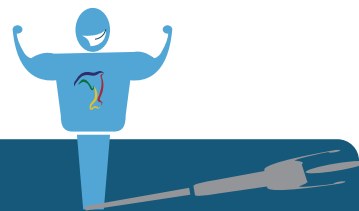


¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
 Ese apoyo que siempre quiso tener...

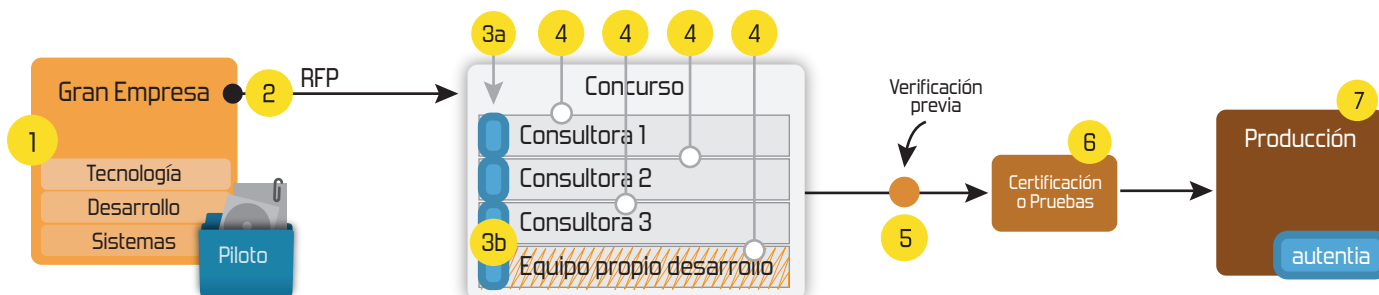
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
 Gestor de contenidos (Alfresco)
 Aplicaciones híbridas

Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

Control de autenticación y
 acceso (Spring Security)
 UDDI
 Web Services
 Rest Services
 Social SSO
 SSO (Cas)

JPA-Hibernate, MyBatis
 Motor de búsqueda empresarial (Solr)
 ETL (Talend)

Dirección de Proyectos Informáticos.
 Metodologías ágiles
 Patrones de diseño
 TDD

BPM (jBPM o Bonita)
 Generación de informes (JasperReport)
 ESB (Open ESB)



[Home](#) | [Quienes Somos](#) | [Empleo](#) | [Foros](#) | [Tutoriales](#) | [Servicios Gratuitos](#) | [Contacte](#)

Tutorial desarrollado por: [Daniel Hernandez del Peso](#)

Puedes encontrarme en Autentia



Descargar este documento en formato PDF [Log4Net.pdf](#)

[Todo para la Hoja Excel](#)

Manuales de macros y funciones.
Aplicaciones y complementos.

[Curso Web J2EE](#)

Curso Avanzado en Desarrollo
Web con J2EE

[Centro de Estudios PFP](#)

Centro especializado en Diseño.
Titulaciones oficiales-homologadas

[Master para comunicadores](#)

Masters en TV, cine, artes visuales
y comunicación de un año en
Madrid

Anuncios Goooooogle

Anunciarse en este sitio

Primeros pasos con Log 4 NET

En el desarrollo de aplicaciones de cualquier tipo, es una práctica muy recomendable dejar trazas de todo lo que pasa en el sistema. El problema es que la información que interese mostrar puede ser distinta si se está en plena construcción o si se está en el entorno de explotación. Para solventar este problema hay dos opciones: se puede modificar el código de la aplicación y recompilarlo; o se puede emplear una librería que se pueda configurar mediante ficheros externos.

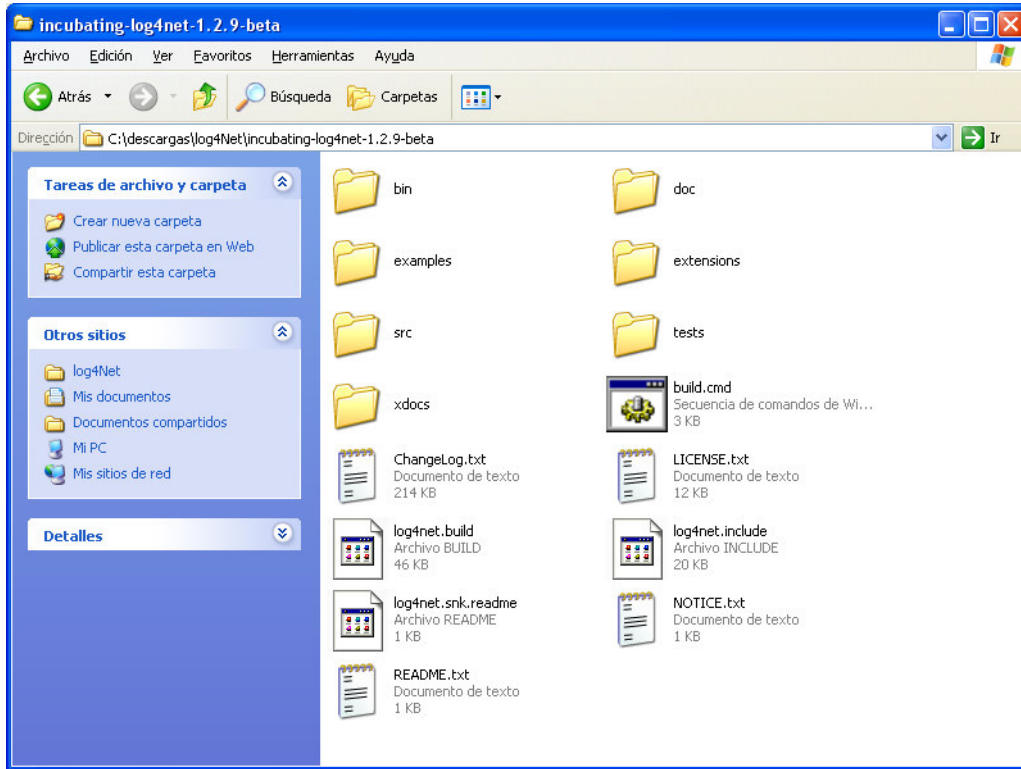
En la empresa en que trabajo, [Autentia](#), hemos optado por la segunda solución, mucho más cómoda y limpia, y empleamos el paquete [log4j](#) (log for Java), un producto gratuito de *Apache.org*, configurable de manera muy sencilla mediante un fichero XML.

Pero como el mundo de la informática no empieza ni acaba en Java, existen paquetes similares para otros entornos, como el que nos va a ocupar en este tutorial: [log4net](#), la versión del producto para la plataforma de Microsoft. A continuación se explicará cómo crear una aplicación web que deje trazas usando esta herramienta.

