INTEGRACIÓN Y UTILIZACIÓN DEL NOTARIO ELECTRÓNICO

Febreo de 2003

Referencia	*Fichero*	*Versión*
NE-Cliente-1102	Integración y utilización del Cliente de Notario Electrónico	V 2.1

Tabla de contenido

1. Introducción

En este documento se detallan los componentes necesarios para realizar las solicitudes al Cliente de Notario Electrónico, así como la configuración del entorno, y las librerías necesarias para su correcto funcionamiento.

1.1. ¿Qué es el Cliente de Notario Electrónico?

El cliente de notario es una interfaz cuya API contiene funciones que permiten realizar las solicitudes que tramitará el Notario Electrónico. Las solicitudes son las siguientes:

1. Solicitud de Sello de Tiempo sobre un documento. Se deberá indicar el documento (fichero) sobre el cual desea obtener el sello de tiempo. El cliente de notario generará el HASH (SHA-1) del documento o dato aportado, lo firmará digitalmente con la clave privada del certificado que lleva asociado el módulo, y enviará la solicitud. Completado el proceso, el cliente devolverá el sello (la estructura firmada por la TSA con el ítem de tiempo) como un array de bytes. Para su visualización se podrá realizar una codificación en Base64.

2. Solicitud de Acuse de Recibo sobre un documento. Es el caso más especial, puesto que debe crearse una estructura de Solicitud de Acuse, para lo que deberán gestionarse una serie de objetos que se detallan en los apartados siguientes. La aplicación deberá seleccionar un tipo de solicitud en función de la política escogida. Finalizado el proceso se devuelve el sello como un array de bytes.

3. Solicitud de Validación de un Sello de Tiempo que fue generado previamente. Se indicará el sello de tiempo que se generó (el array de bytes), a fin de que el Cliente pueda enviarlo para su validación. Se devuelve un true o false, dependiendo de si la validación se completó con éxito o no.

4. Solicitud de Validación de un Acuse de Recibo generado previamente. Es igual al caso anterior, con la particularidad de que debe especificarse la política de Acuse de Recibo asociada, puesto que su estructura depende de ella.

5. Solicitud de Extracción de un Sello de Tiempo o de Acuse de Recibo, generados previamente. La aplicación de la consejería deberá indicar el documento o dato sobre el cual se desea solicitar la extracción de sello o Acuse, junto con el tipo de extracción: de sello o de acuse (esto se

realiza instanciando el objeto que corresponda de dos objetos implementados con este objetivo)

1.2. Métodos de la interfaz del Cliente de Notario Electrónico

Para facilitar las llamadas a las solicitudes, el API suministrado incorpora un objeto MCNE, que implementa los métodos de una interfaz con nombre MCNEInterfaz, donde se encuentran las llamadas que debe realizar la aplicación que desee utilizar al Cliente de Notario Electrónico. Dichas funciones se describen a continuación:

Método	Parámetros	Return
solicitarSelloTiempo	• **String : path** Path absoluto del fichero con el documento a sellar	**byte[]** (Sello de tiempo generado)
solicitarSelloTiempo	• **byte[] : dato** Array de bytes que representan al documento sobre el que se solicita el Sello de Tiempo	**byte[]** (Sello de tiempo generado)
solicitarAcuseRecibo	• **SolicitudAcuse:solicitud** Estructura de Solicitud. Esta estructura depende de la política contenida en ella.	**byte[]** (Solicitud generada)
solicitarExtraccionAcuseRecbio	• **String: numReferencia** Número de referencia del Acuse de Recibo en la Aplicación.	
validarSello	• **byte[] : sello** Array de bytes que representan el sello generado y que se desea validar. • **byte[] : dato** Array de bytes que representan el los datos sobre los que se realizo el sello.	**bolean** (resultado de la validación)
validarAcuse	• **byte[] : acuse** Array de bytes que representan el Acuse de Recibo y que se desea validar. • **String : idPolitica** Política de la validación de acuse, puesto que cada política define una estructura de acuse diferente.	**Bolean** (resultado de la validación)

Todas estas funciones lanzan una excepción denominada **ClienteNotarioException**, de la que habrá que invocar el método getMessage() a fin de obtener información detallada sobre la misma.

