INTEGRACIÓN LIFERAY – PENTAHO

GUÍA DE CONFIGURACIÓN

www.stratebi.com
www.todobi.com
info@stratebi.com
91.788.34.10

MAYO 2010

Índice de contenido

Introducción

El presente documento tiene como objetivo la explicación de forma detallada y explícita los pasos a seguir para la integración de Pentaho y Liferay.

La integración se llevará acabo utilizando ciertas librerías y programas extras que se mencionan en la sección de "Requerimientos" los cuales pueden variar según las necesidades de cada instalación (Por ejemplo: Si se desea utilizar un sistema de base de datos diferente a MySQL)

Luego de seguir los diferentes pasos de este documento será capaz de realizar la integración de Liferay con Pentaho dentro de un contenedor web (Tomcat) utilizando un conjunto de programas y librerías (Ejemplo: CAS Single Sign On).

Requerimientos

Los requerimientos previos para realizar la integración Liferay-Pentaho son los siguientes:

- Descargar y descomprimir Pentaho BI Server CE 3.5.2 **Manual** de sourceforge
- Descargar y descomprimir Liferay Tomcat Bundle (Tomcat 6.0, Liferay 5.2.3) de sourceforge
- Descargar y descomprimir Apache Maven 2 (http://maven.apache.org/download.html)
- Descargar e instalar MySQL Server 5.1 (Si no lo tiene ya en su equipo)
- Descargar e instalar Java 5 o 6 (JDK) (Si no lo tiene ya en su equipo)
- Descargar y descomprimir Apache Ant (http://apache.rediris.es/ant/binaries/apache-ant-1.8.1-bin.zip)

Instalación de Liferay

Una vez que se descargado y descomprimido el paquete integrado de Liferay-Tomcat estos son los pasos que debemos seguir para su configuración:

1. Ir a la carpeta donde descargamos el paquete (Lo llamaremos $DIRECTORIO_PACK de ahora en adelante) y debemos tener una estructura como la siguiente:

2. Ir al directorio $DIRECTORIO_PACK/tomcat-6.0.18/bin, abrir el fichero *setenv.bat* y comentar las 5 primeras líneas de modo que quede de la siguiente forma

```
1  rem if not "%JAVA_HOME%" == "" (
2  rem set JAVA_HOME=
3  rem )
4
5  rem set JRE_HOME=%CATALINA_HOME%/jre1.5.0_17/win
6
7  set JAVA_OPTS=%JAVA_OPTS% -Xmx1024m -XX:MaxPermSize=256m -Dfile.enc
```

(Solo debe quedar sin comentar la linea donde se establece la variable JAVA_OPTS)

3. Abrir la herramienta por defecto para hacer consultas SQL para ejecutar las siguientes sentencias para crear la base de datos de Liferay así como usuario y password de conexión a la misma.

```
create database lportal;

GRANT ALL PRIVILEGES ON lportal.* TO 'lportal'@'%' IDENTIFIED BY 'lportal';
```

4. Ir al directorio $DIRECTORIO_PACK\tomcat-6.0.18\webapps\ROOT\WEB-INF\classes y crear el fichero portal-ext.properties. Una vez creado, debemos colocar los siguiente y guardarlo:

```
## Configuracion bdd

jdbc.default.driverClassName=com.mysql.jdbc.Driver

jdbc.default.url=jdbc:mysql://localhost:3306/lportal?useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false

jdbc.default.username=lportal

jdbc.default.password=lportal

## No encriptar el password

passwords.encryption.algorithm=NONE

#Utilizar screenName para iniciar sesion

company.security.auth.type=screenName
```

5. Borrar la data y aplicaciones de ejemplo que trae Liferay borrando los siguientes archivos y directorios:
    - Borrar el directorio $DIRECTORIO_PACK\tomcat-6.0.18\webapps\sevencogs-hook
    - Borrar el directorio $DIRECTORIO_PACK\tomcat-6.0.18\webapps\sevencogs-theme
    - Borrar el directorio $DIRECTORIO_PACK\tomcat-6.0.18\webapps\wol-portlet
    - Borra el fichero $DIRECTORIO_PACK\data\hsql\lportal.properties
    - Borra el fichero $DIRECTORIO_PACK\data\hsql\lportal.script
6. Ir al directorio $DIRECTORIO_PACK\tomcat-6.0.18\bin y ejecutar el fichero *startup.bat* y debemos estar atentos a que no haya ningún error en la ejecución.

