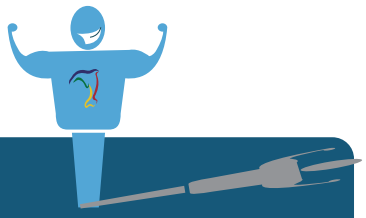
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

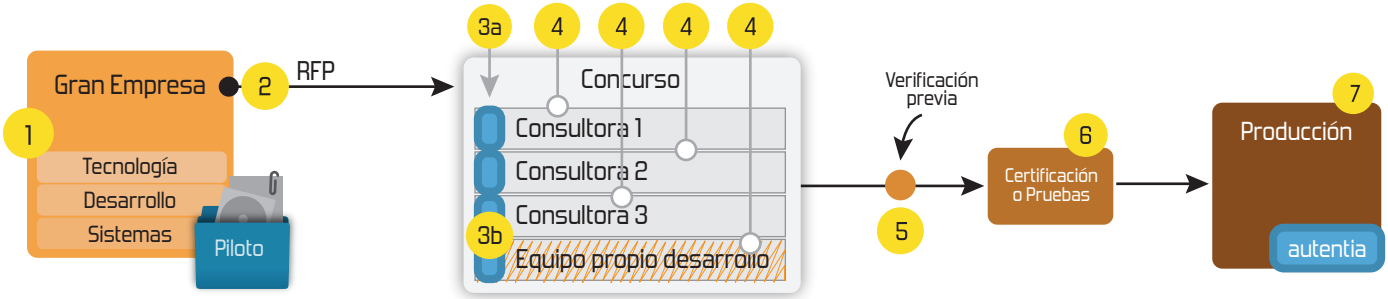
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por

Jose Manuel Sánchez Suárez

Consultor tecnológico de desarrollo de proyectos informáticos. Diseñador de Adictos Al Trabajo 2.0

Puedes encontrarme en Autentia

Somos expertos en Java/J2EE

Descargar este documento en formato PDF: characterencoding-native2ascii.pdf

Fecha de creación del tutorial: 2009-09-08

Sobre las reglas de codificación o... ¿de dónde salen esos caracteres "raros"?.

0. Índice de contenidos.

1. Introducción.

En este tutorial vamos a tratar de dar algo de luz a los errores que tenemos habitualmente con la codificación de caracteres en aplicaciones en las que se ven implicados varios sistemas que intercambian o almacenan información.

Vamos a ir de abajo a arriba, haciendo una pequeña introducción a la codificación de caracteres, mostrando cómo provocar esos errores, para terminar explicando cómo solucionarlos.

2. Codificación de caracteres.

La codificación de caracteres es el método que permite convertir un caracter del lenguage natural, el de los humanos, en un símbolo de otro sistema de representación, aplicando una serie de normas o reglas de codificación. El ejemplo más gráfico suele ser el delcódigo morse, cuyas reglas permiten convertir letras y números en señales (rayas y puntos) emitidas de forma intermitente.

En informática, las normas de codificación permiten que dos sistemas intercambien información usando el mismo código numérico para cada caracter. Las normas más conocidas de codificación son las siguientes:

- ASCII: basado en el alfabeto latino tal como se usa en inglés moderno y en otras lenguas occidentales. Utiliza 7 bits para representar los caracteres, aunque inicialmente empleaba un bit adicional (bit de paridad) que se usaba para detectar errores en la transmisión. Incluye, básicamente, letras mayúsculas y minúsculas del inglés, dígitos, signos de puntuación y caracteres de control, dejando fuera los caracteres específicos de los idiomas distintos del inglés, como por ejemplo, las vocales acentuadas o la letra ñ.
- ISO-8859-1 (Latin-1): es una extensión del código ascii que utiliza 8 bits para proporcionar caracteres adicionales usados en idiomas distintos al inglés, como el español. Existen 15 variantes y cada una cubre las necesidades de un alfabeto diferente: latino, europa del este, hebreo cirílico,... la norma ISO-8859-15, es el Latin-1, con el caracter del euro.
- cp1252 (codepage 1252): Windows usa sus propias variantes de los estándares ISO. La cp1252 es compatible con ISO-8859-1, menos en los 32 primeros caracteres de control, que han usado para incluir, por ejemplo, el caracter del euro.
- UTF-8: es el formato de transformación Unicode, de 8 bits de longitud variable. Unicode es un estándar industrial cuyo objetivo es proporcionar el medio por el cual un texto en cualquier forma e idioma pueda ser codificado para el uso informático. Cubre la mayor parte de las escrituras usadas actualmente.