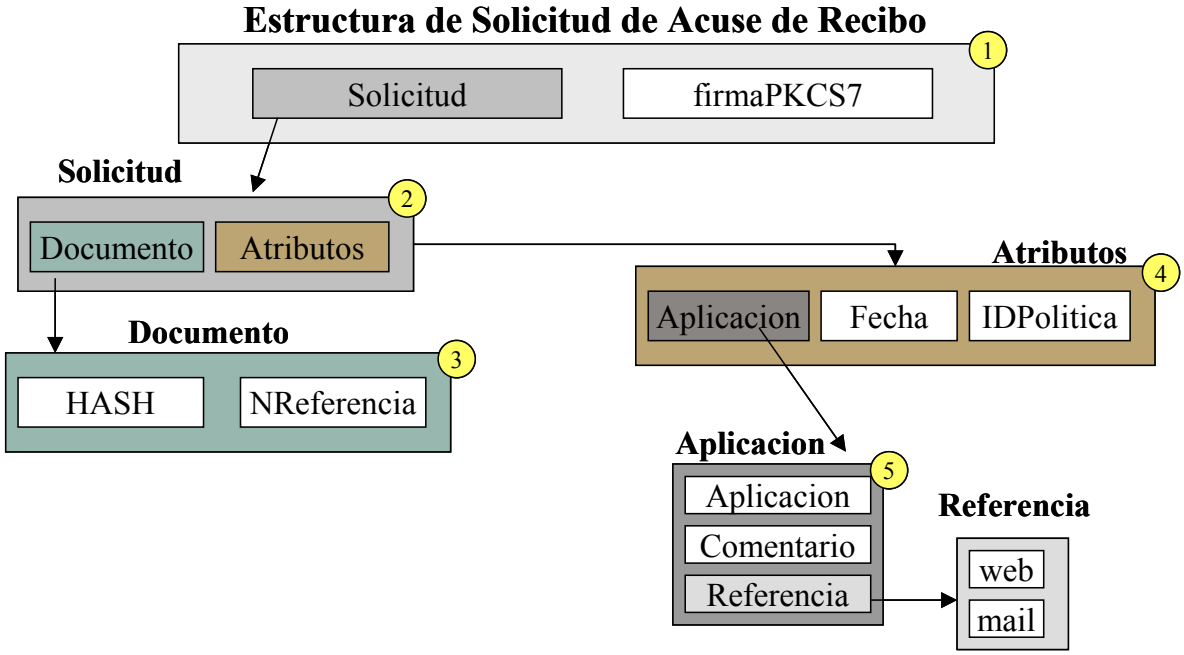
1.3. Estructuras importantes del cliente del Notario Electrónico

A continuación se indican las estructuras que maneja el API y que el lector ha de entender para su posterior uso.

1.3.1. Estructura de solicitud de acuse de recibo

El Acuse de Recibo requiere de una estructura de objetos especial, cuyo contenido dependerá de la política seleccionada.

En la figura siguiente se detallan los elementos constitutivos de una estructura de Acuse de Recibo:



Por claridad, se detallan los componentes que en la figura aparecen numerados con un círculo amarillo.

Puntos	Descripción
1	Representa la estructura principal donde se almacenan los datos de la solicitud de acuse de recibo. Esta estructura esta compuesta de dos campos: ✓ Solicitud: Contiene el conjunto de datos que forman la solicitud de acuse de recibo. ✓ Firma: El propósito de esta firma digital es garantizar la integridad, autenticidad y no repudio de los datos contenidos en la solicitud de acuse de recibo. Para la generación de dicha firma digital se utiliza un certificado emitido para la aplicación que solicita los acuses de recibo al Notario Electrónico. Este ultimo verifica la identidad de la aplicación a través de los procesos de autenticación asociados a la firma digital y se asegura de la integridad de los datos contenidos en la solicitud.
2	La estructura Solicitud esta compuesta por dos campos: ✓ Documento(s): Contiene la información relativa al documento o documentos intercambiados en la relación usuario-aplicación para los cuales se genera una de petición de un acuse de recibo. ✓ Atributos: Contienen un conjunto de información adicional que incrementa el valor informativo de la solicitud de acuse de recibo y del recibo final, especificando las características de la aplicación solicitante, la fecha del servidor donde se ejecuta la aplicación y la política, pues la política determina la estructura de la solicitud.
3	La estructura Documento esta compuesta por dos campos: ✓ HASH: Resumen criptográfico del documento(s) utilizado(s) para las solicitudes de acuses de recibo. ✓ NReferencia: Es un valor alfanumérico, en el cual se almacena la referencia, dentro de la aplicación y su Base de datos, del documento-s original-es a partir de los cuales se genero el HASH.

