# Manual de uso del API suministrado por el Sistema de Notificaciones a las Entidades Emisoras

*Febrero 2005*

*Versión 1.4*

## REVISIONES

| Versión | Fecha | Objeto |
|---|---|---|
| 1.0 | 30/09/03 | Inicio del documento |
| 1.1 | 20/11/03 | Se añade información acerca del funcionamiento del Sistema de Notificaciones. |
| 1.2 | 02/03/04 | Se añaden ejemplos para facilitar la integración del API en las aplicaciones de las Entidades Emisoras |
| 1.3 | 12/09/04 | Se añade información acerca del fichero de configuración y se corrigen algunos errores en el documento. |
| **1.4** | 28/01/05 | • Se añade información acerca de una nueva funcionalidad:<br>o Solicitud que permite comprobar si un conjunto de usuarios están suscritos a un servicio. (3.6.3)<br>• Se describe como se debe llevar a cabo una integración de una aplicación en Notific@ (4) |

## INDICE

# 1. Definición del Sistema de Notificaciones

Es un sistema para realizar el envío y seguimiento de **notificaciones telemáticas fehacientes**, con generación de evidencias comprobables de la entrega por el emisor y la recepción por el destinatario.

Conforme al Real Decreto 209/2003 de 21 de Febrero por el que se regulan los registros y las notificaciones telemáticas:

- Nuevo instrumento de comunicación ciudadano-Administración
- Consentimiento expreso del interesado
- Solo para los procedimientos expresamente señalados por el interesado
- Acreditación de fechas y horas de recepción del aviso de notificación y el acceso del interesado al contenido.

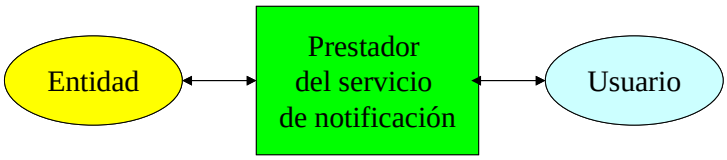
# 2. Arquitectura general del sistema

El sistema de notificación pondrá en relación a tres tipos de entidades que se representan en el esquema siguiente y que designaremos en lo sucesivo por:

Entidades: serán los Organismos de las Administraciones Públicas, las Empresas u otro tipo de entidades con personalidad jurídica propia que constituyen los clientes del servicio de notificaciones y que son los emisores de las notificaciones.

Usuarios: serán los ciudadanos, clientes, proveedores y en general los particulares destinatarios de las notificaciones.

Prestador del servicio de notificación: La Consejería de Justicia y Administración Pública de la Junta de Andalucía.