En la enumeración hemos ido de menos a más, no solo en el tiempo, por el momento de aparición de la norma, sino también por los caracteres que soporta cada una, UTF-8 es la más ambiciosa.

Visto así, la recomendación debería ser el uso de UTF-8 puesto que, escriba en la lengua que escriba, sus caracteres van a ser codificables. Pero, si sólo escribo en castellano, podría limitarme a usar ISO-8859-1, o ISO-8859-15 si necesito el caracter del euro, sin ningún problema.

Más relevante que la norma de codificación que se use para escribir, que no es poco, es ser conscientes que la lectura se debe realizar con la misma norma de codificación con la que se escribió. Y tiene toda la lógica del mundo, puesto que si escribimos un fichero con ISO-8859-1 no debemos esperar que un sistema que lee en UTF-8 lo entienda sin más (aunque realmente entienda gran parte).

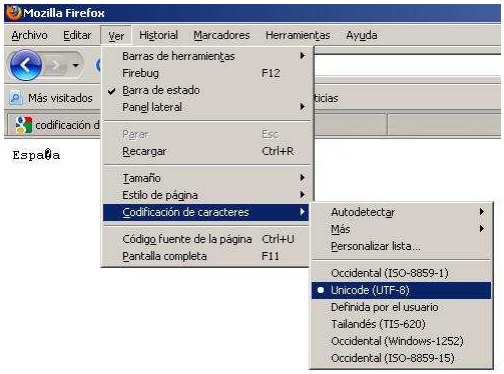
3. ¿Caracteres "raros"?.

Los caracteres "raros" aparecen por una conversión incorrecta entre dos codificaciones distintas. Se suelen producir porque se utiliza la codificación por defecto del sistema o programa y esta no coincide con la original o, directamente, por desconocimiento de la norma de codificación de la fuente de lectura.

Así, por ejemplo, podemos encontrarnos con los siguientes caracteres "raros" escribiendo la misma palabra:

- **España → EspaÃ±a**: si escribimos en UTF-8 y leemos en ISO-8859-1. La letra eñe se codifica en UTF-8 con dos bytes que en ISO-8859-1 representan la A mayúscula con tilde (Ã) y el símbolo más-menos (±).
- **España → Espa◆a**: si escribimos en ISO-8859-1 y leemos en UTF-8. La codificación de la eñe en ISO-8859-1 es inválida en UTF-8 y se sustituye por un caracter de sustitución, que puede ser una interrogación, un espacio en blanco... depende de la implementación.

Podemos provocar un error fácilmente haciendo uso de un editor que permita modificar el formato de escritura, comopspad, y utilizando un lector que permita modificar el de lectura, como por ejemplo un navegador .



Si nos encontramos frente a una aplicación web cliente-servidor tenemos, como mínimo, los siguientes actores implicados: una base de datos con un set de caracteres, una aplicación escrita en un lenguaje que usará su propio encoding para las lecturas y escrituras en esa base de datos y en el sistema de ficheros, un servidor web dinámico o un servidor de aplicaciones que servirá peticiones a un cliente escribiendo en la respuesta con una codificación preestablecida y un cliente que debe leer la respuesta del servidor. Todos esos actores del proceso deben usar la misma norma para leer y escribir, a ser posible estandar, lo deseable: UTF-8.

4. Especificar la codificación de caracteres.

Para evitar problemas con la codificación, siempre debemos indicar explícitamente en nuestros fuentes y sistemas de lectura con qué norma estamos trabajando, con ello le indicaremos al lector la regla de codificación.