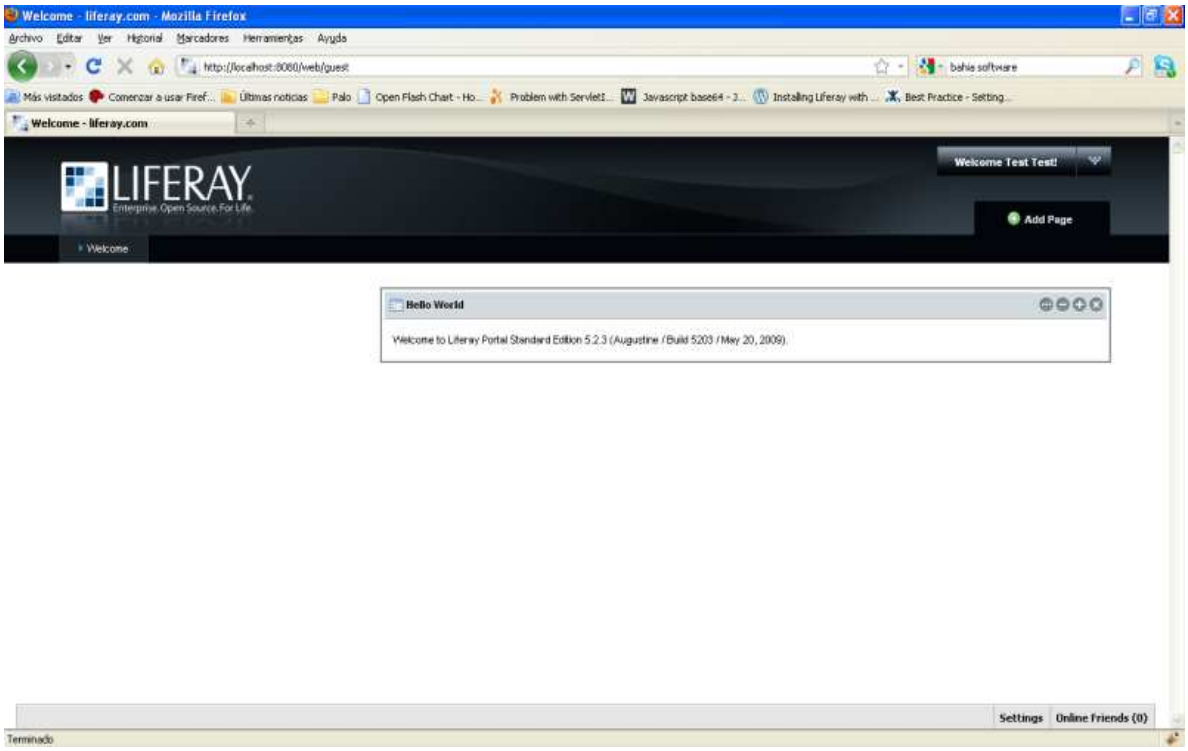
    NOTA: LA PRIMERA VEZ QUE SE EJECUTA SE DEMORA UN POCO YA QUE SE DEBE CREAR LA ESTRUCTUA DE BASE DE DATOS DE LIFERAY.
7. Una vez que inicie el servidor de Tomcat abrir un navegador e ir a la siguiente dirección http://localhost:8080. Una vez desplegada la página,

en la parte superior derecha hacer click en “Sign In” que se encuentra en el combo “Welcome!”. Inciar sesión con la siguiente info:

- usuario: test
- password: test

Una vez iniciada la sesión completar los pasos de creación de la cuenta hasta llegar a la página principal de liferay.

Listo, Liferay está correctamente instalado!!!



Configuración SSL (Conexión segura)

NOTA: Antes de empezar con esta sección es necesario tener instalado Java en nuestra máquina y haber establecido la variable JAVA_HOME como variable de entorno y agregar al PATH la ruta del directorio Bin de Java.

La autenticación a nuestro sistema debe realizarse utilizando conexión segura (https) vía certificado digital, para ello vamos a utilizar una herramienta llamada "keytool" la cual se encuentra en nuestra instalación de Java (%JAVA_HOME%\bin\keytool.exe). A continuación presentamos los pasos a seguir para la configuración de nuestro SSL.

1. Generar el certificado digital para nuestro servidor de la siguiente forma

```
keytool -genkey -alias tomcat --keyalg RSA
```

La contraseña por defecto es "changeit". Una vez ejecutado el comando la herramienta nos preguntará una serie de parámetros, el único realmente importante es cuando nos pregunte ¿Cuáles son su nombre y apellido? Para este ejercicio al estar utilizando una máquina persona colocaremos "localhost", sino deberiamos colocar el DNS del equipo.

2. Al tener el certificado creado lo exportamos a un archivo .cert de la siguiente forma:

```
keytool -export -alias tomcat -file server.cert
```

3. Luego, importar el certificado en el keystore de Java

```
keytool -import -alias tomcat -file server.cert -keystore
$JAVA_HOME/jre/lib/security/cacerts
```

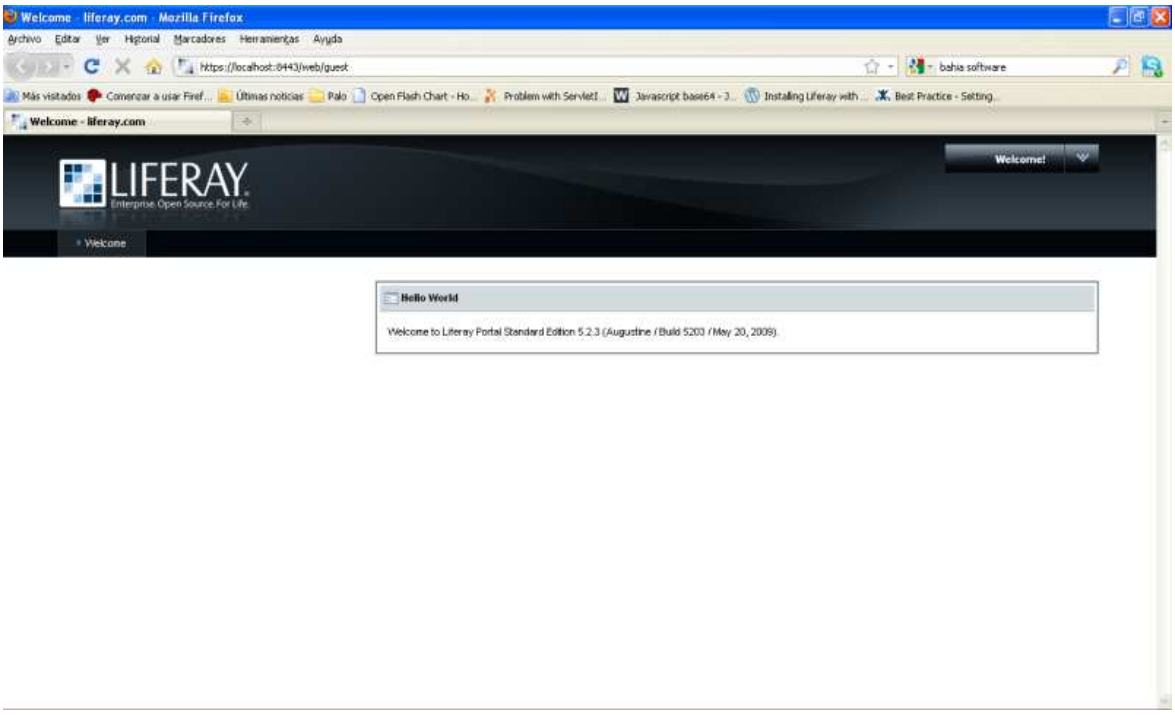
4. Ya el certificado está ubicado donde lo necesitamos así que procedemos a configurar tomcat para que admita conexiones ssl, para ello vamos al fichero $DIRECTORIO_PACK\tomcat-6.0.18\conf\server.xml, ubicamos el

siguiente segmento

```
<!--
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS" />
-->
```

Y lo descomentamos.