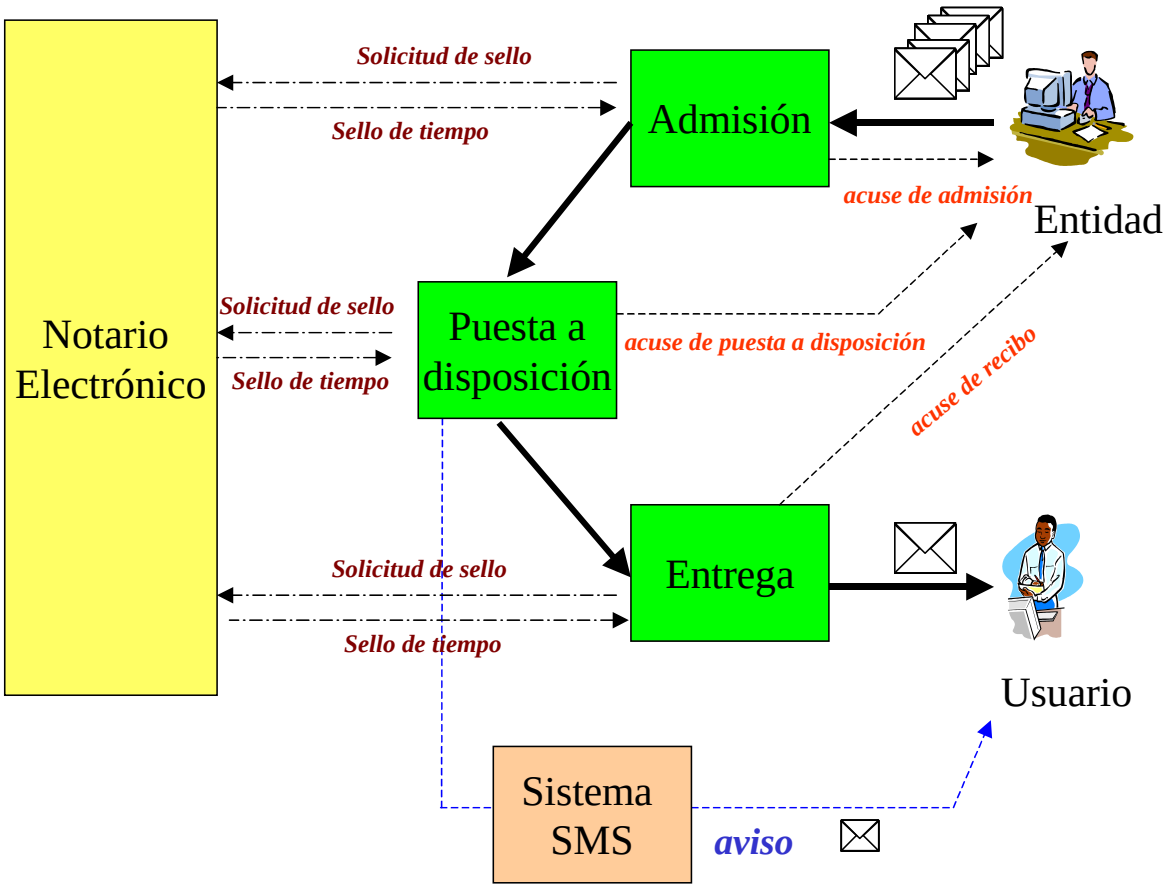

Los tratamientos asociados al envío y entrega de notificaciones pueden representarse mediante una serie de bloques o tratamientos que serán descritos posteriormente de forma mas detallada. De forma resumida son:

Proceso de admisión: Consiste en la entrada al sistema de lotes o remesas de notificaciones provenientes de una entidad determinada, las verificaciones y generación de mensajes de confirmación (acuse de envío)

Proceso de puesta a disposición: Tratamiento que procesa la remesa y deposita cada una de las notificaciones incluidas en ella en la dirección electrónica única del usuario destinatario. Genera acuses de “puesta a disposición” y, opcionalmente, puede generar avisos para los usuarios enviando un mensaje a un buzón de correo electrónico o a un teléfono móvil.

Proceso de entrega: El usuario podrá acceder a una notificación en particular, mediante una interfaz web apropiada. En el proceso se generarán "acuses de recibo" de las notificaciones accedidas.

En el siguiente gráfico se muestra un esquema del proceso seguido desde que una entidad emisora envía una remesa (conjunto de notificaciones) hasta que un usuario destinatario recibe una notificación contenida en la remesa. Como se puede observar como cada uno de los procesos se comunica con el Notario Electrónico para la generación de un sello de tiempo, de tal modo que se tenga constancia de que un proceso finalizó correctamente en un momento dado en el tiempo.

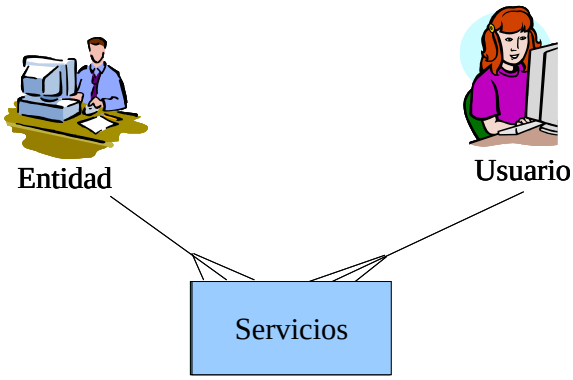



## 2.1 Servicios o suscripciones

Una funcionalidad importante del sistema, es el concepto de "servicio". Un servicio puede ser definido como el nexo de unión entre la entidad emisora y el Usuario. A través de este, el usuario es capaz de recibir notificaciones procedentes de la Entidad Emisora. Como es lógico antes de que un usuario pueda recibir estas notificaciones es necesario que este se suscriba al servicio asociado a la Entidad Emisora por propia iniciativa a través del interfaz web del usuario o suscribiéndole la Entidad Emisora.

El usuario se suscribirá a determinados servicios de entre la lista de los servicios disponibles que le presentará el sistema, y las entidades emisoras a su vez podrán enviar notificaciones correspondientes a aquellos servicios autorizados por el

administrador del sistema. En definitiva: una entidad podrá notificar bajo diferentes servicios (uno único en muchos casos), y los usuarios podrán suscribirse voluntariamente a un número indeterminado de servicios de notificación.




# 3. Funcionalidad del API de Entidades Emisoras

Steria ha desarrollado un API programado en JAVA que permite interactuar con el Sistema de Notificaciones a través del protocolo SOAP (*Simple Object Access Protocol*) contra un *WebService*. El API está compuesto por una serie de métodos que permiten realizar las siguientes funcionalidades:

- Suscripción de un Usuario a un servicio asociado a la Entidad Emisora.
- Baja de un Usuario de un servicio asociado a la Entidad Emisora.
- Construcción de una remesa de notificaciones y envío de la misma a un servicio asociado, previamente contratado con el prestador de servicios de notificación.
- Envío de una circular a todos los Abonados suscritos a un servicio.
- Obtención de información acerca de:
    - Los usuarios que se han dado de alta por iniciativa propia a un servicio asociado de la Entidad Emisora, entre una fecha inicial y final.
    - Los usuarios que se han dado de baja por iniciativa propia de un servicio asociado de la Entidad Emisora, entre una fecha inicial y final.
    - Las remesas enviadas al sistema de notificaciones. Se podrá obtener información sobre la admisión de la remesa, la puesta a disposición de las notificaciones contenidas en la remesa, la lectura o no de una notificación, errores, etc.

