Generalitat de Catalunya
Centre de Telecomunicacions
i Tecnologies de la Informació

Framework Corporatiu J2EE

Servei de traces

Versió 1.1

Barcelona, 3 / octubre / 2006

Històric de modificacions

Data	Autor	Comentaris	Versió
09/12/2005	Atos Origin, sae openTrends	Versió inicial del document	1.0
02/10/2006	Atos Origin	Versió 1.1 openFrame	1.1

Llegenda de Marcadors

Índex

1. INTRODUCCIÓ 4

1.1. PROPÓSIT 4
1.2. CONTEXT I ESCENARIS D'ÚS 4
1.3. VERSIONS I DEPENDÈNCIES 5
1.3.1. Versions 5
1.3.2. Dependències Bàsiques 5
1.3.3. Dependències Adicionals 5
1.4. A QUI VA DIRIGIT 6
1.5. DOCUMENTS I FONTS DE REFERÈNCIA 6
1.6. GLOSSARI 6

2. DESCRIPCIÓ DETALLADA 7

2.1. ARQUITECTURA I COMPONENTS 7
2.1.1. Interfícies i Components Genérics 7
2.1.2. Components implementació de Log4J 8
2.1.3. Components implementació de Commons Logging 10
2.2. INSTAL·LACIÓ I CONFIGURACIÓ 11
2.2.1. Instal·lació 11
2.2.2. Configuració 11
2.3. UTILITZACIÓ DEL SERVEI 24
2.3.1. Generar Missatges 24
2.3.2. Evitar càlculs innecessàris 25
2.3.3. Generar missatges estructurats 26
2.3.4. Generar Informació Contextual 26
2.3.5. Generar nous nivells de traces 28
2.4. EINES DE SUPORT 29
2.4.1. Servidor de proves SNMP 29
2.4.2. Visualització de Logs 32
2.5. INTEGRACIÓ AMB ALTRES SERVEIS 33
2.6. PREGUNTES FREQÜÈNTS 33

3. EXEMPLES 34

3.1. TESTS UNITARIS 34

4. ANNEXOS 37

4.1. INTRODUCCIÓ A LOG4J 37

1. Introducció

1.1. Propósit

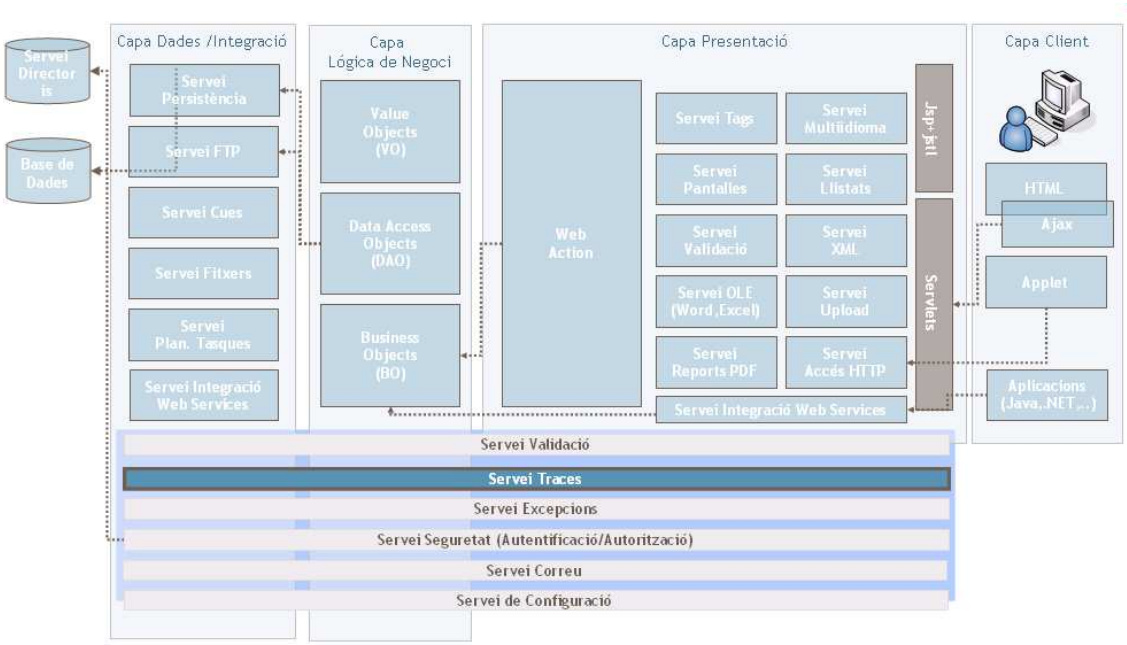
El Servei de Traces té com a missió poder detectar i localitzar amb més facilitat errors de l'aplicació, entrada de dades incorrectes segons la lògica del sistema, i inclús realitzar un seguiment de qui ha fet una determinada operació per a portar a terme una **auditoria.**

Per a que això pugui ser portat a terme, el servei ofereix la possibilitat de:

- Definir nivells de traces (d'informació, fatals, errors, etc.)
- Definir en quines sortides es generarà la traça: consola, fitxers, base de dades, correu electrónic, etc.
- Canviar en qualsevol moment quin és el mínim nivell de traces que volem mostrar, sense haver d'afectar a les classes que generen les traces
- Definir el format de sortida de les nostres traces: incorporar l'hora, el número de línia del codi on s'ha produit i la seva classe, etc.
- Incorporar informació de context a la traça: usuari, ip del client, etc.

1.2. Context i Escenaris d'Ús

El Servei de Traces es troba dins dels serveis de Propósit General de openFrame.



El seu ús és necessari en cas de voler generar traces de la nostra aplicació. Tammateix, molts dels serveis proporcionats per openFrame fan servir aquest servei per a generar les seves traces.

1.3. Versions i Dependències

1.3.1. Versions

En la versió 1.0.2 és va corretgir un bug en PatternXMLLayout que no permetia generar els log en XML en el format de la Generalitat.

1.3.2. Dependències Bàsiques

En el present apartat es mostren quines són les versions i dependències necessàries per fer ús del Servei.

Dins la llista de dependències es mostren diferenciades:

- Dependències bàsiques. Llibreries necessàries per fer ús del servei
- Dependències adicionals. Aquestes dependències són necessàries per poder fer ús de característiques concretes del servei (per exemple, en el cas de voler fer ús de traces a SNMP calen unes dependències adicionals) o per l'ús dels tests unitaris proporcionats amb el servei

Nom	Tipus	Versió	Descripció
commons-logging	jar	1.0.4	http://jakarta.apache.org/commons
log4j	jar	1.2.12	
openFrame-core	jar	1.1	

1.3.3. Dependències Adicionals

- Proves Unitàries del Servei

Nom	Tipus	Versió	Descripció
junit	jar	3.8.1	
jdbcappender	jar	2.1.01	http://www.dankomannhaupt.de/projects/
snmpTrapAppender	jar	1.2.9	
snmp	jar	3.0	
spring	jar	1.2.5	http://www.springframework.org

- Traces a SNMP

Nom	Tipus	Versió	Descripció
snmpTrapAppender	jar	1.2.9	
spring	jar	1.2.5	http://www.springframework.org

- Traces a Base de Dades

Nom	Tipus	Versió	Descripció
jdbcappender	jar	2.1.01	

També cal fer ús de les llibreries específiques de la base de dades utilitzada.

Veure l'apartat 'Instal·lació i configuració' per a més detall.

1.4. A qui va dirigit

Aquest document va dirigit als següents perfils:

- Programador. Per conéixer l'ús del servei
- Arquitecte. Per conéixer quins són els components i la configuració del servei
- Administrador. Per conéixer cóm configurar el servei en cadascun dels entorns en cas de necessitat

1.5. Documents i Fonts de Referència

[1] Log4J Manual http://jakarta.apache.org/log4j/docs/manual.html

1.6. Glossari

Log4J

Un dels framework de traces més extés. Es basa en l'ús de Appenders, Categories i Layouts. Veure l'annex per a més informació.

2. Descripció Detallada

2.1. Arquitectura i Components

openFrame ofereix la possibilitat d'utilitzar diferents implementacions de traces sense que afecti al servei de traces.

