

¿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

Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

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

BPM (jBPM o Bonita)
 Generación de informes (JasperReport)
 ESB (Open ESB)

 Powered by 	Hosting Patrocinado por enREDados.com 
---	--

[Home](#) | [Quienes Somos](#) | [Empleo](#) | [Tutoriales](#) | [Contacte](#)



CoNcept

Lanzado

TNTConcept versión 0.3 (14/05/2007)

Autentia da un paso más en su evolución, hemos lanzado una nueva versión con más de 50 mejoras. Ponemos a vuestra disposición el software que hemos construido (100% gratuito y sin restricciones funcionales) para nuestra gestión interna, llamado TNTConcept (auTeNTia).

Construida con las últimas tecnologías de desarrollo Java/J2EE (Spring, JSF, Hibernate, Maven, Subversion, etc.) y disponible en licencia GPL, seguro que a muchos profesionales independientes y PYMES os ayudará a organizar mejor vuestra operativa.

Las cosas grandes empiezan siendo algo pequeño Saber más en: <http://tntconcept.sourceforge.net/>

<p>Tutorial desarrollado por: Iván Zaera Avellón</p> <p>Puedes encontrarme en Autentia Somos expertos en Java/J2EE Contacta en izaera@autentia.com</p>	<p>www.adictosaltrabajo.com es el Web de difusión de conocimiento de www.autentia.com</p>  <p>autentia real business solutions</p> <p>Catálogo de cursos</p>
---	--

Descargar este documento en formato PDF [officeExtraction.pdf](#)

[Firma en nuestro libro de Visitas](#) <-----> [Asociarme al grupo AdictosAlTrabajo en eConozco](#)

Cursos TIC 100% Gratis

iDesempleados Madrid!
 NET/Cna/Java/Seguridad/Unix/Cobol
www.inforedwbcc.net

SOFTENG

Desarrollo soluciones web y gestión
 Consultoría informática Barcelona.
www.softeng.es

Java Installer Builder

Easy to use, amazingly powerful Creates beautiful installers
www.ej-technologies.com

Fecha de creación del tutorial: 2007-05-22

Extracción de texto de documentos Office desde Java

Introducción

En este tutorial vamos a ver como podemos extraer texto de los documentos de Office (DOC, XLS y PPT) desde Java. Además, emplearemos para ello la suite OpenOffice de Sun que, como sabréis, es un magnífico reemplazo de Microsoft Office y, además, gratis y de código abierto.

La suite OpenOffice es capaz de leer documentos OLE2 (de Microsoft Office) con una compatibilidad cercana al 100%. ésto quiere decir que no todo lo que funciona en Microsoft Office funciona en OpenOffice pero, con pequeños retoques, se pueden migrar los documentos de una plataforma a otra sin problema.

Funcionamiento

Para llevar a cabo la extracción de texto es necesario tener instalado Sun OpenOffice junto con su SDK en Java. En Debian tenemos varios paquetes para ello. Los que no tengáis Debian debéis buscar la web de descargas de OpenOffice y bajaros el programa en si y el SDK. Las librerías Java de OpenOffice estarán en el directorio de instalación de OpenOffice, en el subdirectorio "program/classes". Además, en el subdirectorio "sdk" encontraremos todas la documentación necesaria para desarrollar.

A continuación explicaremos la arquitectura de conexión entre Java y OpenOffice: OpenOffice posee un API llamado UNO para desarrollar tanto aplicaciones Java que llamen a OpenOffice como extensiones Java que son llamadas por OpenOffice. A nosotros nos interesa el primer caso. Cuando una aplicación Java llama al API UNO de OpenOffice, lo primero que se hace es cargar una librería nativa (.DLL en Windows, .so en Unix) que lanza un ejecutable de OpenOffice oculto en segundo plano. A continuación, UNO se conecta al ejecutable de OpenOffice a través de un socket y le envía los comandos pertinentes. Este método tiene la ventaja de que Java se comporta como un cliente de OpenOffice, con lo que se pueden hacer aplicaciones UNO distribuidas e incluso multiplataforma. Es decir, podemos, por ejemplo, conectar un programa Java en un Windows a un OpenOffice remoto en un Solaris. UNO se encarga de aislarnos de todas las diferencias entre plataformas (tamaño de palabra, orden de los bytes, etc.).