Todas las solicitudes que la Entidad Emisora envía al Sistema de Notificaciones a través del API estarán firmadas por un certificado de componente que la Entidad Emisora deberá solicitar a la FNMT y que deberá ser indicado el archivo de configuración del API. Este archivo se explicará en el siguiente punto.

## 3.1 Descripción del contenido del CD entregrado con todo lo necesario para utilizar el cliente de Sistema de Notificaciones

La estructura de directorios es la siguiente:

1. Directorio **lib**, donde se encuentran todas las librerías o ficheros jar necesarios para poder ejecutar la aplicación. El API de Entidades Emisoras se encuentra en la librería *ClientSN.jar*.
2. Directorio **configuración** con los ficheros de configuración del Cliente de Sistema de Notificaciones. Estos ficheros son: a) *mcsn.properties*, para los parámetros del cliente y b) *mcnelog4jConfig.xml*, para la configuración del log. Pueden renombrarse y reconfigurarse.
3. Directorio **doc**, donde se encuentran los ficheros html con la documentación del API desarrollado, y este documento. Los documentos html se encuentran en un directorio llamado html, donde se encuentra un archivo index.html que da entrada a la ayuda.
4. Directorio **Ejemplos**, donde aparece un pequeño programa de ejemplo de utilización, llamado EnviarRemesa.java y CSNTest.java.java.

## 3.2 Instanciación del API

El programador que desee invocar uno de los métodos funcionales del API deberá previamente instanciar la clase *MCSN*, ubicada en el paquete *notificaciones.cliente.api*.

Un ejemplo de instanciación sería el siguiente:

Notificaciones.cliente.api.MCSN mcsn=notificaciones.cliente.api.MCSN(“C:/mcsn.properties, true)

Como se puede observar en la anterior línea la construcción de un objeto de tipo *MCSN*, conlleva el paso de los parámetros siguientes:

- La ruta absoluta del archivo de configuración del módulo cliente del sistema de notificaciones.
- Parámetro lógico. Indica si el API debe configurar el logger LOG4J o por el contrario la aplicación que invoca el API ya ha configurado el mismo el logger.

**Archivo de configuración del módulo cliente del sistema de notificaciones**

En el siguiente cuadro se muestra un ejemplo de archivo de configuración.

```
#Características de la conexión con el sistema de notificaciones.
protocolo=http
direccion_ip=194.224.161.244
puerto=80
path_acceso=axis/*/services/ServicioWEBSN

# ------------------
```

**# Conexión con proxy**

**# ----------**

**# Con conexión proxy a true indica que el acceso es via proxy.**

conexionproxy = false

**# Nombre de servidor proxy o dirección IP:**

proxyhost = nombre del HOST proxy o dirección IP

**# Puerto del servidor proxy:**

proxyport = 8080

**# login conexión al proxy (vacio=>sin autenticación):**

proxylogin =

**# password conexión con proxy:**

proxypassword =

**# ----------**

**# Configuración del Log de trazas**

**# ----------**

**# Fichero de configuración log para la salida con el logger de Log4j**

xml_log=C:/ mcneLog4jConfig.xml

**# ----------**

**# PKCS#12 para la firma de las solicitudes SOAP**

**# ----------**

**# Fichero PKCS#12 que contiene la pareja de claves**

pkcs12.archivo = C:/SoapCertificate.p12

**# Contraseña del PKCS#12 que protege la clave privada**

pkcs12.pass = xmlsignature123

## Parámetros importantes de la configuración del MCSN

| Nombre | Descripción | Posibles valores |
|---|---|---|
| protocolo | Indica el protocolo mediante el cual el módulo se comunicará con el Sistema | • http<br>• https |
| dirección_ip | Dirección IP donde se encuentra el servidor del sistema de notificaciones | • Dirección IP del servidor |
| puerto | Indica el puerto de escucha del servidor web del sistema de | • puerto no seguro de http |

| | | |
|---|---|---|
| | notificaciones que acepta las solicitudes SOAP. Irá en función del protocolo de comunicación utilizado. | • puerto seguro de https |
| path_acceso | Path de acceso al Servicio WEB que atiende las peticiones del Módulo cliente del Sistema de Notificaciones | • Dejar el que viene por defecto. NO TOCAR |
| pkcs12.archivo | Todas las solicitudes enviadas al sistema van firmadas. Por ello es necesario que se indique la ruta absoluta del fichero PKCS#12 que contiene la pareja de claves. | |
| pkcs12.pass | Contraseña de acceso a la clave privada del PKCS#12 | |

El parámetro de configuración *xml_log* permite indicar la ruta absoluta de un fichero en formato XML que permite configurar el logger log4j utilizado en el módulo cliente. Un ejemplo de este fichero podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