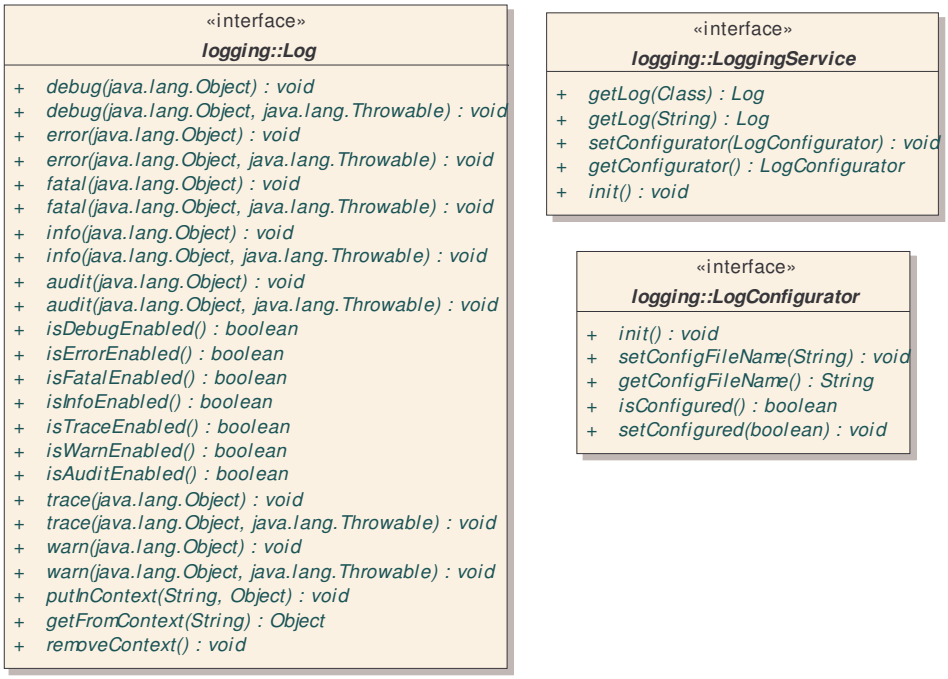
Els components podem classificar-los en:

- Interfícies i Components Genérics. Interfícies del servei i components d'ús general amb independència de la implementació escollida.
- Implementació de les interfícies basada en Log4J
- Implementació de les interfícies basada en Commons Logging

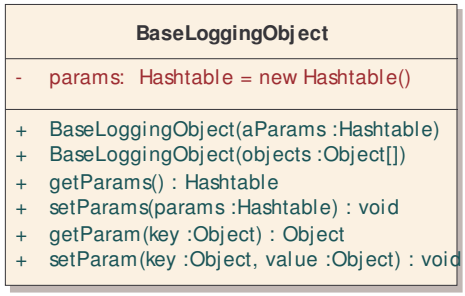
2.1.1. Interfícies i Components Genérics

El servei de traces defineix les següents interfícies:

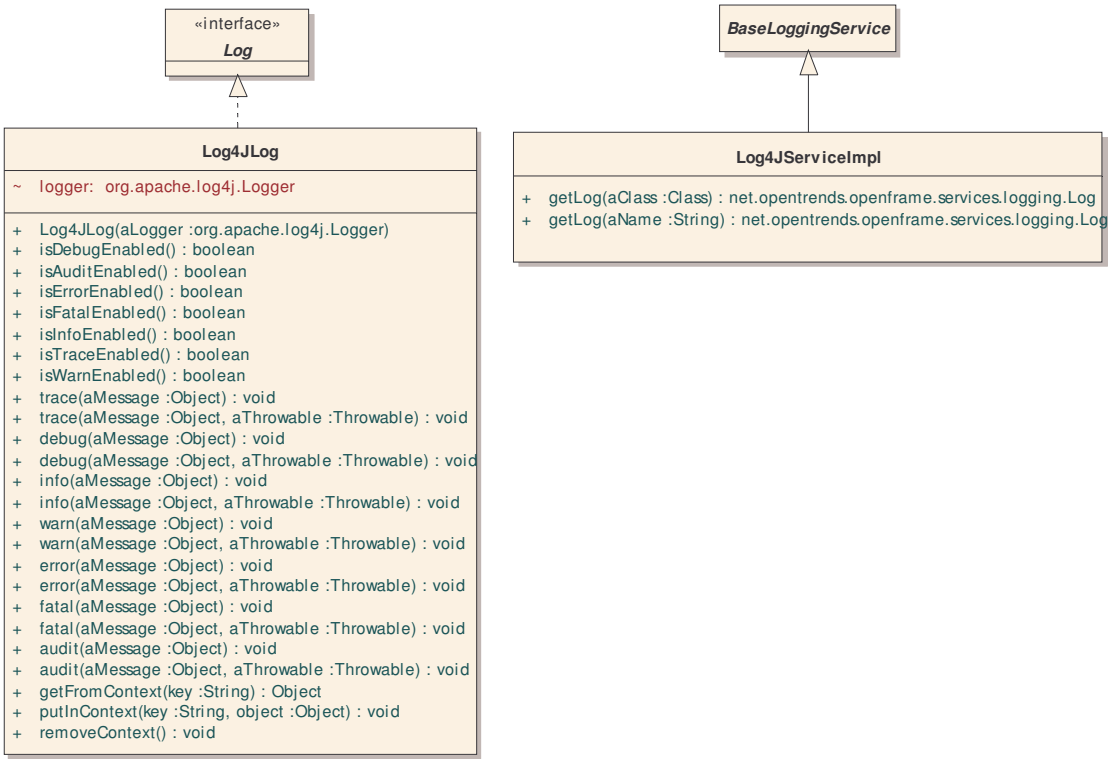
Component	Package	Descripció
Log	net.opentrends.openframe.services .logging	Interfície bàsica que permet volcar missatges a diferents sortides **Ofereix diferents mètodes segons el nivell de traça pel que es volen generar els missatges (*`debug(...)`*, `info(...)`, `error(...)`,** etc.)



Component	Package	Descripció
LoggingService	net.opentrends.openframe.services.logging	**Ofereix entre d'altres (veure la documentació disponible al Javadoc per a més referència):** • **public Log getLog(Class aClass): retorna a partir d'una classe una interfície "net.opentrends.openframe.logging.Log"** • **public Log getLog(String aName): retorna a partir d'un identificador una interfície "net.opentrends.openframe.logging.Log"** • **public void setConfigurator(LogConfigurator aConfigurator)**: estableix el configurador del servei. Aquest configurador establirà les propietats com ara "appenders", categories, etc.. • **public LogConfigurator getConfigurator()**: retorna el configurador associat al servei. • **public void init(): inicialitza el servei (bàsicament cridant al configurador en el cas que no estigui inicialitzat)**
BaseLoggingObject	net.opentrends.openframe.services.logging	Objecte que ens permet generar missatges estructurats (enlloc de cadenes)

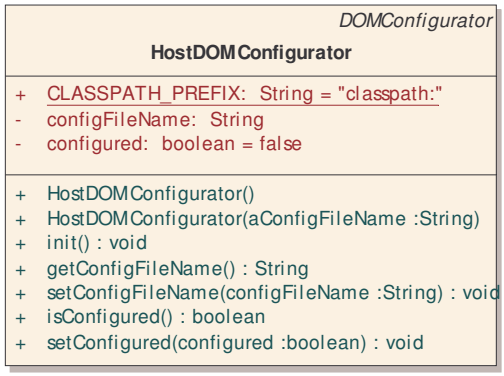


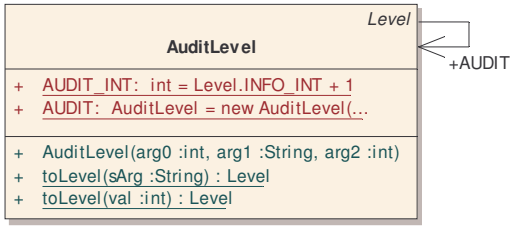
2.1.2. Components implementació de Log4J



Component	Package	Descripció
Log4JLog	net.opentrends.openframe.services.logging.log4j	Implementació de la interfície 'Log' basada en Log4J
Log4JServiceImpl	net.opentrends.openframe.services.logging.log4j	Implementació de la interfície 'LoggingService' basada en Log4J
AuditLevel	net.opentrends.openframe.services.logging.log4j	Nivell adicional als proporcionats per Log4J que permet generar missatges d'auditoria
PatternXMLLayout	net.opentrends.openframe.services.logging.log4j.xml	Layout específic per a poder generar fitxers XML estructurats
HostDomConfigurator	net.opentrends.openframe.services.logging.log4j.xml	Configurador basat en host. Permet que es puguin especificar diferents fitxers de configuració segons l'entorn, de forma que en executar-se el servei s'obtindrà el fitxer corresponent al host en el que s'executa el servei.

