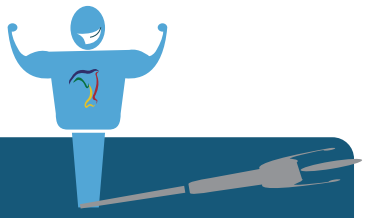
autentia

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...

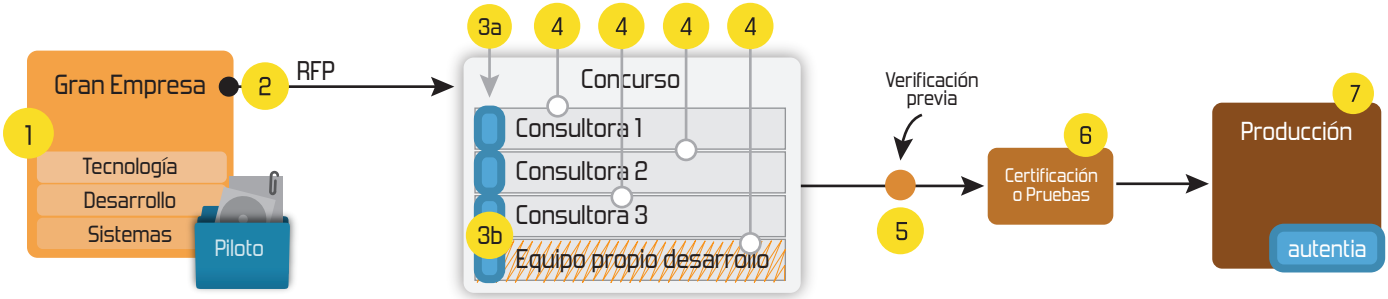
1. Desarrollo de componentes y proyectos a medida



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 conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

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 (au**T**e**NT**ia).

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/

Tutorial desarrollado por: Carlos García Pérez

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

Descargar este documento en formato PDF pushRegistry.pdf

Fecha de creación del tutorial: 2007-05-21

J2ME Push Registry. Activación automática de MIDlets

En este tutorial vamos a tratar una interesante característica que tenemos disponible a partir de MIDP 2.0 para iniciar *MIDlets* sin la intervención del usuario. Se presupone que el lector ya posee conocimientos básicos de programación (J2ME, MIDP, CLDC), compilación e instalación de MIDLets.

Índice de contenidos

Introducción

Normalmente los *MIDlets* son iniciados (ejecutados) por intervención del usuario a través del interfaz de usuario del terminal.

Existen algunas situaciones en donde este modelo se queda cojo y es necesaria una ejecución sin necesidad de la intervención del usuario. Por ejemplo, una aplicación "Agenda" nos podría notificar automáticamente que tenemos una tarea de cumplir.

A está característica se la conoce con el nombre Push Registry.

EL API Push Registry

Push Registry forma parte del *AMS* (Responsable del ciclo de vida de los MIDlets) exponiendo un API para la gestión de estás activaciones.

Básicamente un MIDlet puede ser iniciado de dos formas:

a. A través de una alarma o temporizador. (Sólo se puede registrar una por MIDLet)
b. A través de una conexión entrante.

Las conexiones que pueden activar MIDLets pueden ser una conexión TCP, un datagrama UDP o un SMS. **La norma no especifica cuales deben ser soportadas. Es decisión de los fabricantes. (Normalmente la activación SMS si está soportada)**

A mi personalmente, las conexiónes TCP y UDP no son muy atractivas pues requieren q el terminal esté previamente conectado a Internet para tener una dirección IP, que además es dinámica.

Esta API tiene como corazón la clase `javax.microedition.io.PushRegistry`.

Métodos de la clase:

`public static String getFilter(String connnectionString)`

Devuelve el filtro <Allowed-Sender> para una determinada conexión registrada para el MIDLet.

Por ejemplo, un MIDLet podría ser activado cuando llegara un SMS a un determinado puerto. Pues bien, además puede especificarse un filtro para que sólo sea activado cuando esos SMS provengan de un determinado número de teléfono.

`public static String getMIDlet(String connnectionString)`

Devuelve el nombre de la clase del MIDlet responsable de tratar la conexión especificada como parámetro.

`public static String[] listConnections(boolean available)`

Devuelve una lista con todas las cadenas de conexión registradas por el MIDletSuite del que forma parte el MIDlet.