 <appender name="appender" class="org.apache.log4j.FileAppender">
  <param name="File" value="C:\\logs\\mcneLog4j.txt"/>
  <param name="Append" value="false"/>
  <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%d [%t] %p - %m%n"/>
  </layout>
 </appender>

 <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern" value="%d{HH:mm:ss} [%c] %p - %m%n"/>
        </layout>
</appender>

 <root>
  <priority value ="info"/>
```

```
   <appender-ref ref="STDOUT"/>
  </root>


</log4j:configuration>
```

La forma en la que la aplicación mostrará información puede modificarse cambiando los contenidos del fichero. Como un breve resumen se puede considerar lo siguiente:

- La etiqueta "**appender**" indicará cómo y dónde se muestra la salida textual.

  En el ejemplo anterior, existen dos etiquetas "appender", una para enviar la salida hacia un fichero cuyo path también se especifica, y otro que utiliza la salida estándar, o por pantalla.
- La etiqueta ConversionPattern especifica el formato utilizado por el logger para mostrar la información.
- Por último, con la etiqueta "root" se configuran dos cosas: el nivel del logger (en este caso "info"), y el "appender" de los definidos que se selecciona, en este caso STDOUT, o salida estándar. Dentro del código de la aplicación habrá escritas diferentes llamadas al logger, cada una invocando una de sus funciones que utilizará alguno de los niveles con los que el logger muestra la información: DEBUG, INFO, WARN, ERROR y FATAL (ALL y OFF). El comportamiento del logger es jerárquico conforme al nivel con el que se haya configurado, de tal modo que mostrará todos los mensajes de ese nivel y de los niveles que estén por debajo. Por tanto, con el nivel info, saldrán también los mensajes error, warn y fatal.

Para obtener información más detallada del comportamiento de los logger, consultar la dirección http://jakarta.apache.org/log4j/docs/documentacion.html

## 3.3 Suscripción de un usuario a un servicio.

El proceso de suscripción de un Abonado a un servicio puede ser realizado de dos formas diferentes:

1. Por iniciativa del usuario desde el interfaz web suministrado por el sistema de notificaciones.
2. Desde una aplicación de la consejería utilizando para ello el API de Entidades Emisoras suministrado por Steria.

En el caso de la segunda opción es necesario tener en cuenta los siguientes puntos:

- Si el usuario no está dado de alta en el sistema en el momento de suscribirlo a un servicio, el sistema lo dará de alta automáticamente.
- Es importante antes de poder dar de alta a usuarios en servicios disponer de los códigos de servicios proporcionados por el Administrador del Sistema a la Entidad Emisora.

- Los datos que debe disponer la entidad acerca del Usuario para realizar el alta en un servicio asociado son:
  - Nombre
  - Apellidos
  - Identificador de Abonado. A través de este, el sistema identificará al usuario.

**NOTA**

El proceso de generación de la cadena que identifica unívocamente a un Abonado en el sistema se ha heredado de la Agencia Tributaria, y es el NIF+ANAGRAMA_LARGO, ya que es el único identificador que no tiene duplicados a nivel nacional.
La sintaxis del anagrama largo es:

- 4 primeros caracteres del apellido 1 sin tildes (Quitando las preposiciones y artículos "DE" "LA", ...)
- 3 primeros caracteres del apellido 2 sin tildes (Quitando las preposiciones y artículos "DE" "LA", ...)
- El primer caracter del nombre.

Si los apellidos tuvieran menor longitud de lo necesario (por ejemplo primer apellido "PAZ") se le añaden espacios al final.

El tamaño del NIF del usuario debe tener nueve caracteres, en caso contrario debe rellenarse el NIF con ceros al principio.

El identificador generado deberá estar en mayúsculas y sin tildes.

Ejem.: Supongamos una persona que se llame Raúl de la Paz Robledo y que tiene como nif, el 09678456U. El identificador generado después del proceso explicado con anterioridad será: **09678456UPAZ ROBR**

  - Teléfono móvil. OPCIONAL. En el caso de que se indique se le enviará un mensaje SMS a este móvil, indicando la puesta a disposición de una notificación por parte del sistema de notificaciones.
  - Correo Electrónico. OPCIONAL. En el caso de que se indique se le enviará un correo electrónico a esta dirección, indicando la puesta a disposición de una notificación por parte del sistema de notificaciones.
  - El FormReference y el TransactionID devuelto por el Servidor de Firma, en el momento de firmar el usuario la aceptación de recepción de determinadas notificaciones por vía telemática, según lo dispuesto en el Real Decreto 209/2003 de 21 de febrero. En el caso de que la aplicación de la Entidad Emisora tenga su propio sistema de firma, deberá suministrar el array de bytes de la firma generada por el usuario aceptando los términos de suscripción a un servicio o procedimiento.

El teléfono y el correo electrónico podrán ser posteriormente modificados por el Abonado desde la interfaz web que suministra el sistema para el acceso a las notificaciones recibidas.

**NOTA**

Es responsabilidad de la Entidad Emisora llevar un control de todos los abonados que ha dado de alta en un servicio, almacenando el identificador de abonado generado y el código de servicio en el que está dado de alta.

Un ejemplo de código JAVA que permite el alta de un abonado en el sistema y su suscripción a un servicio sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un Abonado destinatario de la notificación
  Abonado abonado = new Abonado("09678456UPAZ ROBR");
  abonado.setNombre("Raúl");
  abonado.setApellidos("de la Paz Robledo");
  abonado.setTelefonoMovil("654678909");
  abonado.setEmail("luis @yahoo.es");
  FirmaInf firma = new FirmaInf("formreference","TransactionID");
Enviamos la solicitud de Alta de un usuario en el servicio con código 1
  mcsn.solicitarAltaAbonado(abonado, firma, 1);
```