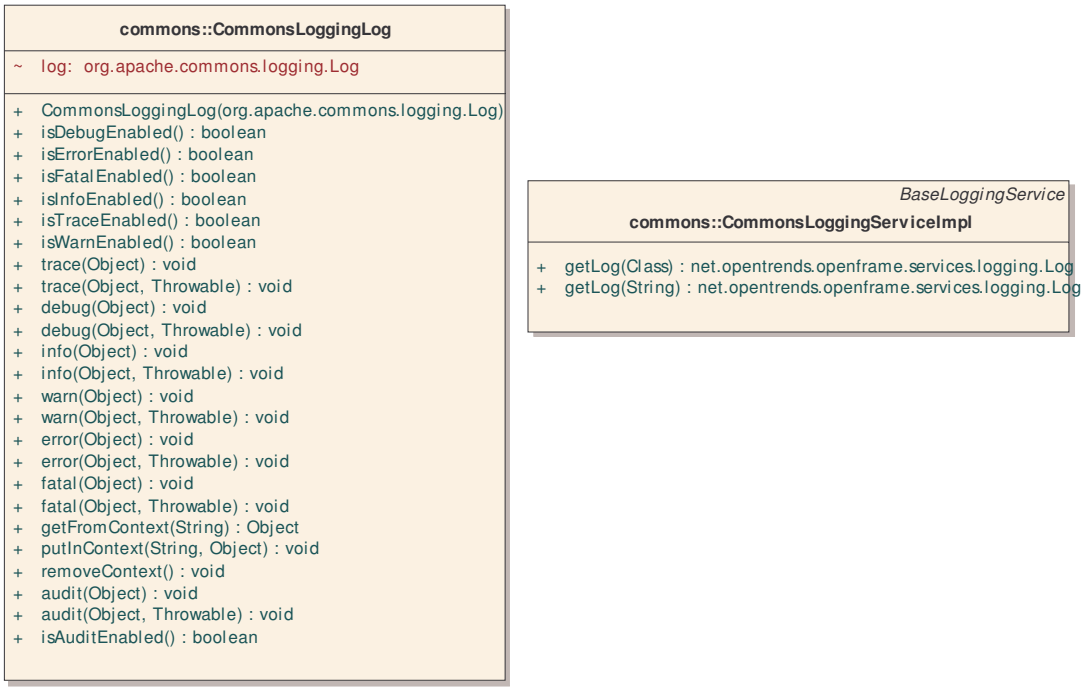
Component	Package	Descripció





Veure la documentació disponible al Javadoc per a més referència.

2.1.3. Components implementació de Commons Logging



Component	Package	Descripció
CommonsLoggingLog	net.opentrends.openframe.services.logging.commons	Implementació de la interfície 'Log' basada en Commons Logging
CommonsLoggingServiceImpl	net.opentrends.openframe.services.logging.commons	Implementació de la interfície 'LoggingService'

2.2. Instal·lació i Configuració

2.2.1. Instal·lació

La instal·lació del servei requereix de la utilització de la llibreria 'openFrame-services-logging' i les dependències indicades a l'apartat 'Introducció-Versions i Dependències'.

2.2.2. Configuració

La configuració del Servei de Traces implica 3 pasos:

1) Definir el configurador del servei per especificar en quin fitxer es defineixen les propietats del servei (sortides, nivells de traces, etc.)
2) Definir el servei i injectar-li el seu configurador
3) Definir les propietats del servei. En aquest pas es farà ús de la configuració concreta de la implementació escollida (log4j,...)

Definició del Configurador del Servei

```
<bean id="loggingConfigurator"
      ...>
```

La definició del servei requereix definir un configurador amb un identificador (es recomana usar 'loggingConfigurator')

Atributs:

Atributs	Requerit	Descripció
class	Sí	Implementació concreta del configurador a utilitzar Opcions: • net.opentrends.openframe.services.logging.log4j.xml.HostDOMConfigurator

- HostDOMConfigurator

  Propietats:

Propietat	Requerit	Descripció
configFileName	Sí	Ruta al fitxer Es pot utilitzar classpath:rutaFitxer, per fer la cerca a partir del classpath Exemple: classpath:log4j.xml

  Exemple:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
   <bean id="loggingConfigurator"
class="net.opentrends.openframe.services.logging.log4j.xml.HostDOMConfigurator"
init-method="init">
   <property name="configFileName"><value>classpath:log4j.xml</value></property>
   </bean>
   ...
</beans>
```

Definició del Servei

```
<bean id=”loggingService”
      ...>
```

La definició del servei requereix configurar un bean amb un identificador (es recomana usar 'loggingService') i els següents atributs:

Atributs:

Atribut	Requerit	Descripció
class	Sí	Implementació concreta del servei a utilitzar Opcions: • net.opentrends.openframe.services.logging.log4j.Log4J ServiceImpl. En aquest cas s'ha d'utilitzar com a configurador la classe 'net.opentrends.openframe.services.logging.log4j.xml. HostDOMConfigurator'
init-method	Sí	Usar 'init'. Mitjançant aquesta definició provoquem que es cridi al mètode 'init' una vegada construída la instància

També es poden configurar les següents propietats:

Propietat	Requerit	Descripció
configurator	Sí	Referència al configurador definit prèviament

Exemple:

```
...
<bean id="loggingService"
class="net.opentrends.openframe.services.logging.log4j.Log4JServiceImpl" init-
method="init">
   <!-- propietat que referència al configurador -->
   <property name="configurator"><ref local="loggingConfigurator"/>
   </property>
</bean>
<!-- bean del configurador -->
<bean id="loggingConfigurator"
class="net.opentrends.openframe.services.logging.log4j.xml.HostDOMConfigurator">
...
</bean>
...
```

Cal destacar que en l'exemple el configurador fa ús d'un localitzador de fitxers depenent del host (*HostDOMConfigurator*). Això vol dir que si en aquest configurador incloem una referència a un fitxer de propietats anomenat "log4j.xml", realment es buscarà un fitxer anomenat "log4j.xml.nom_del_host".

Definició de les Propietats del Servei per Log4J

Fitxer de configuració: **log4j.xml**
Característiques: S'ha d'ubicar en un directori del classpath.

En cas d'haver escollit com a implementació del servei de traces la classe ' net.opentrends.openframe.services.logging.log4j.Log4JServiceImpl', la definició del fitxer de propietats es basa en l'ús de Log4J.

En aquest apartat es mostren les configuracions més comuns de Log4J (mitjançant appenders) i algunes de més específiques que per la seva complexitat són necessàries d'explicar. En alguns casos, es tracten d'implementacions disponibles en la comunitat, en d'altres desenvolupats específicament per openFrame. Es recomana una lectura prèvia del manual de log4j per conéixer en detall la configuració de log4j.

1) Appenders

- Configuració de l'ús de Appender per Consola

  Classe: org.apache.log4j.ConsoleAppender

  S'afegeixen les traces traces al "System.out" o "System.err". Per a infornmació detallada sobre les propietats de l'appender consultar el manual de Log4J.

  Exemple:

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
   <layout class="org.apache.log4j.PatternLayout">
     <param name="ConversionPattern" value="openFrame Message: %-5p [%t] %c -
%m%n"/>
   </layout>
 </appender>
```

- Configuració de l'ús de Appender per Fitxer Rotatiu

  Classe: org.apache.log4j.DailyRollingFileAppender

  Permet establir un tamany màxim de fitxer i cada vegada que el fitxer de log supera aquest tamany es crea un fitxer de traces nou. Aquest procès es fa diariament o segons la freqüència especificada. Per a informació detallada sobre la forma d'especificar les diferents freqüències i propietats de l'appender consultar el manual de Log4J.

El format del fitxer (texte pla o XML) el defineix el *layout* associat a l'*appender* com veurem posteriorment.

- Configuració de l'ús de Appender per Correu Electrònic

  Classe: org.apache.log4j.SMTPAppender

Envia les traces per correu electrònic. Per a informació detallada sobre les propietats de l'appender consultar el manual de Log4J.

- Configuració de l'ús de Appender per SNMP

  Classe: org.apache.log4j.ext.SNMPTrapAppender

Envia les traces com a missatges TRAP pel protocol de control de xarxa SNMP a un administrador de xarxa. El manual de Log4J és molt suscint en definir el seu ús.Degut a la seva complexitat es fa esmena en aquest apartat de cóm utilitzar-lo.

