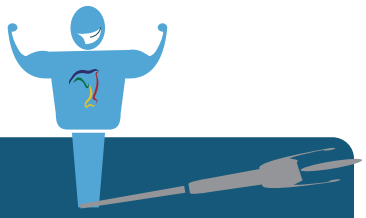
autentia

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...

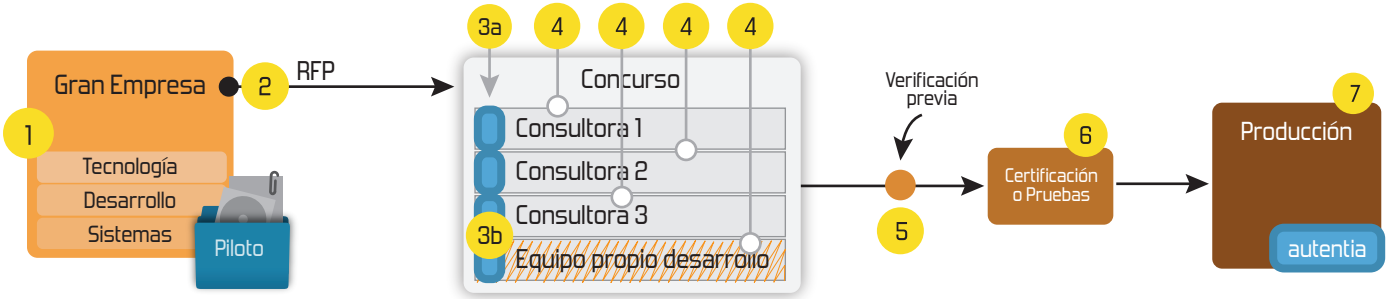
1. Desarrollo de componentes y proyectos a medida



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: Carlos García Pérez

Puedes encontrarme en Autentia
Somos expertos en Java/J2EE
Contacta en:
carlos.garcia@autentia.com

Fecha de creación del tutorial: 2007-07-30

J2ME, FileConnection API. Acceso a tarjetas de memorias desde MIDlets

En la actualidad, debido a la naturaleza multimedia de los terminarles móviles (teléfonos, PDAs, etc.), es muy normal que admitan tarjetas de memoria externas (SD, MultimediaCard, Memory Stick, etc.) que incrementan notablemente sus capacidades de almacenamiento.

En este tutorial voy a intentar hacer una introducción al API FileConection (JSR75) de J2ME que nos permite acceder a estas memorias desde aplicaciones para dispositivos móviles (MIDlets). Se presupone que el lector ya posee conocimientos básicos de programación (J2ME, MIDP, CLDC), compilación e instalación de MIDLets.

Índice de contenidos

1. Introducción
2. El API
3. El ejemplo: Explorando las tarjetas de memoria y abriendo un archivo de texto.
4. Conclusiones y reflexiones.

Introducción

La gran mayoría de las aplicaciones necesitan guardar información en memorias permanentes cuando la aplicación finalice.

CLDC, nos proporciona a través de las clases del paquete `javax.microedition.io` de un mecanismo estándar de almacenamiento de bloques de bytes (`RecordStore`) que sólo pueden ser leídos, modificados y borrados por el MIDlet que lo creó.

Aunque en la mayoría de las aplicaciones este mecanismo es más que suficiente, en otras se ve la necesidad de tener un tratamiento más completo que permita la lectura, la escritura, el borrardo y la búsqueda de archivos y carpetas.

Entre otras cosas, esto último lo que podemos realizar a través del API FileConnection.

FileConnection es una especificación opcional, por lo que la implementación de esta en cada terminal depende del fabricante.

El API

Es un API ligero y sencillo, definido en el paquete `javax.microedition.io.file`.

javax.microedition.io.file.FileSystemRegistry

A través de esta clase podemos enumerar las raices de sistemas de ficheros presentes en el terminal.

> Para hacer una analogía, podría ser como la ventana "MI PC" de los sistemas operativos Windows en donde se muestran las unidades físicas y lógicas presentes sistema.