Descarga

Para comenzar, descargaremos el producto de <http://logging.apache.org/log4net/> y lo descomprimiremos a una carpeta:

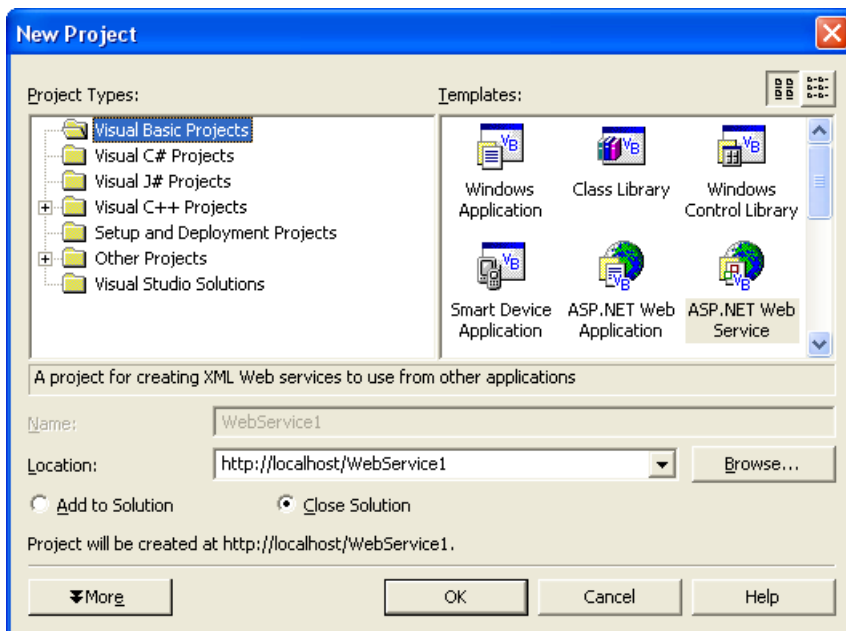
A lo largo del tutorial se va a emplear la versión 1.2.9 beta, que contiene los elementos que se ven en la imagen



La distribución elegida contiene carpetas con documentación, ejemplos e incluso el código sin compilar, pero la carpeta interesante para emplear **log4net** en nuestros proyectos es la carpeta **bin**, que contiene compilaciones para las distintas plataformas que soporta (.NET 1.0, .NET 1.1, SSCLI...)

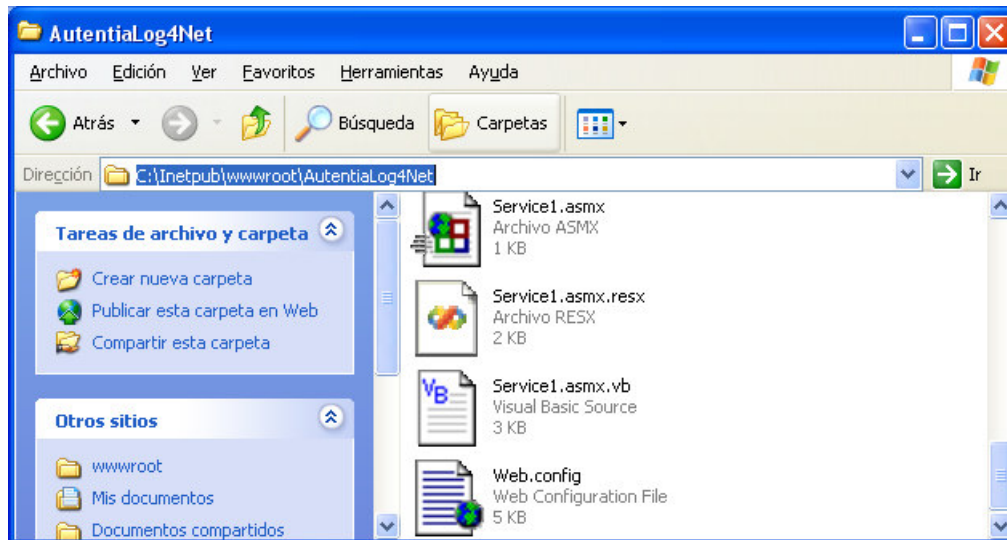
Proyecto .NET

Ahora crearemos una aplicación web en .NET. En este caso vamos a elegir un Web Service escrito en Visual Basic. Para ello, en Visual Studio, vamos al menú **File -> New -> Project**. Se nos abrirá la siguiente ventana:

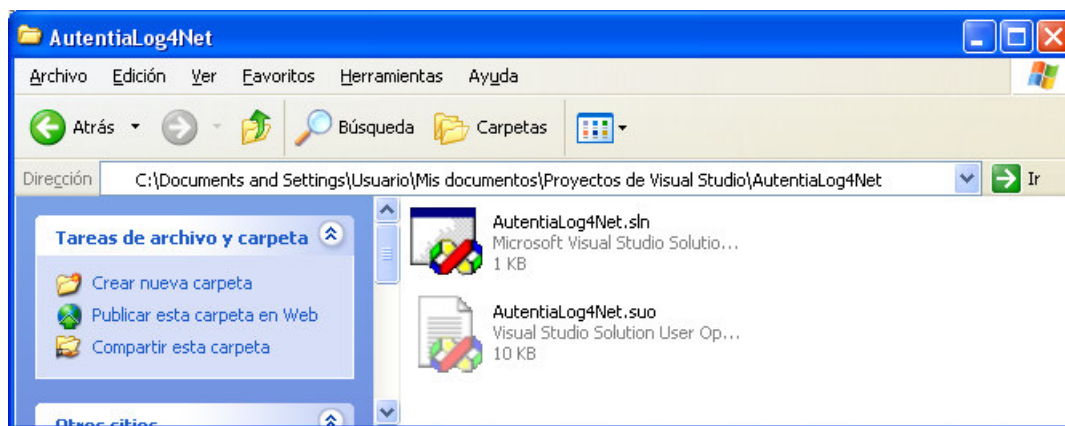


En ella seleccionamos la opción "Visual Basic Projects" y dentro de ella, la opción "ASP .NET Web Service". Vemos que no podemos cambiar el nombre del proyecto, pero sí podemos seleccionar la carpeta virtual (mediante la que se accede a la aplicación por navegador) en la que se desplegará el proyecto (Visual Studio lo hace por nosotros). El proyecto adopta el nombre de esa carpeta virtual.

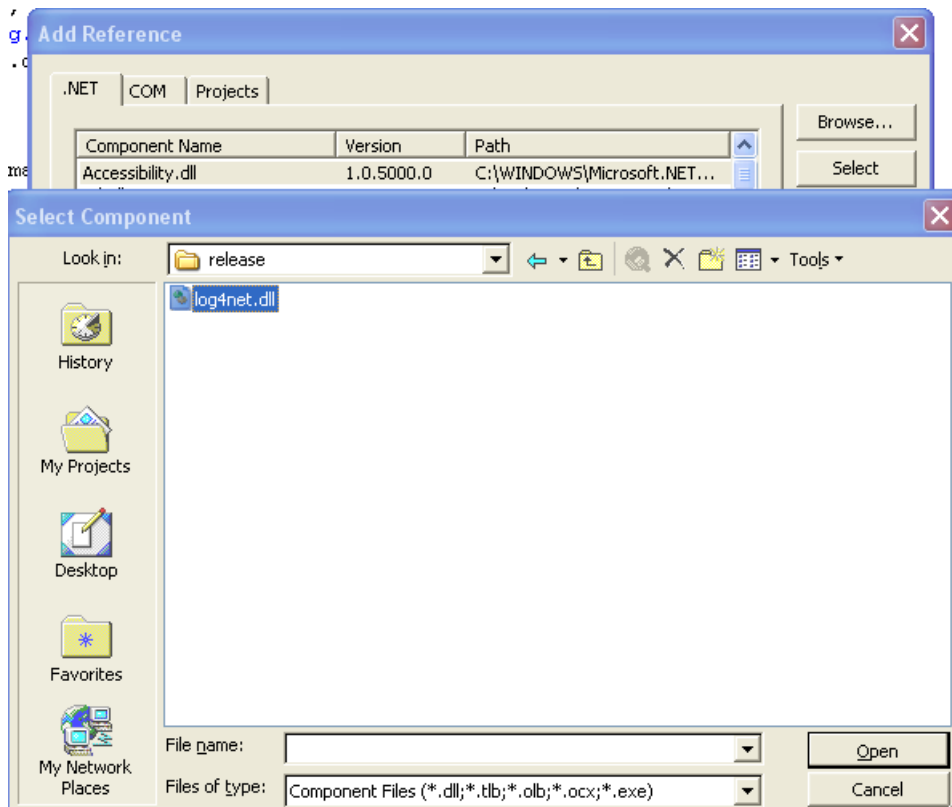
Por tanto, se crean ficheros correspondientes al proyecto en dos ubicaciones: por un lado, los correspondientes a la aplicación web se crean en una carpeta con el nombre dado al proyecto, que se guarda en la ubicación por defecto del IIS (Internet Information Server, que es el que ejecuta este tipo de aplicaciones), *"inetpub/wwwroot"*,