Es poden especificar els següents paràmetres:

Paràmetre	Requerit	Descripció
ImplementationClassName	Sí	Implementació del protocol snmp. Podem escollir entre dos valors en funció de la versió del protocol que volguem fer servir: *org.apache.log4j.ext.WengsoftSNMPTrapSender* (v1,v2 ó v3) ó *org.apache.log4j.ext.JoeSNMPTrapSender* (v1 ó v2)
ManagementHost	Sí	Direcció IP del host a on està l'administrador de xarxa a on s'enviaran els missatges SNMP
ManagementHostTrapListenPort	Sí	Port a on està l'administrador de xarxa a on s'enviaran els missatges SNMP
EnterpriseOID	Sí	paràmetre propi del protocol SNMP
LocalIPAddress	Sí	IP des d'on s'envien els missatges
LocalTrapSendPort	Sí	Port des d'on s'envien els missatges
GenericTrapType	Sí	Paràmetre propi del protocol SNMP
SpecificTrapType	Sí	Paràmetre propi del protocol SNMP
CommunityString	Sí	Paràmetre propi del protocol SNMP
Threshold	Sí	Nivell més baix pel que es geneneraran els missatges TRAP

Exemple:

```
<appender name="snmp" class="org.apache.log4j.ext.SNMPTrapAppender">
   <param name="ImplementationClassName"
value="org.apache.log4j.ext.WengsoftSNMPTrapSender"/>
   <param name="ManagementHost" value="127.0.0.1"/>
```

```
    <param name="ManagementHostTrapListenPort" value="8001"/>
    <param name="EnterpriseOID" value="1.3.6.1.4.1.24.0"/>
    <param name="LocalIPAddress" value="127.0.0.1"/>
    <param name="LocalTrapSendPort" value="161"/>
    <param name="GenericTrapType" value="6"/>
    <param name="SpecificTrapType" value="12345678"/>
    <param name="CommunityString" value="public"/>
    <param name="ForwardStackTraceWithTrap" value="true"/>
    <param name="Threshold" value="DEBUG"/>
    <param name="ApplicationTrapOID" value="1.3.6.1.4.1.24.12.10.22.64"/>
    <layout class="org.apache.log4j.ext.SnmpDelimitedConversionPatternLayout">
           <param name="ValuePairDelim" value="/"/>
           <param name="VarDelim" value=";"/>
           <param name="ConversionPattern" value="%-
5p;1.3.6.1.4.1.24.100.1/%t;1.3.6.1.4.1.24.100.2/%c;1.3.6.1.4.1.24.100.3/%m;1.3.6.1
.4.1.24.100.4" />
    </layout>
   </appender>
```

- Configuració de l'ús de Appender per Base de Dades

  Classe: org.apache.log4j.jdbcplus.JDBCAppender

  Instal·lació: Per a poder fer ús d'aquest appender cal descarregar una llibreria, disponible a la url 'http://www.dankomannhaupt.de/projects/index.html'

  Es poden definir els paràmetres:

Paràmetre	Requerit	Descripció
url	Sí	Url de connexió a la base de dades
username	Sí	Nom usuari connexió a la base de dades
password	Sí	Password connexió a la base de dades
connector	Sí	Connector a la base de dades Els valors disponibles són: • org.apache.log4j.jdbcplus.examples.MySqlConnectionHandler • org.apache.log4j.jdbcplus.examples.OracleConnectionHandler En cas de que no es disposi de connector per la base de dades seleccionada es pot realitzar una implementació de la interfície '*org.apache.log4j.jdbcplus.JDBCConnectionHandler*'
sqlHandler	Sí	Adaptador que proporciona la cadena per fer la inserció d'un registre. Usar: ' org.apache.log4j.jdbcplus.examples.SqlHandler

Paràmetre	Requerit	Descripció
		'
sql	Sí	Patró d'inserció dels valors en la base de dades (Veure patrons a continuació)
table	Sí	Taula en la que s'emmagtzemaran les traces
column	Sí	Crear un paràmetre amb nom 'column' per cada columna a inserir. Indicar com a valor: nom_de_columna~patró~valor On patró correspon a un dels possibles patrons de la sql
buffer	No	Nombre d'insercions a la base de dades que es fan a la vegada Valor per defecte:1
commit	No	Especificar si es vol autocommit en cada inserció. Valor per defecte:true
dbclass	Sí	Driver utilitzat (específic de la BD)
quoteReplace	No	Indicar 'true' si es volen reemplazar les cometes simples (') per poder inserir a la base de dades
layoutPartsDelimiter	No	Delimitador dels patrons de layout. És la cadena que utilitzarem per separar els diferents patrons definits dins del tag ConversionPattern del tag Layout. Podrem escollir el patró que volem usar en cada columna indicant @LAYOUT:num_del_patro@ Ex: <param name="ConversionPattern" value="patro1#-#patro2#-#patro3"/> On layoutPartsDelimiter="#-#"

Patrons del paràmetre sql:

Patró	Descripció
@INC@	Comptador incremental per cada traça
@PRIO@	Prioritat de la traça (DEBUG, INFO,...)
@ID@	Classe que implementa JDBCIDHandler i que proporciona el valor per aquesta columna.
@IPRIO@	**Prioritat de la traça en format numèric**
@DYNAMIC@	Valor dinàmic que proporciona una classe que implementa JDBCColumnHandler

Patró	Descripció
@STATIC@	Valor estàtic
@CAT@	Nom de la categoria
@THREAD@	Nom del thread
@MSG@	Missatge de la traça sense format
@LAYOUT@	Missatge de la traça formatejat amb el patró especificat en el param ConversionPattern.
@LAYOUT:num_patró@	Missatge de la traça formatejat amb el patró especificat en el param ConversionPattern, en la posició indicada per num_patró
@TIMESTAMP@	Moment en que es llença de la traça
@THROWABLE@	Error associat a la traça
@NDC@	Informació contextual de la traça *(nested diagnostic context)*
@MDC:key@	Informació contextual de la traça sota una clau predeterminada *(mapped diagnostic context)*

Exemple:

```
  <appender name="JDBCPooled" class="org.apache.log4j.jdbcplus.JDBCAppender">
   <param name="url" value="jdbc:mysql://localhost/test" />
   <param name="username" value="eulo" />
   <param name="password" value="eulo" />
   <param name="sql" value="INSERT INTO TEST (prio, cat, thread, msg, layout_msg,
throwable, ndc, mdc, mdc2, info, addon, created_by) VALUES ('@PRIO@', '@CAT@',
'@THREAD@', '@MSG@', '@LAYOUT:1@', '@THROWABLE@', '@NDC@', '@MDC:MyMDC@',
'@MDC:MyMDC2@', 'info timestamp: @TIMESTAMP@', 'addon', 'me')"/>
   <param name="buffer" value="1"/>
   <param name="commit" value="true"/>
   <param name="dbclass" value="com.mysql.jdbc.Driver"/>
   <param name="quoteReplace" value="true"/>
   <param name="throwableMaxChars" value="3000"/>
   <param name="layoutPartsDelimiter" value="#-#"/>
   <layout class="org.apache.log4j.PatternLayout">
          <param name="ConversionPattern" value="[%t] %m#-##-#%d{dd.MM.yyyy}#-
#%d{HH:mm:ss}"/>
   </layout>
  </appender>
```

També podem indicar el nom de les columnes i el seu valor amb param name=”column” i posant a value:

nom_de_columna~patró~valor

On patró correspon a un dels possibles patrons.
D’aquesta manera no fa falta el paràmetre sql.

```
```

```
<param name="table" value="logtest" />
<param name="column" value="id~ID~org.apache.log4j.jdbcplus.examples.MyIDHandler"
/>
<param name="column" value="prio~PRIO" />
<param name="column" value="cat~CAT" />
<param name="column" value="thread~THREAD" />
<param name="column" value="msg~MSG" />
<param name="column" value="layout_msg~LAYOUT" />
<param name="column"
value="info~DYNAMIC~org.apache.log4j.jdbcplus.examples.MyColumnHandler" />
<param name="column" value="mdc~MDC~MyMDC" />
<param name="column" value="created_on~TIMESTAMP" />
<param name="column" value="created_by~STATIC~me" />
<param name="usePreparedStatements" value="true"/>


<param name="layoutPartsDelimiter" value="#-#"/>
<!-- layout with conversion pattern -->
<layout class="org.apache.log4j.PatternLayout">
   <!-- conversion pattern with 4 parts separated by #-#, second part is empty -->
<param name="ConversionPattern" value="[%t] %m#-##-#%d{dd.MM.yyyy}#-
#%d{HH:mm:ss}"/>
</layout>
```

2) Layouts

- PatternLayout

  Classe: org.apache.log4j.PatternLayout

  Permet especificar la sortida amb patrons de conversió. El seu ús és equivalent a la funció printf del llenguatje C. Alguns dels caràcters de conversió més interessants són (per a més referència consultar la API de la classe):

Caràcter	Descripció
%c	Nom del "logger"
%C	Mostrar el nom qualificat de la classe que ha llençat el missatge.
%d	Mostrar la data en la que es va produir el missatge. A continuació opcionalment pot aparèixer el patró de conversió de data. Si no s'especifica s'utilitza per defecte **ISO 8601.** Exemple: %d{dd MMM yyyy HH:mm:ss,SSS}.
%L	Número de línia a on es va llençar el missatge (**no fer servir si es necessita un bon rendiment**).
%m	Missatge
%p	Prioritat/nivell del missatge (WARN,...).
%X{clau}	Informació contextual "clau"
%n	Separador de línia (un \n).