Además a través de esta clase podemos registrar y desregistrar oyentes (Listener) que serán invocados cuando estas raices cambien. Por ejemplo, cuando se extraiga la tarjeta de memoria.

public static boolean addFileSystemListener(FileSystemListener listener)
Registra un `FileSystemListener` que será notificado cuando varíen las raices de los sistemas de archivos.

public static boolean removeFileSystemListener(FileSystemListener listener)
Desregistra un `FileSystemListener` que será notificado cuando varíen las raices de los sistemas de archivos.

public static java.util.Enumeration listRoots()
Proporciona una enumeración de `java.lang.String` con las cadenas de conexión necesarias para el acceso a cada uno de las raices de los sistemas de ficheros disponibles.

> Por ejemplo, si devolviese "SDCard/" podríamos acceder a ese sistema de ficheros mediante la instrucción:
> `FileConnection fs = (FileConnection) Connector.open("file:///SDCard/")`

javax.microedition.io.file.FileSystemListener

Es una interfaz que deberán implementar las clases que deseen ser registradas. Contiene un sólo método que será invocado por el sistema cuando se añada o elimine la tarjeta de memoria.

public void rootChanged(int eventType, java.lang.String rootName)
EventType podrá ser una de las dos constantes estáticas definidas en la interface: `ROOT_ADDED` o `ROOT_REMOVED`

javax.microedition.io.file.FileConnection

Se trata de una interfaz que será implementada por cada dispositivo y que nos proporciona decenas de métodos **para crear, leer, escribir, consultar y eliminar información relacionada con archivos y directorios.** Debido a la extensión de la misma sólo voy a enumerar algunos de ellos, debiendo dirigirse usted a la especificación oficial si necesita más información.

`public long availableSize()`
Devuelve el número de bytes dispobibles en la memoria.

`public long directorySize(boolean includeSubDirs) throws java.io.IOException`
Devuelve el número de bytes que ocupa un directorio teniendo en cuenta o no los subdirectorios.

`public void fileSize() throws java.io.IOException`
Devuelve el número de bytes que ocupa el fichero.

`public java.util.Enumeration list() throws java.io.IOException`
Devuelve una enumeracion de `java.lang.String de` todos los archivos y carpetas que contiene el directorio actual. **Los directorios se distinguen fácilmente de los archivos normales por que acaban con el caracter "/".**

`public void setFileConnection(java.lang.String fileName) throws java.io.IOException`
Sirve para reutilizar instancias de FileConnection y así no malgastar recursos. Puede usarse el nombre ".." para ir al directorio padre.

`public void rename(java.lang.String newName) throws java.io.IOException`
Renombra un archivo o carpeta.

`public void delete() throws java.io.IOException`
Elimina un archivo o carpeta.

`public java.io.InputStream openInputStream() throws java.io.IOException`
Obtiene un `java.io.InputStream` para leer el archivo.

`public java.io.OutputStream openOutputStream() throws java.io.IOException`
Obtiene un `java.io.OutputStream` para escribir información en el archivo.

El ejemplo: Explorando las tarjetas de memoria y abriendo un archivo de texto.

En este ejemplo vamos a hacer una aplicación que nos permita **navegar por todas las carpetas de todas las memorias así como abrir y mostrar un archivo de texto.**

Para el desarrollo de aplicaciones para móviles, yo personalmente utilizo el IDE NetBeans con la extensión Mobility

