Recherche dans DEJ avec Google Rechercher

g+1 0

Copyright (C) 1999-2014 Jean-Michel DOUDOUX.

83. JavaDoc

Niveau : Elémentaire

Javadoc est un outil fourni par Sun avec le JDK pour permettre la génération d'une documentation technique à partir du code source.

Cet outil génère une documentation au format HTML à partir du code source Java et des commentaires particuliers qu'il contient. Un exemple concret de l'utilisation de cet outil est la documentation du JDK qui est générée grâce à Javadoc.

Cette documentation contient :

- une description détaillée pour chaque classe et ses membres public et protected par défaut (sauf les classes internes anonymes)
- un ensemble de listes (liste des classes, hiérarchie des classes, liste des éléments deprecated et un index général)
- des références croisées et une navigation entre ces différents éléments.

L'intérêt de ce système est de conserver dans le même fichier le code source et les éléments de la documentation qui lui sont associés. Il propose donc une auto-documentation des fichiers sources de façon standard.

Ce chapitre contient plusieurs sections :

83.1. La mise en oeuvre

Javadoc s'appuie sur le code source et sur un type de commentaires particuliers pour obtenir des données supplémentaires des éléments qui composent le code source.

L'outil Javadoc utilise plusieurs types de fichiers sources pour générer la documentation :

- Les fichiers sources .java
- Les fichiers de commentaires d'ensemble
- Les fichiers de commentaires des packages
- D'autres fichiers tels que des images, des fichiers HTML, ...

En fonction des paramètres fournis à l'outil, ce dernier recherche les fichiers source .java concernés. Les sources de ces fichiers sont scannées pour déterminer leurs membres, extraire les informations utiles et établir un ensemble de références croisées.

Le résultat de cette recherche peut être enrichi avec des commentaires dédiés insérés dans le code avant chaque élément qu'ils enrichissent. Ces commentaires doivent immédiatement précéder l'entité qu'ils concernent (classe, interface, méthode, constructeur ou champ). Seul le commentaire qui précède l'entité est traité lors de la génération de la documentation.

Ces commentaires suivent des règles précises. Le format de ces commentaires commence par /** et se termine par */. Il peut contenir un texte libre et des balises particulières.

Le commentaire peut être sur une ou plus généralement sur plusieurs lignes. Les caractères d'espacement (espace et tabulation) qui précèdent le

premier caractère * de chaque ligne du commentaire ainsi que le caractère lui-même sont ignorés lors de la génération. Ceci permet d'utiliser le caractère * pour aligner le contenu du commentaire.

Exemple :

```
1. /** Description */
```

Le format général de ces commentaires est :

Exemple :

```
1. /**
2.  * Description
3.  *
4.  * @tag1
5.  * @tag2
6.  */
```

Le commentaire doit commencer par une description de l'élément qui peut utiliser plusieurs lignes. La première phrase de cette description est utilisée par javadoc comme résumé. Cette première phrase se termine par un caractère '.' suivi d'un séparateur (espace ou tabulation ou retour chariot) ou à la rencontre du premier tag Javadoc.

Le texte du commentaire doit être au format HTML : les tags HTML peuvent donc être utilisés pour enrichir le formatage de la documentation. Il est donc aussi nécessaire d'utiliser les entités d'échappement pour certains caractères contenus dans le texte tels que < ou >. Il ne faut surtout pas utiliser les tags de titres <Hn> et le tag du séparateur horizontal <HR> car ils sont utilisés par Javadoc pour structurer le document.

Exemple :

```
1. /**
2.  * Description de la classe avec des  <b>mots en gras</b>
3.  */
```

L'utilisation de balises de formatage HTML est particulièrement intéressante pour formater une description un peu longue en faisant usage notamment du tag <p> pour définir des paragraphes ou du tag <code> pour encadrer un extrait de code.

A partir du JDK 1.4, si la ligne ne commence pas par un caractère *, alors les espaces ne sont plus supprimés (ceci permet par exemple de conserver l'indentation d'un morceau de code contenu dans un tag HTML <PRE>).

Le commentaire peut ensuite contenir des tags Javadoc particuliers qui commencent obligatoirement par le caractère @ et doivent être en début de ligne. Ces tags doivent être regroupés ensemble. Un texte qui suit cet ensemble de tags est ignoré.

Les tags prédéfinis par Javadoc permettent de fournir des informations plus précises sur des composants particuliers de l'élément (auteur, paramètres, valeur de retour, ...). Ces tags sont définis pour un ou plusieurs types d'éléments.

Les tags sont traités de façon particulière par Javadoc. Il existe deux types de tags :

- Block tag : ils sont de la forme @tag
- Inline tag : ils sont de la forme{@tag}

Attention un caractère @ en début de ligne est interprété comme un tag. Si un tel caractère doit apparaître en début de ligne dans la description, il faut utiliser la séquence d'échappement HTML @

Le texte associé à un block tag suit le tag et se termine à la rencontre du tag suivant ou de la fin du commentaire. Ce texte peut donc s'étendre sur plusieurs lignes.
Les tags inline peuvent être utilisés n'importe où dans le commentaire de documentation.

83.2. Les tags définis par javadoc

L'outil Javadoc traite de façon particulière les tags dédiés insérés dans le commentaire de documentation. Javadoc définit plusieurs tags qui permettent de préciser certains composants de l'élément décrit de façon standardisée. Ces tags commencent tous par le caractère arobase @.

Il existe deux types de tags :

- Block tag : ils doivent être regroupés après la description. Ils sont de la forme @tag
- Inline tag : ils peuvent être utilisés n'importe où dans le commentaire. Ils sont de la forme {@tag}

Les block tags doivent obligatoirement débuter en début de ligne (après d'éventuels blancs et un caractère *)

Attention : les tags sont sensibles à la casse.

Pour pouvoir être interprétés, les tags standards doivent obligatoirement commencer en début de ligne.