Puntos	Descripción
4	La estructura Atributos esta compuesta por tres campos: ✓ Aplicación: Identificador unívoco de la aplicación que solicita el acuse de recibo. Obligatorio ✓ Fecha: Representa la fecha obtenida del servidor donde ejecuta la aplicación que solicita el acuse de recibo. Esta fecha tiene un valor informativo y no es equiparable a la fecha utilizada en un sello de tiempo. Opcional ✓ IDPolitica: Contiene el identificador de la política de Acuse de recibo utilizada para generar la solicitud.
5	Campos de la estructura Aplicación: ✓ Aplicación: Nombre de la aplicación. ✓ Referencia: Almacena información de contacto relacionada con la aplicación. ✓ Comentario: Contiene datos de carácter informativo.

Cada uno de los elementos anteriores está gestionado en la interfaz mediante un objeto contenido en el paquete **mcne.asn1.wrappers**. Para mayor detalle ver la ayuda del API en formato html. Los objetos son los siguientes:

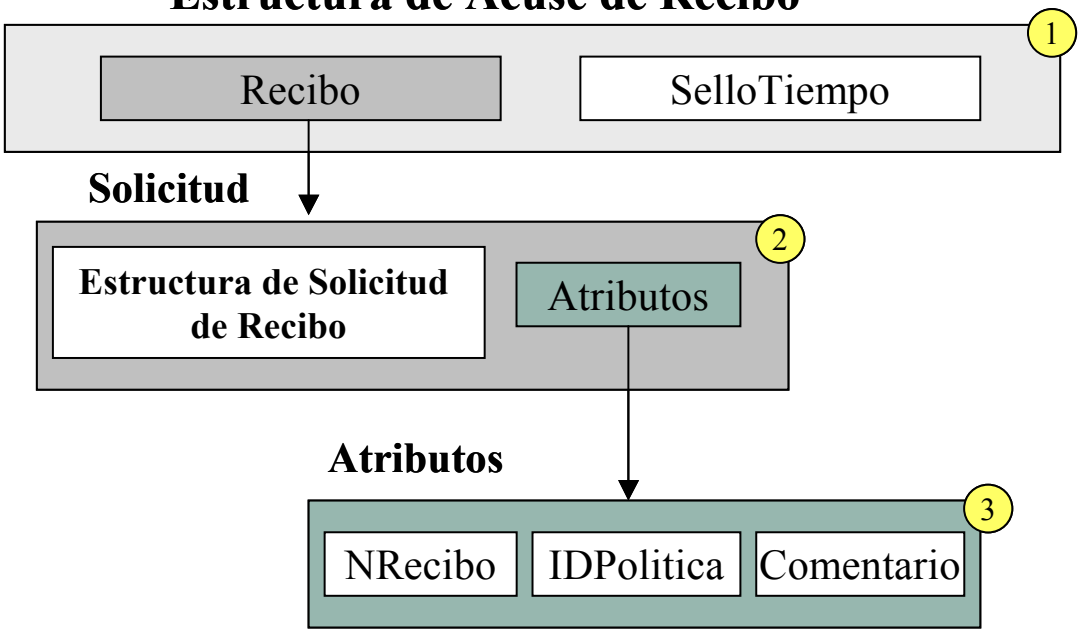
- Referencia: Objeto Referencia
- Aplicación: Objeto Aplicación
- Atributos: Objeto Atributos
- Documento: Objeto Documento
- Documentos: Secuencia de documentos que constituyen la solicitud: Objeto Documentos
- Solicitud: Objeto SolicitudAcuseRecibo
- Solicitud completa del acuse de recibo: Objeto SolicitudAcuse.