mientras que en la carpeta donde se creen habitualmente los proyectos de Visual Studio (por ejemplo, "*Mis Documentos/Proyectos de Visual Studio*") se crean los ficheros propios de la solución a que pertenezca el proyecto:

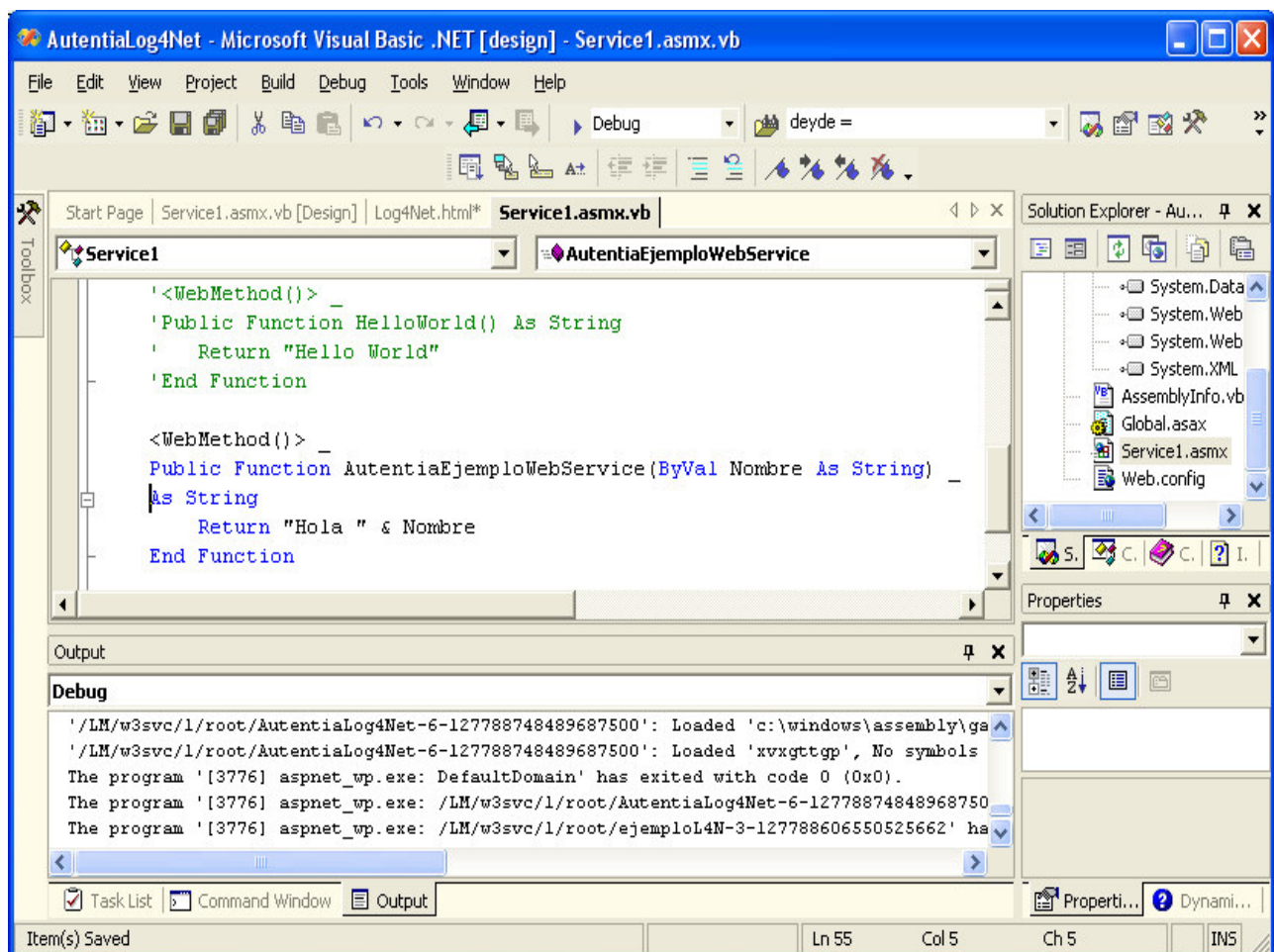


Una vez creado el proyecto, se añade una referencia en el mismo que apunte a **log4net**. Para ello, pinchamos en la opción *Project->Add Reference*, Y se nos abre una ventana, en la que empleamos la opción *Browse* para localizar la DLL que queremos incluir:



Puesto que se va a emplear la versión 1.1 del framework de .NET, seleccionamos la implementación que se encuentra en *[carpeta log4net]binnet1.1release* de entre todas las disponibles, pues este producto también funciona para otros entornos

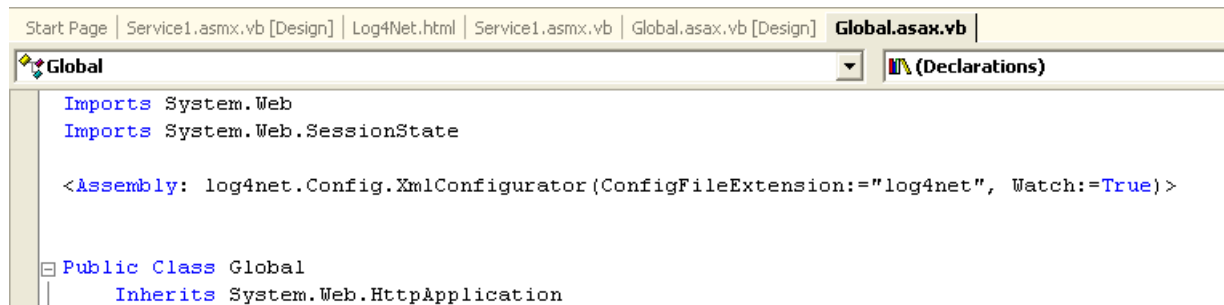
Una vez añadida la referencia implementamos el servicio web



Añadimos la línea siguiente al principio del fichero Global.asax.vb, justo debajo de las importaciones:

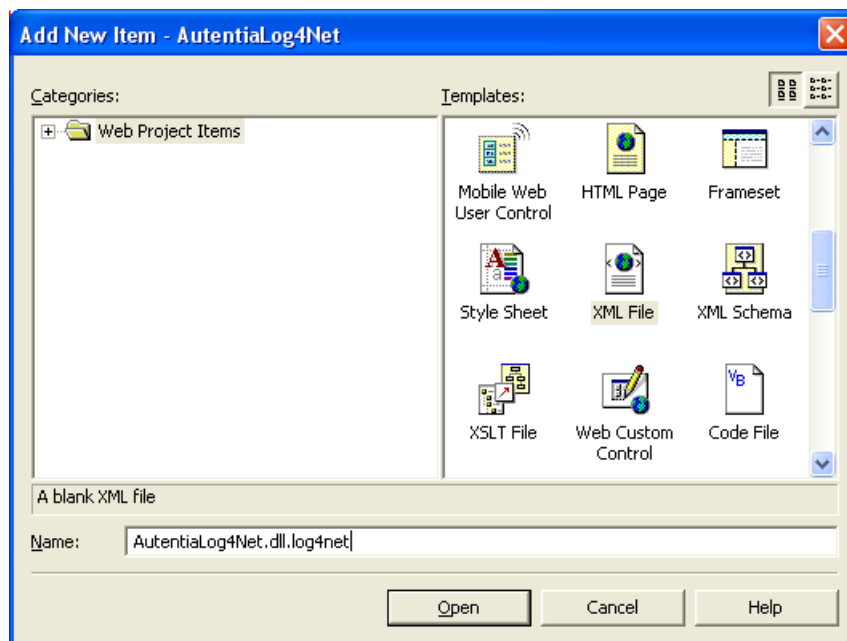
```
<Assembly: log4net.Config.XmlConfigurator(ConfigFileExtension:="log4net", Watch:=True)>
```

Lo que esto quiere decir es que el fichero de configuración de **log4net** tendrá el nombre de la aplicación con la extensión "log4net".

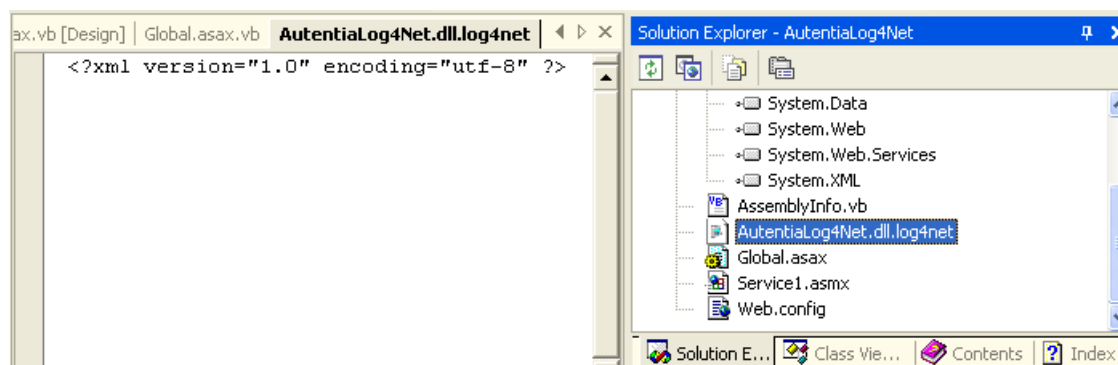


El fichero de configuración

Se añade al proyecto un fichero XML con nombre [nombre_proyecto].dll.log4net, yendo de nuevo al menú *Project* y seleccionando la opción *Add New Item*. En la ventana que se abre, seleccionamos añadir un fichero XML con el nombre que queremos:



Y aceptamos:



El contenido del fichero para este primer ejemplo es el que sigue:

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="LogFile" type="log4net.Appender.FileAppender">
    <file value="./web-serviceExample.log" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <appendToFile value="true" />
  </appender>
</log4net>
```



```

<layout type="log4net.Layout.PatternLayout">
  <conversionPattern value="%date %level %thread %logger - %message%newline" />
</layout>
</appender>
<root>
  <level value="DEBUG" />
  <appender-ref ref="LogFile" />
</root>
</log4net>

```

Dentro de este código, los elementos más importantes son:

- **appender**: indica el medio en que se imprimirán las trazas. en este caso, se emplea un fichero.
 - **file**: nombre del fichero en que se van a almacenar las trazas
 - **appendToFile**: si está a *true*, se anexan las trazas al final del fichero. En caso contrario, éste se sobrescribe
 - **layout**: esta etiqueta se emplea para especificar el formato de las trazas en la salida
- **root**: "logger" genérico. el logger es el elemento que va a escribir las trazas
 - **level**: indica el nivel de trazas que se quieren mostrar. Puede tomar los valores: ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF, en orden ascendente de prioridad
 - **appender-ref**: define el "appender" que se empleará para imprimir las trazas. Se pueden emplear varios "appenders" para el mismo logger

Este es un ejemplo muy sencillo en el que sólo se empleará el logger por defecto, pero se pueden definir otros loggers en el fichero, mediante la etiqueta `<logger>`, que contiene la misma información que el elemento `<root>`. Más adelante se verá un ejemplo de cómo hacer esto.

Añadir las trazas a la aplicación

Ya tenemos todo lo necesario para utilizar **log4Net** en nuestra aplicación. Ahora sólo hay que modificar el servicio web para que deje trazas.

Para ello, en el fichero en el que se encuentre el método declarado como servicio (en nuestro caso, *Service1.asmx.vb*) se importa el espacio de nombres de **log4Net** al principio del fichero, mediante la línea *Imports log4net*.

A continuación se declara un logger en el método:

```
Dim logger As ILog = LogManager.GetLogger("primerEjemplo")
```

Y de esta manera tenemos un logger con el nombre "primerEjemplo". Este nombre tiene dos funcionalidades. Primero, ya se ha visto que en elemento **layout** del fichero de configuración se dice que imprima en la traza el nombre del logger que ha generado el mensaje, con el elemento **%logger**. En este caso, el nombre que empleemos al crear el logger será el que se imprima (en nuestro caso, "primerEjemplo"). En segundo lugar, si el nombre dado al crear el logger coincide (se verá más claro más adelante) con el nombre de un logger declarado en el fichero de configuración, se utilizará ese logger en la aplicación. Ahora que tenemos el logger, hay que añadir las trazas. El código obtenido finalmente será algo así:

```

Imports System.Web.Services
Imports log4net

[...]

<WebMethod()> _
Public Function AutentiaEjemploWebService(ByVal Nombre As String) _
    As String
    Dim logger As ILog = LogManager.GetLogger("primerEjemplo")
    logger.Debug("Mensaje de nivel DEBUG")
    logger.Info("Mensaje de nivel INFO")
    logger.Warn("Mensaje de nivel WARN")
    logger.Error("Mensaje de nivel ERROR")
    logger.Fatal("Mensaje de nivel FATAL")