Tag	Rôle	version du JDK
@author	permet de préciser le ou les auteurs de l'élément	1.0
{@code}		1.5
@deprecated	permet de préciser qu'un élément est déprécié	1.1
{@docRoot}	représente le chemin relatif du répertoire principal de génération de la documentation	1.3
@exception	permet de préciser une exception qui peut être levée par l'élément	1.0
{@inheritDoc}		1.4
{@link}	permet d'insérer un lien vers un élément de la documentation dans n'importe quel texte	1.2
{@linkplain}		1.4
{@literal}		1.5
@param	permet de documenter un paramètre de l'élément	1.0
@return	permet de fournir une description de la valeur de retour d'une méthode qui en possède une : inutile donc de l'utiliser sur une méthode qui retourne void.	1.0
@see	permet de préciser un élément en relation avec l'élément documenté	1.0
@serial		1.2
@serialData		1.2
@serialField		1.2
@since	permet de préciser depuis quelle version l'élément a été ajouté	1.1
@throws	identique à @exception	1.2
@version	permet de préciser le numéro de version de l'élément	1.0
{@value}		1.4

Ces tags ne peuvent être utilisés que pour commenter certaines entités.

Entité	Tags utilisables
Toutes	@see, @since, @deprecated, {@link}, {@linkplain}, {@docroot}
Overview (fichier overview.html)	@see, @since, @author, @version, {@link}, {@linkplain}, {@docRoot}
Package (fichier package.html)	@see, @since, @serial, @author, @version, {@link}, {@linkplain}, {@docRoot}
Classes et Interfaces	@see, @since, @deprecated, @serial, @author, @version, {@link}, {@linkplain}, {@docRoot}
Constructeurs et méthodes	@see, @since, @deprecated, @param, @return, @throws, @exception, @serialData, {@link}, {@linkplain}, {@inheritDoc}, {@docRoot}
Champs	@see, @since, @deprecated, @serial, @serialField, {@link}, {@linkplain}, {@docRoot}, {@value}

Chacun des tags sera détaillé dans les sections suivantes.

Par convention, il est préférable de regrouper les tags identiques ensemble.

83.2.1. Le tag @author

Le tag @author permet de préciser le ou les auteurs d'une entité.

La syntaxe de ce tag est la suivante :

@author texte

Le texte qui suit la balise est libre. Le doclet standard crée une section "Author" qui contient le texte du tag.

Pour préciser plusieurs auteurs, il est possible d'utiliser un seul ou plusieurs tag @author dans un même commentaire. Dans le premier cas, le contenu du texte est repris intégralement dans la section. Dans le second cas, la section contient le texte de chaque tag séparé par une virgule et un espace.

Exemple :

@author Pierre G.

@author Denis T., Sophie D.

Ce tag n'est utilisable que dans les commentaires d'ensemble, d'une classe ou d'une interface.

A partir du JDK 1.4, il est possible au travers du paramètre -tag de préciser que le tag @author peut être utilisé sur d'autres membres

Exemple :

-tag author:a:"Author:"

83.2.2. Le tag @deprecated

Le tag @deprecated permet de préciser qu'une entité ne devrait plus être utilisée même si elle fonctionne toujours : il permet donc de donner des précisions sur un élément déprécié (deprecated).

La syntaxe de ce tag est la suivante :

@deprecated texte

Il est recommandé de préciser depuis quelle version l'élément est déprécié et de fournir dans le texte libre une description de la solution de remplacement, si elle existe, ainsi qu'un lien vers une entité de substitution.

Le doclet standard créé une section "Deprecated" avec l'explication dans la documentation.

Remarque : Ce tag est particulier car il est le seul reconnu par le compilateur : celui-ci prend note de cet attribut lors de la compilation pour permettre d'en informer les utilisateurs. Lors de la compilation, l'utilisation d'entités marquées avec le tag @deprecated générera un avertissement (warning) de la part du compilateur.

Exemple Java 1.1 :

@deprecated Remplacé par setMessage

@see #setMessage

Exemple Java 1.2 :

@deprecated Remplaçé par {@link #setMessage}

83.2.3. Le tag @exception et @throws

Ces tags permettent de documenter une exception levée par la méthode ou le constructeur décrit par le commentaire.

Syntaxe :

@exception nom_exception description

Les tags @exception et @throws sont similaires.

Ils sont suivis du nom de l'exception puis d'une courte description des raisons de la levée de cette dernière. Il faut utiliser autant de tag @exception ou @throws qu'il y a d'exceptions. Ce tag doit être utilisé uniquement pour un élément de type méthode.

Il ne faut pas mettre de séparateur particulier comme un caractère '-' entre le nom et la description puisque l'outil en ajoute un automatiquement. Il est cependant possible d'aligner les descriptions de plusieurs paramètres en utilisant des espaces afin de faciliter la lecture.

Exemple :

@exception java.io.FileNotFoundException le fichier n'existe pas

Le doclet standard crée une section "Throws" qui regroupe les exceptions : l'outil recherche le nom pleinement qualifié de chaque exception si c'est simplement leur nom qui est précisé dans le tag.

Exemple extrait de la documentation de l'API du JDK :

```
public String(char[] value)
```

Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument. The contents of the character array are copied; subsequent modification of the character array does not affect the newly created string.

Parameters:

`value` - the initial value of the string.

Throws:

`NullPointerException` - if `value` is `null`.

83.2.4. Le tag @param

Le tag @param permet de documenter un paramètre d'une méthode ou d'un constructeur. Ce tag doit être utilisé uniquement pour un élément de type constructeur ou méthode.

La syntaxe de ce tag est la suivante :

@param nom_paramètre description du paramètre

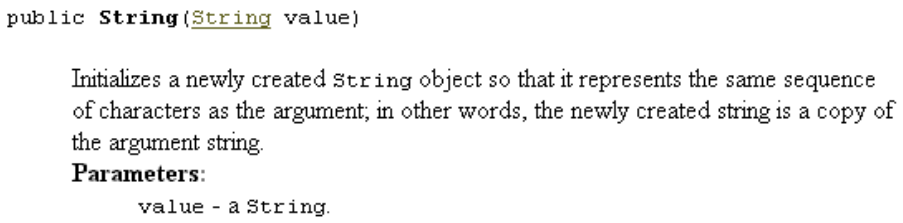
Ce tag est suivi du nom du paramètre (ne pas utiliser le type) puis d'une courte description de ce dernier. A partir de Java 5, il est possible d'utiliser le type du paramètre entre les caractères < et > pour une classe ou une méthode.

Il ne faut pas mettre de séparateur particulier comme un caractère '-' entre le nom et la description puisque l'outil en ajoute un automatiquement. Il est cependant possible d'aligner les descriptions de plusieurs paramètres en utilisant des espaces afin de faciliter la lecture.

Il faut utiliser autant de tag @param que de paramètres dans la signature de l'entité concernée. La description peut être contenue sur plusieurs lignes.

Le doclet standard crée une section "Parameters" qui regroupe les tags @param du commentaire. Il génère pour chaque tag une ligne dans cette section avec son nom et sa description dans la documentation.

Exemple extrait de la documentation de l'API du JDK :


public String(String value)

Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

Parameters:

value - a String.

Par convention les paramètres doivent être décrits dans leur ordre dans la signature de la méthode décrite

Exemple :

@param nom nom de la personne

@param message chaîne de caractères à traiter. Si cette valeur est <code>null</code> alors une exception est levée

Exemple 2 : /** * @param <E> Type des elements stockés dans la collection */
public interface List<E> extends Collection<E> { }

83.2.5. Le tag @return

Le tag @return permet de fournir une description de la valeur de retour d'une méthode qui en possède une.

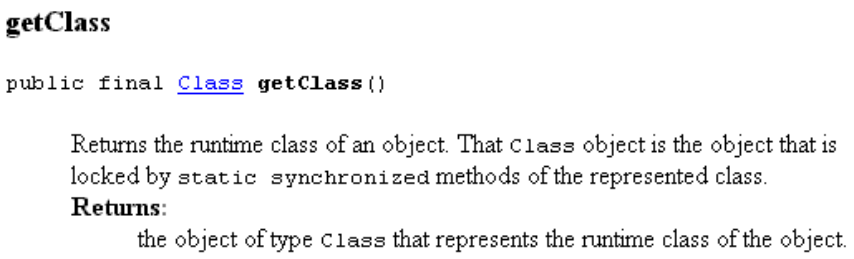
La syntaxe de ce tag est la suivante :

@return description_de_la_valeur_de retour_de_la_méthode

Il ne peut y avoir qu'un seul tag @return par commentaire : il doit être utilisé uniquement pour un élément de type méthode qui renvoie une valeur.

Avec le doclet standard, ce tag crée une section "Returns" qui contient le texte du tag. La description peut tenir sur plusieurs lignes.

Exemple extrait de la documentation de l'API du JDK :


getClass

public final Class **getClass**()

Returns the runtime class of an object. That Class object is the object that is locked by static synchronized methods of the represented class.

Returns:

the object of type Class that represents the runtime class of the object.

Il ne faut pas utiliser ce tag pour des méthodes ne possédant pas de valeur de retour (void).

Exemple:

@return le nombre d'occurrences contenues dans la collection

@return <code>true</code> si les traitements sont correctement exécutés sinon <code>false</code>

83.2.6. Le tag @see

Le tag @see permet de définir un renvoi vers une autre entité incluse dans une documentation de type Javadoc ou vers une url.

La syntaxe de ce tag est la suivante :

@see référence à une entité suivie d'un libellé optionnel ou lien ou texte entre double quote

@see package
@see package.Class
@see class
@see #champ
@see class#champ
@see #method(Type,Type,...)
@see class#method(Type,Type,...)
@see package.class#method(Type,Type,...)
@see ...
@see " ... "

Le tag génère un lien vers une entité ayant un lien avec celle documentée.

Il peut y avoir plusieurs tags @see dans un même commentaire.

L'entité vers laquelle se fait le renvoi peut être un package, une classe, une méthode ou un lien vers une page de la documentation. Le nom de la classe doit être de préférence pleinement qualifié.

Le caractère # permet de séparer une classe d'un de ses membres (champ, constructeur ou méthode). Attention : il ne faut surtout pas utiliser le caractère "." comme séparateur entre une classe ou une interface et le membre précisé.

Pour indiquer une version surchargée particulière d'une méthode ou d'un constructeur, il suffit de préciser la liste des types d'arguments de la version concernée.

Il est possible de fournir un libellé optionnel à la suite de l'entité. Ce libellé sera utilisé comme libellé du lien généré : ceci est pratique pour forcer un libellé à la place de celui généré automatiquement (par défaut le nom de l'entité).

Si le tag est suivi d'un texte entre double cote, le texte est simplement repris avec les cotes sans lien.

Si le tag est suivi d'un tag HTML <a>, le lien proposé par ce tag est repris intégralement.

Le doclet standard crée une section "See Also" qui regroupe les tags @see du commentaire en les séparant par une virgule et un espace.

Exemple extrait de la documentation de l'API du JDK :

```
public static String valueOf(Object obj)
```

Returns the string representation of the `Object` argument.

Parameters:

`obj` - an `Object`.

Returns:

if the argument is `null`, then a string equal to `"null"`; otherwise, the value of `obj.toString()` is returned.

See Also:

`Object.toString()`

Remarque : pour insérer un lien n'importe où dans le commente, il faut utiliser le tag {@link}

Exemple :

@see String
@see java.lang.String
@see String#equals
@see java.lang.Object#wait(int)
@see MaClasse nouvelle classe
@see Test
@see "Le dossier de spécification détaillée"

Ce tag permet de définir des liens vers d'autres éléments de l'API.