En HTML con la siguiente etiqueta en la cabecera del documento

view plain print ?

```
01. <META httpequiv="ContentType" content="text/html; charset=UTF-8'>
```

En XML con el valor del atributo encoding (por defecto es UTF-8):

view plain print ?

```
01. <?xml version="1.0" encoding="ISO-8859-1" ?>
```

Con ello reducimos los posibles problemas pero no los evitamos, puesto que no sirve de nada indicar en el fuente de un fichero html que su encoding es utf-8 si, al guardarlo en disco, lo guardo con un encoding distinto o el servidor lo lee con un encoding diferente para servirlo.

Y, ¿que ocurre con aquellos fuentes en los que no se indica el encoding?, como fuentes java o ficheros de propiedades. Los primeros son leídos por el compilador y los segundos por el propio fuente java compilado, en runtime.

La Máquina Virtual Java (JVM) utiliza una codificación por defecto, que suele ser la del sistema operativo donde se está ejecutando, en windows cp1252. La codificación por defecto se puede conocer obteniendo la propiedad del sistema "file.encoding" y se puede modificar programáticamente (asignando valor a dicha propiedad) o en la compilación. Si bien, volvemos a tener el mismo problema, puesto que, de nada sirve que se asigne dicha propiedad si, al escribir el fuente java, estoy guardando con una codificación distinta.

Con lo dicho, la preocupación no es sólo el encoding asignado al documento en su fuente, sino el encoding con el se escribe el mismo.

5. Solución al problema de codificación en lenguajes de marcado.

La solución al problema de codificación en lenguajes de marcado como html o xml pasa por hacer uso de entidades que sustituyen los caracteres no incluidos en la códificación básica (ascii). Dichas entidades son una clave escrita con los propios caracteres de la norma ascii, aunque también podemos hacer uso de la correspondiente clave del caracter en unicode o de su cógido en decimal. Con ello, son interpretados independientemente de la codificación, puesto que no usan ningún caracter fuera del código ascii.

Así, podemos acudir a cualquier tabla de referencia en las que se incluyen los caracteres y su correspondiente entidad, clave en unicode y en decimal. Por incluir algún ejemplo:

Caracter	Entidad	Unicode	Decimal
"	quot	U+0022	34
á	aacute	U+00E1	225
ñ	ntilde	U+00F1	241

En cualquier fuente xml o html, independientemente del encoding asignado a la cabecera y del usado para guardar el documento, podemos incluir cualquiera de esas tres claves para sustituir su correpondiente caracter de la siguiente forma:

view plain print ?

```
01. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dt>
02. <html xmlns="http://www.w3.org/1999/xhtml' xml:lang="es" lang="es">
03. <head>
04.   <title>Character encoding</title>
05.   <META httpequiv="ContentType" content="text/html; charset=UTF-8'>
06. </head>
07. <body>
08. <p>Usando el nombre de la entidad: Espa&ntilde;</p>
09. <p>Usando el código unicode: Espa&#x00F1;</p>
10. <p>Usando el código decimal: Espa&#241;</p>
11. </body>
12. </html>
```

Podéis probar a guardar el documento en UTF-8 y visualizarlo desde un navegador en ISO-8859-1, los únicos caracteres que no se mostrarán correctamente son las oes acentuadas, puesto que no están escapadas.

Algo tedioso si hubiese que hacerlo manualmente... para ello las herramientas de edición de páginas web realizan esa conversión a entidades automáticamente.

E insisto, esta conversión es del todo innecesaria si especificamos el encoding UTF-8 en la cabecera de nuestros fuentes, guardamos el fichero con ese mismo encoding, y todos los actores implicados en el proceso de lectura usan esa misma norma de codificación.

6. Solución al problema de codificación en java.

Volvemos a los fuentes java y ficheros de propiedades que nos permiten internacionalizar nuestra aplicación; ahora se almacenan en un repositorio de código tipo subversion y son compartidos por un equipo de desarrollo más o menos dimensionado. ¿En qué codificación se almacenarán esos ficheros en el repositorio de código?, trataremos que sea en UTF-8, pero más importante será que todo el equipo de desarrollo tenga asignado esa misma codificación en sus entornos de desarrollo o IDEs.