Si se especifica available a true, sólo se devolverán aquellas cadenas de conexión cuyo `javax.microedition.io.Connection` asociado tengan datos pendientes de ser tratados.

`public static long registerAlarm(String midlet, long time)`

Registra una alarma que activará el MIDlet.

El parámetro `midlet` especifica el MIDlet que debe ser activado. (Normalmente es redundante si sólo hay un MIDlet por MIDletsuite, pero bueno son lentejas.)

El parámetro `time` especifica el milisegundo contando a partir del 1/1/1970 en el que debe ser activado el MIDlet. (No se ni en el día que vivo como para saber de milisegundos) Usando la clase `va.util.Date` para facilitarte calcular el instante deseado.

Si se pasa el valor 0 en parámetro time, desregistrará la alarma.

`public static void registerConnection(String connection, String midlet, String filter)`

Registrar de manera dinámica una conexión.

- connection: Cadena de conexión en la que se registra el MIDlet.

  Ejemplos:

  `sms://:6000` Cadena de conexión válida para que el MIDlet tratará los SMS dirigidos al puerto 6000 del terminal.

  `datagram://:6000` Cadena de conexión válida para que el MIDlet tratará los datagramas dirigidos al puerto 6000 del terminal.

  `socket://:6000` Cadena de conexión válida para que el MIDlet tratará las conexiones TCP que deseen establecerse en el puerto 6000 del terminal.

- midlet: Nombre de la clase del MIDlet que será activado.

  Por ejemplo, si especificamos `this.getClass().getName()` el estariamos registrando el MIDlet que está en ejecución.

- filter: Los filtros que deben cumplir los emisores de la información que activa el MIDlet.

  Por ejemplo, si especificamos `*`, cualquier emisor podría activar el MIDlet.

Normalmente las conexiones se registran estáticamente a través del fichero *JAD* o de manifest del JAR del MIDlet, es decir cuando el MIDlet es instalado en el terminal.

Las alarmas siempre deben ser registradas dinámicamente. No se puede hacer de forma estática.

`public static boolean unregisterConnection(String connection)`

Desregistra dinámicamente una conexión registrada para el MIDlet que lo invoca.

Devuelve `true` en caso de que la operación se haya realizado correctamente `false` en caso contrario.

El *AMS* desregistrará automáticamente todas los registros creados de forma estática o dinámica cuando el MIDlet sea eliminado del terminal.

¿Cómo sabemos si un MIDlet ha sido iniciado por el usuario o por un evento externo?

Pues muy fácil, debemos escribir las siguientes líneas de código en el método startApp() del MIDlet.

```
public void startApp() throws MIDletStateChangeException {
          // Obtenemos la lista de conexiones del MIDlet
          String[] connections = javax.microedition.io.PushRegistry.listConnections(true);

          if ((connections == null) || (connections.length == 0)){
                    // El MIDlet ha sido iniciado por el usuario a través del interfaz gráfico del terminal
          } else {
                    // El MIDlet ha sido iniciado por una conexión entrante o una alarma.
          }
}
```

El código anterior NO es válido para las alarmas. De hecho no hay forma de saber mediante el API si un MIDlet ha sido iniciado mediante una alarma o mediante el interfaz de usuario.

Yo, personalmente cuando necesito saberlo, lo que hago es guardar el instante en el que debe ser activado el MIDlet en la memoria permanente del dispositivo para luego comparar en el startApp() entre con el instante actual... si son más o menos iguales (con un margen de milisegundos) es que el MIDlet fue activado con la alarma.

Registros estáticos a través del archivo JAD

Todos los MIDlets tienen asociado un descriptor (Archivo JAD, Java Archive Descriptor) en donde se definen una gran variedad de información relacionada con la aplicación útil para el terminal durante el proceso de instalación del MIDlet.
De esta manera el terminal puede, por ejemplo, rechazar la instalación del MIDlet si no posee las características mínimas necesarias para ejecutarlo.