## 3.4 Baja de un usuario de un servicio.

A través de esta funcionalidad la Entidad Emisora podrá dar de baja un usuario de un servicio suscrito previamente. Para ello la Entidad Emisora deberá suministrar al API un objeto de tipo Abonado donde se haya indicado el identificador del usuario (Ejem. *09678456UPAZ ROBR*).

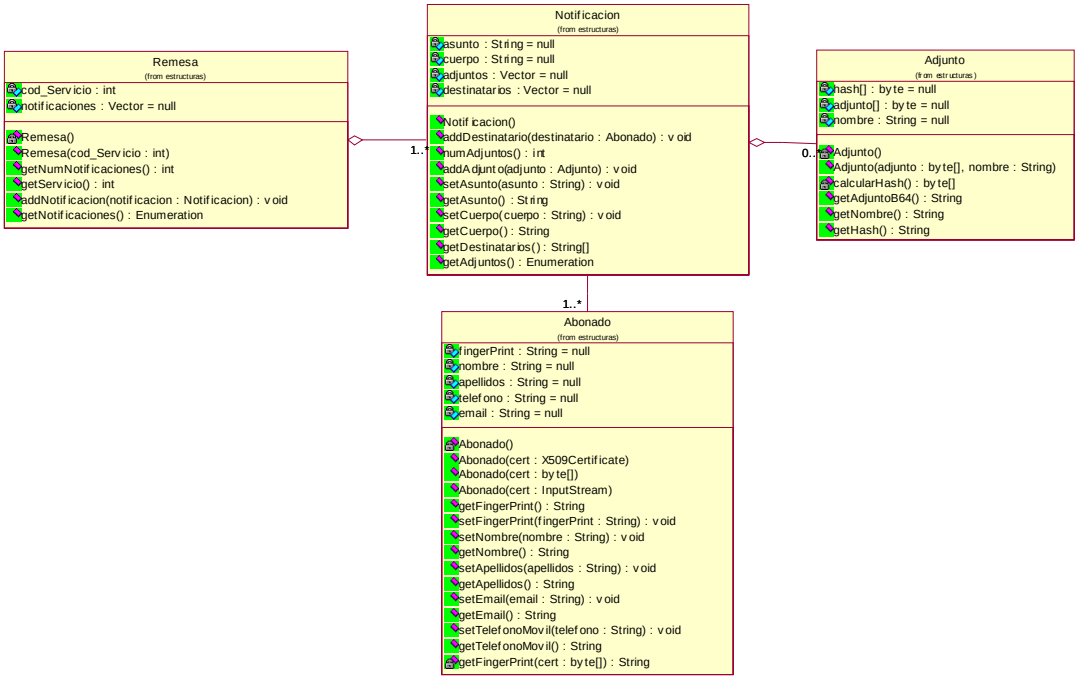
Un ejemplo de código JAVA que permite la baja de un abonado de un servicio sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un objeto Abonado (Usuario que queremos dar de baja)
  Abonado abonado = new Abonado("09678456UPAZ ROBR");
Enviamos la solicitud de Alta de un usuario en el servicio con código 1
  mcsn.solicitarBajaAbonado(abonado,1);
```

## 3.5 Envío de Remesas de notificaciones

Una entidad emisora podrá construir una remesa utilizando para ello las clases suministradas en el API dado por el prestador de servicios de notificación. Una vez construida podrá enviarla al sistema de notificaciones, utilizando para ello el método de la clase MCSN: ***enviarRemesa(Remesa remesa)***.

A continuación se muestra un diagrama de clases en UML, de las clases que intervienen en el proceso de construcción de una remesa:




Un ejemplo de código JAVA que permite la construcción y envío de una remesa sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un Abonado destinatario de la notificación
  Abonado abonado = new Abonado("09678456UPAZ ROBR");
Leemos el fichero que vamos a adjuntar a la notificación
  java.io.File adjunto = new java.io.File("C:/adjunto.pdf");
  if (!adjunto.exists())
        throw new Exception("No se ha encontrado el adjunto en " + pathAdjunto);
  java.io.FileInputStream fis = new java.io.FileInputStream(adjunto);
  java.io.ByteArrayOutputStream baos = new java.io.ByteArrayOutputStream();
  byte[] buffer = new byte[1024];
  int i = 0;
  while ((i = fis.read(buffer)) != -1){
        baos.write(buffer,0, i);
   }
   fis.close();
```

```
Construimos el adjunto
  Adjunto adj1 = new Adjunto(baos.toByteArray(),adjunto.getName());
  baos.close();
Construimos la notificación
  Notificacion notif1 = new Notificacion(false);
  notif1.setAsunto("Información");
  notif1.setCuerpo("Discurso del Presidente del CC.AA. de Andalucía ");
  notif1.addAdjunto(adj1) ; //Podemos adjuntar tantos adjuntos como queramos
  notif1.addDestinatario(abonado); //Podemos añadir tantos destinatarios como queramos
Construimos la remesa que contiene el adjunto creado
  Remesa remesa =new Remesa(1);
  remesa.addNotificacion(notif1);
Enviamos la remesa al Sistema de Notificaciones
  mcsn.enviarRemesa(remesa);
```