5. Si Tomcat se encuentra levantado, es necesario reiniciarlo para que los cambios surjan efecto.
6. Probar que el certificado esté correctamente instalado colocando en el URL de nuestro navegador web https://localhost:8443, el cual debe redirigirnos a la página de inicio de sesión de Liferay



Instalación y Configuración de CAS

Para la instalación y configuración de CAS utilizaremos Apache Maven, para ello debemos tenerlo descomprimido en nuestro equipo y ***opcionalmente*** agregar a la variable de entorno PATH la ruta hacia los binarios del mismo. Los pasos a seguir son los siguientes:

1. Detener el servidor de tomcat si se encuentra corriendo.
2. Crear un directorio de trabajo para nuestro proyecto de CAS (De ahora en adelante lo llamaremos $PROYECTO_CAS).
3. En el debemos crear el fichero pom.xml con la siguiente configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
       <modelVersion>4.0.0</modelVersion>
       <groupId>com.stratebi.cas</groupId>
       <artifactId>local-cas</artifactId>
       <packaging>war</packaging>
       <version>1.0</version>

       <build>
              <plugins>
                     <plugin>
                          <artifactId>maven-war-plugin</artifactId>
                       <configuration>
                            <warName>cas</warName>
                       </configuration>
                   </plugin>
                    </plugins>
       </build>
```

```
<properties>
	<cas.version>3.3.3</cas.version>
</properties>

<dependencies>
	<dependency>
		<groupId>org.springframework.security</groupId>
		<artifactId>spring-security-cas-client</artifactId>
		<version>2.0.4</version>
		<scope>runtime</scope>
		<exclusions>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-dao</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-tx</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-jdbc</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-beans</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-context</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-support</artifactId>
			</exclusion>
		</exclusions>
	</dependency>

	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-aop</artifactId>
```

```
        <version>2.5.6</version>
    </dependency>


    <dependency>
        <groupId>quartz</groupId>
        <artifactId>quartz</artifactId>
        <version>1.5.2</version>
        <type>jar</type>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.1.2</version>
        <type>jar</type>
    </dependency>

    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
        <type>jar</type>
    </dependency>

    <dependency>
        <groupId>ognl</groupId>
        <artifactId>ognl</artifactId>
        <version>2.6.9</version>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.15</version>
        <type>jar</type>
        <scope>runtime</scope>
  <exclusions>
     <exclusion>
         <groupId>javax.mail</groupId>
```

```
            <artifactId>mail</artifactId>
        </exclusion>
        <exclusion>
            <groupId>javax.jms</groupId>
            <artifactId>jms</artifactId>
        </exclusion>
        <exclusion>
            <groupId>com.sun.jdmk</groupId>
            <artifactId>jmxtools</artifactId>
        </exclusion>
        <exclusion>
            <groupId>com.sun.jmx</groupId>
            <artifactId>jmxri</artifactId>
        </exclusion>
    </exclusions>
</dependency>
    <dependency>
          <groupId>org.jasig.cas</groupId>
          <artifactId>cas-server-webapp</artifactId>
          <version>${cas.version}</version>
          <type>war</type>
          <scope>runtime</scope>
    </dependency>

    <dependency>
          <groupId>org.jasig.cas</groupId>
          <artifactId>cas-server-support-jdbc</artifactId>
          <version>${cas.version}</version>
          <type>jar</type>
          <scope>runtime</scope>
    </dependency>
    <dependency>
          <groupId>commons-dbcp</groupId>
          <artifactId>commons-dbcp</artifactId>
          <version>1.2.1</version>
          <scope>runtime</scope>
     </dependency>
     <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.12</version>
```

```
        </dependency>
            </dependencies>
</project>
```

4. Crear en $PROYECTO_CAS el fichero deployerConfigContext.xml dentro de la siguiente estructura de directorios:

$PROYECTO_CAS ☐ src ☐ main ☐ webapp ☐ WEB-INF

El fichero debe tener la siguiente configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
        <bean id="authenticationManager" class="org.jasig.cas.authentication.AuthenticationManagerImpl">
            <property name="credentialsToPrincipalResolvers">
                <list>
                    <bean
class="org.jasig.cas.authentication.principal.UsernamePasswordCredentialsToPrincipalResolver" />
                    <bean
            class="org.jasig.cas.authentication.principal.HttpBasedServiceCredentialsToPrincipalResolver" />
                </list>
            </property>
            <property name="authenticationHandlers">
                <list>
                    <bean
class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
                        p:httpClient-ref="httpClient" />
                    <bean
class="org.jasig.cas.adaptors.jdbc.SearchModeSearchDatabaseAuthenticationHandler">
                        <property
name="tableUsers"><value>user_</value></property>
```