    Return "Hola " & Nombre
End Function
[...]

```

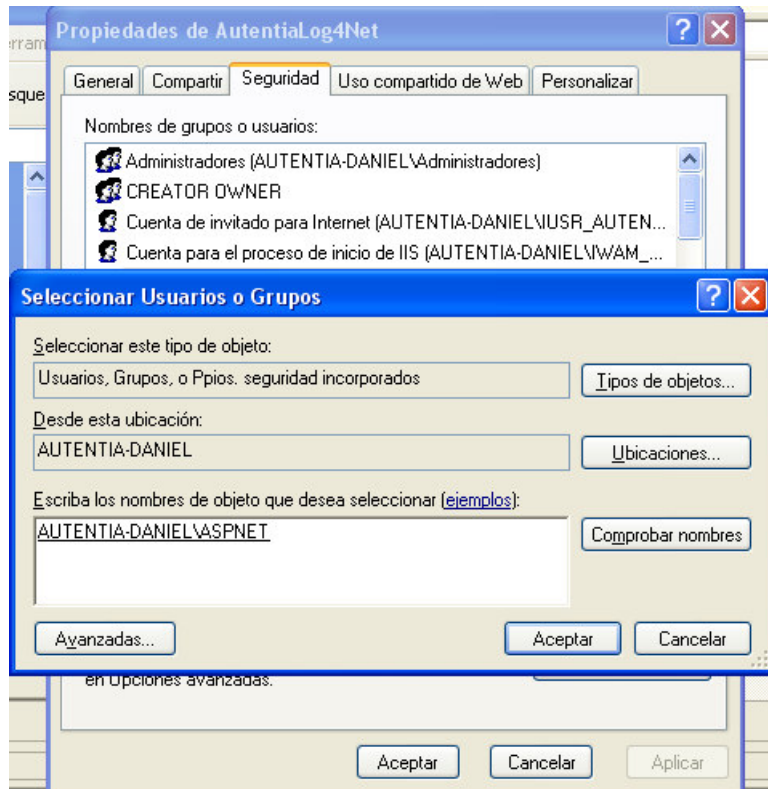
Bueno pues... parece que ya está todo... ejecutamos y... no funciona ¿POR QUÉ?

Conceder permiso de escritura

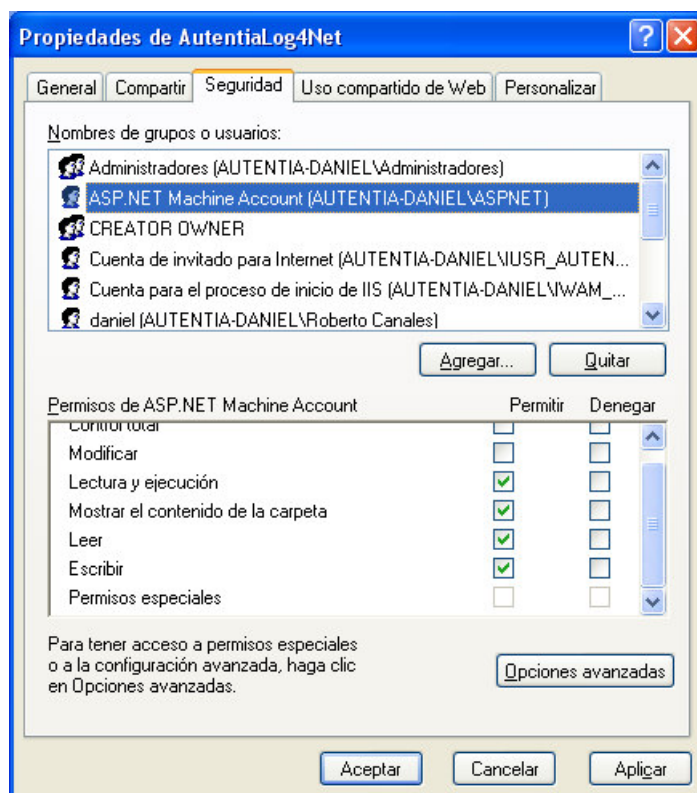
Las aplicaciones de .NET son ejecutadas por un usuario especial, el usuario ASPNET, que no tiene privilegios de escritura en el servidor. Por tanto, si queremos que puedan dejarse las trazas en un fichero, hay que darle al usuario ASPNET los permisos necesarios para escribir en la carpeta donde queramos guardar los logs. En nuestro caso, los

vamos a guardar en la raíz de la aplicación

Seleccionamos el usuario ASPNET...

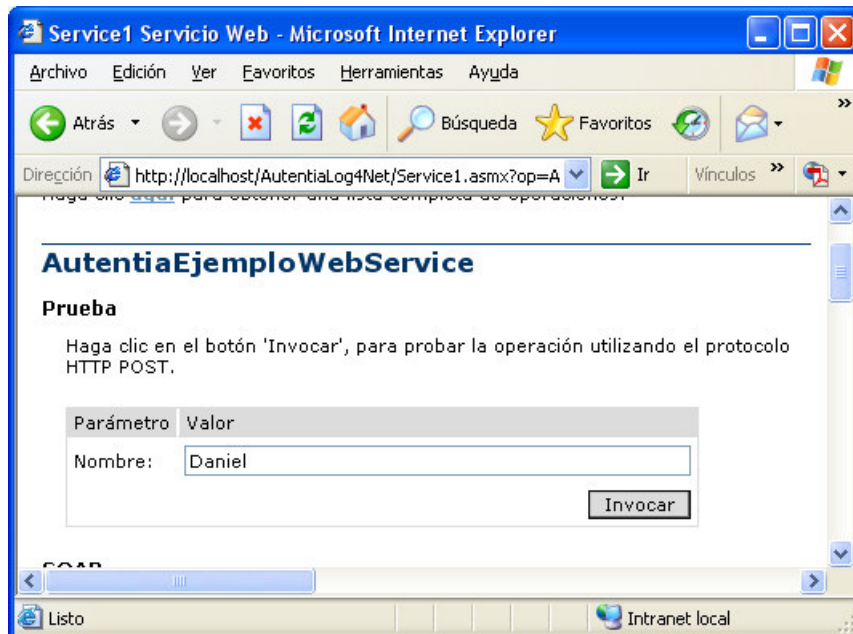


Y nos aseguramos de que tenga permiso de escritura (el resto de permisos necesarios ya los tendrá asignados)

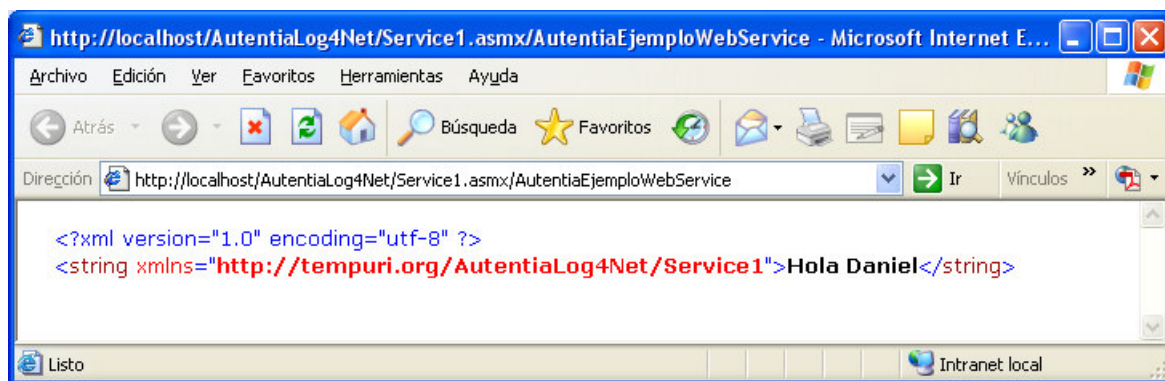


Segundo Intento

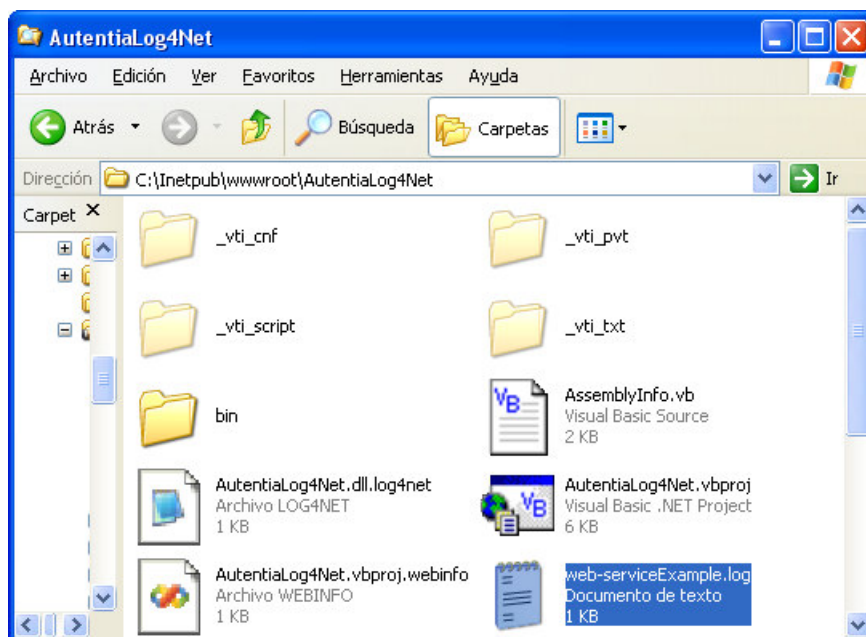
Una vez concedidos los permisos necesarios, invocamos otra vez el servicio...



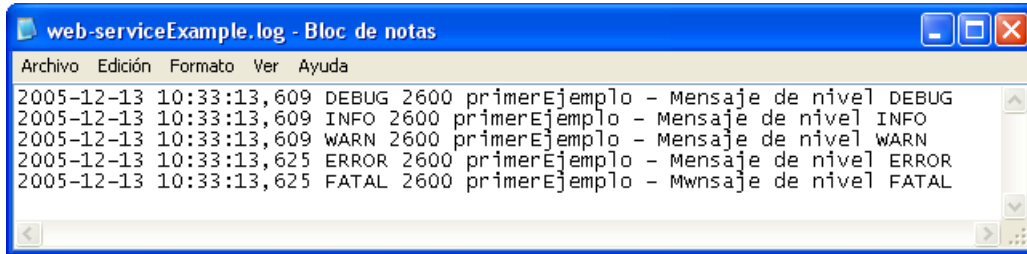
Obtenemos la respuesta...



Y esta vez SÍ obtenemos el fichero de log correctamente creado:



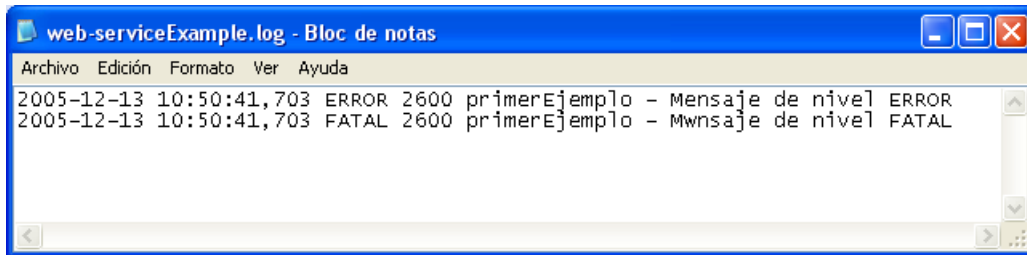
El contenido del fichero es:



Es decir, todas las trazas que mandamos imprimir.

Otras pruebas

Hemos visto que pasaría si fijáramos el nivel de traza a DEBUG, pero ¿y si cambiáramos el fichero de configuración para que usara el nivel ERROR? ¿Cuál habría sido el resultado? En ese caso, el contenido del fichero contendría:



Esto es así porque **log4net** sólo imprime las trazas de nivel superior o igual al asignado. El orden de menor a mayor prioridad que se aplica a los niveles es:

- DEBUG (Menos prioritaria)
- INFO
- WARN
- ERROR
- FATAL (Más prioritaria)