Una peculiaridad del API UNO es que es orientado a componentes. A los que hayáis programado ActiveX o cualquier componente en la plataforma COM de Microsoft, os resultaran muy familiares los mecanismos de obtención de interfaces de UNO. Básicamente, la única diferencia entre UNO y Java estándar es que en UNO no hay objetos en si sino componentes proveedores de interfaces. Por ejemplo: un documento de texto de OpenOffice es un componente que implementa las interfaces XTextDocument, XComponent, XTextRange, etc. A los componentes sólo se puede acceder a través de alguna de sus interfaces. A algunas interfaces, Sun las llama servicios y al principio de la guía de desarrollo hay una disquisición sobre la diferencia entre servicio e interfaz. Para el desarrollo es algo que nos da más o menos igual: las interfaces se comportan como servicios y viceversa.

Con lo dicho hasta ahora parecería que UNO es igual que Java estándar, pero no es así. En Java podemos tener clases que implementen múltiples interfaces y nos basta con hacer un casting del objeto al interfaz en cuestión para obtenerla. Por ejemplo, dada la clase:

```
public class MiClase implements Interfaz1, Interfaz2
{
    .
    .
    .
}
```

Si creamos un objeto le podemos hacer casting directo a sus dos interfaces:

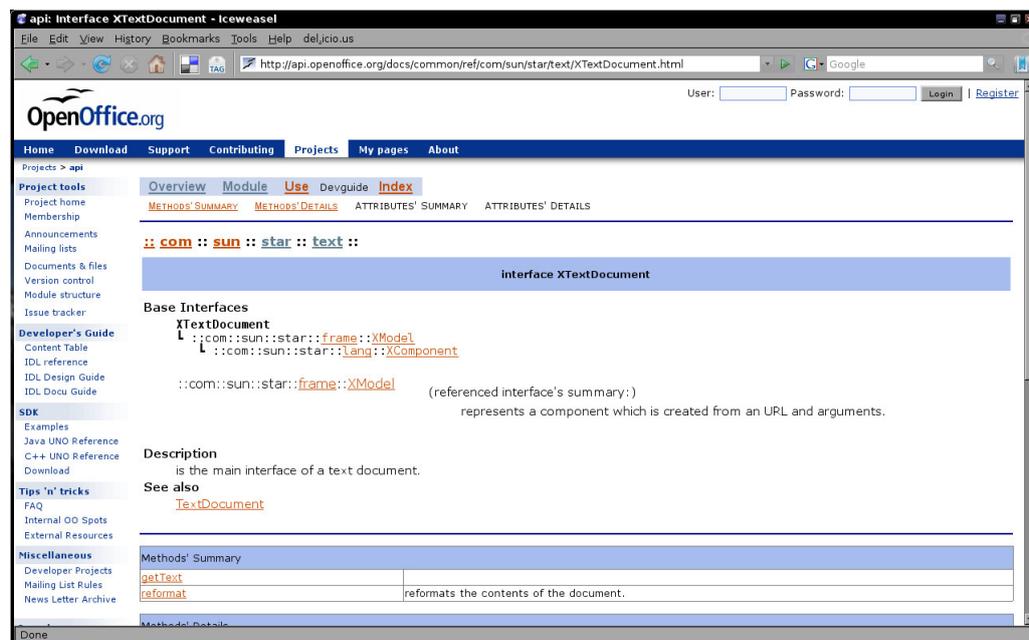
```
MiClase miObjeto = new MiClase();
Interfaz1 if1 = (Interfaz1)miObjeto;
Interfaz2 if2 = (Interfaz2)miObjeto;
```

ésto en UNO no se puede hacer. Como el API es multiplataforma y distribuido, los objetos con los que trabajamos son proxis (al estilo de los de EJB). ésto quiere decir que, dado un componente, por ejemplo un documento de texto, para obtener sus distintas interfaces tenemos que llamar a un método especial del runtime de UNO. Ese método se llama *UnoRuntime.queryInterface()* Veamos un ejemplo:

```
.
.
.
// Cargo un documento de texto (.DOC)
XComponent xComponent = xComponentLoader.loadComponentFromURL(
    "MiDocumento.doc", "_blank", 0, new PropertyValue[0] );

// Obtengo la interfaz XTextDocument del documento
XTextDocument xTextDocument = (XTextDocument)UnoRuntime.queryInterface(
    com.sun.star.text.XTextDocument.class, xComponent );
.
.
.
```

Como se ve en la segunda instrucción, llamamos al método *UnoRuntime.queryInterface()* para hacer un casting de XComponent a XTextDocument.



