

# Utilizando NLog con MonoDevelop en .NET

por Martín A. O Márquez <xomalli@gmail.com>

Para cualquier aplicación de software a nivel producción es indispensable tener un componente que escriba los eventos más significativos en una bitácora. Esta acción que se conoce como logging (escribir en la bitácora) se define técnicamente como:

*“Una forma sistemática y controlada para obtener el estado de una aplicación en tiempo de ejecución.”*

Pensando en esto, los diseñadores de .NET incorporaron un mecanismo de **logging** de forma predeterminada dentro del ensamblado **System.Diagnostics**, en clases como *Trace*, *TraceListener*, *Swicth* y todas sus clases derivadas. Aunque el empleo de estas clases es efectivo, no deja de ser rudimentario y carecer de muchas funcionalidades que terminan siendo un límite.

Teniendo en cuenta esto, en el ecosistema .NET han surgido a lo largo de los años una cantidad de componentes propietarios y opensource para el logging de aplicaciones. Dentro de ese conjunto hay un componente que se destacó y es del que trataré de resumir en este tutorial: **Nlog**.

## ¿Qué es NLog?

**Nlog** (<http://nlog-project.org/>) es un componente open source de logging para .NET, que entre sus características se encuentran:

- Muy fácil de configurar.
- Extremadamente personalizable con plantilla (layouts)
- Altamente extensible

Hay tres características que se deben conocer antes de empezar su utilización:

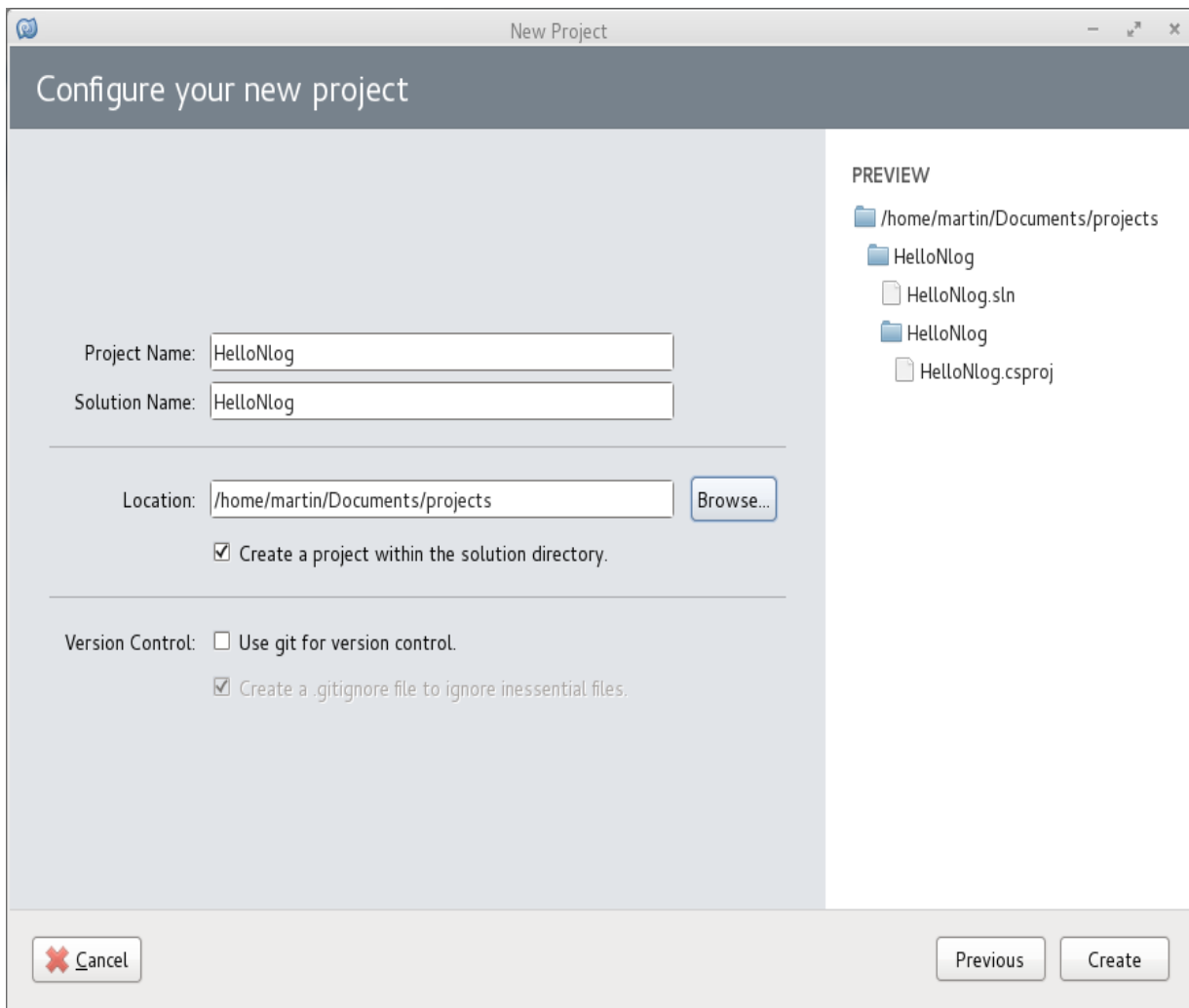
1. **Targets:** Se utilizan para enviar los mensajes hacia otro destino, entiéndase aquí un archivo, una base de datos, un email, la consola, un webservice, etc.
2. **Layouts:** Con los comandos de layout podemos definir la estructura o el molde de como acomodar la información escrita en un determinado target.
3. **Levels:** Es una forma de asignar una prioridad al mensaje, los niveles permitidos son los siguientes:
  1. **Fatal:** Sucedió algo que causo que el todo el sistema entero falle, se debe detener la ejecución.
  2. **Error:** Un problema ha ocurrido pero no es fatal, el sistema puede seguir funcionando.
  3. **Warn:** Un problema ha ocurrido, pero puede ser recuperable.
  4. **Info:** Mensajes de información muy útiles como cambios de estado, login/logout, etc.
  5. **Debug:** Se utiliza durante el desarrollo del sistema.
  6. **Trace:** Para indicar el inicio y final de una rutina.