Un problema común se da trabajando con Eclipse en una plataforma windows, puesto que viene configurado por defecto con el encoding cp-1252, porque es el de por defecto de la JVM en ese SO. Si los fuentes están en el repositorio en UTF-8 y me los descargo sin preasignar dicha codificación, he perdido todas las tildes de los comentarios del fuente java (y quizás el valor de alguna constante de tipo cadena) y, peor aún, de los literales internacionalizados de los ficheros de propiedades. No es mayor problema, si me doy cuenta y modifico el encoding a UTF-8, pero si no caigo en ello y hago commit sin modificarlo, he cambiado la codificación del fuente en el repositorio de código y lo he subido con errores de codificación, con caracteres "raros".

Para solucionar este problema podemos hacer uso de las claves unicode en el fuente de nuestros .java o .properties. Así, por ejemplo:

view plain print ?

```
01. path = path.replace('\u00E1','a');
```

donde **00E1** se correponde con la a acentuada, y el código reemplazaría de la cadena las aes acentuadas por aes sin acento.

Al igual que el punto anterior, algo tedioso para hacerlo manualmente... para ello la propia JDK distribuye una utilidad llamada native2ascii que covierte un fichero de una codificación nativa (no Unicode o no Latin-1) a otro en ascii con caracteres codificados en unicode.

Previene el problema de la codificación de caracteres convirtiendo el contenido de los ficheros en un formato que puede ser almacenado como ascii. Y se suele usar para la internacionalización de los ficheros de mensajes (ficheros de propiedades) de manera que puedan ser leidos por las clases Properties y ResourceBundle, sin necesidad de una configuración específica del entorno.

Es una utilidad por línea de comandos que puedes encontrar en el mismo directorio que el compilador, y que permite:

- introducir caracteres para que los convierta y les de salida por consola
- introducir un path a fichero y que lo convierta dándole salida por consola o a fichero.

```
C:\Program Files\Java\jdk1.6.0_07\bin>native2ascii.exe -encoding UTF-8 c:\tmp\messages_es.properties c:\tmp\messages_es-ascii.properties
```

Volvemos a recordar la importancia de que exista una sincronía entre la codificación de lectura y escritura, puesto que native2ascii es una utilidad java que usará para leer el encoding por defecto de la JVM, si no lo modificamos, será el del SO. Como consecuencia, si nuestro fichero de propiedades está escrito en UTF-8, y no le indicamos lo contrario, la utilidad le leerá como cp1252 en windows, y sustituirá todos los caracteres que no estén incluidos en dicha codificación como \ufffd. De ahí la necesidad de indicar el parámetro **-encoding**.

7. Solución al problema de codificación en el entorno de un proyecto gestionado por maven.

2009-07-16
Gestión de Usuarios en TNT Concept versión 0.16.1

2009-07-16
Continuación del Tutorial: JMonkeyEngine, Creación de nuestro primer juego.

2009-07-16
Como implementar el Scene Monitor para analizar las escenas en jMonkeyEngine

2009-02-26
Transformaciones de escena en JMonkeyEngine

2009-07-15
Detalles del juego de la moto en jMonekyEngine.

2009-07-14
JMonkeyEngine, Creación de nuestro primer juego.

2009-07-13
Ajax tests con Selenium: prototype.js e ICEfaces.

2009-07-08
AOP con AspectJ y Maven

2009-07-07
Instalación y configuración de Eclipse Galileo

2009-07-07
Iniciarse en el manejo de JME, Creación de un Cloth.

2009-07-06
Primeros pasos con Blender: Pintando nuestra mascota en 3D

2009-07-06
DBUnit-Exportar e Importar BBDD

2009-07-05
JMeter, Pruebas de stress sobre aplicaciones web: Grabando y reproduciendo navegaciones

2009-07-02
Axis2: Invocación de Servicios Web usando distintos MEP

2009-07-02
Instalación OpenOffice

2009-07-02
Juegos 3D en Java: Blender y JMonkeyEngine

2009-06-20
StAX (Xml Pull Parser): Streaming API para XML

2009-06-15
Configuración de la desconexión de usuarios con ICEFaces

2009-06-10
LWUIT: Una librería gráfica tipo AWT o Swing para J2ME

2009-06-10
Mapas mentales con XMind

2009-02-26
Redimensionar Imagenes en Windows Vista

2009-06-08
UploadFile con Icefaces + Hibernate + Anotaciones

2009-06-05
Habilitar exportación en Liferay

2009-06-01
Registrar Liferay en Eclipse

2009-05-29
Liferay Social Office

2009-05-28
Broadcast con Ustream

2009-05-25
Tabla datos accesible con ordenación y paginación

2009-05-21
Primeros pasos con Audacity: Un editor de sonido libre y multiplataforma.