La documentación del SDK de OpenOffice (<http://api.openoffice.org/docs/common/ref/com/sun/star/module-ix.html>) explica que interfaces cumple cada objeto. Por ejemplo, la documentación del método *loadComponentFromURL()* nos dice que devuelve siempre un XComponent y que, en el caso de haber cargado un documento de texto (.DOC) se puede pasar el XComponent a XTextDocument. Aparte de ésto, la documentación de cada interfaz también incluye enlaces a la documentación explicativa del API UNO.

La miga: extracción del texto

Una vez entendido como funciona UNO, vamos a ver los tres trozos de código que permiten extraer texto de un documento .DOC (documento de texto de MS Word), de un documento .XLS (hoja de cálculo de MS Excel), y de un documento .PPT (presentación de MS PowerPoint). Para ejecutar estos fragmentos de código es necesario incluir en nuestro proyecto las librerías Java de UNO: aunque normalmente no son necesarios todos, lo más sencillo es incluir todos los archivos .JAR que hay en el subdirectorio program/classes del directorio de instalación de OpenOffice.

Código común a los tres ejemplos

```
// Arrancar OpenOffice a través de UNO
XComponentContext xContext = Bootstrap.bootstrap();

// Obtener la factoría de servicios de OpenOffice
XMultiComponentFactory xMCF = xContext.getServiceManager();

// Obtener la ventana principal (Desktop) de OpenOffice
Object oRawDesktop = xMCF.createInstanceWithContext( "com.sun.star.frame.Desktop", xContext );
```

```

XDesktop oDesktop = (XDesktop) UnoRuntime.queryInterface(XDesktop.class,oRawDesktop);

// Obtener interfaz XComponentLoader del XDesktop
XComponentLoader xCompLoader = (XComponentLoader)
    UnoRuntime.queryInterface(com.sun.star.frame.XComponentLoader.class, oDesktop);

// Definir URL del fichero a cargar
String sUrl = "file:///home/ivan/MiDocumento.doc";

// Cargar el documento en una nueva ventana oculta del XDesktop
PropertyValue[] loadProps = new PropertyValue[1];
loadProps[0] = new PropertyValue();
loadProps[0].Name = "Hidden";
loadProps[0].Value = new Boolean(true);
XComponent xComp = xCompLoader.loadComponentFromURL(sUrl, "_blank", 0, loadProps);

// Extraer el texto (esta parte depende del tipo de archivo que hayamos cargado).
// Consultar los siguientes tres apartados para ver el código que habría que introducir aquí
:
:
:

// Cerrar el documento abierto
xComp.dispose();

// Opcionalmente, cerrar el ejecutable de OpenOffice (solo si no vamos a extraer nada más)
oDesktop.terminate();

```

Extracción de texto de un archivo .DOC

Para este tipo de documentos obtendremos una selección de todo el texto del documento, la convertiremos a cadena y ése será el texto extraído.

```

// Hacer casting del documento cargado a XTextDocument
XTextDocument doc = (XTextDocument)UnoRuntime.queryInterface(
    com.sun.star.text.XTextDocument.class, xComp);

// Obtener el rango de texto (XTextRange) que representa a todo el documento
XTextRange range = (XTextRange) UnoRuntime.queryInterface( XTextRange.class, doc.getText() );

// Imprimir el texto del rango, que será todo el texto del documento
System.out.println( range.getString() );

```

Extracción de texto de un archivo .XLS

Este tipo de archivos tienen el problema de que, al no ser el contenido texto secuencial, se puede extraer texto de distintas formas. Por ejemplo, para una hoja de cálculo podemos recorrer todas las hojas y, dentro de cada una, todas las celdas e ir concatenando su contenido. Este proceso es excesivamente lento y puede tardar mucho incluso para hojas muy sencillas (por ello se recomienda emplear el método alternativo, explicado después del siguiente ejemplo de código). Se haría con el siguiente código:

```

// Variable donde almacenaremos el texto
StringBuilder text = new StringBuilder();

// Hacemos casting del documento cargado a hoja de cálculo
XSpreadsheetDocument xls = (XSpreadsheetDocument)UnoRuntime.queryInterface(
    com.sun.star.sheet.XSpreadsheetDocument.class, xComp);

// Obtenemos las hojas del documento y las recorremos por nombre
XSpreadsheets sheets = xls.getSheets();
String[] sheetNames = sheets.getElementNames();
for( String sheetName : sheetNames )
{
    // Obtenemos la hoja actual
    XSpreadsheet sheet = (XSpreadsheet)UnoRuntime.queryInterface(
        XSpreadsheet.class, sheets.getByNames(sheetName) );

    // Obtenemos un cursor que representa todas las celdas de la hoja
    XSheetCellCursor cursor = sheet.createCellCursor();

    // Obtenemos el número de filas y columnas del cursor (hoja)
    XCellRangeAddressable range = (XCellRangeAddressable)UnoRuntime.queryInterface(
        XCellRangeAddressable.class, cursor );
    CellRangeAddress addr = range.getCellRangeAddress();

    // Recorremos las celdas existentes por sus coordenadas
    for( int x=addr.StartColumn ; x<=addr.EndColumn ; x++ )
    {
        for( int y=addr.StartRow ; y<=addr.EndRow ; y++ )
        {
            // Obtenemos la celda actual
            XCell cell = cursor.getCellByPosition(x,y);

            // Miramos el contenido de la celda
            if( cell.getType()==CellContentType.EMPTY )
            {
                // Celda vacía: no hacemos nada
            }
            else
            {
                if( cell.getType()==CellContentType.VALUE || cell.getType()==CellContentType.FORMULA )
                {
                    // Celda con valor constante o fórmula: obtenemos su valor
                    text.append(" ");
                    text.append(cell.getValue());
                }
                else
                {
                    if( cell.getType()==CellContentType.TEXT )
                    {
                        // Celda con texto: obtenemos su contenido
                        XText cellText = (XText)UnoRuntime.queryInterface(XText.class, cell);
                    }
                }
            }
        }
    }
}