**Notas:**

- El texto del mensaje puede contener palabras con tildes y URLs completas. Además el sistema permite el uso de retornos de carro "\n", tabuladores "\t" y caracteres especiales como el signo de mayor ">" y menor "<".
- Si se desea enviar una notificación a todos los usuarios suscritos a un servicio, es decir una circular, se podrá realizar esta acción tan sólo indicando *true* en el constructor de la notificación.

## 3.6 Sistema de información para entidades

Dentro de este subsistema se agrupan los procesos encaminados a extraer y servir informaciones del sistema de notificación a las entidades que lo hayan contratado.

Las informaciones disponibles para las entidades serán:

- Usuarios suscritos por iniciativa propia, entre una fecha inicial y final, a servicios de notificaciones contratados por la entidad.
- Usuarios dados de baja por iniciativa propia, entre una fecha inicial y final, a servicios de notificaciones contratados por la entidad.
- Acuses de admisión. Mediante estos acuses se informará tanto de las remesas admitidas por el sistema de notificación como de los posibles errores detectados en la admisión.
- Acuses de puesta a disposición de los mensajes. Asimismo se generarán errores de puesta a disposición de aquellos mensajes que no se puedan depositar en el buzón del usuario, indicando el error detectado (usuario desconocido, usuario no suscrito al servicio, etc).
- Acuses de recibo de los mensajes de aquellos servicios que lo requieran.

### 3.6.1 Información acerca de los usuarios dados de alta en un servicio

Existe en el Sistema de Notificaciones la posibilidad de que un Usuario previamente dado de alta en el sistema opte por suscribirse a un servicio específico asociado a una Entidad Emisora. Pues bien, esta entidad podrá conocer que nuevos abonados se han dado de alta en un servicio asociado invocando un método del API. El método en cuestión será el siguiente:

*AbonadoInf[] solicitarInformacionAltasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)*

En la invocación del anterior método será necesario que la Entidad Emisora indique el código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quieren obtener los Abonados dados de alta por iniciativa propia. Además será necesario que indique el rango de fechas en los que los abonados se dieron de alta en el servicio indicado.

Como resultado de la invocación del método se obtendrá información de los usuarios dados alta por propia iniciativa. Dicha información estará compuesta por lo siguiente:

- Nombre del usuario
- Apellidos del usuario
- Identificador del usuario en el sistema

### 3.6.2 Información acerca de los usuarios dados de baja de un servicio

Existe en el Sistema de Notificaciones la posibilidad de que un Usuario previamente dado de alta en el sistema opte por darse de baja de un servicio específico asociado a una Entidad Emisora. Pues bien, esta entidad podrá conocer que abonados se han dado de baja de un servicio asociado invocando un método del API. El método en cuestión será el siguiente:

*AbonadoInf[] solicitarInformacionBajasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)*

En la invocación del anterior método será necesario que la Entidad Emisora indique el código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quieren obtener los Abonados dados de alta por iniciativa propia. Además será necesario que indique el rango de fechas en los que los abonados se dieron de baja del servicio indicado.

Como resultado de la invocación del método se obtendrá información de los usuarios dados baja por iniciativa propia. Dicha información estará compuesta por la siguiente:

- Nombre del usuario
- Apellidos del usuario
- Teléfono móvil (Opcional). Puede ser que no desee que se le notifique por teléfono la puesta a disposición de una notificación.
- Correo electrónico (Opcional)

### 3.6.3 Información acerca del estado de un conjunto de abonados en un servicio

Las entidades emisoras podrán en todo momento conocer si un conjunto de usuarios están suscritos a un servicio asociado a la Entidad, gracias a la invocación del método siguiente:

*public int[] solicitarEstadoAbonadoServicio(String[] anagramasAbons, int cod_Servicio)*

En la invocación del anterior método será necesario que la Entidad Emisora indique los siguientes parámetros:

- AnagramasAbons: Es el conjunto de anagramas fiscales que son los identificadores unívocos de los usuarios que queremos conocer su estado en un servicio.
- Código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quiere comprobar si el usuario está suscrito.