Por consiguiente, para obtener una Solicitud de Acuse de Recibo se irán construyendo dichos objetos conforme a la jerarquía que se ha mostrado en la figura, y se invocará al método solicitarAcuseRecibo del objeto MCNE con dicho objeto.

1.3.2. Estructura de Acuse de Recibo

En la siguiente figura se muestra el conjunto de estructuras que componen el formato recibo final obtenido.

Estructura de Acuse de Recibo



Puntos	Descripción
1	Es la estructura principal del Recibo. Esta estructura esta compuesta por dos campos: ✓ Recibo: Estructura que contiene todos los datos que componen el Recibo. Esta estructura se utiliza para la generación del sello de tiempo contenido en el "Recibo". ✓ Sello de Tiempo: Contiene el sello de tiempo que se realiza sobre la estructura Recibo. Con ello se garantiza la integridad, autenticidad y no repudio de los datos contenidos en el Recibo.
2	La estructura Recibo se compone de dos campos: ✓ Solicitud de recibo: Contiene todos los datos contenidos en la solicitud de recibo enviada por la aplicación. Esta estructura se ha explicado con detalle en el apartado 2.1. ✓ Atributos: Conjunto de datos adicionales aportados por el servidor de Notario Electrónico.
3	La estructura Atributos se compone de tres campos: ✓ Nrecibo: Almacena un identificador único para cada recibo emitido por el Notario Electrónico. ✓ IDPolitica: Almacena el identificador de la política de recibos utilizada. ✓ Comentario: Permite almacenar información de carácter aclaratorio o meramente informativo.

2. Integración del Software de Cliente de Notario Electrónico. Guía del Programador.

Se detallan los componentes desarrollados para la utilización e invocación del Cliente de Notario Electrónico, la manera de instalarlos, de configurar el sistema, así como ejemplos de utilización.

Como toda aplicación Java, se distribuyen unas librerías en formato JAR, se indica la manera de incluirlas en el CLASSPATH, así como los ficheros que configuran los parámetros de la aplicación de Cliente de Notario.

2.1. Descripción de los contenidos del Documento

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. La aplicación desarrollada para el cliente de notario electrónico se encuentra en la librería *ClientNE.jar*.
2. Directorio **Configuración** con los ficheros de configuración del Cliente de Notario Electrónico. Estos ficheros son: a) *mcne.properties*, para los parámetros del cliente y b) *mcnelog4jConfig.xml*, para la configuración del log. Pueden renombrarse y reconfigurarse.
3. Directorio **Documentación**, 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 MCNETest.java y CheckNotario.java.

2.2. Descripción del Interfaz del Cliente de Notario Electrónico.

La interfaz del cliente está conforme a la estructura de las Solicitudes ya comentada. Para más detalles sobre las funciones y sus parámetros, visualizar los ficheros generados con javadoc en formato html remitidos. Se comentan los objetos necesarios:

A grosso modo, la jerarquía de los paquetes java es la siguiente:

mcne

asn1

wrappers (objetos para construir la Solicitud de Acuse de Recibo)

api (objetos para invocar al Cliente de Notario mediante el objeto MCNE)

excepciones (objeto para controlar las excepciones)

La documentación html suministrada completa los detalles referentes a la organización de los paquetes.