```

```

        text.append(" ");
        text.append( cellText.getString() );
    }
}
}
}

// Imprimimos el resultado de la extraccion
System.out.println(text.toString());

```

El método alternativo, mucho más rápido, consiste en salvar el archivo en formato CSV (Comma Separated Values), quitarle las comas y las comillas y, de esa forma, obtener el texto de las celdas. Para quitar dichos caracteres emplearemos expresiones regulares y el método `replaceAll()` de la clase `String` de Java. El código es el siguiente:

```

// Preparar un fichero temporal de trabajo
File tmp = File.createTempFile("extract", "openoffice", new File("/tmp"));

// Salvar la hoja de calculo como CSV
XStorable file = (XStorable)UnoRuntime.queryInterface(
    XStorable.class, xComp);
PropertyValue[] saveProps = new PropertyValue[2];
saveProps[0] = new PropertyValue();
saveProps[0].Name = "FilterName";
saveProps[0].Value = "Text - txt - csv (StarCalc)";
saveProps[1] = new PropertyValue();
saveProps[1].Name = "Overwrite";
saveProps[1].Value = new Boolean(true);
file.storeToURL( tmp.toURL().toString(), saveProps );

// Releer el fichero temporal con el contenido CSV almacenandolo en un StringBuilder
StringBuilder text = new StringBuilder();
InputStream is = new FileInputStream(tmp);
byte[] buffer = new byte[8192];
int c;
while( (c=is.read(buffer))!=-1 )
{
    // OJO: habria que leer como char, no como bytes para respetar el encoding
    text.append(new String(buffer,0,c));
}
is.close();
tmp.delete();

// Quitar caracteres no deseados del texto cargado
String finalText = text.toString();
finalText = finalText.replaceAll("\\",+\"\", \" \");
finalText = finalText.replaceAll("\\\",+\"\", \"\");
finalText = finalText.replaceAll(",+\"\", \"\");
finalText = finalText.replaceAll("\\\",+\"\", \"\");

// Imprimir resultado
System.out.println(finalText);

```

Extracción de texto de un archivo .PPT

Este es sin duda el archivo más complejo de los tres. El problema de las presentaciones es que no podemos recorrerlos todos los objetos gráficos buscando texto (bueno, si podemos, pero sería un poco de locos). Además, tampoco podemos salvar las presentaciones como texto, al estilo de lo que hemos hecho con las hojas de cálculo. ¿Entonces? Entonces tenemos que recurrir a métodos indirectos. El método que se propone aquí es exportar la presentación como documento PDF y utilizar la librería PDFBox para extraer texto del PDF. La librería PDFBox es probablemente la mejor librería de manejo de PDFs en Java y se puede descargar de www.pdfbox.org. Después de descargar, se descomprime el ZIP en un directorio y se añaden los .JAR de los directorios `lib` y `external` a nuestro proyecto. A continuación usamos el código siguiente para extraer el texto:

```

// Preparar un fichero temporal de trabajo
File tmp = File.createTempFile("extract", "openoffice", new File("/tmp"));

// Salvar la presentacion como PDF
PropertyValue[] saveProps = new PropertyValue[2];
saveProps[0] = new PropertyValue();
saveProps[0].Name = "FilterName";
saveProps[0].Value = "impress_pdf_Export";
saveProps[1] = new PropertyValue();
saveProps[1].Name = "Overwrite";
saveProps[1].Value = new Boolean(true);
XStorable file = (XStorable)UnoRuntime.queryInterface(
    XStorable.class, xComp);
file.storeToURL( tmp.toURL().toString(), saveProps );

// Obtener el texto del PDF utilizando PDFBox
PDFTextStripper st = new PDFTextStripper();
PDDocument doc = PDDocument.load(tmp);

// Imprimir resultado
System.out.println( st.getText( doc ) );