Maven gestiona todo el ciclo de vida de nuestro proyecto: compilación, tests, empaquetamiento, versionado,... en base a plugins que configuramos en nuestros pom.xml. Esos plugins estan escritos en java y leen y escriben nuestros ficheros, para lo cuál necesitan la especificación de un encoding. Por defecto, usan el de la JVM, esto es, el del SO, con lo que si estamos en windows será cp-1252. Todos esos plugins aceptan la configuración del encoding de lectura y escritura, y es necesario configurarlo.

view plain print ?

```
01.  <build>
02.    <plugins>
03.      <plugin>
04.        <artifactId>maven-compiler-plugir</artifactId>
05.        <configuration>
06.          <source>1.5</source>
07.          <target>1.5</target>
08.        <encoding>UTF-8</encoding>
09.        </configuratior>
10.      </plugin>
11.      <plugin>
12.        <groupId>org.apache.maven.plugins</groupId>
13.        <artifactId>maven-resources-plugir</artifactId>
14.        <configuration>
15.          <encoding>UTF-8</encoding>
16.        </configuratior>
17.      </plugin>
18.      <plugin>
19.        <groupId>org.apache.maven.plugins</groupId>
20.        <artifactId>maven-site-plugir</artifactId>
21.        <configuration>
22.          <inputEncoding>UTF-8</inputEncoding>
23.          <outputEncoding>UTF-8</outputEncoding>
24.        </configuratior>
25.      </plugin>
26.    </plugins>
27.  </build>
```

Podéis comprobar que lo que hemos hecho es escribir UTF-8 en la configuración de todos los plugins que aceptan la asignación de encoding. Si bien, para no repetir la configuración podríamos haber declarado una propiedad y usarla como una variable o, mejor aún, usar directamente una propiedad preestablecida que asigna unencoding por defecto para todos los plugins de nuestros pom.xml.

view plain print ?

```
01.  <properties>
02.    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
03.    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
04.  </properties>
```

Con ello ahorramos configuración, puesto que ya no es necesaria la configuración individual de cada uno de ellos.

Y para nuestros ficheros de propiedades existe un plugin que permite invocar de manera automática, en la fase de empaquetación, a la utilidad native2ascii permitiendo realizar la conversión de caracteres de ciertos ficheros dentro del propio ciclo de vida del proyecto.

view plain print ?

```
01.  <plugin>
02.        <groupId>org.codehaus.mojo</groupId>
03.        <artifactId>native2ascii-maven-plugir</artifactId>
04.        <version>1.0-alpha-1</version>
05.        <configuratior>
06.             <src>src/main/resources</src>
07.             <dest>target/classes</dest>
08.        </configuration>
09.        <executions>
10.             <execution>
11.                  <goals>
12.                       <goal>native2ascii</goal>
13.                  </goals>
14.                  <configuration>
15.                       <encoding>UTF8</encoding>
16.                       <includes>messages*.properties</includes>
17.                  </configuration>
18.             </execution>
19.        </executions>
20.  </plugin>
```

Se incluye un directorio de origen y destino de los ficheros, el encoding de los fuentes y un patrón para detectar qué ficheros deben ser convertidos, en el ejemplo, todos los ficheros de propiedades que comiencen por "messages".

Aún todo lo anterior, los problemas no estarán del todo solucionados porque siempre existe la posibilidad de que alguien suba al repositorio de código un fichero de propiedades en un formato distinto a UTF-8, con lo que el encoding de escitura difiere del de lectura y la conversión de native2ascii será errónea. En tal caso solo queda identificar al culpable y proceder su escarnio público, con la posibilidad de establecer una sanción económica por haber "roto" el proyecto.