El resultat de l'aplicació d'aquests patrons són missatges de texte pla uniformement formatejats.

- XMLLayout

Classe: org.apache.log4j.XMLLayout

La sortida d'aquest "layout" consisteix en una sèrie de "log4j:event" nodes XML. Aquest "layout" no deixa un fitxer XML ben format, sinó que està pensat per a ser inclòs com a entitat externa a un altre fitxer per a formar un fitxer XML correcte.
Per exemple, si `abc` és el nom del fitxer a on van les traces, el fitxer XML correcte seria:

```
<?xml version=1.0"?>
<!DOCTYPE log4j:eventSet
   <!ENTITY data SYSTEM "file:///abc">
]>
<log4j:eventSet version="1.2" xmlns:log4j="http://jakarta.apache.org/log4j/">
   &data;
</log4j:eventSet>
```

L'atribut *version* de l'element *eventSet* indica la versió de Log4J que es fa servir. 1.1 indica una versió de Log4J anterior a la 1.2 i un valor per l'atribut 1.2 indica una versió 1.2 o posterior.
Cal tenir en compte que aquest "layout" no permet qualsevol estructura de XML que podem pensar, sinó que segueix l'especificació de "log4j.dtd" que defineix quatre atributs per cada node XML: *logger, timestamp, level i thread.*

```
<log4j:event
logger="net.opentrends.openframe.services.logging.snmp.test.TrapReceiver"
timestamp="1127743279027" level="DEBUG" thread="main">
   <log4j:message><![CDATA[missatge...]]></log4j:message>
</log4j:event>
```

Qualsevol informació de tipus contextual (com ara l'userId) s'ha d'incloure dins la informació contextual del log i automàticament s'inclourà en la traça:

```
<log4j:event logger="net.opentrends.openframe.services.logging.test.LoggingClient"
timestamp="1127746730256" level="DEBUG" thread="main">
   <log4j:message><![CDATA[Log property file: log4j-test.xml.es-
c7pym1j]]></log4j:message>
   <log4j:NDC><![CDATA[userId: me]]></log4j:NDC>
</log4j:event>
```

- PatternXMLLayout

Classe: net.opentrends.openframe.services.logging.xml.PatternXMLLayout

La sortida d'aquest "layout" és una combinació de PatternLayout i XMLLayout.
A l'igual que l'XMLLayout, aquest "layout" no deixa un fitxer XML ben format, si no que també està pensat per a ser inclòs com a entitat externa a un altre fitxer per a formar un fitxer XML correcte.
El fitxer xml resultant tindrà una estructura semblant a XMLLayout, amb la diferència que el atributs no seran els de l'XMLLayout(*loggin, timestamp, level i thread*), sinó que els podrem definir nosaltres mateixos, seguint els patrons de PatternLayout.

Es permet definir els següents paràmetres:

Paràmetre	Requerit	Descripció
NameSpace	Sí	Indica el namespace que l'xml mostrarà en els tags (<name_space:event>). Si no es defineix, mostrarà únicament el nom del tag (<event>)
NodeName	Sí	Indica el nom del node principal(<registre>). Per defecte és "event" (<event>)
AttrValuePairDelim	Sí	Indica quin caràcter servei de separador entre un atribut i el seu valor (usar '=' per defecte).
AttrVarDelim	Sí	Indica quin caràcter serveix de separador entre els diferents atributs
AttrConversionPattern	Sí	Defineix els diferents atributs dels events del log. El valor dels atributs pot seguir els mateixos patrons que en ***PatternLayout.***
NodesValuePairDelim	Sí	Indica quin caràcter serveix de separador entre els noms dels nodes i el seu valor
NodesVarDelim	Sí	Indica quin caràcter serveix de separador entre els diferents nodes
NodesConversionPattern	Sí	Defineix els diferents nodes dels events del log. El valor dels nodes també pot seguir els mateixos patrons que en PatternLayout.
Throwable	Sí	Indica que es mostraran les excepcions llançades. A value definirem el nom del tag que inclourà l'excepció.
ExceptionMaxLength	No	Indica el nombre màxim de caràcters que mostrarem de l'excepció. Per defecte són 256 caràcters.

Exemple:

```
<appender name="file" class="org.apache.log4j.RollingFileAppender">
```

```
    <param name="File" value="D:/logs/logging4.log.xml"/>
    <param name="Append" value="true"/>
    <param name="MaxFileSize" value="50KB"/>
    <param name="maxBackupIndex" value="3"/>
    <layout
class="net.opentrends.openframe.services.logging.xml.PatternXMLLayout">
          <param name="NameSpace" value="log4j"/>
          <param name="NodeName" value="registre"/>

          <param name="AttrValuePairDelim" value="="/>
          <param name="AttrVarDelim" value=";"/>
          <param name="AttrConversionPattern" value="attr1=%-5p;attr2=%t;attr3=%c"
/>

          <param name="NodesValuePairDelim" value="="/>
          <param name="NodesVarDelim" value=";"/>
          <param name="NodesConversionPattern" value="data=%d{yyyy MM dd
HH:mm:ss};errorType=0;aplicacio=%c;logger=%F;nivell=%p;classe=%c;metode=metode;pid
=%t;missatge=%m;user_id=%X{userId};pagOrigen=%X{pagOrigen};" />

          <param name="Throwable" value="textException"/>
          <param name="ExceptionMaxLength" value="256"/>

    </layout>
  </appender>
```

Un exemple de sortida amb aquesta configuració és:

```
<log4j:registre
   attr1="ERROR"
   attr2="SNMP Server"
attr3="net.opentrends.openframe.services.logging.snmp.test.TestListener">

   <log4j:data>2005 10 13 13:29:21</log4j:data>
   <log4j:errorType>0</log4j:errorType>
   <log4j:aplicacio>net.opentrends.openframe.services.logging.snmp.test.TestListen
er</log4j:aplicacio>
   <log4j:logger>Log4JLog.java</log4j:logger>
   <log4j:nivell>ERROR</log4j:nivell>
   <log4j:classe>net.opentrends.openframe.services.logging.snmp.test.TestListener<
/log4j:classe>
   <log4j:metode>metode</log4j:metode>
   <log4j:pid>SNMP Server</log4j:pid>
   <log4j:textException><![CDATA[java.lang.Exception: Exception Message
          at
net.opentrends.openframe.services.logging.snmp.test.TestListener.snmpRequest(TestL
istener.java:38)
          at
ca.wengsoft.snmp.Core.SnmpDispatcher.notifyEvent(SnmpDispatcher.java:164)
          at ca.wengsoft.snmp.Core....]]>
   </log4j:textException>

</log4j:registre>
```

En cas que no s'hagués defint el paràmetre ***NameSpace*** obtindrem aquest altre format:

```
<registre
   attr1="ERROR"
   attr2="SNMP Server"
   attr3="net.opentrends.openframe.services.logging.snmp.test.TestListener">

   <data>2005 10 13 13:30:40</data>
   <errorType>0</errorType>
   <aplicacio>net.opentrends.openframe.services.logging.snmp.test.TestListener</ap
licacio>
   <logger>Log4JLog.java</logger>
   <nivell>ERROR</nivell>
   <classe>net.opentrends.openframe.services.logging.snmp.test.TestListener</class
e>
   <metode>metode</metode>
   <pid>SNMP Server</pid>
   <textException><![CDATA[java.lang.Exception: Exception Message
          at
net.opentrends.openframe.services.logging.snmp.test.TestListener.snmpRequest(TestL
istener.java:38)
          at
ca.wengsoft.snmp.Core.SnmpDispatcher.notifyEvent(SnmpDispatcher.java:164)
          at ca.wengsoft.snmp.Core....]]>
   </textException>

</registre>
```

Incorporació de valors comuns a l'aplicació

Utilitzant aquest layout podem crear logs amb valors provinents de variables ja especificades en qualsevol punt de l'aplicació.