```

Problemas

En principio es bastante fácil hacer funcionar la integración Java-OpenOffice; no obstante se pueden presentar algunos problemas típicos. Es altamente recomendable leerse la documentación del SDK antes de empezar. La documentación es amplia, buena y trae muchos ejemplos, y nos puede ahorrar gran cantidad de tiempo.

Los problemas habituales con la plataforma UNO se dan al intentar arrancar OpenOffice. Dado que esto se hace mediante una librería nativa y JNI, éste suele ser el punto débil. Hay dos problemas principales:

- **No se puede cargar la librería nativa de OpenOffice (java.lang.UnsatisfiedLinkError: createJNI):** La librería nativa que Java intenta cargar mediante JNI está en el subdirectorio `program` del directorio de instalación de OpenOffice. UNO busca dicha librería en el `CLASSPATH`, por lo que no sirve de nada poner el directorio `program` en la propiedad de sistema de Java

java.library.path, o en las variables de sistema PATH o LD_LIBRARY_PATH (en Unix). Hay que añadir el directorio program al CLASSPATH de Java. Por ejemplo, en Debian/Unix, basta con añadir /usr/lib/openoffice/program al CLASSPATH.

- **No se puede lanzar el ejecutable de OpenOffice (com.sun.star.comp.helper.BootstrapException: no office executable found!)**: Este problema se produce porque Java no encuentra el ejecutable de OpenOffice. Para solucionarlo no sirve poner el ejecutable en el PATH del sistema, sino que hay que tener el directorio del ejecutable en el CLASSPATH de Java, que es donde lo busca UNO. Por ejemplo, en Debian/Unix basta con añadir /usr/bin al CLASSPATH.



This work is licensed under a [Creative Commons Attribution-NonCommercial-No Derivative Works 2.5 License](https://creativecommons.org/licenses/by-nc-nd/2.5/).
[Puedes opinar sobre este tutorial aquí](#)



Recuerda

que el personal de [Autentia](#) te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#))

¿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?

info@autentia.com

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

Autentia = Soporte a Desarrollo & Formación

Gestión de contenidos

[Autentia S.L.](#) Somos expertos en:
J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ..
 y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
<i>e-mail</i>	<input type="text"/>
	<input type="button" value="Enviar"/>

Otros Tutoriales Recomendados ([También ver todos](#))

Nombre Corto	Descripción
Creación de documentos PDF en sitios web utilizando el componente AspPDF	En este tutorial se muestran las características generales del componente ASP-PDF que permite gestionar documentos PDF desde una aplicación web
Generación de Informes pdf con DataVision	Os mostramos como simplificar la creación de informes un múltiples formatos, incluyendo pdf, con una fantástica herramienta visual (DataVision). Os mostramos como integrarla con Servlets
Creación y configuración de firmas y plantillas de fondos para Microsoft Office - Outlook	En el siguiente tutorial, se explicará como crear y configurar archivos HTML para utilizarlos como firmas y fondos para Microsoft Outlook.
Crear pdfs a partir de páginas HTML	Os mostramos como agrupar y convertir páginas HTML que os interesen en documentos PDF a través de herramientas gratuitas
Librería PDFBOX de Java	En este tutorial os mostramos como utilizar algunas de las utilidades de línea de comandos que incorpora la librería Java PDFBOX, para manejar documentos en formato pdf
Exportar PDF multiidioma con iReport	Este tutorial pretende solucionar los problemas que pueden ocasionarnos la exportación de informes en PDF usando la herramienta iReport en diferentes idiomas

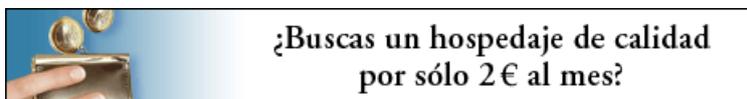
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

[Patrocinados por enredados.com Hosting en Castellano con soporte Java/J2EE](#)



www.AdictosAlTrabajo.com Optimizado 800X600