Por tanto, al poner el nivel en ERROR obtendremos trazas de nivel ERROR y de nivel FATAL, que es la única que está por encima de ERROR

¿Y si lo hubiéramos puesto a OFF? Entonces no se habría generado ningún fichero, puesto que le habríamos indicado a **log4net** que no deben mostrarse trazas de ningún tipo.

Estas pruebas nos han servido además para comprobar que sólo con cambiar el fichero de configuración los resultados se notan de manera inmediata en la aplicación sin necesidad de cambiar nada más

Empleando varios loggers

Ya hemos dicho que se pueden declara varios loggers en un mismo fichero de configuración. Ahora veremos un ejemplo de como se hace y como funciona.

Para empezar, modificaremos el fichero de configuración para incluir el segundo logger, y usaremos un segundo appender para ilustrar mejor el ejemplo. Con todos estos cambios, el fichero de configuración queda así:

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="LogFile" type="log4net.Appender.FileAppender">
    <file value="./web-serviceExample.log" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <appendToFile value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date %level %thread %logger - %message%newline" />
    </layout>
  </appender>
  <appender name="OtroLogFile" type="log4net.Appender.FileAppender">
    <file value="./pruebas-dos-loggers.log" />
    <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
    <appendToFile value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date %level %thread %logger - %message%newline" />
    </layout>
  </appender>
</root>
<level value="DEBUG" />
```

```

    <appender-ref ref="LogFile" />
  </root>
  <logger name="segundoLogger">
    <level value="ERROR" />
    <appender-ref ref="OtroLogFile" />
  </root>
</log4net>

```

Ahora tendremos que modificar el código del servicio web para que emplee el segundo logger:

```

<WebMethod(> _
Public Function AutentiaEjemploWebService(ByVal Nombre As String) _
As String
    Dim logger As ILog = LogManager.GetLogger("primerEjemplo")
    Dim segundoLogger As ILog = LogManager.GetLogger("segundoLogger")

    logger.Debug("Mensaje de nivel DEBUG")
    logger.Info("Mensaje de nivel INFO")
    logger.Warn("Mensaje de nivel WARN")
    logger.Error("Mensaje de nivel ERROR")
    logger.Fatal("Mensaje de nivel FATAL")

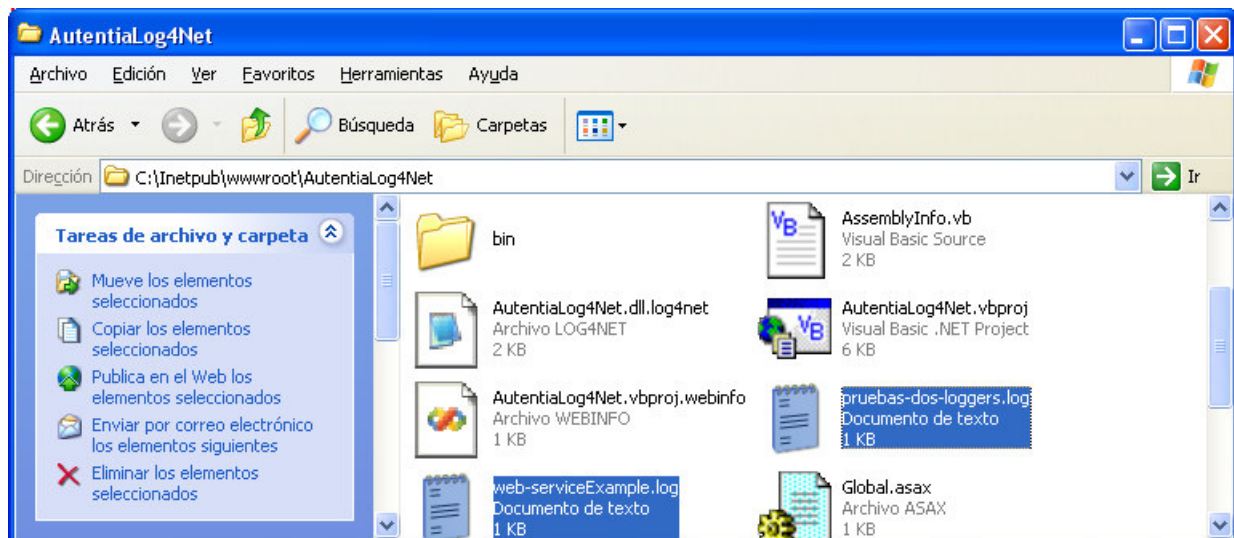
    segundoLogger.Debug("Mensaje de nivel DEBUG del segundo logger")
    segundoLogger.Info("Mensaje de nivel INFO del segundo logger")
    segundoLogger.Warn("Mensaje de nivel WARN del segundo logger")
    segundoLogger.Error("Mensaje de nivel ERROR del segundo logger")
    segundoLogger.Fatal("Mensaje de nivel FATAL del segundo logger")

    Return "Hola " & Nombre
End Function

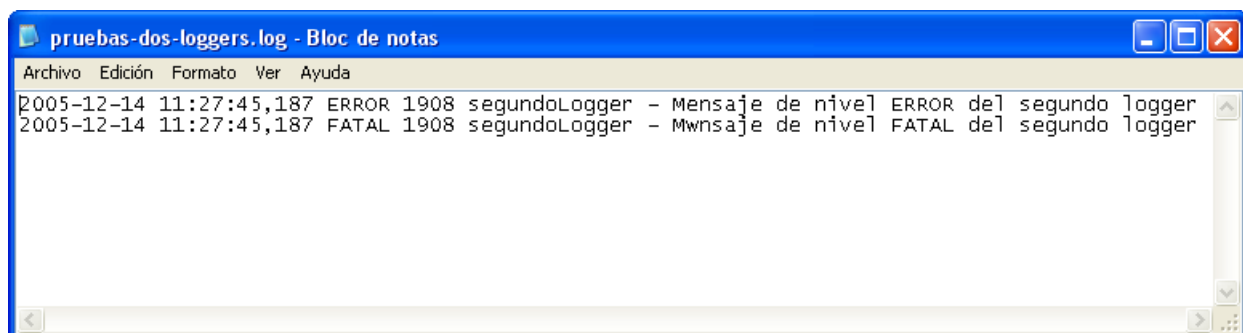
```

Como se observa, el segundo logger usa el mismo nombre que el declarado en el fichero de configuración, por lo que utilizará el appender y el nivel declarados por él. Como no hay ningún logger con el nombre "primerEjemplo", éste tomará los valores determinados en el logger **root**.

Si invocamos ahora el servicio como se ha visto anteriormente, tenemos los siguientes resultados en cuanto a ficheros de log se refiere:

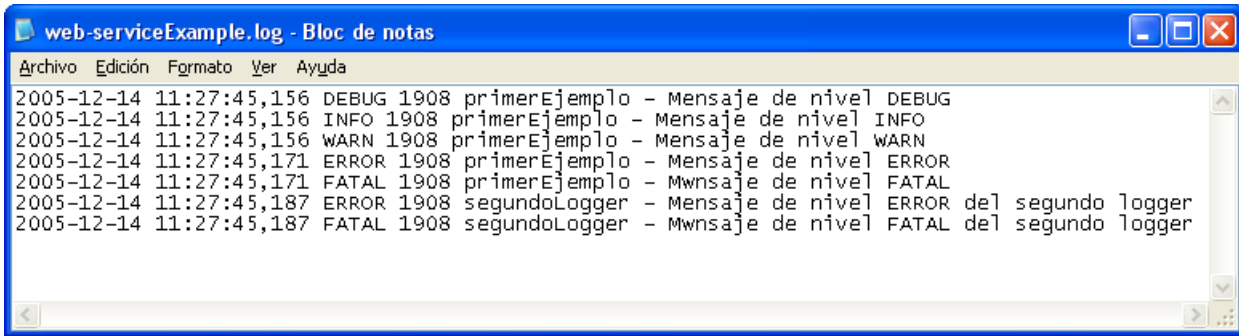


Como se ve, se han creado los dos ficheros de log. Si abrimos el fichero nuevo, vemos que contiene lo siguiente:



Es decir, lo esperado, los mensajes de nivel ERROR y FATAL, pues son los que concuerdan con la configuración del segundo logger

Si abrimos ahora el otro fichero de log, vemos lo siguiente:



```

web-serviceExample.log - Bloc de notas
Archivo Edición Formato Ver Ayuda
2005-12-14 11:27:45,156 DEBUG 1908 primerEjemplo - Mensaje de nivel DEBUG
2005-12-14 11:27:45,156 INFO 1908 primerEjemplo - Mensaje de nivel INFO
2005-12-14 11:27:45,156 WARN 1908 primerEjemplo - Mensaje de nivel WARN
2005-12-14 11:27:45,171 ERROR 1908 primerEjemplo - Mensaje de nivel ERROR
2005-12-14 11:27:45,171 FATAL 1908 primerEjemplo - Mwnsaje de nivel FATAL
2005-12-14 11:27:45,187 ERROR 1908 segundoLogger - Mensaje de nivel ERROR del segundo logger
2005-12-14 11:27:45,187 FATAL 1908 segundoLogger - Mwnsaje de nivel FATAL del segundo logger
  
```

Se puede observar que en él están los mensajes del primer logger, tal y como esperábamos, pero que también están los del segundo logger, cuando éste no especifica más que el uso de un `append`...

Esto se debe a que los loggers se organizan de manera jerárquica. Por ejemplo, todos los loggers que se declares descenden de **root**, por tanto, aunque el nivel de trazas deseado se sobrescribe, los `appenders` usados se heredan, y el resultado es que, aunque no explícitamente, el segundo logger declara dos `appenders`: el suyo y el que hereda de **root**. También pueden declararse hijos de otros loggers. Si declaramos un logger con el nombre "unLogger.otroLogger" tanto en el fichero de configuración como en el código del servicio web, heredará los valores que se declaren en "unLogger".

Por ejemplo, si añadimos al fichero de configuración el fragmento:

```

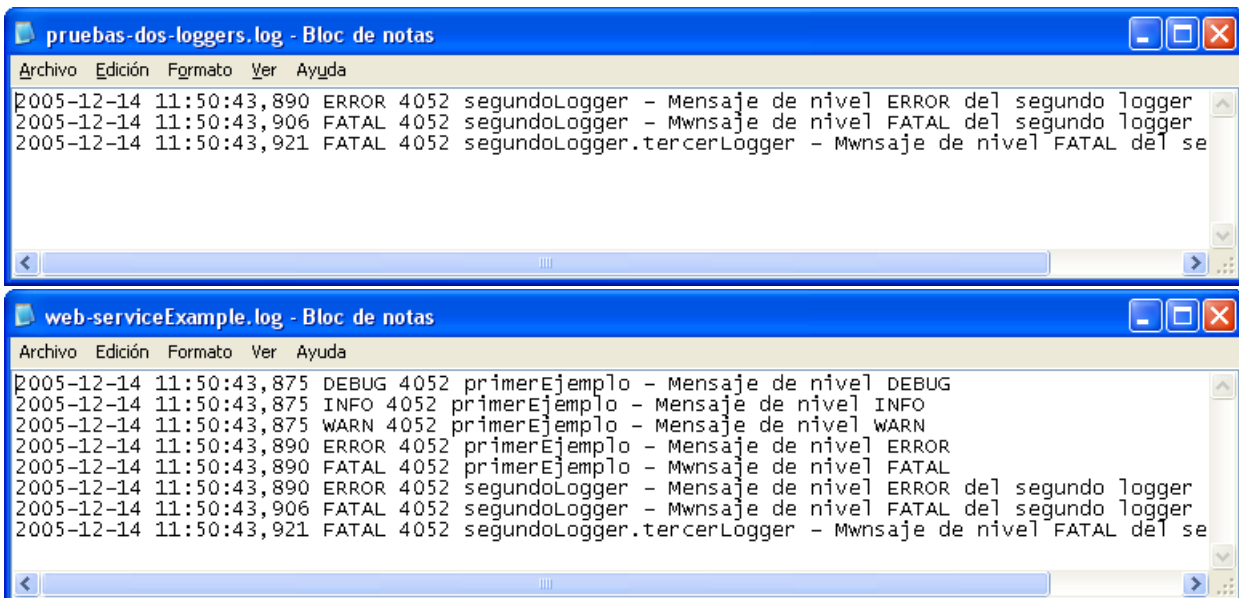
<logger name="segundoLogger.tercerLogger">
  <level value="FATAL" />
</logger>
  
```

y al código le añadimos las líneas siguientes:

```

[...]  
Dim tercerLogger As ILog = LogManager.GetLogger("segundoLogger.tercerLogger")  
[...]  
tercerLogger.Error("Mensaje de nivel ERROR del segundo logger")  
tercerLogger.Fatal("Mwnsaje de nivel FATAL del segundo logger")  
[...]  
  
```

y lo ejecutamos, vemos que, aunque no le hemos especificado ningún `append`, va a heredar los de "segundoLogger" e imprimirá en sus mismos `appenders`:



```

pruebas-dos-loggers.log - Bloc de notas
Archivo Edición Formato Ver Ayuda
2005-12-14 11:50:43,890 ERROR 4052 segundoLogger - Mensaje de nivel ERROR del segundo logger
2005-12-14 11:50:43,906 FATAL 4052 segundoLogger - Mwnsaje de nivel FATAL del segundo logger
2005-12-14 11:50:43,921 FATAL 4052 segundoLogger.tercerLogger - Mwnsaje de nivel FATAL del se

web-serviceExample.log - Bloc de notas
Archivo Edición Formato Ver Ayuda
2005-12-14 11:50:43,875 DEBUG 4052 primerEjemplo - Mensaje de nivel DEBUG
2005-12-14 11:50:43,875 INFO 4052 primerEjemplo - Mensaje de nivel INFO
2005-12-14 11:50:43,875 WARN 4052 primerEjemplo - Mensaje de nivel WARN
2005-12-14 11:50:43,890 ERROR 4052 primerEjemplo - Mensaje de nivel ERROR
2005-12-14 11:50:43,890 FATAL 4052 primerEjemplo - Mwnsaje de nivel FATAL
2005-12-14 11:50:43,890 ERROR 4052 segundoLogger - Mensaje de nivel ERROR del segundo logger
2005-12-14 11:50:43,906 FATAL 4052 segundoLogger - Mwnsaje de nivel FATAL del segundo logger
2005-12-14 11:50:43,921 FATAL 4052 segundoLogger.tercerLogger - Mwnsaje de nivel FATAL del se
  
```

Gracias a esta organización jerárquica, aparecen posibilidades muy atractivas y de implementación sencilla. Por ejemplo, se puede decir que todo en la aplicación deje trazas en fichero declarando este `append` en el elemento **root**, pero se puede hacer de manera muy fácil que una clase deje además trazas en base de datos, creando un logger que herede de **root** (por lo que escribirá en fichero) y que utilice otro `append`.

Conclusión

Ya hemos dado los primeros pasos con **log4net**. Como habéis visto es muy sencillo, y os animo a que sigáis investigando y descubriendo todo el potencial de este producto, puesto que con él podréis no sólo dejar trazas en un fichero, sino que seréis capaces de escribirlas en base de datos, en pantalla... prácticamente donde queráis.

Y ya sabéis que en mi empresa, [Autentia](#), estaremos encantados de echaros una mano, así que no dudéis en poneros en contacto con nosotros si necesitáis soluciones para vuestros problemas de desarrollo

Si desea contratar formación, consultoría o desarrollo de piezas a medida puede contactar con

Creatividad Internet

[Autentia S.L.](#) Somos expertos en:
J2EE, C++, OOP, UML, Vignette, Creatividad ..
y muchas otras cosas

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados ([También ver todos](#))

Nombre Corto	Descripción
Pruebas Web con JWebUnit	Os mostramos como automatizar las pruebas de caja negra (desde el punto de vista de usuario final) de vuestro Web con el Framework gratuito JWebUnit. Esta técnica es perfecta para crear test de regresión de aplicaciones Web complejas.
Detección de errores Java con FindBugs	Os mostramos como instalar y utilizar FindBugs, una excelente herramienta para analizar, de un modo estático, posibles problemas en vuestro código Java
Test con JUnit	Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit
Consola de administración de Struts	En este tutorial aprenderemos a simplificar la gestión de Struts a través de una consola gráfica gratuita
Escritura log con Fichero UDP y JMS	Os mostramos ejemplos para cuantificar el coste de escritura de Logs por pantalla, fichero, UDP y JMS (describiendo como configurar el entorno)
Aplicaciones con el framework de Microsoft .NET	Ejemplo de desarrollo de una aplicación con el framework de Microsoft .NET (creación de un servicio de Encuestas Web)
Gestión de errores con IIS	Ismael caballero nos enseña como evitar que aparezcan pantallas genericas de error cuando trabajamos con IIS
Gestión de Errores con Bugzilla	Alejandro Pérez nos enseña como instalar (en Debian) y utilizar Bugzilla, una herramienta gratuita de gestión de errores.
Introducción a log4j	En un desarrollo Java es vital normalizar los logs para posteriormente poder depurar el funcionamiento de nuestra aplicación. Os mostramos como usar Log4J.
Introducción a ANT	En el mundo Java, la compilación, verificación e instalación de aplicaciones se ha normalizado con este potente paquete llamado ANT.

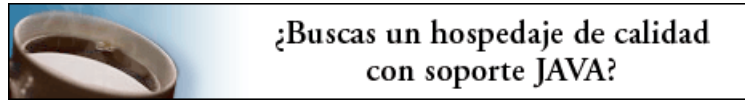
Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.

[Patrocinados por enredados.com Hosting en Castellano con soporte Java/J2EE](#)



www.AdictosAlTrabajo.com Optimizado 800X600