

# Gestion de l'encodage dans une application J2EE

par P. Durville ([Accueil](#)) ([Blog](#))

Date de publication : 18/02/08

Dernière mise à jour : 18/02/08

Suite aux nombreux problèmes que pose la gestion de l'encodage au sein d'une application web J2EE, au foisonnement de messages et de pseudo-règles échangées entre développeurs et au manque de synthèse sur le sujet, j'ai eu l'idée de récapituler ici un ensemble de bonnes pratiques à mettre en oeuvre pour éviter d'avoir à faire face à de véritables casse-têtes.

- I - Introduction
- II - Les pages (HTML, JSP, XSL, ...)
  - II-A - Les fichiers
  - II-B - Les pages JSP et JSPX
  - II-C - Le rendu (X)HTML
  - II-D - Les feuilles XSL
- III - Du côté du code Java
  - III-A - Utilisation d'un filtre dédié
  - III-B - Une servlet générique
  - III-C - Gestion des entrées/sorties fichiers
  - III-D - Gestion des entrées/sorties avec une base de données
- IV - Du côté des serveurs d'application J2EE
  - IV-A - Spécifier l'encodage de la JVM
  - IV-B - Gestion spécifique à chaque serveur
- V - Bonus
  - V-A - Bon à connaître...

## I - Introduction

Commençons par un petit rappel : qu'est-ce qu'un encodage et à quoi sert-il ? Les ordinateurs travaillant uniquement avec des suites d'octets (un octet = 8 bits, donc un octet peut se représenter comme un nombre entier compris entre 0 et 255), toute chaîne de caractères va devoir être codée sous forme d'octets. Il faut donc choisir un procédé de transformation qui, à chaque caractère, associe de manière unique un ensemble d'octets (de 1 jusqu'à 4 parfois) afin de pouvoir stocker puis relire cette chaîne de caractères. C'est ce procédé que l'on nomme *encodage*. **UTF-8** est l'un de ces encodages. C'est celui qui est utilisé pour coder le texte que vous êtes entrain de lire. C'est, en fait, l'encodage le plus pratique pour échanger des textes constitués de caractères Unicode (= appartenant à l'ensemble de caractères standard décrit par le Consortium Unicode. Ce consortium a pour objectif de répertorier tous les caractères existants dans les différentes langues parlées et d'associer, à chacun de ces caractères, un numéro que l'on note en hexadécimal. Ainsi, par exemple, la lettre "a" minuscule aura pour valeur Unicode U+0061). Pourquoi UTF-8 est-il l'encodage le plus pratique ? Car il est compatible avec un autre encodage, ancien mais très répandu : l'encodage ASCII. (Pour les plus chevronnés, cela signifie que les caractères dont les numéros Unicode sont compris entre 32 et 126 sont représentés par la même suite d'octets en UTF-8 et en ASCII). Pour plus de détails sur les différents **encodages existants** et sur **Unicode**, consulter l'article très détaillé de **Wikipedia à propos d'Unicode** et/ou le **tutoriel complet (en anglais)** hébergé par l'université de technologie de Tampere.

Suite aux nombreux problèmes que pose la gestion de l'encodage au sein d'une application web J2EE, au foisonnement de messages et de pseudo-règles échangées entre développeurs et au manque de synthèse sur le sujet, j'ai eu l'idée de récapituler ici un ensemble de bonnes pratiques à mettre en oeuvre pour éviter d'avoir à faire face à de véritables casse-têtes.

La gestion de l'encodage d'une application J2EE nécessite d'ajuster plusieurs (trop nombreux ?) paramètres. Il ne faut pas en oublier un seul (et c'est là la seule difficulté) sous peine d'être confronté au problème récurrent de l'affichage d'un hiéroglyphe à la place de l'accent attendu par exemple ! L'objectif de ce document est donc de lister, en dix points, ces différents "paramètres" à prendre en compte.

## II - Les pages (HTML, JSP, XSL, ...)

### II-A - Les fichiers

💡 » La première contrainte à vérifier est que TOUS les fichiers (JSP, JSPX, XSL, XML, ...) de l'application soient dans l'encodage voulu.

La plupart des IDE peuvent se paramétrer pour que leurs éditeurs soient dans l'encodage désiré. Vérifier donc ce paramètre avant toute chose. Si vous éditez certains de vos fichiers de façon externe (à l'IDE en question), n'oubliez pas de vérifier également l'encodage de l'éditeur externe. A noter que certains éditeurs ne gèrent pas tous les encodages (voire n'en gèrent qu'un seul ! :- ) auquel cas, changez d'éditeur !