Como resultado de la invocación del método se obtendrá un *array* de tipo entero por cada anagrama consultado. Dependiendo de su valor indicará:

- **-1**, El usuario **no** está dado de alta en el sistema
- **0**, El usuario **no** está suscrito al servicio indicado.
- **1**, El usuario **está suscrito** al servicio indicado

Un ejemplo de código JAVA que permite la invocación de esta funcionalidad sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Enviamos la solicitud de estado de un abonado en el servicio con código 1
 String[] ids = new String[1];
  Ids[0]= “09678456UPAZ ROBR”;
  Int[] res = mcsn.solicitarEstadoAbonadosServicio(ids,1);

Procesamos el resultado
  if (res[0] == -1)
      logger.info("El usuario NO está dado de alta en el sistema");
 else if (res[0] == 0)
      logger.info("El usuario NO está suscrito en el SERVICIO " + 1 + ", pero si esta dado de alta en el sistema");
  else if (res[0] == 1)
        logger.info("El usuario SI está suscrito al SERVICIO " + 1 );
```

### 3.6.4 Información de remesas enviadas

A través del API suministrado, la Entidad Emisora podrá obtener información acerca de las remesas enviadas y las notificaciones contenidas en las mismas. Esta información

podrá ser obtenida a través de un conjunto de funciones filtro que a continuación se indican:

> **Nota**
>
> Sólo con el método *obtenerInfRemesas(int[], boolean conAcuses)* existe la posibilidad de obtener los acuses generados por el sistema.

- Información acerca de las remesas enviadas a partir de sus códigos devueltos en el envío de remesa.

Método:

*RemesaInf[] obtenerInfRemesas(int[] remesas, bolean conAcuses)*

El parámetro *conAcuses* indica si se desea obtener los acuses generados, por el sistema y por el abonado en la lectura de la notificación, en el *array* de *RemesaInf*

- Información acerca de las remesas asociadas a un servicio y procesadas entre una fecha inicial y final, que contengan notificaciones leídas por el Usuario.

Método:

RemesaInf[] obtenerInfRemesaConNotifLeidas(Date fechaIni, Date fechaFin, int cod_Suscripcion)

- Información acerca de las remesas asociadas a un servicio y procesadas entre una fecha inicial y final, que contengan notificaciones **no** leídas por el Usuario.

Método:

RemesaInf[] obtenerInfRemesaConNotifNoLeidas(Date fechaIni, Date fechaFin, int cod_Suscripcion, boolean cumplidoPlazo)

El parámetro *cumplidoPlazo* es un modificador del método que permite, si es *true,* obtener información acerca de las remesas y notificaciones que no han sido leídas en el plazo máximo exigible de 10 días en el Real Decreto 209/2003 desde la recepción de la notificación, siendo notificada o avisada por correo electrónico o por el envío de un SMS dicha recepción. En el caso de que su valor sea *false,* se obtendrán todas las notificaciones no leídas, tanto rechazadas como no.

- Obtención de información acerca de las remesas entregadas al Sistema de Notificaciones y procesadas por el mismo entre una fecha inicial y final, y asociadas a una suscripción concreta de la entidad.

Método:

*RemesaInf[] obtenerInfRemesa(Date fechaIni, Date fechaFin, int cod_Suscripcion)*

- Información acerca de las remesas entregadas al Sistema de Notificaciones y procesadas por el mismo entre una fecha inicial y final, asociadas a una suscripción concreta de la entidad, y que contienen notificaciones destinadas a un grupo de abonados.

Método:

*RemesaInf[] obtenerInfRemesa(Date fechaIni, Date fechaFin, int cod_Servicio, Abonado[] abonados)*

# 4. Integración de Notific@ en una aplicación de un Organismo

Como es lógico pensar el uso de este API requiere de un pequeño desarrollo de integración. Para llevar a cabo una integración exitosa será necesario tener presente los siguientes puntos:

1. **Suscripción de usuarios a servicios asociados.** Pueden existir dos opciones.
   a. Usuario suscrito mediante API. En este caso será necesario disponer de datos importantes acerca del usuario, como son:
      *i.* Anagrama fiscal largo, que permite identificar al usuario de forma unívoca. Esta cadena podrá ser obtenida invocando la utilidad *notificaciones.common.util.GeneradorIDUsuario* al que se le facilitará el nombre, el primer apellido, el segundo apellido y el NIF. **Es muy importante que todos estos datos coincidan en caracteres y espacios con los aparecidos en el certificado del usuario.**

      Existe otra posibilidad de obtener el anagrama fiscal largo, siempre y cuando se utilice el servidor de autenticación de la plataforma de @firma para autenticar a los usuarios que se conectan a la aplicación del Organismo. Entre los datos devueltos por esta plataforma se encuentra el anagrama fiscal obtenido a partir del certificado autenticado.

      ii. Nombre

      iii. Apellidos

      iv. Y opcionalmente Correo Electrónico y SMS, si se desea que el sistema automáticamente envíe un aviso de llegada de notificación.

   **Nota**

   Para la suscripción de un usuario mediante el API, es necesario que el usuario firme un formulario dando su consentimiento expreso de que desea que le notifiquen vía telemática todas las notificaciones generadas por un procedimiento.