En estos descriptores también se definen los permisos que debe de tener la aplicación para ejecutarse (**MIDlet-Permissions**) y también las conexiones que pueden activar el MIDlet **(MIDlet-Push-n)**

(El siguiente fichero JAD no representa al JAD del ejemplo.)

```
MIDlet-1: PushRegistryDemo,,autentia.tutoriales.pushregistry.SMSPushRegistryDemo
MIDlet-Jar-Size: 1682
MIDlet-Jar-URL: PushRegistry.jar
MIDlet-Name: PushRegistry
MIDlet-Permissions: javax.microedition.io.PushRegistry, javax.wireless.messaging.sms.send,
javax.wireless.messaging.sms.receive
MIDlet-Push-1: sms://:6000,autentia.tutoriales.pushregistry.SMSPushRegistryDemo,*
MIDlet-Vendor: Carlos García. Autentia
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.0
```

El atributo MIDlet-Push tiene el formato: `MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>,<AllowedSender>`

Algunas consideraciones:

- Hay una relación directa entre MIDlet-**n** con MIDlet-Push-**n**. Por ejemplo MIDlet-1 está asociado con MIDlet-Push-1 y no con MIDlet-Push-2.
- Se pueden especificar varios MIDlet-Push, incluso para el mismo MIDlet
- `<ConnectionURL>` Conexión que activará el MIDlet.
- `<MIDletClassName>` Nombre completo de la clase principal del MIDlet que tratará la petición.
- `<AllowedSender>` Especifica quienes puede activar el MIDlet. Los carácteres * y ? son válidos. Por ejemplo: 192.100.?3.*

En el ejemplo anterior el MIDlet será activado por la clase **autentia.tutoriales.pushregistry.SMSPushRegistryDemo**, cuando se reciban SMS en el puerto 600 **sms://:6000** por parte de cualquier emisor.

Un ejemplo del uso de alarmas con Push Registry

En el siguiente ejemplo, vamos a registrar una alarma para que active el MIDlet cuando pasen 30 segundos desde que se estableció la alarma.

La siguiente clase no necesita ningún comentario especial, simplemente es la clase principal del MIDlet del ejemplo que muestra la ventana principal de la aplicación.

```
package autentia.tutoriales.pushregistry;

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

/**
 * Ejemplo del uso del API Push Registry para establecer una alarma
 * @author Carlos García. Autentia.
```

```
 */
public class PushRegistryDemo extends javax.microedition.midlet.MIDlet  {

    /**
     * Constructor
     */
    public PushRegistryDemo() {
    }

    /*
     * @see javax.microedition.midlet.MIDlet#startApp()
     */
    public void startApp() throws MIDletStateChangeException {
         Display.getDisplay(this).setCurrent(new PushRegistryMainForm(this));
    }

    /*
     * @see javax.microedition.midlet.MIDlet#pauseApp()
     */
    protected void pauseApp() {
        // En este ejemplo no se requiere ninguna tarea cuando el MIDLet es pausado.
    }

    /*
     * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
     */
    protected void destroyApp(boolean b) throws MIDletStateChangeException {
        this.destroyApp(true);
    }
}
```

La siguiente clase representa la ventana principal de la aplicación, básicamente se compone de tres botones para activar y desactivar la alarma y otro para finalizar la aplicacion.

```
package autentia.tutoriales.pushregistry;

import javax.microedition.io.Connector;
import javax.microedition.lcdui.*;
import java.io.*;
import java.util.Date;
import javax.microedition.io.ConnectionNotFoundException;


/**
 * Ventana principal de la aplicación
 * @author Carlos García. Autentia
 */
public class PushRegistryMainForm
                extends javax.microedition.lcdui.Form
                    implements javax.microedition.lcdui.CommandListener,
                            java.lang.Runnable {

          /**
           * Establece la alarma cada 30 segundos.
           */
          private static final long ALARM_INTERVAL = 30000L;

          /**
           * Referencia al MIDLet
           */
          private javax.microedition.midlet.MIDlet midlet;

          /**
           * Establece la alarma
           */
          private javax.microedition.lcdui.Command cmdSetAlarm;

          /**
           * Eliminar la alarma
           */
          private javax.microedition.lcdui.Command cmdUnsetAlarm;

          /**
           * Finaliza la aplicación
           */
          private javax.microedition.lcdui.Command cmdExit;

        /**
         * Las tareas de establecer/quitar alarmas deben realizarse en un
         * hilo distinto al Thead principal que ejecuta el MIDlet
         */
        private javax.microedition.lcdui.Command btnLastCommand;
        private java.lang.Thread t;

          /**
           * Constructor
           * Ventana principal de la aplicación
           */
          public PushRegistryMainForm(javax.microedition.midlet.MIDlet midlet) {
            super("PushRegistry Demo1");
            this.midlet = midlet;
            this.createUI();

            this.append("MIDlet de ejemplo del uso de activación automática
            mediante alarmas y PushRegistry");
          }

          /**
```

```
         * Crea y configura el interfaz gráfico de la ventana
         */
        private void createUI(){
          this.setTicker(new Ticker("Carlos García. Autentia Real Business Solutions"));
          this.cmdSetAlarm   = new Command("Establecer", Command.OK,   1);
          this.cmdUnsetAlarm = new Command("Quitar", Command.OK,   1);
          this.cmdExit       = new Command("Salir",  Command.STOP, 1);


          this.addCommand(cmdSetAlarm);
          this.addCommand(cmdUnsetAlarm);
          this.addCommand(cmdExit);
          this.setCommandListener(this);
        }


        /**
         * Establece la alarma.
         */
        private void setAlarm() throws java.lang.ClassNotFoundException,
              javax.microedition.io.ConnectionNotFoundException {
      String fullClassName = midlet.getClass().getName();
      long   nextAlarmTime = (new Date().getTime() + ALARM_INTERVAL);


      javax.microedition.io.PushRegistry.registerAlarm(fullClassName, nextAlarmTime);
      this.append(new StringItem("Estado", "La alarma ha sido establecida correctamente.
      Cierre la aplicación y se autoactivará en 30 segundos"));
        }

        /**
         * Quita una alarma.
         */
        private void unsetAlarm() throws java.lang.ClassNotFoundException,
              javax.microedition.io.ConnectionNotFoundException {
      String fullClassName = midlet.getClass().getName();
      javax.microedition.io.PushRegistry.registerAlarm(fullClassName, 0L);
        }


       /**
        * @see java.lang.Runable#run();
        */
       public void run(){
           try {
               if (btnLastCommand == cmdSetAlarm){
                   this.setAlarm();
               } else if (btnLastCommand == cmdSetAlarm){
                   this.unsetAlarm();
               } else if (btnLastCommand == cmdExit){
                   this.midlet.notifyDestroyed();
               }
           } catch (Exception ex){
               this.append(ex.toString());   // En caso de error lo imprimimos por pantalla
           }
       }

         /*
          * @see javax.microedition.lcdui.CommandListener#commandAction(
          *    javax.microedition.lcdui.Command,javax.microedition.lcdui.Displayable)
          */
         public void commandAction(Command arg0, Displayable arg1) {
           try {
               this.btnLastCommand = arg0;
               t = new Thread(this);
               t.start();
           } catch (Exception ex){
               this.append(ex.toString());   // En caso de error lo imprimimos por pantalla
           }
         }
}
```

Para probar el ejemplo, establezca la alarma y rápidamente cierre la aplicación para ver que se activa automáticamente cuando transcurren 30 segundos.

Importante: Si el terminal está desconectado o el MIDlet iniciado LA ALARMA NO SALTA. Si deseas realizar tareas periódicas cuando el MIDlet este activo, puede utilizar la clase `java.util.TimerTask`.

Conclusiones y reflexiones.

Personalmente le veo un potencial amplísimo a esta característica, con un gran abanico de posibilidades para crear software útil, así como para ampliar los servicios que proporcionamos a nuestros clientes. Uniendo está junto con otras tecnologías como las que os presentamos desinteresadamente desde Autentia a través de tutoriales se pueden crear sistemas verdaderamente interesantes.

Aunque pueda parecer sencillo a primera vista, os animo a que practiquen y se pegen con el tema, pues desarrollar software para móviles y que este funcione correctamente tanto en un terminal de hace varios años como en uno actual no es tarea fácil. Yo, en mi formación personal en el tema, me he llevado muchas sorpresas desagradables que me han costado mucho tiempo entender y solventar.

Espero que os haya parecido interesante este tutorial.
Hasta otra.

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	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
Establecimiento de conexiones HTTP desde un MIDLET	Este tutorial nos va a ayudar para resolver los problemas de conexión HTTP desde un MIDLET.
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.
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.
Java en tu movil con J2ME	Os enseñamos como construir una aplicación Java capaz de correr en tu Movil gracias a J2ME
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).

www.AdictosAlTrabajo.com Opimizado 800X600