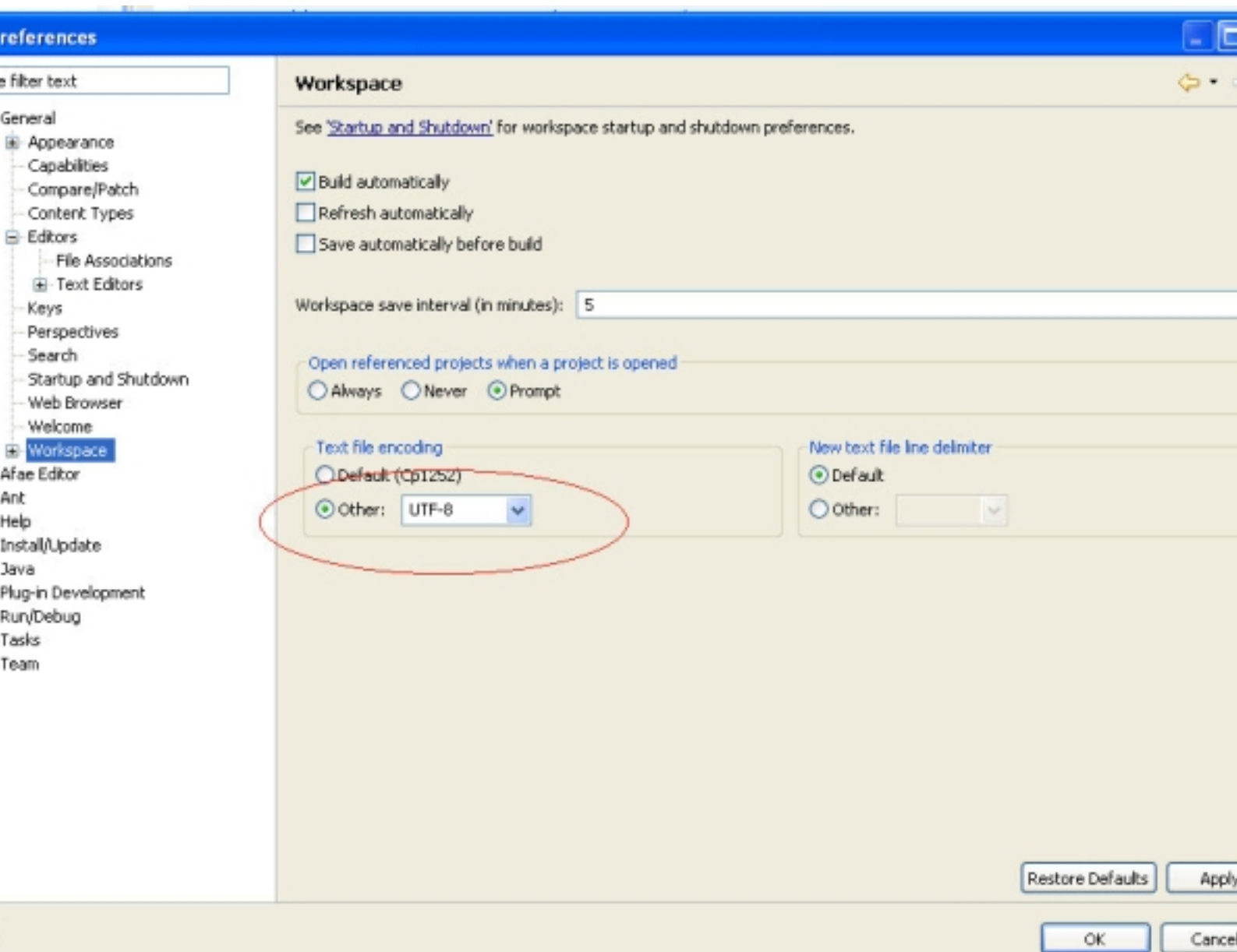


Fig. 1 : Configuration de l'encodage par défaut sous Eclipse (Window puis Préférences...)