   *b.* Usuario suscrito por iniciativa propia. En este caso solo existe un mecanismo para averiguar que usuarios están suscritos a un servicio y es invocando el método *AbonadoInf[] solicitarInformacionAltasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)*. Este método devolverá información (Nombre, apellidos, anagrama fiscal) de todos los usuarios suscritos al servicio indicado.

   Será responsabilidad de la aplicación almacenar en base de datos, con el objetivo de enviar en un futuro una remesa de notificaciones, el anagrama fiscal junto con otros datos de interés, de todos los usuarios suscritos a un servicio asociado, y comprobar periódicamente, que usuarios se han dado de baja del

servicio asociado, invocando al método *AbonadoInf[] solicitarInformacionBajasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio).*

2. **Envío de Remesa de Notificaciones.** Una vez que sabemos que usuarios están suscritos a un servicio asociado, bien porque los hemos suscritos a través del API, o bien porque se han suscrito ellos por iniciativa propia, podremos enviarles notificaciones, confeccionando una remesa.

> **Nota**
>
> Es importante antes de enviar una remesa con notificaciones, comprobar que los destinatarios están suscritos al servicio destino de la remesa. Para ello se podrá invocar al método *int[] solicitarEstadoAbonadosServicio(String[] anagramasAbons, int cod_Servicio).* Solo en aquellos destinatarios en los que devuelva '1', podrá enviarse una notificación. Para aquellos que devuelva '0' o '-1' será necesario darles de alta antes de enviarles una notificación.

Es responsabilidad de la aplicación almacenar en base de datos el código devuelto por el sistema en el envío de la remesa, ya que este código permitirá verificar posteriormente que la remesa ha sido procesada correctamente y que las notificaciones contenidas en la misma, han sido leídas o rechazadas por los usuarios o rechazadas por el sistema transcurridos el plazo de diez días desde la puesta a disposición y la no lectura por el destinatario.

Además es conveniente registrar las notificaciones que se envían, con el objetivo de, en un futuro, registrar cualquier evento que se produzca sobre la misma, ya sea por la puesta a disposición, como por su lectura o rechazo. Por ello es conveniente identificar la notificación en el origen, utilizando el método *setId(int)* de la clase *Notificacion*, ya que este identificador servirá para cuando se obtenga información de las remesas enviadas, hacer el casamiento y así poder identificar la notificación que se envío.

3. **Obtener información de remesas enviadas.** Tiene por objetivo verificar que el procesamiento de las remesas ha sido satisfactoria. Hay que tener presente que el procesamiento de una remesa se hace de forma asíncrona, lo que supondría una pérdida de tiempo verificar una remesa nada más enviarla.

   Tiempo para verificar el procesamiento de una remesa

   Lo idóneo, sería verificar una remesa enviada a partir de las **cinco horas** desde que se envío, permitiendo de esta manera dejar suficiente tiempo al sistema para procesarla adecuadamente. La forma de verificar su procesamiento sería invocando al método *RemesaInf[] obtenerInfRemesas(int[] remesas, bolean conAcuses)*

   Por cada *remesaInf* existe un array de *NotificacionInf.* Cada *NotificacionInf* tiene un estado que es devuelto por el método *getEstado()* y que puede devolver cualquier constante definida en la clase *notificaciones.common.util. EstadoNotificacion.* Entre los estados que puede devolver se encuentran:

| Estado | Descripción |
|---|---|
| *0* | Notificación puesta a disposición del abonado. |
| *5* | Notificación puesta a disposición y leída por el usuario. |
| *6* | Notificación puesta a disposición y rechazada por el usuario. |
| *7* | Notificación puesta a disposición y rechazada por el sistema automáticamente, por no lectura de la notificación por el usuario durante los diez días desde su puesta a disposición. |
| *-1* | Se ha enviado una notificación a un destinatario no dado de alta en el sistema. |
| *-2* | Se ha enviado una notificación a un destinatario no suscrito al servicio destino de la remesa. |

Tiempo para verificar la lectura o rechazo de una notificación por el destinatario

Para verificar la lectura o rechazo de la notificación contenida en la remesa es necesario también invocar al anterior método, una vez transcurridos los **once días** desde que se envío la remesa. Esto permitirá que el sistema haya rechazado[1] automáticamente aquellas notificaciones que no hayan sido leídas o rechazadas por el usuario y por lo tanto se tendrá información fidedigna acerca del estado de las notificaciones enviadas, pudiendo estar estas en los siguientes estados posibles:

- Notificación leída por el usuario.
- Notificación rechazada por el usuario.
- Notificación rechazada por el sistema.

[1] Según el Real Decreto 209/2003 de 21 de Febrero, el usuario tendrá un plazo de diez días desde la puesta a disposición de la notificación para leerla o rechazarla, transcurridos estos diez días el sistema automáticamente rechazará la notificación haciendo que esta ya no pueda leerla el usuario.