- El objeto MCNE implementa las funciones que se invocan para llamar al cliente. Se encuentra en el paquete mcne.api:
    - Constructores:
        - MCNE(int idAplicacion, String ficheroPropiedades, boolean log).
            - El fichero de propiedades es el path absoluto de dicho fichero, y el log indica si se desea configurar el log4j con un fichero indicado en el fichero de propiedades o no.
        - MCNE(int idAplicacion, Properties configuración, boolean log)
            - A diferencia del anterior constructor, no es necesario indicar un fichero de configuración del cliente de notario electrónico, sino que se le pasa un objeto Properties con los parámetros de configuración. A continuación se indica un ejemplo de instanciación.

```
Instanciamos un Properties
Properties propiedades = new Properties();

Indicamos el protocolo de acceso al servidor de Notario (http/https)
propiedades.setProperty("protocolo", "http");

Indicamos la dirección IP del servidor de notario
propiedades.setProperty("direccion_ip", "notalinux");

Establecemos el puerto de escucha del servidor de notario
propiedades.setProperty("puerto", "80");

y el path de acceso al servidor de notario
propiedades.setProperty("path_acceso", "jboss-
net/services/ServidorNE");

Indicamos si el acceso al notario es via Proxy o no (true/false)
propiedades.setProperty("conexionproxy", "false");
propiedades.setProperty("proxyhost", "proxy");
propiedades.setProperty("proxyport", "8080");
propiedades.setProperty("proxylogin", "user");
propiedades.setProperty("proxypassword", "pass");

Path del fichero de configuración del log4j
propiedades.setProperty("xml_log","c:/mcneLog4jConfig.xml");

Finalmente instanciamos el API
MCNE mcne = new MCNE(idAplicacion, propiedades, false);
```

- Funciones:
  - boolean validarSello(byte[] sello, byte [] datosDocumento)
  - boolean validarSello(byte[] sello, String ficheroDoc)
  - boolean validadarAcuse(byte[] acuse, String idPolitica)
  - byte[] solicitarSelloTiempo(byte[] datosDocumento)
  - byte[] solicitarSelloTiempo(String ficheroDoc)
  - byte[] solicitarAcuseRecibo(SolicitudAcuse)
  - byte[] solicitarExtraccionAcuse(String referencia)

- Debe controlarse la excepción ClienteNotarioException, donde se indicará el mensaje en caso de que ocurra algún error

- El paquete asn1.wrappers, contiene los objetos para construir una Solicitud de Acuse de recibo.

A continuación se explican fragmentos de código de ejemplo de cómo invocar a las principales funciones del API.

2.2.1. Solicitar un Sello de Tiempo

1. Instanciamos el módulo de cliente de notario electrónico, para ello es necesario indicarle un id de aplicación que previamente nos habrá suministrado la persona encargada del Notario Electrónico.

```
MCNE mcne = new MCNE(idAplicacion, configurationFile, true);
```

2. Solicitamos el sello de tiempo. Este método puede aceptar la ruta de un documento a sellar o un array de bytes que representa los datos que se quieren sellar.

```
byte[] selloTiempo =
      mcne.solicitarSelloTiempo("Prueba a realizar".getBytes());
```

3. Una vez obtenido el sello de tiempo se puede obtener la fecha realizando el siguiente proceso:

```
mcne.asn1.TimeStampResp tsResp =
      mcne.util.TimeStampDecoder.decodeTimeStampResp(selloTiempo);

Date fechaSello =
mcne.util.TimeStampDecoder.getFechaYHoraSelloTiempo(tsResp);
System.out.println("Fecha del sello obtenido: "+ fechaSello);
```

2.2.2. Solicitar Acuse de Recibo

1. Instanciamos el módulo de cliente de notario electrónico, para ello es necesario indicarle un id de aplicación que previamente nos habrá suministrado la persona encargada del Notario Electrónico.

```
MCNE mcne = new MCNE(idAplicacion, configurationFile, true);
```