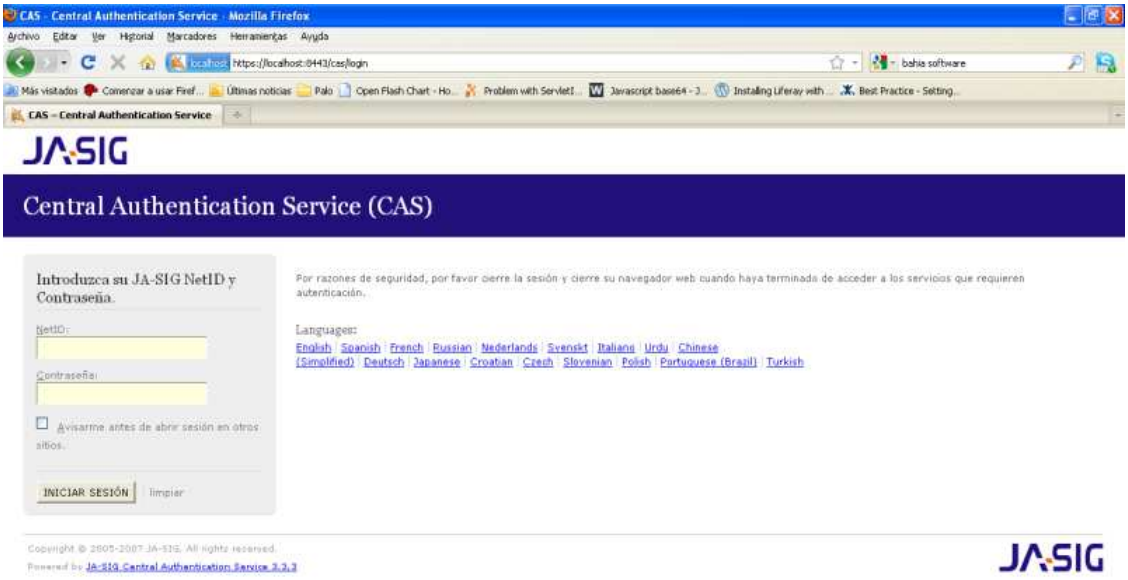
```
                                        <property
name="fieldUser"><value>screenname</value></property>
                                        <property
name="fieldPassword"><value>password_</value></property>
                                        <property name="dataSource" ref="dataSource"/>
                                </bean>
                        </list>
                </property>
        </bean>
        <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
                <property name="driverClassName">
                        <value>com.mysql.jdbc.Driver</value>
                </property>
                <property name="url">
                        <value>jdbc:mysql://localhost:3306/lportal</value>
                </property>
                <property name="username">
                        <value>lportal</value>
                </property>
                <property name="password">
                        <value>lportal</value>
                </property>
        </bean>
        <bean id="userDetailsService" class="org.springframework.security.userdetails.memory.InMemoryDaoImpl">
                <property name="userMap">
                        <value>    </value>
                </property>
        </bean>
        <bean id="attributeRepository"
                class="org.jasig.services.persondir.support.StubPersonAttributeDao">
                <property name="backingMap">
```

```
                <map>
                    <entry key="uid" value="uid" />
                    <entry key="eduPersonAffiliation" value="eduPersonAffiliation" />
                    <entry key="groupMembership" value="groupMembership" />
                </map>
            </property>
        </bean>
        <bean
            id="serviceRegistryDao"
            class="org.jasig.cas.services.InMemoryServiceRegistryDaoImpl" />
</beans>
```

5. Luego, desde una línea de comandos (Command Prompt) ejecturar la siguiente sentencia en el directorio $PROYECTO_CAS

```
mvn clean package
```

6. Esto creará dentro de $POYECTO_CAS un directorio “target” y dentro de el conseguiremos el archivo cas.war que vamos a utilizar. Debemos colocarlo en $DIRECTORIO_PACK\tomcat-6.0.18\webapps y levantar el servidor de Tomcat.
7. Probar que esté funcionando CAS. Ir a https://localhost:8443/cas e iniciar sesión con el usuario y password de liferay.


CAS - Central Authentication Service - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
https://localhost:8443/cas/login
CAS – Central Authentication Service
JA-SIG
Central Authentication Service (CAS)
Introduzca su JA-SIG NetID y Contraseña.
NetID:
Contraseña:
Avisarme antes de abrir sesión en otros sitios.
INICIAR SESIÓN
limpiar
Por razones de seguridad, por favor cierre la sesión y cierre su navegador web cuando haya terminado de acceder a los servicios que requieren autenticación.
Languages:
English | Spanish | French | Russian | Nederlands | Svenskt | Italiano | Urdu | Chinese (Simplified) | Deutsch | Japanese | Croatian | Czech | Slovenian | Polish | Portuguese (Brazil) | Turkish
JA-SIG

Instalación y configuración de Pentaho

Una vez descargado y descomprimido el paquete **manual** de pentaho es necesario realizar una serie de paso para configurar el war de pentaho que vamos a obtener.

El directorio en el que se descomprimió pentaho lo llamaremos de aquí en adelante $PENTAHO_FUENTE.

Para configurar nuestro pentaho debemos seguir los siguientes pasos:

1. Ir al directorio $PENTAHO_FUENTE\custom-pentaho-webapp\META-INF y crear el fichero context.xml. Luego de su creación, debemos agregar lo siguiente en el:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/pentaho" docbase="webapps/pentaho/">
    <Resource name="jdbc/Hibernate" auth="Container" type="javax.sql.DataSource"
        factory="org.apache.commons.dbcp.BasicDataSourceFactory" maxActive="20"
maxIdle="5"
        maxWait="10000" username="hibuser" password="password"
        driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/hibernate"
        validationQuery="/* ping */ select 1"/>

    <Resource name="jdbc/Quartz" auth="Container" type="javax.sql.DataSource"
        factory="org.apache.commons.dbcp.BasicDataSourceFactory" maxActive="20"
maxIdle="5"
        maxWait="10000" username="pentaho_user" password="password"
        driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/quartz"
        validationQuery="/* ping */ select 1"/>
</Context>
```

2. Descargar el Cas Client (URL: http://www.ja-sig.org/downloads/cas-clients/ descomprimirlo y copiar el fichero cas-client-core-3.1.10.jar) y colocarlo en el directorio $PENTAHO_FUENTE\pentaho-third-party
3. Copiar el fichero spring-security-cas-client-2.0.4.jar de la carpeta lib de CAS y colocarlo en el directorio $PENTAHO_FUENTE\pentaho-third-party
4. Crear el directorio $PENTAHO_FUENTE\build

5. Ir al directorio $PENTAHO_FUENTE y construir el paquete de pentaho utilizando Ant (Es parte de las librerías de Java así como lo es keytool de la sección de SSL) de la siguiente forma:

```
ant war-pentaho-tomcat
```

6. Esto colocara nuestro .war en el directorio $PENTAHO_FUENTE\build\pentaho-wars\tomcat, el cual debemos colocar en el directorio webapps de Tomcat así como el pentaho-style.war.
7. Es necesario configurar pentaho-solutions para mysql
8. Crear las bases de datos de hibernate y quartz en mysql, utilizando los ficheros que se encuentran en el directorio $PENTAHO_FUENTE\pentaho-data\mysql5 (create_quartz_mysql.sql, create_reporsitory_mysql.sql, create_sample_datasource_mysql.sql)
9. Abrir el fichero pentaho-spring-beans.xml ubicado en $PENTAHO_FUENTE\pentaho-solutions\system y editarlo para que quede de la siguiente forma:

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

<!--+
  | This should be the only file specified in web.xml's contextConfigLocation. It
should only contain imports.
  +-->

<beans>

  <import resource="pentahoSystemConfig.xml" />
  <import resource="adminPlugins.xml" />
  <import resource="systemListeners.xml" />
  <import resource="sessionStartupActions.xml" />
  <import resource="applicationContext-spring-security.xml" />
  <import resource="applicationContext-common-authorization.xml" />
```

```
  <import resource="pentahoObjects.spring.xml" />

  <import resource="applicationContext-spring-security-jdbc.xml"/>
       <import resource="applicationContext-pentaho-security-jdbc.xml"/>
       <import resource="applicationContext-spring-security-cas.xml"/>

</beans>
```

10. Crear el archivo applicationContext-spring-security-cas.xml en la ruta $PENTAHO_FUENTE\pentaho-solutions\system con la siguiente configuración:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--+
        | Application context containing FilterChainProxy. This version overrides
        | certain beans from applicationContext-spring-security.xml to enable CAS.
        +--><!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans default-autowire="no" default-dependency-check="none" default-lazy-init="false">

        <!-- ========================= FILTER CHAIN ======================== -->

        <!-- overridden from applicationContext-spring-security.xml to enable CAS -->
        <bean autowire="default" class="org.springframework.security.util.FilterChainProxy" dependency-
check="default" id="filterChainProxy" lazy-init="default">
                <property name="filterInvocationDefinitionSource">
                        <value>
                                <![CDATA[CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                                PATTERN_TYPE_APACHE_ANT

        /**=securityContextHolderAwareRequestFilter,httpSessionContextIntegrationFilter,logoutFilter,casPro
cessingFilter,basicProcessingFilter,requestParameterProcessingFilter,anonymousProcessingFilter,pentahoSecurit
yStartupFilter,exceptionTranslationFilter,filterInvocationInterceptor]]>
                        </value>
```

```
		</property>
	</bean>

	<!-- ====================== HTTP REQUEST SECURITY ====================== -->

	<bean autowire="default" class="org.springframework.security.ui.cas.ServiceProperties" dependency-check="default" id="serviceProperties" lazy-init="default">
		<property name="service" value="http://localhost:8080/pentaho/j_spring_cas_security_check"/>
		<property name="sendRenew" value="false"/>
	</bean>

	<!-- replaces authenticationProcessingFilter in filterChainProxy above -->
	<bean autowire="default" class="org.springframework.security.ui.cas.CasProcessingFilter" dependency-check="default" id="casProcessingFilter" lazy-init="default">
		<property name="authenticationManager">
			<ref bean="authenticationManager"/>
		</property>
		<property name="authenticationFailureUrl" value="/public/casFailed"/>
		<property name="defaultTargetUrl" value="/"/>
		<property name="filterProcessesUrl" value="/j_spring_cas_security_check"/>
	</bean>

	<!-- overridden from applicationContext-spring-security.xml -->
	<bean autowire="default" class="org.springframework.security.ui.ExceptionTranslationFilter" dependency-check="default" id="exceptionTranslationFilter" lazy-init="default">
		<property name="authenticationEntryPoint">
			<ref local="casProcessingFilterEntryPoint"/>
		</property>
		<property name="accessDeniedHandler">
			<bean autowire="default" class="org.springframework.security.ui.AccessDeniedHandlerImpl" dependency-check="default" lazy-
```

```
init="default"/>
                </property>
        </bean>

        <bean autowire="default" class="org.springframework.security.ui.cas.CasProcessingFilterEntryPoint"
dependency-check="default" id="casProcessingFilterEntryPoint" lazy-init="default">
                <property name="loginUrl" value="https://localhost:8443/cas/login"/>
                <property name="serviceProperties">
                        <ref local="serviceProperties"/>
                </property>
        </bean>

        <!-- overridden from applicationContext-spring-security.xml -->
        <bean autowire="default" class="org.springframework.security.providers.ProviderManager"
dependency-check="default" id="authenticationManager" lazy-init="default">
                <property name="providers">
                        <list>
                                <!--ref bean="daoAuthenticationProvider" /-->
                                <ref bean="anonymousAuthenticationProvider"/>
                                <ref bean="casAuthenticationProvider"/>
                        </list>
                </property>
        </bean>

        <bean autowire="default"
class="org.springframework.security.providers.cas.CasAuthenticationProvider" dependency-check="default"
id="casAuthenticationProvider" lazy-init="default">
                <property name="userDetailsService">
                        <ref bean="userDetailsService"/>
                </property>
    <property name="serviceProperties">
      <ref local="serviceProperties"/>
```

```
    </property>
                <property name="ticketValidator">
                        <ref local="ticketValidator"/>
                </property>
                <property name="key" value="my_password_for_this_auth_provider_only"/>
        </bean>

        <bean autowire="default" class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator"
dependency-check="default" id="ticketValidator" lazy-init="default">
    <constructor-arg index="0" value="https://localhost:8443/cas"/>
        </bean>