---

```
package autentia.tutoriales.fileconnections;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

/**
 * Clase principal del ejemplo.
 * Explorador de ficheros para terminales móviles
 * @author Carlos García. MobileTest
 */
public class FileExplorerMIDlet extends javax.microedition.midlet.MIDlet {

        private FileSelector fileSelector;


    /**
    * Constructor
    */
        public FileExplorerMIDlet(){
                this.fileSelector = new FileSelector(this);
        }


    /*
    * Punto de inicio del MIDlet
    * @see javax.microedition.midlet.MIDlet#startApp()
    */
        public void startApp(){
                Displayable current = Display.getDisplay(this).getCurrent();

                if (current == null){
                        // Aueriguamos si el dispositivo implementa la especificación FileConnection
                        boolean isFCAvailable = System.getProperty("microedition.io.file.FileConnec

                        if (! isFCAvailable) {
                                Alert splashScreen = new Alert(null,
                                "El terminal no permite esta característica", null, AlertType.INFO)
                                Display.getDisplay(this).setCurrent(splashScreen);
                                return;
                        }

                        Display.getDisplay(this).setCurrent(this.fileSelector);
                } else {
                        Display.getDisplay(this).setCurrent(current);
                }
        }

    /*
    * @see javax.microedition.midlet.MIDlet#pauseApp()
    */
        public void pauseApp(){
                // No se requiere la realización de ninguna tarea aqui cuando el MIDlet es pausado.
        }


    /*
    * Liberamos recursos y salimos
    * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
    */
        public void destroyApp(boolean unconditional){
                this.fileSelector.stop();
                this.notifyDestroyed();
        }
}
```

```
package autentia.tutoriales.fileconnections;

import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.io.file.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

/**
 * Explorador de ficheros para terminales móviles
 * @author Carlos García. MobileTest
 */
public class FileSelector extends List implements CommandListener, FileSystemListener, Runnable {

        /**
         * Separador de ficheros
         */
        private final static String FILE_SEPARATOR = (System.getProperty("file.separator") != null)
                                                     System.getProperty("file.separator") : "/";

        /**
         * Directorio padre
         */
        private final static String UPPER_DIR = "..";

        // Códigos de las posibles operaciones: Inicialización y apertura de ficheros
        private final static int LOAD_OP = 1;
        private final static int OPEN_OP = 2;


        private MIDlet midlet;
        private Vector roots;
        private FileConnection currentRoot;

        // Las tareas de I/O deben realizarse en un hilo independiente
        private int    operationCode;
        private Thread task;

        // Atributos gráficos
        private Command cmdOpen, cmdExit;


     /**
      * Constructor
      * @param midlet Referencia al MIDlet
      */
        public FileSelector(MIDlet midlet) {
                super("File Explorer", Choice.IMPLICIT);
                this.midlet = midlet;
                this.roots  = new Vector();

                this.createUI();

                // Nos registramos para que el sistema nos notifique cuando se inserten o extraigan memorias ext
                FileSystemRegistry.addFileSystemListener(FileSelector.this);

                // Mostramos todas las posibles raices de sistemas de ficheros.
                this.operationCode = FileSelector.LOAD_OP;
                task = new Thread(this);
                task.start();
        }

     /**
      * Crea e inicializa el interfaz gráfico
      */
        private void createUI(){
                this.cmdOpen = new Command("Abrir", Command.ITEM, 1);
                this.cmdExit = new Command("Salir", Command.EXIT, 1);

                this.addCommand(this.cmdOpen);
                this.addCommand(this.cmdExit);
                this.setCommandListener(this);
        }
```

Conclusiones y reflexiones

Es increíble la evolución de las capacidades de los dispositivos móviles. Actualmente es normal tener varios cientos de Mb de almacenamiento disponibles.

Aunque normalmente esta memoria es usada para el almacenamiento de imagen y sonido, dejando volar la imaginación se pueden crear aplicaciones que requieran grandes capacidades de almacenamiento y que antes eran inviables... Gestión de mapas, diccionarios, etc. Aun así deberiamos de poner todo nuestro empeño en hacer las cosas bien más aun en este tipo de aplicaciones en donde los recursos de CPU y memoría volátil son limitados.

Bueno, espero que os haya parecido interesante. Desde Autentia nos gusta compartir el conocimiento. aquí tenéis un poquito más de nuestra aportación.

Puedes opinar sobre este tutorial aquí

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	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Desarrollo dispositivos móviles con WindowsCE	Con este tutorial conocerás, pantalla a pantalla, todos los pasos para crear y probar tu primera aplicación Visual C++ para Windows CE, sin ningún conocimiento.
J2ME, Java Wireless Message API (WMA)	En este tutorial se intenta hacer una introducción de las características más importantes que nos proporciona Java para el envío y recepción de SMS desde aplicaciones para móviles (MIDLets).
J2ME Push Registry. Activación automática de MIDlets	En este tutorial se va a tratar una interesante característica que está disponible a partir de MIDP 2.0 para iniciar MIDlets sin la intervención del usuario.
Java en tu movil con J2ME	Os enseñamos como construir una aplicación Java capaz de correr en tu Movil gracias a J2ME
Páginas WML facil para tu movil Wap	Os enseñamos como usar una herramienta para construir de un modo sencillo páginas WML para dispositivos Wap
Desarrollando portales para móviles con FireFox	En este tutorial, se va a presentar User Agent Switcher una extensión para el navegador Web FireFox que nos permite de una forma sencilla emular y probar aplicaciones Web sobre cualquier teléfono móvil a través del propio navegador.