Exemple: Imaginem que el filtre d'autentificació obté l'usuari i volem mostrar l'usuari connectat a les nostres traces.

Una possible opció és que la classe que generés la traça accedís prèviament a l'obtenció de l'usuari connectat (a la sessió, cridant a la classe d'autentificació, etc.) i afegís aquesta informació a la traça. Aquesta opció és costosa i poc mantenible. Si volem mostrar a totes les traces de la nostra aplicació l'usuari connectat haurem de realitzar aquest procediment. I si després ens demanen que s'afegeixi altra informació contextual a totes les traces de l'aplicació?

Per evitar aquestes problemàtiques openFrame proporciona la classe ' net.opentrends.openframe.core.threadlocal.ThreadLocalProperties'.

```
ThreadLocalProperties.put("userId","Usuari1");
```

Una vegada introduit en qualsevol punt de l'aplicació aquest atribut, podem configurar al layout que es mostri aquesta informació utilitzant el caràcter '$' seguit del nom del atribut.

Exemple:

```
<param name="NodesConversionPattern" value="data=%d{yyyy MM dd
HH:mm:ss};user_id=$userId;" />
```

Aquest paràmetre crearia un tag amb un node '***data***' obtingut amb el patró de conversió indicat i un tag '***user_id***' amb el valor del paràmetre *userId* obtingut de l'objecte 'ThreadLocalProperties':

```
<log4j:registre>
   <log4j:data>2005 10 19 11:29:40</log4j:data>
   <log4j:user_id>Usuari1</log4j:user_id>
</log4j:registre>
```

Incorporació de missatges estructurats a la traça

Per lo general els missatges de traces són cadenes. Si volem generar informació adicional estructurada farem ús de l'objecte BaseLoggingObject per afegir la informació (veure apartat 'Utilització - Generar logs amb paràmetres adicionals a un missatge').

Per mostrar els atributs de l'objecte BaseLoggingObject s'usarà la sintaxi '**@**' seguida del nom del paràmetre.

```
<param name="NodesConversionPattern" value="idTramit=@idTramit;" />
```

2.3. Utilització del Servei

2.3.1. Generar Missatges

Tal i com s'ha comentat a l'apartat 'Arquitectura i Components' les classes principals per a la generació de les traces es troben al package 'net.opentrends.openframe.services.logging'.

Per generar un missatge seguirem un patró com el mostrat en el següent exemple:

```
if(this.loggingService!=null){
    String missatge = "Missatge";
     Log log = this.loggingService.getLog(this.getClass());
    log.info(missatge);
}
```

Realitzarem doncs els següents pasos:

1) En primer lloc comprovarem que s'ha realitzat correctament la injecció a la nostra classe del servei de logging
2) Construir el missatge que generarem a la sortida

   Aquest missatge és per lo general un String, però també es permet l'ús de la classe 'BaseLoggingObject' per generar un missatge estructurat (veure apartat 'Generar missatges estructurats').

3) Obtenir la interfície 'Log' configurada pel servei de logging mitjançant la crida 'loggingService.getLog(nomCategoria)'

   En el cas de fer ús de la implementació de log4j, el nom de la categoria ha de correspondre al nom especificat en l'atribut 'name' del tag 'category' del fitxer de propietats de log4j.

```
<category name="net.opentrends">
    <appender-ref ref="console"/>
    <appender-ref ref="file"/>
</category>
```

4) Per últim, farem ús dels mètodes que indiquen per quin nivell es genera la traça:

   log.info(), log.debug(),....

2.3.2. Evitar càlculs innecessàris

Una de les qüestions més importants del sistema de traces és el cost de computació. Si en un moment donat es desactiven determinats nivells de traces, tot i que aquesta traça no es generi a la sortida el missatge és avaluat (ja que es troba com un paràmetre de la crida). Si l'avaluació del missatge comporta càlculs estem afegint un cost innecessari.

Per evitar afegir costos adicionals de cómput innecessari es recomana prèviament comprovar que es troba activat el nivell pel qual generarem el missatge. Així, per exemple, enlloc de:

```
log.debug("....");
```

Haurem sempre de realitzar:

```
if (log.isDebugEnabled())
```

```
log.debug("....");
```

2.3.3. Generar missatges estructurats

Per lo general les traces enviades són missatges purs (cadenes de tipus String). En cas de que volguem generar informació estructurada del missatge es pot fer ús de l'objecte 'net.opentrends.openframe.services.logging.log4j.BaseLoggingObject' com si del missatge es tractés.

La utilització d'aquesta classe és especialment útil en l'ús del layout 'PatternXMLLayout', el qual ens permet generar tags específics per estructurar la nostra informació.

Per a utilitzar-ho passaram a cadascun dels mètodes de traça l'objecte creat de la classe 'BaseLoggingObject'. Aquesta classe permet 2 constructors:

- Constructor basat en un array de Objects
- Constructor basat en una Hashtable

Exemple:

```
        log.debug(new BaseLoggingObject(new
Object[]{"idTramit","34","missatge","missatge del log"}));
```

O bé:

```
        Hashtable params = new Hashtable();
        params.put("idTramit","34");
        params.put("missatge","missatge del log");

        log.debug(new BaseLoggingObject(params));
```

Veure la secció 'PatterXMLLayout - Configuració Incorporació de missatges estructurats a la traça' per configurar que se'ns generi la informació introduïda a l'objecte BaseLoggingObject.

2.3.4. Generar Informació Contextual

Moltes vegades no és suficient amb la informació estàndard que es mostra a les traces i és necessari complementar-la amb informació contextual cóm: usuari que fa la petició, temps d'inici i final, etc. Per afegir aquesta informació, és necessari inicialitzar-la en algun moment de la petició per què estigui disponible durant tot el processat de la mateixa. Per tant, el patró de treball a seguir és:

- Inicialització de l'informació adicional abans de processar la petició
- Processat de la petició (utilització de l'informació adicional en el servei de traces)
- Finalització de la petició

openFrame suporta la inserció d'aquesta informació mitjançant el mètode `public void putInContext(String key, Object object)` de la interfície "net.opentrends.openframe.services.logging.Log". Cal remarcar que el contexte manegat és una solució basada en "contexte per fil" (*context per thread*), per tant al finalitzar la petició el programador s'ha de preocupar de buidar el contexte del "thread" amb el mètode `public void removeContext()`

Una possible solució per seguir el patró de treball abans esmentat, és fer servir els filtres de l'especificació de l'API Servlet. Així, podríem tenir un filtre que ens insertès, per exemple, el nom d'usuari, tal i com es mostra a continuació:

```
...
public class RequestLoggingFilter implements Filter {
...
    private LoggingService loggingService = null;
    public void init(FilterConfig filterConfig) throws ServletException
{
  // retrieves the loggingService
     ...
    }

    public void doFilter(ServletRequest arg0, ServletResponse arg1,
FilterChain chain) throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) arg0;
        HttpServletResponse response = (HttpServletResponse) arg1;


         String userLogin =
(String)request.getSession().getAttribute("userLogin");
         if (userLogin!=null) {

this.loggingService.getLog(this.getClass()).putInContext("userLoginId",u
serLoginId);
         }


         String remoteHost = request.getRemoteHost();
         String remoteAddr = request.getRemoteAddr();


  this.loggingService.getLog(this.getClass()).putInContext("remoteHost",
remoteHost);

  this.loggingService.getLog(this.getClass()).putInContext("remoteAddr",
remoteAddr);

     try {
     chain.doFilter(arg0,arg1);
     }
  finally {
```

```
        this.loggingService.getLog(this.getClass()).removeContext();
    }
      }
...
}
```

2.3.5. Generar nous nivells de traces

El servei de traces afegeix varis nivells de traces (debug, info, warn, error, fatal y audit). En cas de voler afegir un nou nivell es pot seguir el següent procediment:

1) Extendre la interfície 'net.opentrends.openframe.services.logging.Log' amb una nova interfície
2) Definir a la nova interfície els mètodes de traces segons el nou nivell

   Exemple :

```
package net.opentrends.openframe.services.logging;
...
public interface Log {
...
    public void audit(java.lang.Object message);
    public void audit(java.lang.Object message, java.lang.Throwable);
    public boolean isAuditEnabled();
...
}
```