        <!-- overridden from applicationContext-spring-security.xml to specify logoutSuccessUrl as CAS
logout page -->
        <bean autowire="default" class="org.springframework.security.ui.logout.LogoutFilter" dependency-
check="default" id="logoutFilter" lazy-init="default">
                <constructor-arg
value="https://localhost:8443/cas/logout?url=http://localhost:8080/pentaho/Home"/>
                <!-- URL redirected to after logout -->
                <constructor-arg>
                        <list>
                                <bean autowire="default"
class="org.pentaho.platform.web.http.security.PentahoLogoutHandler" dependency-check="default" lazy-
init="default"/>
                                <bean autowire="default"
class="org.springframework.security.ui.logout.SecurityContextLogoutHandler" dependency-check="default"
lazy-init="default"/>
                        </list>
                </constructor-arg>
                <property name="filterProcessesUrl" value="/Logout"/>
        </bean>

</beans>
```

11. Editar el fichero pentaho.xml ubicado en la ruta $PENTAHO_FUENTE\pentaho-solutions\system, reemplazando “Admin” por “Administrator” en todas las ocurrencias del archivo y “User” por “Authenticated”

12. Editar el fichero applicationContext-spring-security-jdbc.xml ubicado en la ruta $PENTAHO_FUENTE\pentaho-solutions\system que debe quedar de la siguiente forma:

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

<!--+
	| Application context containing JDBC AuthenticationProvider
	| implementation.
	+-->

<beans>

	<bean id="daoAuthenticationProvider"

	class="org.springframework.security.providers.dao.DaoAuthenticationProvider"
>
		<property name="userDetailsService">
			<ref bean="userDetailsService" />
		</property>
		<property name="passwordEncoder">
			<ref bean="passwordEncoder" />
		</property>
	</bean>
```

```
	<bean id="userDetailsService"
		class="org.springframework.security.userdetails.jdbc.JdbcDaoImpl">
		<property name="dataSource">
			<ref local="dataSource" />
		</property>
		<property name="authoritiesByUsernameQuery">
			<value>
				<![CDATA[SELECT user_.screenname as username, role_.name
as authority FROM role_, user_, users_roles where role_.roleId =
users_roles.roleId AND user_.userId = users_roles.userId AND user_.screenname = ?
ORDER by role_.name]]>
			</value>
		</property>
		<property name="usersByUsernameQuery">
			<value>
				<![CDATA[select screenname as username, password_ as
password, 1 as enabled from user_ where screenname = ? order by username]]>
			</value>
		</property>
	</bean>
	<!--  This is only for Hypersonic. Please update this section for any other
database you are using -->
	<bean id="dataSource"
		class="org.springframework.jdbc.datasource.DriverManagerDataSource">
		<property name="driverClassName" value="com.mysql.jdbc.Driver" />
		<property name="url"
			value="jdbc:mysql://localhost:3306/lportal" />
		<property name="username" value="lportal" />
		<property name="password" value="lportal" />
	</bean>
```

```
  <bean id="passwordEncoder"

class="org.springframework.security.providers.encoding.PlaintextPasswordEncoder"
/>

</beans>
```

13. Editar el fichero applicationContext-pentaho-security-jdbc.xml ubicado en la ruta $PENTAHO_FUENTE\pentaho-solutions\system que debe quedar de la siguiente forma:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springsource.org/dtd/spring-beans.dtd">


<!--+

       | Application context containing JDBC UserRoleListService

       | implementation.

       +-->


<beans>


       <bean id="jdbcUserRoleListService"


       class="org.pentaho.platform.plugin.services.security.userrole.jdbc.JdbcUserR
oleListService">

              <constructor-arg index="0" ref="userDetailsService" />

              <property name="allAuthoritiesQuery">

                     <value>

                            <![CDATA[select distinct(name) as authority from role_
order by authority]]>

                     </value>

              </property>

              <property name="allUsernamesInRoleQuery">
```

```
                        <value>
                                <![CDATA[SELECT user_.screenname as username FROM role_,
user_, users_roles where role_.roleId = users_roles.roleId AND user_.userId =
users_roles.userId AND role_.name = ? ORDER by role_.name]]>
                        </value>
                </property>
                <property name="allUsernamesQuery">
                        <value>
                                <![CDATA[SELECT distinct(user_.screenname) as username
from user_ order by username]]>
                        </value>
                </property>
                <property name="dataSource" ref="dataSource" />
        </bean>

        <bean id="pentahoUserRoleListService"

        class="org.pentaho.platform.engine.security.userrole.UserDetailsRoleListServ
ice">
                <property name="userRoleListService">
                        <ref local="jdbcUserRoleListService" />
                </property>
        </bean>