2. Construimos la solicitud de acuse de recibo.

```
SolicitudAcuseRecibo solicitud = null;

Una solicitud de acuse de recibo consta de atributos y una colección o
sequencia de documentos:

ATRIBUTOS

Referencia referencias = new Referencia(new String("Dirección Web
Aplicación Junta Andalucía"), new String("Correo electrónico de
contacto"));

Los comentarios son opcionales, es decir, en caso que no se desee
introducirlos, se pondrá una cadena vacía ""

String identificadorApp = new String("Nombre de la Aplicación. P.e.:
Aplicacion 1 JA");
```

```
Construimos el objeto aplicación
mcne.asn1.wrappers.Aplicacion aplicacion = new
mcne.asn1.wrappers.Aplicacion(2,"Comentario sobre la aplicación 1 de
la JA", referencias);

Creamos la política:
String identificadorPol = "1.2.3.4"; // el que corresponda.
Politica politica = new Politica( identificadorPol, "Comentario sobre
la política de Acuse Recibo de la JA" );

Atributos atributos = new Atributos(aplicacion, new java.util.Date(),
politica, "Comentario sobre los atributos" );

DOCUMENTOS
Suponemos que el documento está en un fichero, en cuyo caso se
suminstra el path absoluto del mismo, por ejemplo:
String pathDocumento = "C:\\solicitudinscripcion.pdf";

Y que tenemos una referencia en la base de datos de la aplicación del
mismo (que servira para identificarlo en el proceso)
String refDocumento = "ref3";

Creamos el objeto documento
 mcne.asn1.wrappers.Documento documento = new
mcne.asn1.wrappers.Documento(pathDocumento, refDocumento);

En realidad una solicitud está constituida por una Sequencia de
Documentos, que podrá incluir uno o más documentos. Por tanto hay
que contruir el objeto con la secuencia de documentos:
        mcne.asn1.wrappers.Documentos documentos = new
mcne.asn1.wrappers.Documentos(documento);

Los objetos construidos anteriormente son los que constituyen una
solicitud, por lo que debe construirse el objeto:
El identificador de políticas se pasa como un String y por eso se
realiza la conversión:

        solicitud = new SolicitudAcuseRecibo(documentos, atributos,
null);

   3. Solicitamos el acuse de recibo con la solicitud creada

byte[] acuseRecibo = mcne.solicitarAcuseRecibo(solicitud);

Podemos decodificar el acuse realizando el siguiente proceso:

java.io.ByteArrayInputStream bais=new
java.io.ByteArrayInputStream(acuseRecibo);

org.bouncycastle.asn1.DERInputStream deris=new
org.bouncycastle.asn1.DERInputStream(bais);

org.bouncycastle.asn1.DERObject obj = deris.readObject ();


AcuseRecibo recibo = AcuseRecibo.getInstance (obj);
```

```
En este punto ya tendríamos el acuse de recibo, y a partir de este
podríamos extraer el recibo y el sello de tiempo generado.

Recibo recib = recibo.getRecibo();
byte[] sello = recibo.getSelloTiempo();

Si queremos podemos decodificar el sello de tiempo para obtener la
fecha y hora asociada.
TimeStampResp resp =
TimeStampDecoder.decodeTimeStampResp(selloTiempo);

Date fechaAcuseSello =
TimeStampDecoder.getFechaYHoraSelloTiempo(tsResp);

System.out.println("Fecha del sello del acuse de recibo obtenido: "+
fechaAcuseSello);
```

2.3. Ficheros asociados a la aplicación

La aplicación lleva asociados dos ficheros. El primero configura los parámetros del Cliente, como son las características de la llamada al Servidor del Notario (su URL). El segundo es el fichero que configura del log de la aplicación, a fin de obtener información de los errores o del funcionamiento del sistema. El path de este fichero es una propiedad del fichero de configuración: xml_log.

2.3.1. Fichero de configuración del Cliente

Puede tener cualquier nombre, puesto que es un parámetro pasado en la construcción del objeto MCNE. En nuestro caso se llama, mcne.properties y tiene el siguiente contenido:

```
# Características de la conexión con el servidor del notario.
protocolo=http
direccion_ip=10.241.254.16
puerto=80
path_acceso=axis/*/services/ServidorNE

#
# ------------------
# Conexión con proxy
# ------------------
# Con conexionproxy 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 =

# Fichero de configuración log para la salida con el logger de Log4j
xml_log=c:/mcneLog4jConfig.xml
```

NOTA
Para aumentar la velocidad de respuesta a la hora de obtener una acuse de recibo o un sello de tiempo, es MUY RECOMENDABLE configurar los parámetros relativos al protocolo y puerto a ***http*** y ***80***, respectivamente. Esta configuración no influirá en la seguridad, ya que todas las solicitudes que van al Notario Electrónico van firmadas electrónicamente.

2.3.2. Fichero de configuración de los Log

Se trata de un fichero con formato XML, cuyo path absoluto se especifica en la entrada xml_log del fichero de propiedades, y cuyo formato se muestra a continuación:

```
<?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 contendidos 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