8. Un problema de dificil detección.

Sigo en el mismo proyecto, gestionado por Maven, y tengo que hacer una modificación en un fuente. Se me ocurre que por ser lunes y las 11:15 no voy a hacerlo desde el IDE, desde Eclipse, voy a acceder mediante el explorador de windows para hacerlo desde el block de notas. El fuente estaba en UTF-8 y el notepad al guardar en ese encoding introduce una marca BOM (byte order mark) en el comienzo del fichero. Esos caracteres pueden provocar que el compilador java falle y, si el fuente es un xml, como puede ser un pom.xml de maven, el proyecto dejará de compilar puesto que se trata de un xml mal formado.

view plain print ?

```
01.  ï»¿<?xml version="1.0" encoding="UTF-8"?>
```

La solución pasa por encontrar un editor que permita primero leer esos caracteres y eliminarlos del fuente después. Para ello,jedit tiene un plugin para editar en hexadecimal.

9. Conclusiones.

La codificación es uno de esos temas de los que sí debemos ser conscientes y espero que este tutorial haya ayudado a ello o, al menos, a prevenir ciertos errores comunes.

Un saludo.

Jose

mailto:jmsanchez@autentia.com

- Puedes inscribirte en nuestro servicio de notificaciones haciendo clic aquí.
- Puedes firmar en nuestro libro de visitas haciendo clic aquí.
- Puedes asociarte al grupo AdictosAlTrabajo en XING haciendo clic aquí.
- Añadir a favoritos Technorati.

Recuerda

Autentia te regala la mayoría del conocimiento aquí compartido (Ver todos los tutoriales). Somos expertos en: J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ... y muchas otras cosas.

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en tu empresa?, ¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos ...

Autentia = Soporte a Desarrollo & Formación.

info@autentia.com

Gestión de contenidos

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	Valoración	Votos	Pdf
Cómo hacer deploy del site de Maven en SourceForge	Este tutorial nos enseña un poco mas sobre Maven	2009-08-28	263	Muy bueno	1	
AOP con AspectJ y Maven	Programacion orientada a aspectos con AspectJ y Maven	2009-07-08	972	Bueno	1	
DBUnit-Exportar e Importar BBDD	DBUnit como complemento de los test unitarios con carga a una base de datos	2009-07-06	1214	Muy bueno	6	
Configuración de la desconexión de usuarios con ICEFaces	Este tutorial muestra la manera de configurar y traducir la ventana de desconexión o pérdida de sesión del usuario en ICEFaces.	2009-06-15	1975	Muy bueno	11	
Plugin Hibernate3 para Maven	En este tutorial veremos las posibilidades que nos ofrece el plugin de Hibernate3 para Maven, como por ejemplo, la generación del esquema de base de datos desde clases con anotaciones.	2009-05-02	1496	Bueno	8	
Empaquetamiento de aplicaciones de escritorio (standalone) con Maven	En este tutorial vamos a aprender a empaquetar aplicaciones de escritorio con Maven para ser distribuidas	2009-03-29	2572	Muy bueno	15	
JasperReports Maven Plugin	JasperReports Maven Plugin es un plugin de Maven que nos permite compilar nuestros informes JasperReports.	2009-03-20	3028	Bueno	10	
Integración Selenium / Maven 2 / Surefire / Cargo / Tomcat 6	Con este tutorial se pretende integrar en nuestro proyecto : Maven, Selenium, Surefire, Cargo y Tomcat 6 con el objetivo de incluir y ejecutar las pruebas de integración dentro del ciclo de vida de Maven.	2009-02-26	1506	Muy bueno	3	
Maven JXR Plugin: publica el código fuente en el site	JXR es un plugin para maven, de tipo reporting, que genera en el site un informe con los fuentes java de tu aplicación: "Source Xref".	2009-02-26	1875	Muy bueno	5	
EJB 3.0 y pruebas unitarias con Maven, JUnit 4 y Embedded JBoss sobre Java 6	En este tutorial aprenderemos a configurar Maven para ejecutar test de EJB sobre Embedded JBoss con Java 6	2009-02-09	2684	Bueno	4	

Nota: