retornado.

## 8.5 Documentando effects, events e styles

Para documentar as *metadatas* [Effect], [Event] e [Style] coloque um comentário ASDoc antes das metatags como no exemplo abaixo:

```
/**
 * Definindo um estilo
 */
[Style "name"]
```

Para as metadatas [Effect] e [Event] incluímos o nome da classe associada com o evento e/ou efeito:

```
/**
 * Documentando um evento criado.
 *
 * @eventType br.com.dclick.events.DClickEvent.MY_EVENT
 */
[Event (name="myEvent",
type="br.com.dclick.events.DClickEvent.MY_EVENT")]
```

Agora, ao documentarmos a constante `br.com.dclick.events.DClickEvents.MY_EVENT` do exemplo acima, **é uma boa prática definirmos uma tabela com os valores bubbles, cancelable, target e currentTarget da classe Event**. Para que este tipo de documentação funcione, devemos colocar o tag @eventType para que o ASDoc possa encontrar o comentário da constante:

```
/**
 * Minha constante para definir um tipo de evento disparado
 *
 * <p>A propriedade tem os seguintes valores:</p>
 * <table class=innertable>
 * <tr><th>bubbles</th><th>>true</th></tr>
 * <tr><th>cancelable</th><th>>true</th></tr>
 * <tr><th>target</th><th> br.com.dclick.events.DClickEvent
 * </th></tr>
 * <tr><th>currentTarget</th><th> br.com.dclick.events.DClickEvent
 * </th></tr>
 * </table>
 *
 * @eventType myEvent
 */
public static const BUTTON_DOWN:String = "myEvent";
```

Se fizermos isso da forma correta o ASDoc irá criar um link para a classe de evento da classe que declara a tag [Event] e também irá fazer uma cópia da descrição da constante para a documentação desta mesma classe. Um bom exemplo do uso desse tipo de documentação veja as classes *mx.controls.Button* e *mx.events.FlexEvent*.

## 8.6 Documento arquivos MXML

Segundo a documentação fornecida pela Adobe, podemos usar o que foi discutido até o momento para arquivos .mxml no código ActionScript contido na tag <mx:Script>. Comentários feitos fora do bloco <mx:Script> não aparecerão na documentação. Entretanto vemos que esse comportamento não funciona hoje com a ferramenta. Independente disso, **sugerimos que todo código ActionScript seja documentado dentro do bloco <mx:Script>.**

## 9. Usando a Tag @see

A tag @see permite que você faça referência entre elementos da mesma classe, classes diferentes, do mesmo pacote ou de outros pacotes. Podemos também fazer referência a URLs fora da documentação. Esta tag tem a seguinte sintaxe:

```
@see link [texto exibido]
```

A palavra link deve ser substituída pelo o destino da classe e o texto exibido é a string que irá aparecer na documentação (opcional). A localização do link depende do prefixo do mesmo:

- Prefixo **#** - ASDoc procura na mesma classe pelo link;
- Prefixo **ClassName** - ASDoc procura por uma classe no mesmo pacote pelo link;
- Prefixo **PackageName** - ASDoc procura em um pacote diferente pelo link.

Exemplo	Resultado
@see http://www.dclick.com.br	Site externo
@see meuHtml.html	HTML local
@see Array	Classe pai
@see AccessibilityProperties	Classe no mesmo pacote
@see flash.display.TextField	Classe em um pacote diferente
@see Array#length	Propriedade da classe pai
@see flash.ui.ContextMenu#customItems	Propriedade em uma classe em outro pacote
@see #updateProperties()	Método na mesma classe
@see Array#pop()	Método da classe pai
@see flash.ui.ContextMenu#clone()	Método em uma classe diferente

## 10. Executando a ferramenta ASDoc

Existem diversas maneiras de configurar o script para gerar a documentação final. As principais são usando os comandos `doc-classes` para especificar as classes que queremos gerar a documentação, `doc-namespaces` que consiste em dizer para a ferramenta onde se encontra o arquivo de manifest (útil para bibliotecas de componentes) e `doc-sources` que consiste em especificar um diretório para documentação.

A maneira mais fácil e a melhor de se usar é usando `doc-sources`, pois a ferramenta irá documentar todas as classes do diretório que apontarmos recursivamente:

```
asdoc -source-path . -doc-sources .
```

### 10.1 Excluindo classes

Como vimos anteriormente a melhor maneira de se gerar a documentação é através de `doc-sources`. Entretanto, é comum não quereremos documentar algumas classes de alguns pacotes. Para tanto, usamos a opção `exclude-classes`. O exemplo abaixo ilustra o uso:

```
asdoc -source-path . -doc-sources . -exclude-classes  
br.com.dclick.Classe1 br.com.dclick.Classe2
```

Devemos lembrar que classes contendo a tag `@private` não são documentadas, o que torna o uso do parâmetro `exclude-classes` pouco usado. Outro ponto importante é que classes encontradas em um SWC também não são documentadas.

### 10.2 Pastas

Como convenção criamos a pasta ***asdoc*** na raiz de nosso projeto. Dentro desta pasta criamos mais duas pastas: ***docs*** e ***examples***. "Dizemos" para a ferramenta gerar a documentação na pasta ***asdoc/docs*** através da opção `-output-path`. O porquê da pasta ***examples*** explicaremos mais para frente.

Também, é comum e recomendável em nossos projetos termos a pasta **src** (source) com nossos arquivos fonte. Geralmente, colocamos o arquivo de **build** da documentação (recomendamos o nome **asDocBuild.xml**) na raiz do nosso projeto. Dessa forma, temos que dizer para a ferramenta ASDoc qual a pasta dos arquivos fonte. Para tanto temos que definir o parâmetro `-source-path` e indicar a pasta **src**.

Abaixo temos um exemplo de organização de projeto. Veja as pastas dentro da pasta **asdoc** e onde está arquivo **asDocBuild.xml**. Neste exemplo temos dois arquivos a serem documentados *DownloadUploadManager.as* e *ASDoc.mxml*.

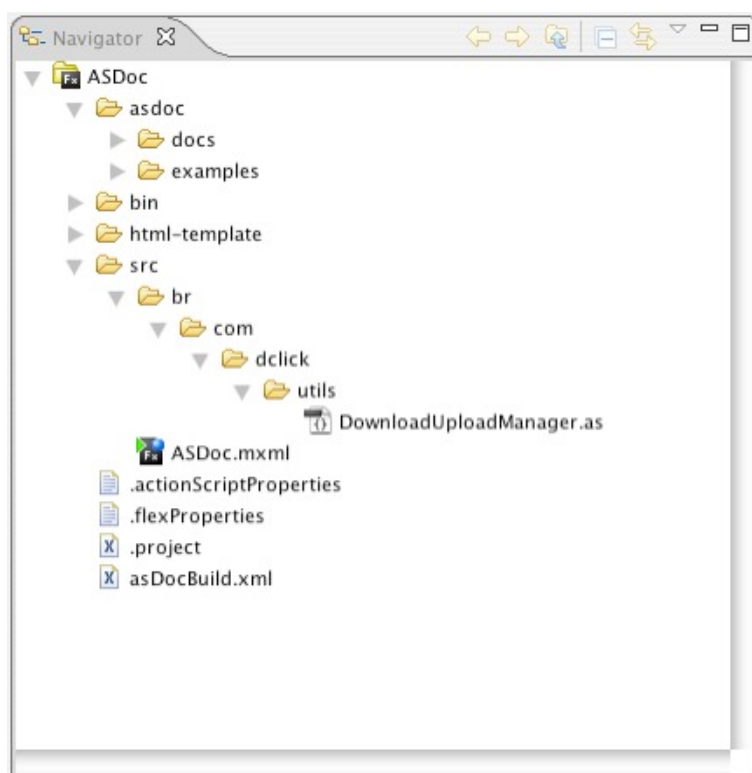


Figura 2 – Exemplo de organização de projeto Flex.

### 10.3 Exemplos de Código

A ferramenta ASDoc possibilita colocarmos exemplos no meio da documentação. Os exemplos podem ser colocados de duas maneiras: diretamente na documentação ou apontando para um arquivo externo de exemplo.

O primeiro caso é bem simples de implementar e é recomendado quando o exemplo é curto. Para tanto usamos o a tag `@example` no meio do comentário. Devemos cercar o

exemplo com a tag `<listing version="3.0"></listing>` como mostra o exemplo abaixo:

```
/**
 * Start o download do arquivo chamando um servlet via post.
 *
 * @param parameters Objeto que será assado na URL do servlet
 * @param suggestedFileName Arquivo sugerido na caixa de diálogo de download
 * do browser.
 * @param fileName Nome do arquivo do download. O parametro passado no
 * servlet é ARQUIVO.
 * @param servlet URL completa do servlet que será chamado.
 * @param controller Nome do controlador. Caso não passado será o
 * controlador default do servlet.
 * @param service Nome da atividade. Caso não passado será a atividade
 * default do servlet.
 * @example Exemplo de construção de um objeto no parameter
 * <listing version="3.0" >
 * var obj:Object = new Object();
 * var.PARAM1 = "param1";
 * var.PARAM2 = 10;
 * //Os objetos serão passados como ?PARAM1=param1&PARAM2=10
 * </listing>
 */
public function download(parameters:Object,
                        suggestedFileName:String, fileName:String, servlet:String,
                        controller:String = null, service:String = null):void
{
}
```

No arquivo final, a ferramenta irá criar uma área de exemplo dentro do método `download` e colocará o exemplo dentro de uma caixa cinza.

Como vimos, o exemplo dado deste modo deve ser simples. Imagine agora que queremos dar uma classe inteira ou um exemplo muito grande como nos exemplos da API de Flex da Adobe? Para tanto devemos usar a tag `@includeExample`. Através dela iremos apontar para um arquivo externo (.as ou .mxml) e a ferramenta ASDoc irá incluir o arquivo no final da documentação. A sintaxe de uso é:

```
@includeExample arquivo [-noswf]
```

Perceba que existe um segundo parâmetro chamado `-noswf`. Temos a opção de colocar um `swf` funcionando de exemplo ao invés de código em classes `.as` ou arquivos `.mxml`.

Devemos também colocar outro parâmetro em nosso arquivo de build para dizer onde o ASDoc irá procurar pelos arquivos de exemplo. Esse parâmetro é o `-examples-path`. Recomendamos que a pasta de exemplo seja a pasta **`asdoc/examples`** como da figura 2.

A ferramenta ASDoc irá procurar o arquivo de exemplo a partir da pasta `-examples-path` especificado. Imaginem em um projeto real onde documentamos nossas classes e começamos a ter vários arquivos externos de exemplo. Se começamos a jogos todos estes arquivos em apenas uma pasta, começamos a ter certa desorganização e podemos nos perder com o tempo. Desta forma, **recomendamos que o package do exemplo seja o mesmo package da classe**. Ao fazermos isso, não precisamos especificar todo o caminho do nosso exemplo, pois a ferramenta irá procurar pelo arquivo de exemplo no mesmo package do arquivo com a tag `@includeExample`. Assim, o comentário apontando para o exemplo externo da classe `br.com.dclick.util.DownloadUploadManager` ficaria:

```
/**
 * Classe responsável por fazer download e upload de arquivos
 * de forma genérica através de um servlet via POST.
 *
 * @see flash.events.EventDispatcher
 * @see br.com.dclick.utils.FileDownload
 * @see br.com.dclick.utils.FileUpload
 * @playerversion
 *
 * @includeExample FileDownload.mxml -noswf
 * @includeExample FileUpload.mxml -noswf
 */
public class DownloadUploadManager extends EventDispatcher
{
}
```

A imagem abaixo mostra como ficaria as pastas com os exemplos:

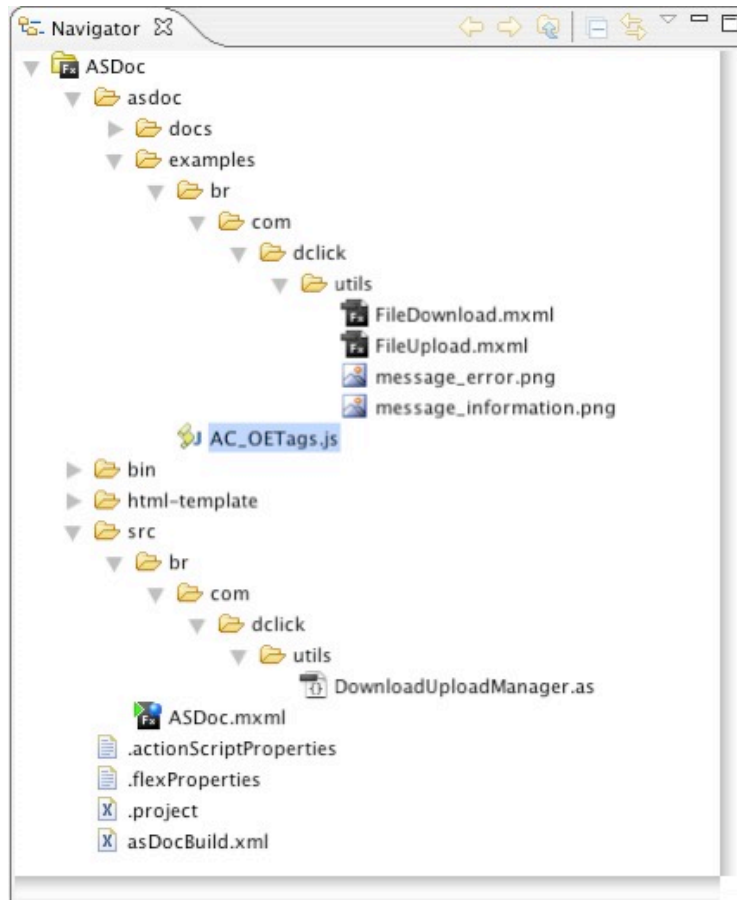


Figura 3 – Exemplo de organização de projeto Flex com exemplos de documentação.

Perceba que o package dos exemplos é o mesmo da classe e que no pacote de exemplo existem duas imagens. Isso porque a ferramenta ASDoc valida os arquivos de exemplo e os mesmos precisam das duas imagens.

## 11. Arquivo de exemplo final

Abaixo temos a classe `br.com.dclick.util.DownloadUploadManager` totalmente documentada para termos como exemplo. Foram removidos os códigos dos métodos, pois os mesmos não são relevantes para o objetivo deste documento:

```
package br.com.dclick.utils
{
    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.events.ProgressEvent;

    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    import mx.controls.Alert;
    import mx.controls.ProgressBar;
    import mx.core.Application;

    import flash.events.EventDispatcher;
    import flash.events.SecurityErrorEvent;
    import flash.events.ProgressEvent

    /**
     * Disparado quando o usuário seleciona um arquivo para upload.
     * @eventType flash.events.Event.SELECT
     */
    [Event(name="select", type="flash.events.Event")]

    /**
     * Disparado ao longo do download do arquivo.
     * @eventType flash.events.ProgressEvent.PROGRESS
     */
    [Event(name="progress", type="flash.events.ProgressEvent")]
}
```

```
/**
 * Disparado ao completar o download/upload.
 * @eventType flash.events.Event.COMPLETE
 */
[Event(name="complete", type="flash.events.Event")]

/**
 * Disparado se algum erro ocorre na operação de download/upload.
 * @eventType flash.events.IOErrorEvent.IO_ERROR
 */
[Event(name="ioError", type="flash.events.IOErrorEvent")]

/**
 * Disparado se ocorre algum erro de segurança.
 * @eventType flash.events.SecurityErrorEvent.SECURITY_ERROR
 */
[Event(name="securityError",
type="flash.events.SecurityErrorEvent")]

[Bindable]
/**
 * Classe responsável por fazer download e upload de arquivos
 * de forma genérica através de um servlet via POST.
 *
 * @see flash.events.EventDispatcher
 * @see br.com.dclick.utils.FileDownload
 * @see br.com.dclick.utils.FileUpload
 * @playerversion
 *
 * @includeExample FileDownload.mxml -noswf
 * @includeExample FileUpload.mxml -noswf
 */
public class DownloadUploadManager extends EventDispatcher
{
    //*****CONSTANTES*****
    private const CONTROLLER_URL_PARAM:String = "CONTROLADOR=";
    private const SERVICE_URL_PARAM:String = "SERVICO=";
```

```
    private var _fr:FileReference;
    private var _uploadEnabled:Boolean = false
    private var _progressBarMessage:String;
    private var pb:ProgressBar;
    private var request:URLRequest;
/**
 * Cria uma nova instância do FileReference e o listeners
 * disparados
 * pelo o FielReference.
 */
public function DownloadUploadManager()
{

}

/**
 * Cancela Operacao o download ou upload.
 */
public function cancel():void
{

}

/**
 * Start o download do arquivo chamando um servlet via post.
 *
 * @param parameters Objeto que será assado na URL do servlet
 * @example Exemplo de construção de um objeto no parameter
 * <listing version="3.0" >
 * var obj:Object = new Object();
 * var.PARAM1 = "param1";
 * var.PARAM2 = 10;
 * //Os objetos serão passados como ?PARAM1=param1&PARAM2=10
 * </listing>
 * @param suggestedFileName Arquivo sugerido na caixa de
 * diálogo de download do browser.
 * @param fileName Nome do arquivo do download. O parametro
```

```
* passado no servlet é ARQUIVO.
* @param servlet URL completa do servlet que será chamado.
* @param controller Nome do controlador. Caso não passado será
* o controlador default do servlet.
* @param service Nome da atividade. Caso não passado será a
* atividade default do servlet.
*/
public function download(parameters:Object,
                        suggestedFileName:String,
                        fileName:String,
                        servlet:String,
                        controller:String = null,
                        service:String = null):void
{
}

/**
 * Gera o request que será usado pelos métodos download e
 * upload.
 * @param servlet
 * @param controller
 * @param service
 * @param parameters
 * @param fileName
 * @return
 */
private function generateRequest(servlet:String,
                                controller:String = null,
                                service:String = null,
                                parameters:Object = null,
                                fileName:String = null
                                ):URLRequest
{
}
```

```
/**
 * Abre a caixa de seleção para o usuário escolher o arquivo de
 * upload.
 * @param filters Array de Filters para restringir as extensões
 * que o usuário pode escolher.
 * @default null
 */
public function browse(filters:Array = null):void
{
}

/**
 * Evento disparado ao selecionar um arquivo
 * @param event
 */
private function selectHandler(event:Event):void
{
}

/**
 * Start o upload de um arquivo chamando um servlet via post.
 * @param servlet URL completa do servlet que será chamado.
 * @param controller Nome do controlador. Caso não passado será o
 * controlador default do servlet.
 * @param service Nome da atividade. Caso não passado será a
 * atividade default do servlet.
 * @param parameters Objeto que será assado na URL do servlet.
 */

public function upload(servlet:String,
                      controller:String = null,
                      service:String = null,
                      parameters:Object = null):void
{
}
```

```
/**
 * Método disparado ao abrir a caixa de escolha de arquivo.
 * Coloca o ProgressBar visível.
 * @param event
 */
private function openHandler(event:Event):void
{
}

/**
 * Método chamado ao longo do download do arquivo.
 * @param event
 */
private function progressHandler(event:ProgressEvent):void
{
}

/**
 * Método disparado ao completar o download. Coloca o ProgressBar
 * invisível e dispara um Alert de Download Concluído.
 * @param event
 */
private function completeHandler(event:Event):void
{
}

/**
 * Método disparado se algum erro ocorre nas operações de
 * download
 * ou upload.
 * @param event
 */
private function ioErrorHandler(event:IOErrorEvent):void
{
}
```

```
/**
 * Método disparado se ocorre algum erro de segurança.
 * @param event
 *
 */

private function securityErrorHandler(
    event:SecurityErrorEvent):void
{
}

//GETs e SETs
/**
 * Progress bar que indica o andamento do upload ou download.
 * @see mx.controls.ProgressBar
 */
public function get progressBar():ProgressBar
{
    return this.pb;
}
/**
 * @private
 */
public function set progressBar(pb:ProgressBar):void
{
    this.pb = pb;
    this.pb.mode = "manual";
}
/**
 * Indica se o processo e upload/download está disponível.
 * Se o componente estiver fazendo upload/download está
 * propriedade é falsa.
 *
 */
public function get uploadEnabled():Boolean
{
    return this._uploadEnabled;
}
```

```
    }

    /**
     * @private
     */

    public function set uploadEnabled(b:Boolean):void
    {
        this._uploadEnabled = b;
    }

    /**
     * Retorna a referência do FileReference.
     * @see flash.net.FileReference
     */
    public function get fileReference():FileReference
    {
        return this._fr;
    }

    /**
     * Mensagem que será mostrada no ProgressBar
     */
    public function get progressBarMessage():String
    {
        return this._progressBarMessage;
    }

    /**
     * @private
     */
    public function set progressBarMessage(s:String):void
    {
        this._progressBarMessage = s;
    }
}
}
```

Abaixo temos o arquivo asDocbuild.xml de exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="DClick ASDoc" default="_generate" basedir=".">
  <target name="_generate" depends="delete,create"/>

  <!-- caminho da ferramenta ASDoc -->
  <property name="asdocExec" location="/dev/sdks/flex/bin/asdoc" />
  <!-- pasta de source do projeto -->
  <property name="sourcePath" location="src" />
  <!-- pasta para ser documentada recursivamente -->
  <property name="docSources" value="src"/>
  <!-- pasta que será gerada a documentação -->
  <property name="outputPath" location="${basedir}/asdoc/docs" />
  <!-- pasta que contém os exemplos externos -->
  <property name="examplesPath" location="${basedir}/asdoc/examples" />

  <!-- deleção da pasta asdoc -->
  <target name="delete">
    <delete dir="${outputPath}" includeEmptyDirs="true"/>
    <mkdir dir="${outputPath}" />
  </target>

  <!-- execução do asdoc com os parâmetros definidos -->
  <target name="create">
    <exec executable="${asdocExec}" failonerror="true">
      <arg line="-doc-sources ${docSources}" />
      <arg line="-source-path ${sourcePath}" />
      <arg line="-output ${outputPath}" />
      <arg line="-examples-path ${examplesPath}" />
    </exec>
  </target>
</project>
```

Existem ainda outros parâmetros úteis que podemos usar como `-main-title`, `-window-title`, `-footer` para configurar o nome na aplicação que aparece no top do HTML, o nome que aparece na barra do browser e o nome que aparece no rodapé das páginas HTML respectivamente. A lista completa de parâmetro pode ser encontrada em [http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs\\_Book\\_Parts&file=asdoc\\_127\\_9.html](http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=asdoc_127_9.html)

## 12. Cabeçalho de arquivos

Recomendamos colocar em todos os arquivos (.as e .mxml) algo que identifique alguns parâmetros do arquivo com **autor**, **versão**, **última alteração** etc. Este espaço poderá conter qualquer comentário pertinente para a classe. Lembre-se que isso não sairá na documentação, **é apenas uma boa prática que estamos recomendando**. Temos como exemplo:

### 12.1 AS

```
////////////////////////////////////  
// 2007 DClick Desenvolvimento de Software LTDA. Todos os direitos  
// reservados.  
//  
// @autor Rafael M. Martinelli  
// @version 1.0  
// @lastModified 21/06/2007  
//  
////////////////////////////////////  
package br.com.dclick.utils  
{  
}
```

## 12.2 MXML

```
<?xml version="1.0" encoding="utf-8"?>
<!--
////////////////////////////////////
//
// 2007 DClick Desenvolvimento de Software LTDA. Todos os direitos
reservados.
//
// @autor Rafael M. Martinelli
// @version 1.0
// @lastModified 21/06/2007
//
////////////////////////////////////
-->
<mx:Application xmlns:mx=http://www.adobe.com/2006/mxml layout="absolute">
```

## 13. Conclusão

A ferramenta ASDoc é bem completa e contém todas as funcionalidades para fazermos uma boa documentação. O que nos resta agora é documentar tudo que fazemos e nos conscientizar da importância de uma ótima documentação.

## 14. Apêndices

Lista completa de tags do ASDoc:

[http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs\\_Book\\_Parts&file=asdoc\\_127\\_6.html](http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=asdoc_127_6.html)

Resumo das tags HTML mais usadas em comentários:

[http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs\\_Book\\_Parts&file=asdoc\\_127\\_8.html](http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=asdoc_127_8.html)