3) Extendre la classe específica de cadascuna de les implementacions de la interfície base 'Log' i incorporar el tractament del nou nivell.

```
package net.opentrends.openframe.services.logging.log4j;
...
public class Log4JLog implements Log {
...
    public void audit(Object aMessage) {
        logger.log(AuditLevel.AUDIT,aMessage);

    }
    public void audit(Object aMessage, Throwable aThrowable) {
        logger.log(AuditLevel.AUDIT,aMessage, aThrowable);
    }
    public boolean isAuditEnabled() {
        return logger.isEnabledFor(AuditLevel.AUDIT);
    }
...
```

```
}
```

4) Crear el nou nivell

```
package net.opentrends.openframe.services.logging.log4j;
...
public class AuditLevel extends Level {
    final static public int AUDIT_INT = Level.INFO_INT + 1;
    final static public Level AUDIT = new AuditLevel(AUDIT_INT, "AUDIT",
8);
    public AuditLevel(int arg0, String arg1, int arg2) {
        super(arg0, arg1, arg2);
    }
    public static Level toLevel(String sArg) {
        return (Level) toLevel(sArg, AuditLevel.AUDIT);
    }

    public static Level toLevel(int val) {
        return (Level) toLevel(val, AuditLevel.AUDIT);
    }
}
```

NOTA: En aquest exemple es mostra un exemple d'aplicació que ja es troba incorporar a openFrame

Per últim, la referència al nou nivell des del fitxer de propietats es farà mitjançat l'ús del tag '<level>' indicant com a value el nou nivell:

```
...
<category name="net.opentrends.openframe.services.logging">
    <level value="audit"
class="net.opentrends.openframe.services.logging.log4j.AuditLevel"/>
    <appender-ref ref="file"/>
</category>
...
```

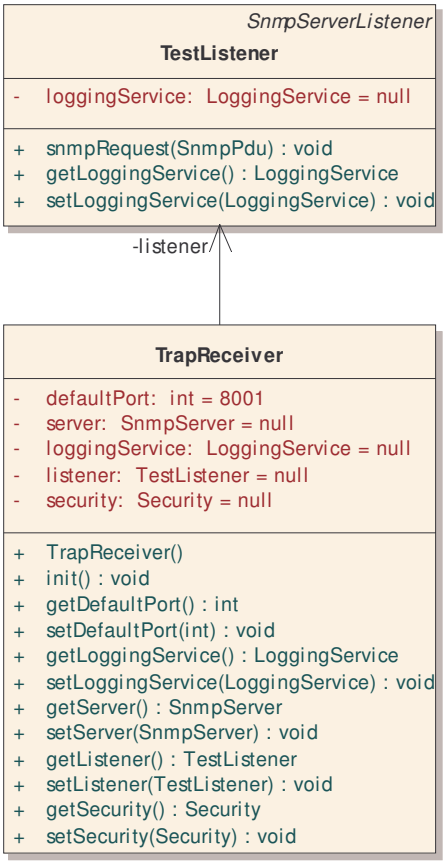
2.4. Eines de Suport

2.4.1. Servidor de proves SNMP

En el cas de voler instal·lar un servidor SNMP local per a realitzar proves amb SNMP, openFrame proporciona unes classes de test per a fer ús d'una implementació opensource de servidor.
Realitzar els següents pasos:

1) Descarregar el fitxer comprimit de l'última versió des de la pàgina
http://www.m2technologies.net/asp/snmpTrapAppender.asp

2) Del fitxer descarregat incorporar a la llibreria del projecte el fitxer inclós



'snmpTrapAppender_X_X_X.jar' (on X_X_X correspon a la versió del fitxer)

3) Definir la classe listener que s'encarregarà d'analitzar els missatges

Classe:'net.opentrends.openframe.services.logging.snmp.test.TestListener'

Definir les següents propietats:

Propietat	Requerit	Descripció
loggingService	Sí	Incorporar una referència al servei de traces

Exemple:

```
<bean id="listener"
class="net.opentrends.openframe.services.logging.snmp.test.TestListener">
   <property name="loggingService"><ref   local="loggingService"/></property>

   </bean>
```

4) Definir un bean de seguretat.

Classe: 'net.opentrends.openframe.services.logging.snmp.test.Security'

Definir les següents propietats:

Propietat	Requerit	Descripció
loggingService	Sí	Incorporar referència al servei de traces

Exemple:

```
<bean id="security"
class="net.opentrends.openframe.services.logging.snmp.test.Security">
          <property name="loggingService"><ref
local="loggingService"/></property>
   </bean>
```

5) Per últim definir el servei que rebrà els missatges enviats per l'appender configurat

Classe: 'net.opentrends.openframe.services.logging.snmp.test.TrapReceiver'
Indicar adicionalment:

Atribut	Requerit	Descripció
init-method	Sí	Indicar 'init' per tal que s'inicialitzi yb servidor amb un listener.

Com a propietats configurarem:

Propietat	Requerit	Descripció
loggingService	Sí	Incorporar referència al servei de traces
listener	Sí	Referència al listener configurat al pas 3
security	Sí	Referència a l'objecte security configurat al pas 4
defaultPort	Sí	Indicar el port especificat al paràmetre 'ManagementHostTrapListenPort' del appender (veure 'Configuració de l'ús de Appender per SNMP')

Exemple:

```
<bean id="trapReceiver"
class="net.opentrends.openframe.services.logging.snmp.test.TrapReceiver" init-
method="init">
   <property name="loggingService"><ref local="loggingService"/></property>
   <property name="listener"><ref local="listener"/></property>
   <property name="security"><ref local="security"/></property>
   <property name="defaultPort"><value>8001</value></property>
</bean>
```

2.4.2. Visualització de Logs

Per visualitzar els logs provinents de la nostra aplicació podem utilitzar una eina anomenada **chainsaw**, que trobem dins del paquet de **log4j** (*org.apache.log4j.chainsaw*). Aquesta eina ens permet visualitzar tant els events generats per l'aplicació com els provinents d'un fitxer xml generat per *XMLLayout*.

Per visualitzar un fitxer xml només caldrà executar l'eina **chainsaw** i obrir el fitxer. Aquest fitxer no ha d'ésser un fitxer xml ben format. Ha de tenir l'estructura que presenta quan és generat per XMLLayout, és a dir, només haurà de tenir els nodes dels events, sense la capçalera xml, ja que chainsaw ja la introdueix abans de parsejar el fitxer.

Així, aquest fitxer no podrà ser visualitzat pel chainsaw:

```
<?xml version=1.0”?>
<!DOCTYPE log4j:eventSet
   <!ENTITY data SYSTEM “file:///abc”>
]>
<log4j:eventSet version=”1.2” xmlns:log4j=”http://jakarta.apache.org/log4j/”>

   <log4j:event
logger="net.opentrends.openframe.services.logging.test.LoggingClient"
timestamp="1127746730256" level="DEBUG" thread="main">
          <log4j:message><![CDATA[Log property file: log4j-test.xml.es-
c7pym1j]]></log4j:message>
          <log4j:NDC><![CDATA[userId: me]]></log4j:NDC>
   </log4j:event>

</log4j:eventSet>
```

Però sí podrà visualitzar fitxers amb aquest format:

```
   <log4j:event
logger="net.opentrends.openframe.services.logging.test.LoggingClient"
timestamp="1127746730256" level="DEBUG" thread="main">
          <log4j:message><![CDATA[Log property file: log4j-test.xml.es-
c7pym1j]]></log4j:message>
          <log4j:NDC><![CDATA[userId: me]]></log4j:NDC>
   </log4j:event>
```

Per visualitzar els events provinents d'una aplicació haurem de configurar l'arxiu **log4j.xml.nom_del_host** de la següent manera:

```
<appender name="chainsaw"          class="org.apache.log4j.net.SocketAppender">
        <param name="remoteHost" value="localhost"/>
        <param name="port" value="4445"/>
        <param name="locationInfo" value="true"/>
</appender>
```

Aquí creem un appender amb el nom **chainsaw** i amb els següents paràmetres:

remoteHost = nom del host on executem el chainsaw
port = port per on establim la connexió
localinfo = true

Seguidament hem d'especificar quines classes enviaran els seus events cap a **chainsaw**, com fem amb els altres appenders:

```
<category name="net.opentrends.openframe.services">
  <appender-ref ref="chainsaw"/>
  <appender-ref ref="console"/>
  <appender-ref ref="file"/>
  <appender-ref ref="snmp"/>
</category>
```

Un cop configurat el fitxer tots els events que produeixin aquestes classes seran automàticament enviats cap al **chainsaw**.