</beans>
```

14. Editar el fichero applicationContext-spring-security.xml ubicado en la ruta $PENTAHO_FUENTE\pentaho-solutions\system, reemplazando “Admin” por “Administrator” en todas las ocurrencias del archivo y “Authenticated” por “User”

15. Copiar el directorio pentaho-solutions de $PENTAHO_FUENTE y

colocarlo en $DIRECTORIO_PACK

16. Iniciar Tomcat e ir a la dirección http://localhost:8080/pentaho, la cual deberia redirigirnos a la página de inicio de sesión de CAS y luego

de iniciar sesión a la pantalla principal de Pentaho

Configuración de Liferay (CAS)

Luego de tener Liferay instalado y CAS correctamente configurado es necesario configurar Liferay para que su autenticación se haga atraves de CAS; para ello es necesario seguir los siguientes pasos:

1. Detener el servidor de tomcat
2. Editar el fichero portal-ext.properties que se encuentra en la ruta $DIRECTORIO_PACK\tomcat-6.0.18\webapps\ROOT\WEB-INF\classes y agregar la siguiente configuración (luego de lo que ya se encuentre en el fichero):

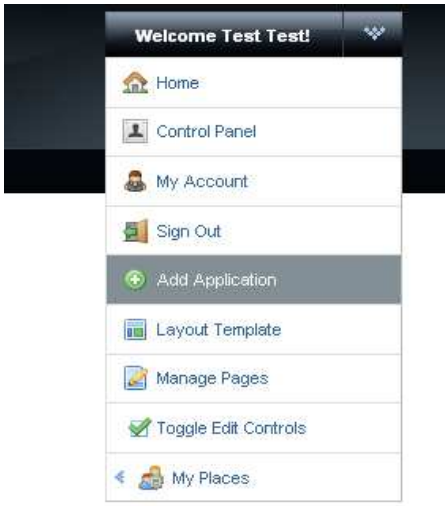
```
cas.auth.enabled=true
cas.login.url=https://localhost:8443/cas/login
cas.logout.url=https://localhost:8443/cas/logout?url=http://localhost:8080
cas.service.url=http://localhost:8080/c/portal/login
cas.server.name=localhost:8080
cas.validate.url=https://localhost:8443/cas/proxyValidate
auto.login.hooks=com.liferay.portal.security.auth.CASAutoLogin
```

3. Deshabiliar el portlet de inicio de sesión de Liferay (ya que es necesario hacer un desarrollo sobre el para que funcione con CAS); para ello es necesario editar el fichero liferay-portlet que se encuentra en la ruta $DIRECTORIO_PACK\tomcat-6.0.18\webapps\ROOT\WEB-INF\ y comentar la entrada del portlet de login (<portlet-name>58</portlet-name>)
4. Iniciar el servidor de Tomcat y probar iniciar sesión (Debe redirigirnos a CAS y luego de suministrar los datos volver a la página de inicio de Liferay.

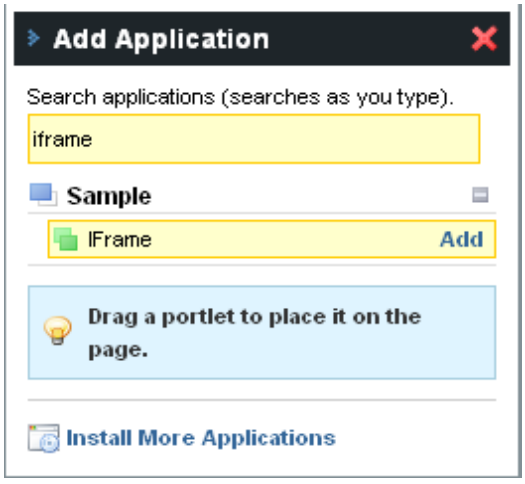
Prueba de integración

Luego de realizar la integración Pentaho-Liferay-CAS vamos a hacer un pequeño ejemplo práctico que nos permita visualizar la funcionalidad de esta integración. Para ello vamos a hacer lo siguiente:

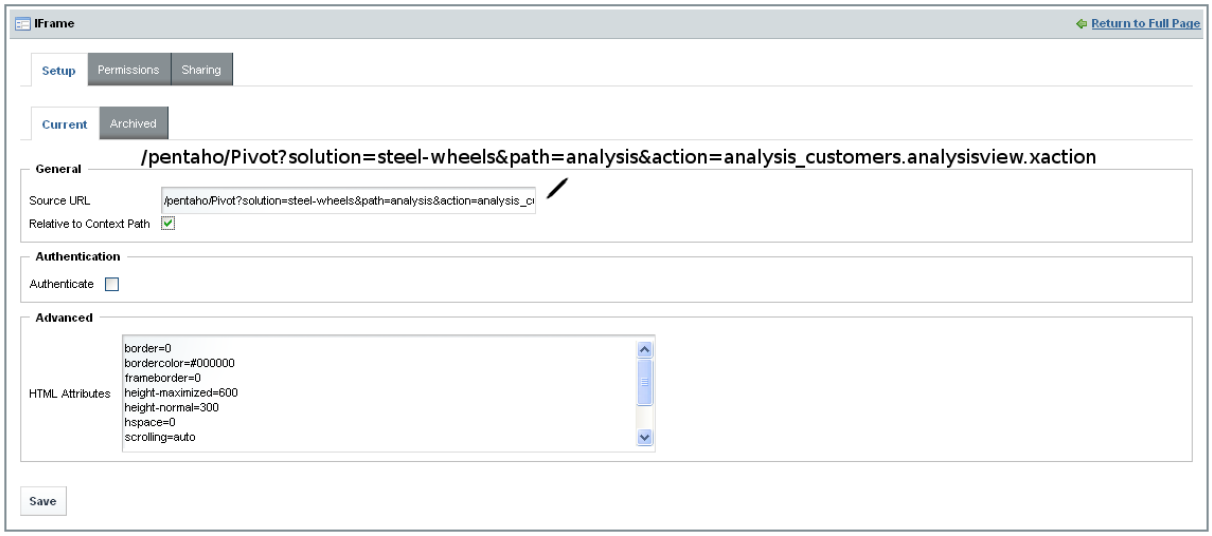
1. Copiar una solución existente de pentaho (en nuestro caso steel-wheels) y colocarla en la carpeta pentaho-solutions.
2. Chequear que en base de datos existe el datasource hacia la base de datos SampleData (Base de datos de hypersonic).
3. Levantar la base de datos de hypersonic
4. Si el servidor de Tomcat se encontraba levantado detenerlo y luego volverlo a iniciar.
5. Una vez que el servicio se encuentre disponible ir al url http://localhost:8080/pentaho, iniciar sesión, refrescar el cache de la solución y refrescar el repositorio del cache de mondrian.
6. Ir a la url http://localhost:8080 (Nos debe redirigir a la página principal de liferay con la sesión ya iniciada)



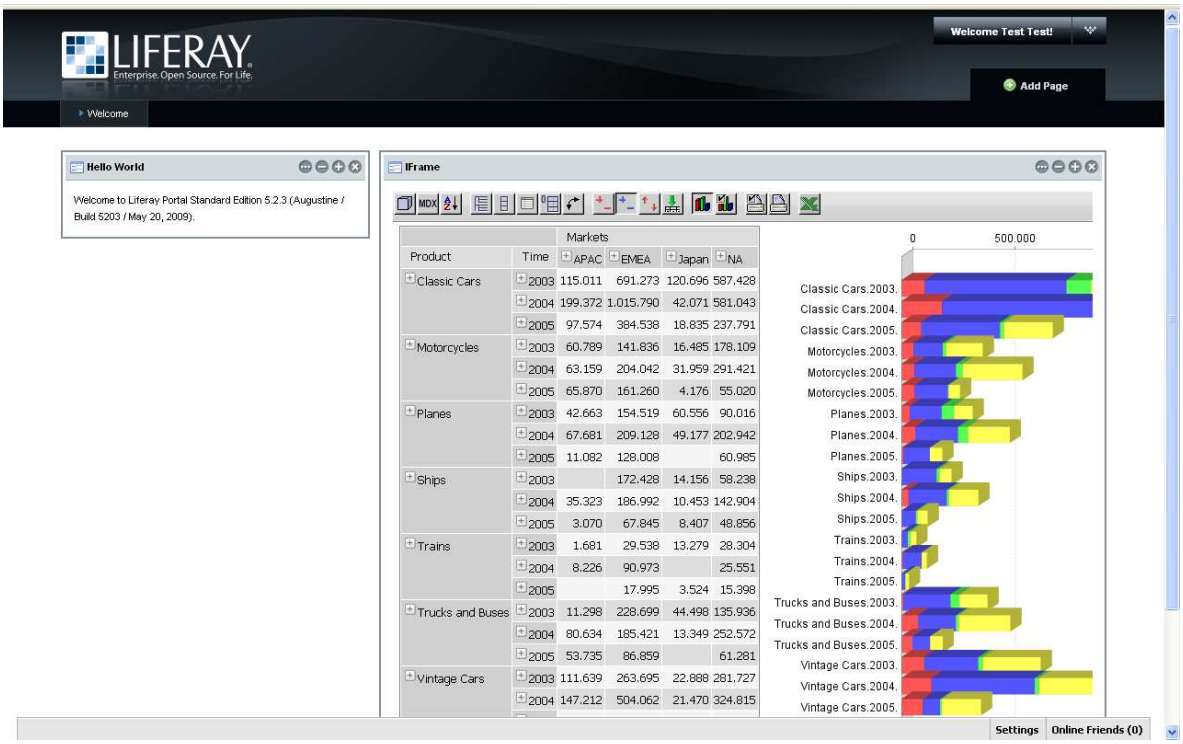