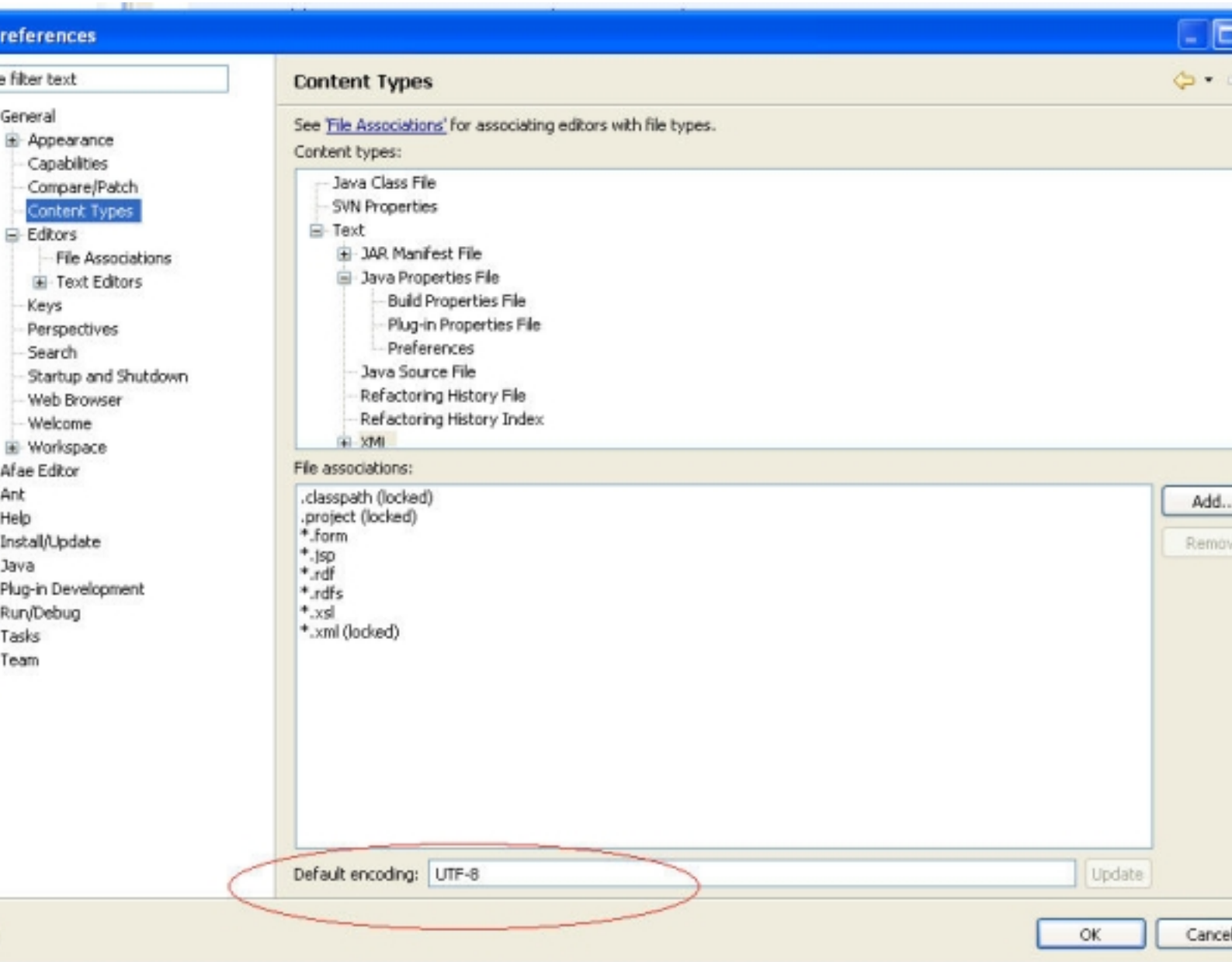




Fig. 2 : Configuration de l'encodage des différents type de fichiers sous Eclipse (Window puis Préférences...)

 Dans les exemples de ce document, on a choisi l'encodage UTF-8, mais toutes les règles valent pour n'importe quel encodage !

## II-B - Les pages JSP et JSPX

 » Déclarer explicitement l'encodage des pages JSP (et JSPX).

Cela se fait dans chaque page, dans l'en-tête de la page elle-même :

```
<jsp:directive.page contentType="text/html; charset=UTF-8" />
```


équivalent à, dans l'ancienne syntaxe (JSP1.2),

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

ou, de façon centralisée, dans le fichier de déploiement de l'application *web.xml* :

```
<jsp-config>
  <jsp-property-group>
    <description>Config. de l'encodage des pages JSP</description>
    <url-pattern>*.jsp</url-pattern>
    <page-encoding>UTF-8</page-encoding>
  </jsp-property-group>
  ...
</jsp-config>
```


## II-C - Le rendu (X)HTML

 » *Prévenir le navigateur client de l'encodage qu'il doit utiliser pour afficher la page (X)HTML.*

Cette directive lui est donnée via la balise "meta" suivante, insérée dans l'en-tête de la page (X)HTML (existante ou générée) renvoyée par le serveur :

```
<head>
  <meta equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Mon tr&#232;s joli titre</title>
  ...
</head>
```

A noter que cette balise "meta" doit être placée avant la balise "titre", surtout si le titre contient des accents par exemple !