Per visualitzar fitxers xml que no segueixin l'estructura dels que genera *XMLLayout*, com els de *PatternLayout* o *PatternXMLLayout*, no podran ser visualitzats pel **chainsaw**. Haurem d'utilitzar editors de text o bé d'xml, pel cas dels generats per *PatternXMLLayout*.

2.5. Integració amb Altres Serveis

2.6. Preguntes Freqüènts

3. Exemples

3.1. Tests Unitaris

Un exemple d'utilització del servei de traces són els tests unitaris, a on s'obté el bean del servei a partir del fitxer de definició (applicationContext.xml) i s'envia un log a les sortides especificades.

```
package net.opentrends.openframe.services.mail.test;

...

public class LoggingServiceTest extends TestCase {
...
    public void testLogging(){
   /**
    * Obtenim les definicions dels beans
    */
        ClassPathXmlApplicationContext appContext = new
ClassPathXmlApplicationContext("applicationContext.xml");
   /**
    * Obtenim el servei de logs
    */
        BeanFactory factory = (BeanFactory) appContext;
   /**
    * Obtenim un client de test per llençar la traça
    */
        LoggingService service=(LoggingService)factory.getBean("loggingService");
        FileSystemResource resource = new FileSystemResource(
client.getLoggingService().getConfigurator().getConfigFileName() );
        assertTrue("Log property file doesn't exist:
"+resource.getFilename(),resource.exists());
        if(resource.exists()){

   /**
    * Llencem la traça
    */
          Log log = service.getLog(this.getClass());
          if(log.isDebugEnabled())
                log.debug("Log property file: "+resource.getFilename());
        }
    }
...
}
```

La utilització del servei es independent de la configuració del mateix. Així, un possible fitxer de configuració del servei seria:

```
<?xml version="1.0"?>
<!DOCTYPE log4j:configuration PUBLIC "-//APACHE//DTD LOG4J//EN"
"http://logging.apache.org/log4j/docs/api/org/apache/log4j/xml/log4j.dtd">

<log4j:configuration>
   <appender name="JDBCPooled" class="org.apache.log4j.jdbcplus.JDBCAppender">
```

```
          <param name="connector"
value="org.apache.log4j.jdbcplus.examples.FirebirdPoolConnectionHandler"/>
          <param name="sql" value="INSERT INTO LOGTEST (id, prio, cat, thread,
msg, layout_msg, throwable, ndc, mdc, mdc2, info, addon, the_date, the_time,
the_timestamp, created_by) VALUES (@INC@, '@PRIO@', '@CAT@', '@THREAD@', '@MSG@',
'@LAYOUT:1@', '@THROWABLE@', '@NDC@', '@MDC:MyMDC@', '@MDC:MyMDC2@', 'info
timestamp: @TIMESTAMP@', 'addon', cast ('@LAYOUT:3@' as date), cast ('@LAYOUT:4@'
as time), cast ('@LAYOUT:3@ @LAYOUT:4@' as timestamp), 'me')"/>
          <param name="buffer" value="1"/>
          <param name="commit" value="true"/>
          <param name="dbclass" value="org.firebirdsql.jdbc.FBDriver"/>
          <param name="quoteReplace" value="true"/>
          <param name="throwableMaxChars" value="3000"/>
          <param name="layoutPartsDelimiter" value="#-#"/>
          <layout class="org.apache.log4j.PatternLayout">
                  <param name="ConversionPattern" value="[%t] %m#-##-
#%d{dd.MM.yyyy}#-#%d{HH:mm:ss}"/>
          </layout>
   </appender>
   <appender name="SYSTEMOUT" class="org.apache.log4j.ConsoleAppender">
          <param name="target" value="System.out"/>
          <layout class="org.apache.log4j.PatternLayout">
                  <param name="ConversionPattern" value=" %d{dd.MM.yyyy HH:mm:ss}
- %m\n"/>
          </layout>
   </appender>
   <appender name="SNMP" class="org.apache.log4j.ext.SNMPTrapAppender">
          <param name="ImplementationClassName"
value="org.apache.log4j.ext.WengsoftSNMPTrapSender"/>
          <param name="ManagementHost" value="127.0.0.1"/>
          <param name="ManagementHostTrapListenPort" value="8001"/>
          <param name="EnterpriseOID" value="1.3.6.1.4.1.24.0"/>
          <param name="LocalIPAddress" value="127.0.0.1"/>
          <param name="LocalTrapSendPort" value="161"/>
          <param name="GenericTrapType" value="6"/>
          <param name="SpecificTrapType" value="12345678"/>
          <param name="CommunityString" value="public"/>
          <param name="Threshold" value="DEBUG"/>
          <layout
class="org.apache.log4j.ext.SnmpDelimitedConversionPatternLayout">
                  <param name="ValuePairDelim" value="/"/>
                  <param name="VarDelim" value=";"/>
                  <param name="ConversionPattern" value="%-
5p;1.3.6.1.4.1.24.100.1/%t;1.3.6.1.4.1.24.100.2/%c;1.3.6.1.4.1.24.100.3/%m;1.3.6.1
.4.1.24.100.4" />
          </layout>
   </appender>
   <appender name="MAIL" class="org.apache.log4j.net.SMTPAppender">
          <param name="threshold" value="ERROR"/>
          <param name="SMTPHost" value="smtp.opentrends.net"/>
          <param name="subject" value="Error!"/>
          <param name="to"
value="xavi.xicota@opentrends.net,jordi.negrete@opentrends.net"/>
          <param name="from" value="xavi.xicota@opentrends.net"/>
   </appender>
   <appender name="FILE" class="org.apache.log4j.DailyRollingFileAppender">
          <param name="File" value="/var/logs/daily_application.log"/>
          <param name="DatePattern" value="'.'yyyy-MM-dd"/>
          <param name="Append" value="true"/>
          <param name="MaxFileSize" value="50KB"/>
          <layout class="org.apache.log4j.PatternLayout">
                  <param name="ConversionPattern" value="openFrame Message: %-5p
[%t] %c - %m%n"/>
          </layout>
   </appender>
   <root>
          <appender-ref ref="SYSTEMOUT" />
```

```
            <appender-ref ref="JDBCPooled"/>
            <appender-ref ref="FILE"/>
            <appender-ref ref="MAIL"/>
    </root>
</log4j:configuration>
```

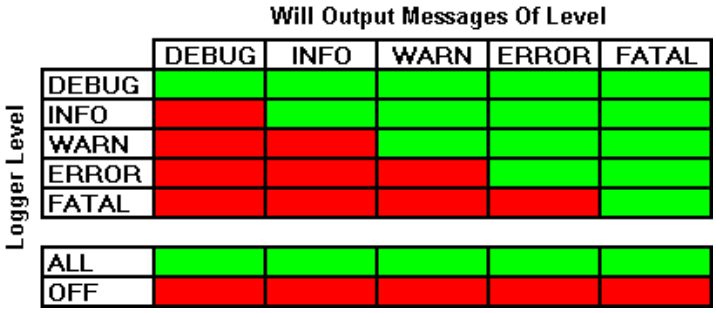
4. Annexos

4.1. Introducció a Log4J

En aquest annex s'ofereix una breu introducció a conceptes de Log4J que cal entendre per utilitzar el Servei de Traces.

A Log4J, existeixen varis elements clau:

1) Prioritats o nivells de les traces. Log4J defineix per defecte 5 nivells: DEBUG, INFO, WARN, ERROR i FATAL. Entre ells existeix una jerarquia (DEBUG<INFO<WARN<ERROR<FATAL), de forma que si en l'aplicació s'ha configurat que només es mostrin traces de nivell WARN, també es mostrarien els de nivell ERROR i FATAL.



2) Categories. Les categories donen control al programador per a determinar quins events requereixen ser capturats i quins no. Dins de l'aplicació es poden definir diferents categories organitzades per jerarquia que en la majoria dels casos es mapejen amb el nom qualificat de les classes java, a on cada classe tindria la seva categoria. També poden crear-se categories per nom, de manera que podria definir-se una categoria que fes traces de les conexions RMI de l'aplicació, o de totes les operacions del usuari, etc.

3) Appenders. Els "appenders" especifiquen a on es desarà la traça definida per a la categoria. Una categoria pot definir diferents sortides o "appenders" a la vegada (consola, fitxer, correu electrònic, base de dades,etc.)
4) Layouts. Permeten especificar com es mostren els events. Per exemple podem especificar que ens aparegui l'hora en la que s'ha produït l'event, qui ho ha llençat, la línia en la que s'ha fet, etc.