7. En la página donde nos encontremos ir al combo que se encuentra en la parte superior izquierda de nuestra pantalla y hacer click en la opción "Add Aplication" y nos aparecerá un panel en donde escribiremos "iframe" y lo agregamos a nuestra página.



8. Configurar el "iframe" portlet de la siguiente manera:



9. Guardar los cambios efectuados en la configuración del portlet y la misma nos redirigirá a nuestra pantalla de visualización donde obtendremos el resultado de nuestra integración.



Sobre Stratebi

Stratebi es una empresa española, radicada en Madrid y oficinas en Barcelona, creada por un grupo de profesionales con amplia experiencia en sistemas de información, soluciones tecnológicas y procesos relacionados con soluciones de Open Source y de inteligencia de Negocio.

Esta experiencia, adquirida durante la participación en proyectos estratégicos en compañías de reconocido prestigio a nivel internacional, se ha puesto a disposición de nuestros clientes a través de Stratebi.

En Stratebi nos planteamos como **objetivo** dotar a las compañías e instituciones, de herramientas escalables y adaptadas a sus necesidades, que conformen una estrategia Business Intelligence capaz de rentabilizar la información disponible. Para ello, nos basamos en el desarrollo de soluciones de Inteligencia de Negocio, mediante tecnología Open Source.

Stratebi **son profesores y responsables de proyectos del Master en Business Intelligence de la Universidad UOC.**

Los profesionales de Stratebi son los creadores y autores del primer weblog en español sobre el mundo del Business Intelligence, Data Warehouse, CRM, Dashboards, Scorecard y Open Source.

Todo Bi, se ha convertido en una referencia para el conocimiento y divulgación del Business Intelligence en español.

TodoBI <business intelligence>

Stratebi ha sido elegida como **Caso Éxito del Observatorio de Fuentes Abiertas de Cenatic**.
http://observatorio.cenatic.es/index.php?option=com_content&view=article&id=429:stratebi&catid=2:empresas&Itemid=41

Observatorio Nacional del Software de Fuentes Abiertas | cenatic

Asociaciones empresariales de Software Libre empresarial en las que participamos.

TECNOLOGIAS CON LAS QUE TRABAJAMOS

ALGUNAS REFERENCIAS STRATEBI

DEMOS e INFO

- Creadores del principal Portal sobre Business Intelligence en castellano (TodoBI.com)
- Demo Tablero Futbolero (http://www.tablerofutbolero.es)(Cuadros de Mando) pedir clave en info@stratebi.com
- Demo BI Open Source Sector Público, (http://demo.stratebi.es)pedir clave en info@stratebi.com
- BI Termometer. Checklist gratuito (más de 1.500 Kpis), para el éxito de un Proyecto BI. http://todobi.blogspot.com/2010/04/checklist-para-hacer-un-proyecto.html
- Video entrevista en Portal BI-Spain, http://todobi.blogspot.com/2010/04/entrevista-sobre-business-intelligence.html
- Zona YouTube Stratebi. YouTube , http://www.youtube.com/user/Stratebi
- Catálogo de Soluciones Verticales. Encuentra la tuya!!, http://www.stratebi.com/Inteletter.htm

(si encontráis cualquier errata o mejora sobre el documento, por favor, hacédnoslo saber, escribiendo a: info@stratebi.com)