 » *Eventuellement, prévenir le navigateur client de l'encodage qu'il doit utiliser pour charger une feuille de style externe.*

Il est possible de préciser l'encodage d'une feuille de style (CSS) externe (c'est-à-dire dans un fichier à elle toute seule) à l'aide de la directive suivante insérée en tout début de fichier (aucun caractère ne doit précéder cette directive) :


```
@charset "UTF-8";
```

Mais il est à noter que les navigateurs clients ont pour règle de respecter les priorités suivantes pour déterminer l'encodage d'une feuille de style externe :

- 1 le content-type (vu ci-dessus) de la page (X)HTML appelante,
- 2 la directive @charset de la feuille de style elle-même.

Par conséquent, si vos fichiers (X)HTML contiennent un "content-type", il n'est pas nécessaire de spécifier la directive "@charset" dans vos feuilles de style. Mais encore faut-il que ce soit le même encodage qui soit utilisé !

## II-D - Les feuilles XSL

 » *Déclarer explicitement l'encodage du xml/html généré.*

Cette directive est donnée via la balise suivante, insérée dans l'en-tête de la feuille xsl :

```
<xsl:output method="xml" omit-xml-declaration="yes"
            encoding="UTF-8" indent="yes" />
```

## III - Du côté du code Java

### III-A - Utilisation d'un filtre dédié

» *Utiliser un filtre dédié à l'encodage qui forcera votre serveur d'applications web à lire les paramètres des requêtes dans l'encodage voulu et à renvoyer ces réponses également dans ce même encodage.*

*Ceci est nécessaire avec le serveur Tomcat, par exemple qui, sous Windows (France), lit les paramètres en ISO8895-1 par défaut si on ne lui force pas la main au moment de la lecture des paramètres.*

Le code Java du filtre est le suivant :

```
package monappli.filters;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class EncodingFilter implements Filter {

    public static final String ENCODING = "encoding"; //key for encoding.
    private String encoding;

    public void init(FilterConfig filterConfig) throws ServletException {
        this.encoding = filterConfig.getInitParameter(ENCODING);
    }

    public void doFilter(ServletRequest req, ServletResponse resp,
        FilterChain filterChain)
        throws IOException, ServletException {
        req.setCharacterEncoding(encoding);
        resp.setContentType("text/html;charset="+encoding);
        filterChain.doFilter(req, resp);
    }


    public void destroy() {}
}
```

Il faut ensuite déclarer ce filtre dans le fichier de déploiement de l'application *web.xml* :

```
<filter>
  <filter-name>encodingfilter</filter-name>
  <filter-class>monappli.filters.EncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>
```


et s'assurer qu'il mappe TOUTES les pages :

```
<filter-mapping>
  <filter-name>encodingfilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

 **NOTE IMPORTANTE** : ce mapping doit être le **PREMIER**, de tous les mappings, déclaré dans le fichier web.xml. Pourquoi ? En fait, les filtres vont s'appliquer dans l'ordre dans lequel les mappings sont déclarés dans le fichier de déploiement. Or, notre filtre d'encodage doit s'exécuter avant que quoique ce soit ne soit lu dans la requête ou écrit dans la réponse sans quoi il n'aurait aucun effet.

### III-B - Une servlet générique

De même, il est nécessaire d'effectuer ce même traitement dans le cas des servlets.

 » Pour cela, il faut que toutes les servlets de l'application héritent d'une servlet générique qui gère l'encodage.

Cette servlet générique peut s'inspirer du code suivant :

```
package monappli.servlets;

import java.io.IOException;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public abstract class EncodingServlet extends HttpServlet {

    public static final String ENCODING = "encoding"; //key for encoding.
    private String encoding;

    public void init(ServletConfig servletConfig)
        throws ServletException {
        this.encoding = servletConfig.getInitParameter(ENCODING);
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        req.setCharacterEncoding(encoding);
        resp.setContentType("text/html;charset="+encoding);
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws IOException, ServletException {
        request.setCharacterEncoding(encoding);
        response.setContentType("text/html;charset="+encoding);
    }
}
```

### III-C - Gestion des entrées/sorties fichiers

Enfin, pour la gestion des entrées/sorties fichiers (lecture de fichiers et écriture dans un fichier) à l'aide de l'API Java, il est nécessaire d'utiliser les "bonnes" méthodes c'est-à-dire celles qui sont de suffisamment bas niveau pour prendre en compte l'encodage spécifié et ne pas faire d'assomptions intempestives et malheureuses.



» Pour cela, ne pas utiliser de *FileWriter* et/ou *FileReader* mais préférer les classes et méthodes qui travaillent directement sur les flux de caractères comme la classe *FileInputStream* et *FileOutputStream*.

Ainsi pour récupérer un *Writer* sur un fichier (ce qui peut être nécessaire lors d'application de transformations XSLT par exemple), le code suivant doit être utilisé :

```
public static final Writer getFileWriter(File file, String encoding) {
    try {
        if (!file.exists()) {
            file.createNewFile();
        }
        return new BufferedWriter(new OutputStreamWriter(
            new FileOutputStream(file), encoding));
    } catch (FileNotFoundException fnfe) {
    } catch (IOException ioe) {
    }
    return null;
}
```

A l'inverse, pour lire un fichier, le code suivant peut-être utilisé :


```
public static final Reader getFileReader(File file, String encoding) {
    try {
        return new BufferedReader(new InputStreamReader(
            new FileInputStream(file), encoding));
    } catch (FileNotFoundException fnfe) {
    } catch (UnsupportedEncodingException uee) {
    }
    return null;
}
```

### III-D - Gestion des entrées/sorties avec une base de données

Les bases de données utilisant également un encodage donné, qui n'est pas forcément celui utilisé dans l'application J2EE. Il faut également se renseigner pour identifier cet encodage et effectuer les transformations éventuellement nécessaires lors de la récupération et du stockage de données de/vers la base.

## IV - Du côté des serveurs d'application J2EE

### IV-A - Spécifier l'encodage de la JVM

 » *Exécuter la jvm dans l'encodage désiré.*

Ceci afin que le traitement, en Java, des chaînes de caractères soit dans le même encodage que le reste de l'application. Cela simplifie le code et évite des problèmes potentiels.

C'est au lancement de la JVM (donc au lancement du serveur d'application J2EE) que l'encodage de celle-ci est spécifié à l'aide d'un argument :


```
-Dfile.encoding=UTF-8
```

Par exemple, dans Tomcat, cet argument est à spécifier dans le fichier de lancement *catalina.sh* sous Linux (ou *catalina.bat* sous Windows), dans les options Java.

```
JAVA_OPTS="$JAVA_OPTS -Dfile.encoding=utf-8" // sous Linux
```

### IV-B - Gestion spécifique à chaque serveur

Enfin, chaque serveur d'applications a (malheureusement ?) ses propres caractéristiques de gestion d'encodage et, par conséquent, ses propres paramétrages.

 » *Il faut donc absolument se renseigner et connaître les spécificités du serveur d'applications que l'on utilise.*

Prenons l'exemple de Tomcat. L'encodage des URIs doit être déclaré explicitement via l'attribut *URIEncoding* sur le connecteur de base nommé Coyote, comme suit (dans le fichier *TOMCAT\_HOME/conf/server.xml*) :

```
<Connector port="8080" maxHttpHeaderSize="8192"  
    ...  
    URIEncoding="UTF-8" />
```

## V - Bonus

### V-A - Bon à connaître...

A ce stade, normalement, vous ne rencontrerez plus aucun problème d'encodage. Cela dit, lorsqu'on parle d'encodage, il existe une petite manip qu'il est bon de connaître (car elle peut être utile pour "débugger" par exemple). Cette manip consiste en la transformation d'une chaîne de caractères dans un encodage précis, en cette même (sémantiquement parlant) chaîne MAIS dans un autre encodage !

```
public static String transformStringEncoding(String init,
                                           String encodingBefore,
                                           String encodingAfter) {
    try {
        return new String(init.getBytes(encodingBefore), encodingAfter);
    } catch (UnsupportedEncodingException uee) {
        return null;
    }
}
```

Et pour finir, une dernière recommandation : lorsqu'en Java, vous devez travailler avec des sources (fichiers ou chaîne de caractères) XML, toujours utiliser l'API DOM et ne jamais les manipuler à l'aide d'objets String.

Merci à [vbrabant](#), à [ZedroS](#) et à [Jeannot45](#) pour leur relecture et leurs remarques.