83.2.7. Le tag @since

Le tag @since permet de préciser un numéro de version de la classe ou de l'interface à partir de laquelle l'élément décrit est disponible. Ce tag peut être utilisé avec tous les éléments.

La syntaxe de ce tag est la suivante :

@since texte

Le texte qui représente le numéro de version est libre. Le doclet standard crée une section "Since" qui contient le texte du tag.

Exemple extrait de la documentation de l'API du JDK :

```
public byte[] getBytes()
```

Convert this `String` into bytes according to the platform's default character encoding, storing the result into a new byte array.

Returns:

the resultant byte array.

Since:

JDK1.1

Par convention, pour limiter le nombre de sections Since dans la documentation, lorsqu'une nouvelle classe ou interface est ajoutée, il est préférable de mettre un tag @since sur le commentaire de la classe et de ne pas le reporter sur chacun de ses membres. Le tag @since est utilisé sur un membre uniquement lors de l'ajout du membre.

Dans la documentation de l'API Java, ce tag précise depuis qu'elle version du JDK l'entité décrite est utilisable.

Exemple :

@since 2.0

83.2.8. Le tag @version

Le tag @version permet de préciser un numéro de version. Ce tag doit être utilisé uniquement pour un élément de type classe ou interface.

La syntaxe de ce tag est la suivante :

@version texte

Le texte qui suit la balise est libre : il devrait correspondre à la version courante de l'entité documentée. Le doclet standard crée une section "Version" qui contient le texte du tag.

Il ne devrait y avoir qu'un seul tag @version dans un commentaire.

Par défaut, le doclet standard ne prend pas en compte ce tag : il est nécessaire de demander sa prise en compte avec l'option -version de la commande javadoc.

Exemple :

@version 1.00

83.2.9. Le tag {@link}

Ce tag permet de créer un lien vers un autre élément de la documentation.

La syntaxe de ce tag est la suivante :

{**@link** package.class#membre texte }

Le mode de fonctionnement de ce tag est similaire au tag @see : la différence est que le tag @see crée avec le doclet standard un lien dans la section "See also" alors que le tag {@link} crée un lien à n'importe quel endroit de la documentation.

Si une accolade fermante doit être utilisée dans le texte du tag il faut utiliser la séquence d'échappement }.

Exemple :

Utiliser la {@link #maMethode(int) nouvelle méthode}

83.2.10. Le tag {@value}

Ce tag permet d'afficher la valeur d'un champ.

La syntaxe de ce tag est la suivante :

{@value}

{@value package.classe#champ_static}

Lorsque le tag {@value} est utilisé sans argument avec un champ static, le tag est remplacé par la valeur du champ.

Lorsque le tag {@value} est utilisé avec comme argument une référence à un champ static, le tag est remplacé par la valeur du champ précisé. La référence utilisée avec ce tag suit la même forme que celle du tag @see

Exemple :

{@value}

{@value #MA_CONSTANTE}

83.2.11. Le tag {@literal}

Ce tag permet d'afficher un texte qui ne sera pas interprété comme de l'HTML.

La syntaxe de ce tag est la suivante :

{**@literal** texte}

Le contenu du texte est repris intégralement sans interprétation. Notamment les caractères < et > ne sont pas interprétés comme des tags HTML.

Pour afficher du code, il est préférable d'utiliser le tag {@code}

Exemple :

{@literal 0<b>10}

83.2.12. Le tag {@linkplain}

Ce tag permet de créer un lien vers un autre élément de la documentation dans une police normale.

Ce tag est similaire au tag @link. La différence réside dans la police d'affichage.

83.2.13. Le tag {@inheritDoc}

Ce tag permet de demander explicitement la recopie de la documentation de l'entité de la classe mère la plus proche correspondante.

La syntaxe de ce tag est la suivante:

{@inheritDoc}

Ce tag permet d'éviter le copier/coller de la documentation d'une entité.

Il peut être utilisé :

- dans la description d'une entité : dans ce cas tout le commentaire de l'entité de la classe mère est repris
- dans un tag @return, @tag, @throws : dans ce cas tout le texte du tag de l'entité de la classe mère est repris

83.2.14. Le tag {@docRoot}

Ce tag représente le chemin relatif à la documentation générée.

La syntaxe de ce tag est la suivante :

{@docRoot}

Ce tag est pratique pour permettre l'inclusion de fichiers dans la documentation.

Exemple :

Historique

83.2.15. Le tag {@code}

Ce tag permet d'afficher un texte dans des tags <code> ... </code> qui ne sera pas interprété comme de l'HTML.

La syntaxe de ce tag est la suivante :

{@code texte}

Le contenu du texte est repris intégralement sans interprétation. Notamment les caractères < et > ne sont pas interprétés comme des tags HTML.

Le tag {@code texte} est équivalent à <code>{@literal texte}</code>

Exemple :

{@code 0<b>10}

83.3. Un exemple

Exemple :

```
01. /**
02.  * Résumé du rôle de la méthode.
03.  * Commentaires détaillés sur le role de la methode
04.  * @param val la valeur a traiter
05.  * @return la valeur calculée
06.  * @since 1.0
07.  * @deprecated Utiliser la nouvelle methode XXX
08.  */
09.   public int maMethode(int val) {
10.     return 0;
11.   }
```

Résultat :

---

maMethode

```
public int maMethode(int val)
```

Deprecated. *Utiliser la nouvelle methode xyz*

Résumé du rôle de la méthode. Commentaires détaillés sur le role de la methode

Parameters:
`val` - la valeur a traiter
Returns:
la valeur calculée
Since:
1.0

83.4. Les fichiers pour enrichir la documentation des packages

Javadoc permet de fournir un moyen de documenter les packages car ceux-ci ne disposent pas de code source particulier : il faut définir des fichiers dont le nom est particulier.

Ces fichiers doivent être placés dans le répertoire désigné par le package.

Le fichier package.html contient une description du package au format HTML. En plus, il est possible d'utiliser les tags @deprecated, @link, @see et @since.

Le fichier overview.html permet de fournir un résumé de plusieurs packages au format html. Ce fichier doit être placé dans le répertoire qui inclut les packages décrits.

83.5. La documentation générée

Pour générer la documentation, il faut invoquer l'outil javadoc. Javadoc recrée à chaque utilisation la totalité de la documentation.

Pour formater la documentation, javadoc utilise une doclet. Une doclet permet de préciser le format de la documentation générée. Par défaut, Javadoc propose une doclet qui génère une documentation au format HTML. Il est possible de définir sa propre doclet pour changer le contenu ou le format de la documentation (pour par exemple, générer du RTF ou du XML).

La génération de la documentation avec le doclet par défaut crée de nombreux fichiers et des répertoires pour structurer la documentation au format HTML, avec et sans frame.

La documentation de l'API Java fournie par Sun/Oracle est réalisée grâce à Javadoc. La page principale est composée de trois frames :

Par défaut, la documentation générée contient les éléments suivants :

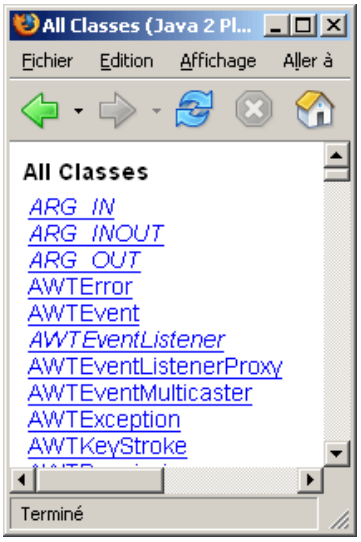
- un fichier html par classe ou interface qui contient le détail de chaque élément de la classe ou de l'interface
- un fichier html par package qui contient un résumé du contenu du package
- un fichier overview-summary.html
- un fichier overview-tree.html
- un fichier deprecated-list.html
- un fichier serialized-form.html
- un fichier overview-frame.html
- un fichier all-classe.html
- un fichier package-summary.html pour chaque package
- un fichier package-frame.html pour chaque package
- un fichier package-tree.html pour chaque package

Tous ces fichiers peuvent être regroupés en trois catégories :

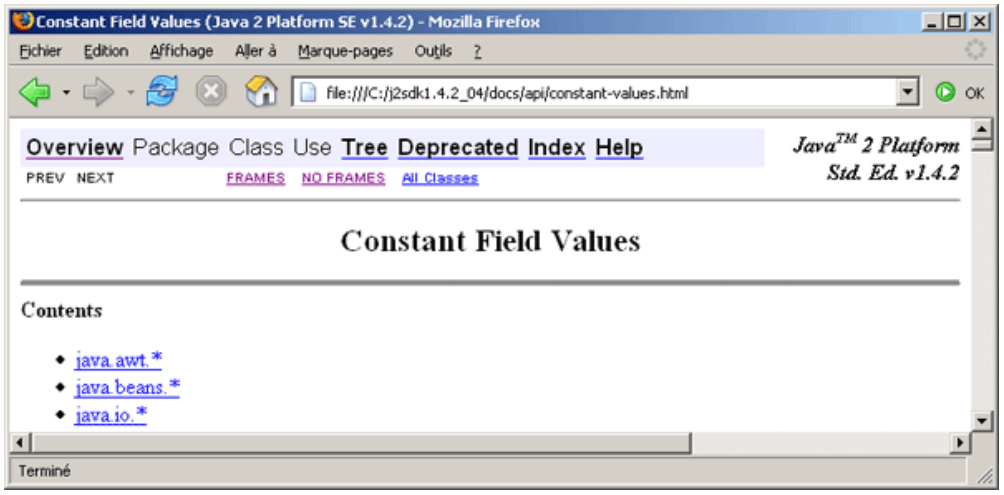
- Les pages de base : les pages des classes et interfaces et les résumés
- Les pages des références croisées : les pages index, les pages de hiérarchie, les pages d'utilisation et les pages deprecated-list.html, constant-values.html et serialized-form.html
- Les fichiers de structure : la page principale, les frames, la feuille de style

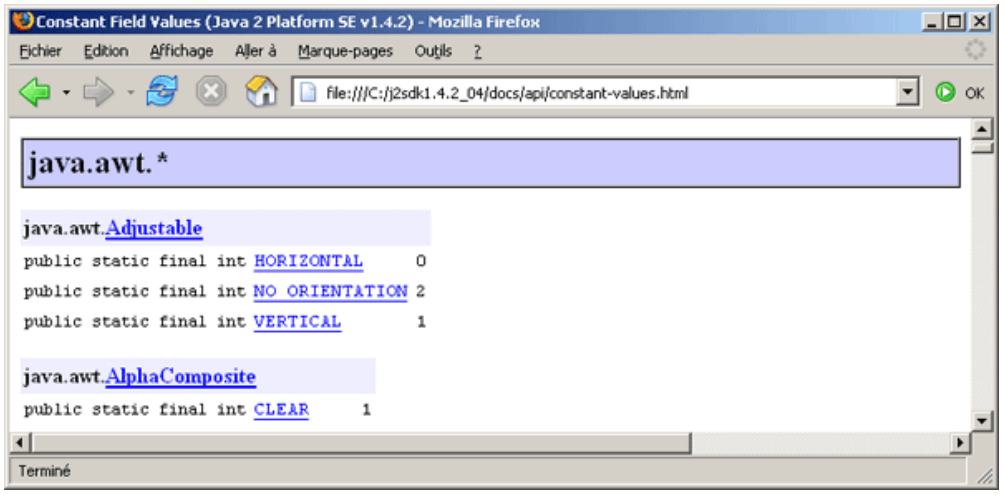
Il y a plusieurs fichiers générés à la racine de l'application :

Le fichier allclasses-frame.html affiche toutes les classes, interfaces et exceptions de la documentation avec un lien pour afficher le détail. Cette page est affichée en bas à gauche dans le fichier index.html

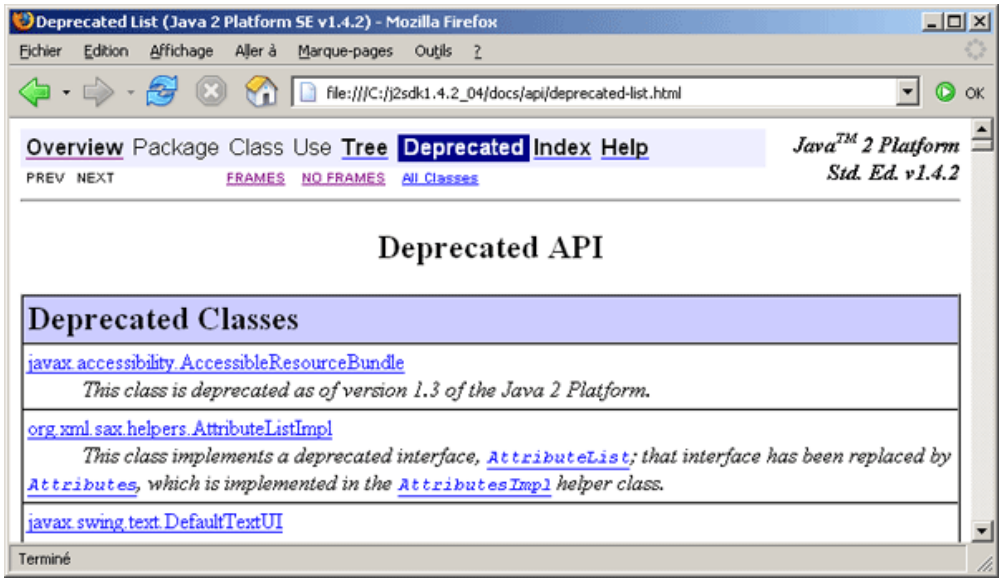


Le fichier constant-values.html affiche la liste de toutes les constantes avec leurs valeurs.

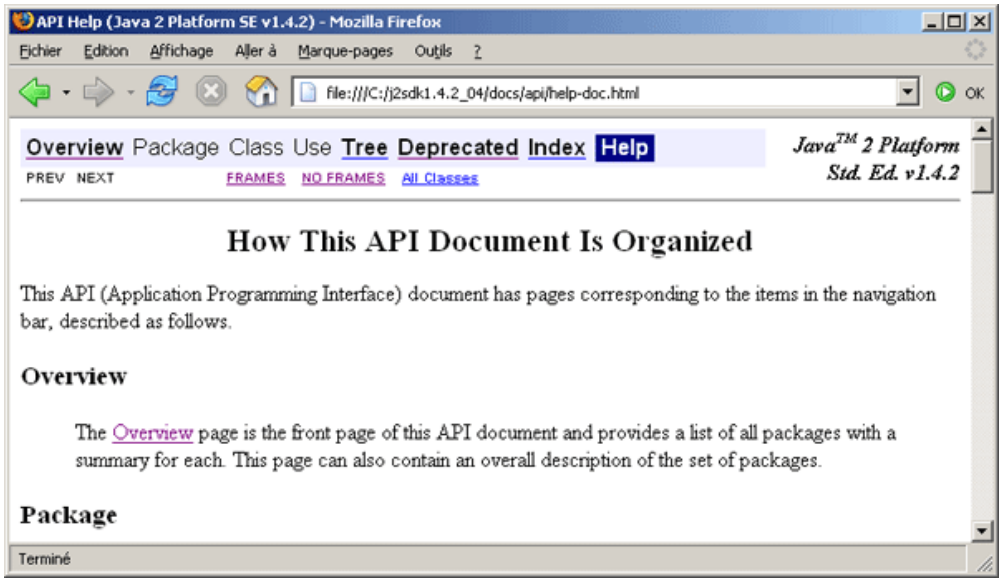




Le fichier deprecated-list.html affiche la liste de tous les membres déclarés deprecated. Le lien Deprecated de la barre de navigation permet d'afficher le contenu de cette page.



Le fichier help-doc.html affiche l'aide en ligne de la documentation. Le lien Help de la barre de navigation permet d'afficher le contenu de cette page.

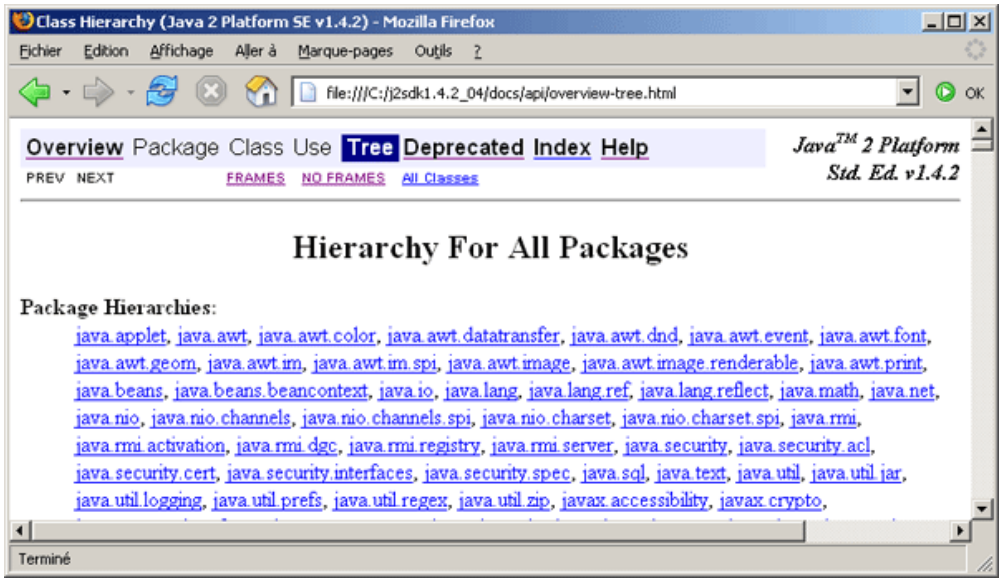


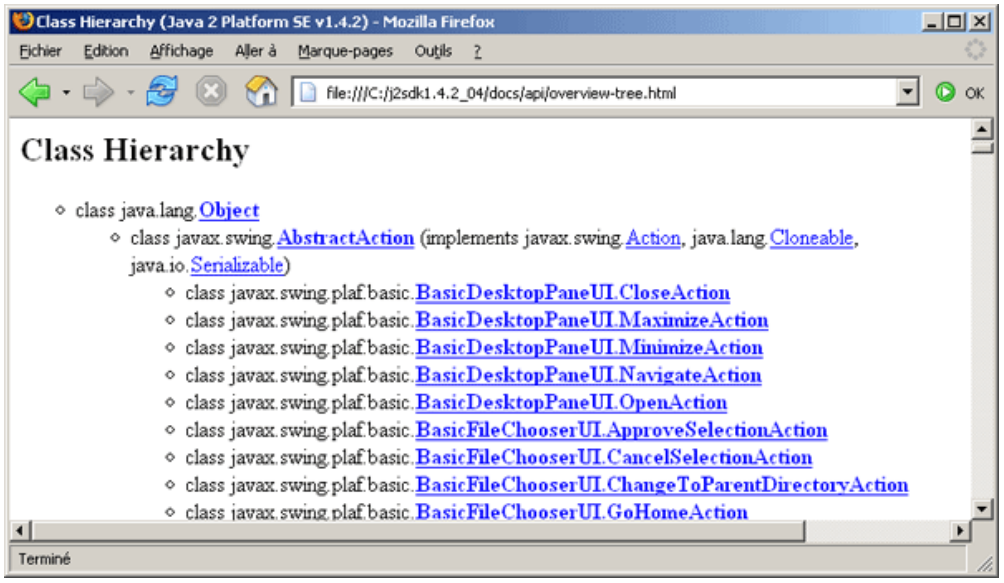
Le fichier index.html est la page principale de la documentation composée de 3 frames

Le fichier overview-frame.html affiche la liste des packages avec un lien pour afficher la liste des membres du package. Cette page est affichée en haut à gauche dans le fichier index.html

Le fichier overview-summary.html affiche un résumé des packages de la documentation. Cette page est affichée par défaut dans la partie centrale de la page index.html

Le fichier overview-tree.html affiche la hiérarchie des classes et interfaces. Le lien Tree de la barre de navigation permet d'afficher le contenu de cette page.

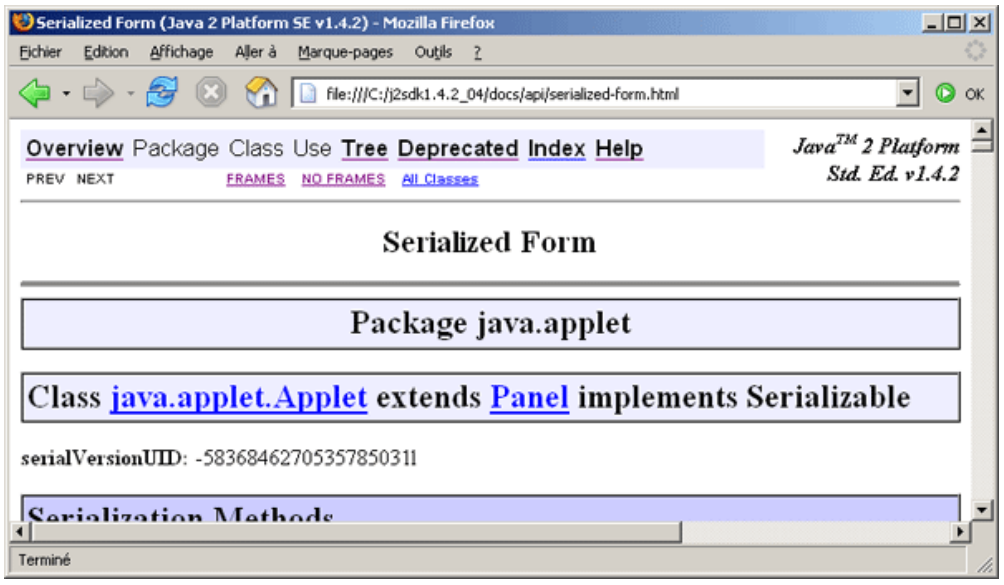




Le fichier package-list est un fichier texte contenant la liste de tous les packages (non affiché dans la documentation).

Le fichier packages.html permet de choisir entre les versions avec et sans frame de la documentation

Le fichier serialized-form.html affiche la liste des classes qui sont sérialisables



Le fichier stylesheet.css est la feuille de style utilisée pour afficher la documentation.

Le fichier allclasses-noframe.html affiche la page allclasses-frame.html sans frame.

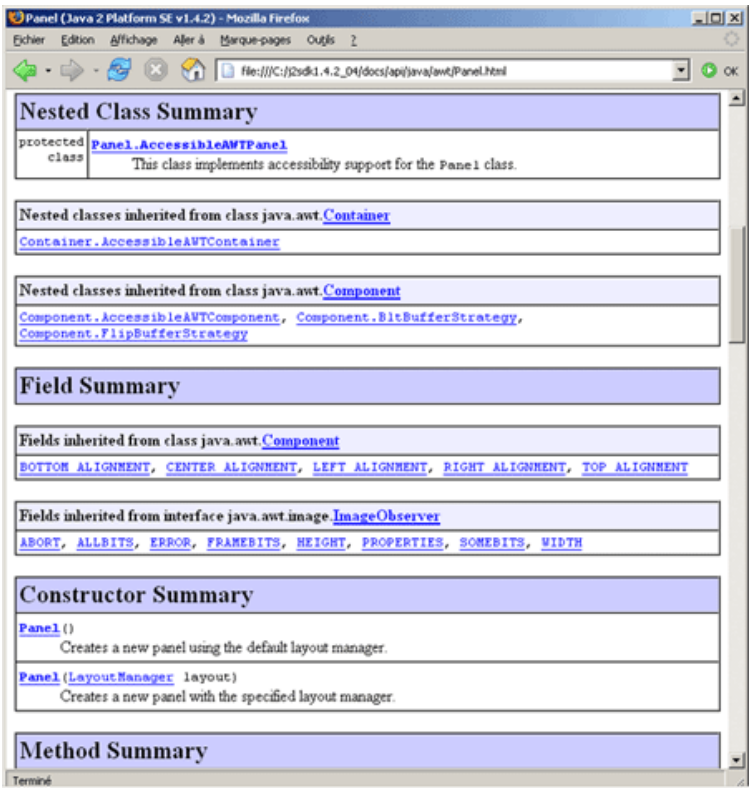
Il y a un répertoire par package. Ce répertoire contient plusieurs fichiers :

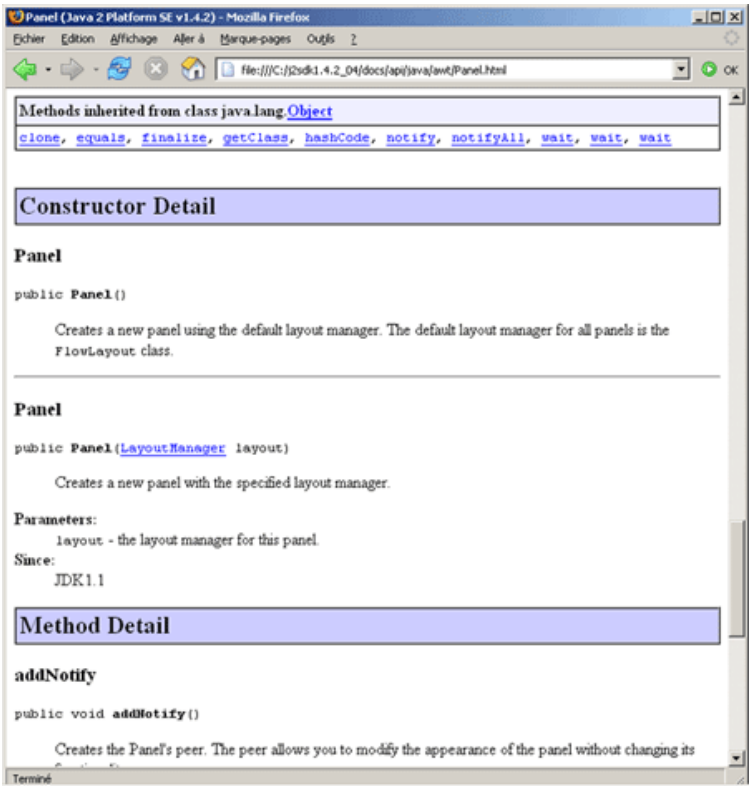
- classe.html : un fichier html contenant la définition de chaque classe du package
- package-frame.html : contient la liste de toutes les interfaces, classes et exceptions du package
- package-summary.html : contient un résumé de toutes les interfaces, classes et exceptions du package
- package-tree.html : contient l'arborescence de toutes les interfaces et classes du package

Cette structure est reprise pour les sous-packages.

La page détaillant une classe possède la structure suivante :


Panel (Java 2 Platform SE v1.4.2) - Mozilla Firefox
Fichier Edition Affichage Aller à Marque-pages Outils ?
file:///C:/j2sdk1.4.2_04/docs/api/java/awt/Panel.html
Overview Package Class Use Tree Deprecated Index Help
Java™ 2 Platform Std. Ed. v1.4.2
PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
java.awt
Class Panel
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Panel
All Implemented Interfaces:
Accessible, ImageObserver, MenuContainer, Serializable
Direct Known Subclasses:
Applet
public class Panel
extends Container
implements Accessible
Panel is the simplest container class. A panel provides space in which an application can attach any other component, including other panels.
The default layout manager for a panel is the FlowLayout layout manager.
Since:
JDK1.0
See Also:
FlowLayout, Serialized Form
Terminé


Panel (Java 2 Platform SE v1.4.2) - Mozilla Firefox
Fichier Edition Affichage Aller à Marque-pages Outils ?
file:///C:/j2sdk1.4.2_04/docs/api/java/awt/Panel.html
Nested Class Summary
protected class Panel.AccessibleAWTPanel
This class implements accessibility support for the Panel class.
Nested classes inherited from class java.awt.Container
Container.AccessibleAWTContainer
Nested classes inherited from class java.awt.Component
Component.AccessibleAWTComponent, Component.BltBufferStrategy, Component.FlipBufferStrategy
Field Summary
Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT
Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH
Constructor Summary
Panel()
Creates a new panel using the default layout manager.
Panel(LayoutManager layout)
Creates a new panel with the specified layout manager.
Method Summary
Terminé



Si l'option -linksource est utilisée, les fichiers sources sont stockés dans l'arborescence du sous-répertoire src-html de la documentation.





Copyright (C) 1999-2014 Jean-Michel DOUDOUX.