Como un primer acercamiento a su utilización, escribí una aplicación de consola en C# que solicita una cadena de conexión, con esta cadena trata de conectarse a una base de datos PostgreSQL, si la cadena de conexión no es correcta se utiliza **Nlog** para enviar la excepción a consola, si logra conectarse solicita una consulta SELECT para ejecutar y mostrar los resultados en la consola. Aquí

de nuevo si ocurre una excepción utiliza **Nlog** para notificarla.

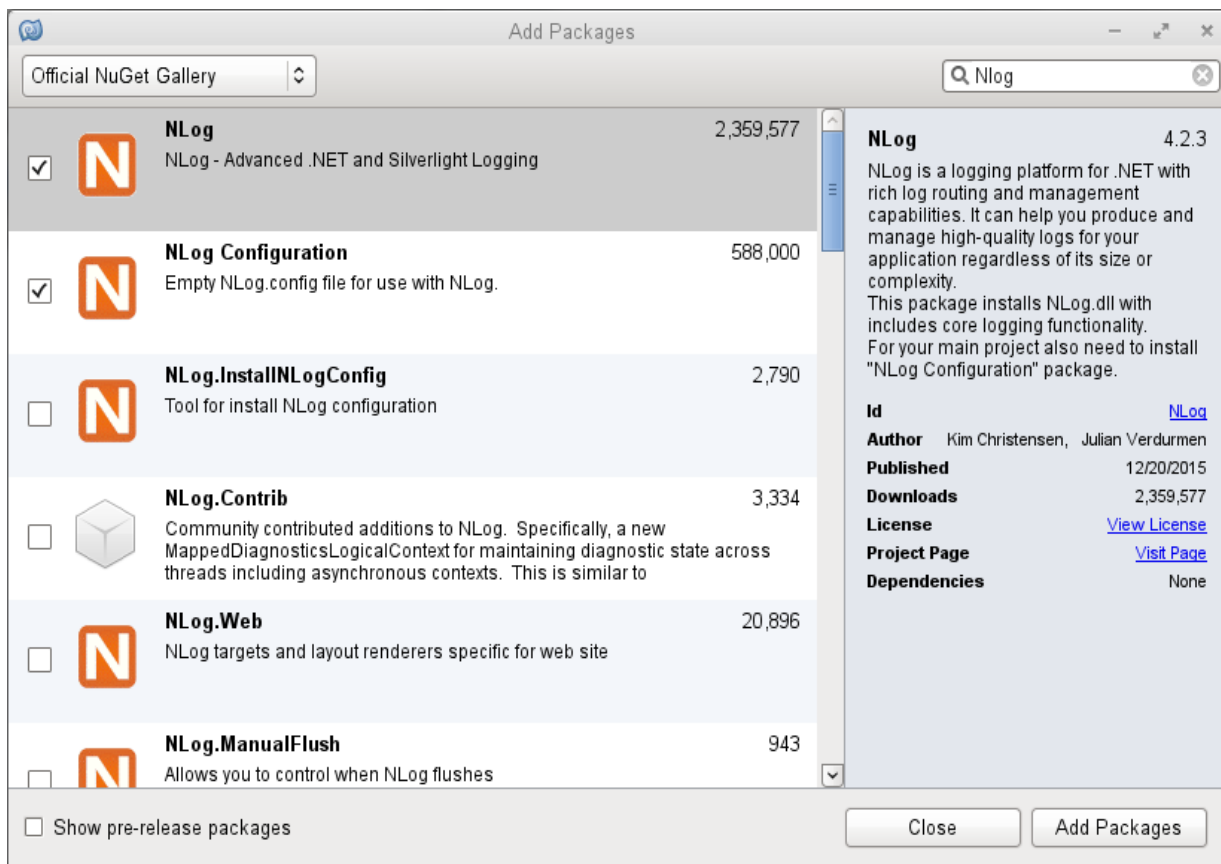
1-. Ejecutar **Monodevelop** y seleccionar un proyecto de consola y nombrarlo como **HelloNlog**

**Fig. 1 Crear un proyecto llamado HelloNlog**



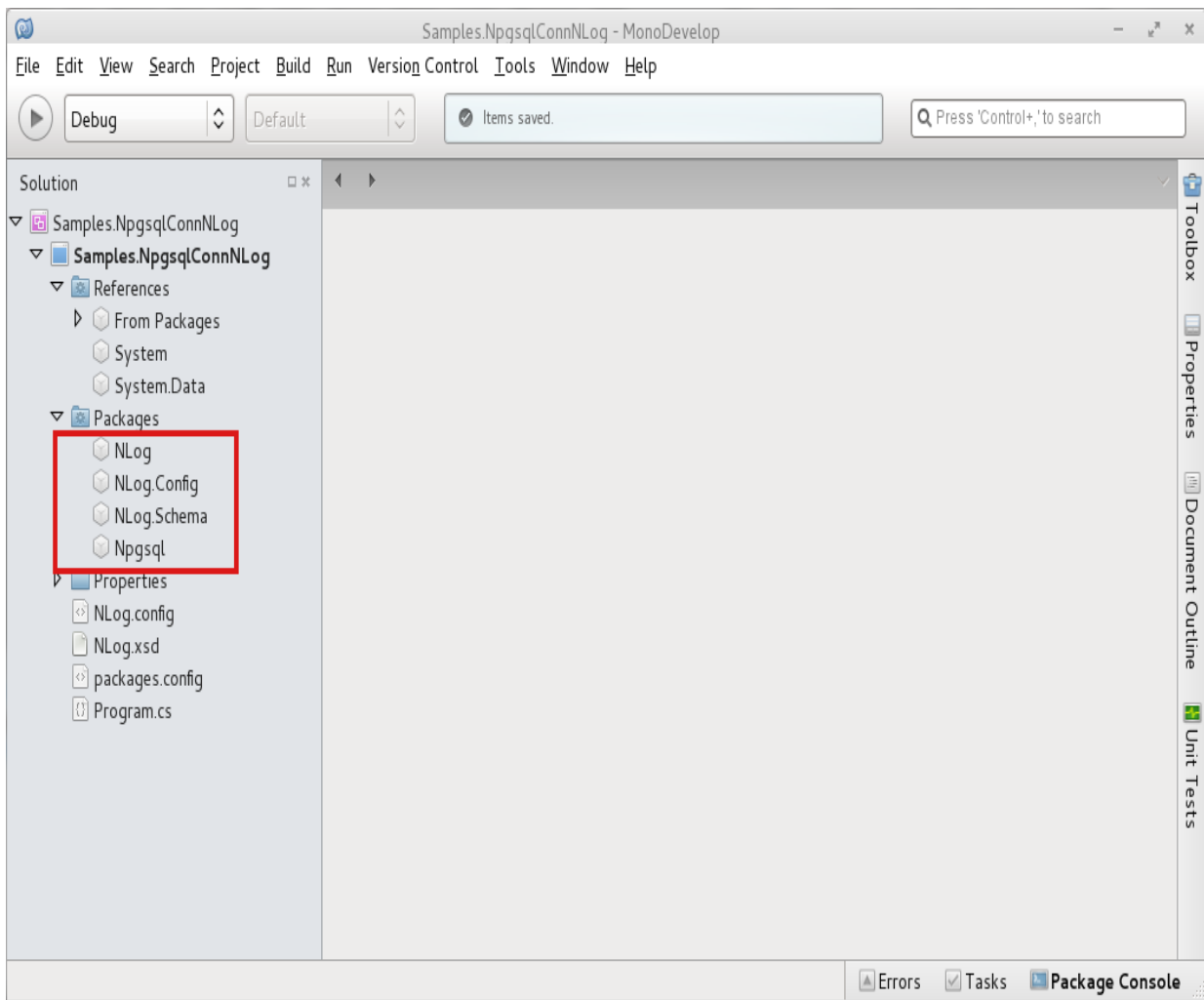
2-. Dentro del **Solution Explorer** has click con el botón derecho y has click en **Add Packages** , entonces aparecerá la pantalla **Add Packages**, ya en esa pantalla usa el buscador para encontrar el paquete **Nlog**, y seleccionar los paquetes: **Nlog**, **Nlog Configuration** y **Npgsql** respectivamente, presionar el botón **Add Packages** para agregar los ensamblados al proyecto.

**Fig. 2 Seleccionar los paquetes NLog y NLog Configuration**



3-. Ahora que ya se tiene una estructura en la solución como se muestra en la siguiente imagen:

**Fig. 3 La estructura de la solución con los ensamblados.**



4-. Agregar al proyecto una clase llamada **NloggerWrapper** y escribir el siguiente código:

```
using System;
using System.Data;
using NLog;
using Npgsql;

namespace HelloNlog
{
    public static class NLoggerWrapper
    {
        static Logger logger = LogManager.GetCurrentClassLogger();
        public static void ConnectionStateChanged(object
sender, StateChangeEventArgs args)
        {
            logger.Info ("Connection Original state: " + args.OriginalState);
            logger.Info ("Connection Current state: " + args.CurrentState);
        }

        public static void Warn(string message){
            logger.Warn (message);
        }

        public static void Trace(string message){
            logger.Trace (message);
        }
    }
}
```

```

    }

    public static void LogException(Exception ex)
    {
        if (ex is ArgumentException)
            logger.Warn (ex.Message);
        else
            if (ex is NpgsqlException)
                logger.Error (ex.Message);
            else
                logger.Fatal (ex.Message);
    }
}
}

```

5-. Bien ahora hay que completar el código de la clase Program con el siguiente código para completar el programa:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;
using System.Data;

namespace HelloNlog
{
    class Program
    {
        static NpgsqlConnection conn = null;
        static NpgsqlConnectionStringBuilder builder = null;

        static void Main(string[] args)
        {
            Console.WriteLine ("- A simple demo for NLog trying to connect a
PostgreSQL Server -");
            Console.WriteLine (Environment.NewLine);
            bool keepGoing = true;
            do {
                try {
                    Console.WriteLine("Enter the Connection String or 'Q' to
quit > ");
                    string connString = Console.ReadLine();
                    if(connString.Equals("Q"))
                        keepGoing = false;
                    else
                    {
                        builder = new NpgsqlConnectionStringBuilder(connString);
                        conn = new NpgsqlConnection(builder.ConnectionString);
                        NLoggerWrapper.ConnectionStateChanged;
                        conn.Open();
                        if(conn.State == ConnectionState.Open)
                        {
                            Console.WriteLine("Please enter a SELECT query or
'Q' to quit >");
                            string commandText = Console.ReadLine();
                            if(commandText.Equals("Q"))
                                keepGoing = false;
                            else
                            {
                                if(!commandText.ToUpper().StartsWith("SELECT"))

```



```

<targets>
  <target xsi:type="File"
    name="fileLog"
    fileName="${basedir}/logs/${shortdate}.log"
    layout="${longdate} ${machinename} ${uppercase:${level}} ${message}" />

  <target xsi:type="ColoredConsole"
    name="coloredConsoleLog"
    layout="${message}"
    useDefaultRowHighlightingRules="true" />

  <target xsi:type="Console"
    name="consoleLog"
    layout="${longdate} ${machinename} ${message}" />
</targets>

<rules>
  <logger name="*" levels="Error,Fatal" writeTo="fileLog"></logger>
  <logger name="*" level="Trace" writeTo="consoleLog"></logger>
  <logger name="*" levels="Error,Warn,Info"
writeTo="coloredConsoleLog"></logger>
</rules>
</nlog>

```

## Targets

En este archivo de configuración defino tres **targets**, el primero hacia un archivo, el segundo hacia una consola con salida de color y el último hacia una consola de salida normal.

```

<targets>
  <target xsi:type="File"
    name="fileLog"
    fileName="${basedir}/logs/${shortdate}.log"
    layout="${longdate} ${machinename} ${uppercase:${level}} ${message}" />

  <target xsi:type="ColoredConsole"
    name="coloredConsoleLog"
    layout="${message}"
    useDefaultRowHighlightingRules="true" />

  <target xsi:type="Console"
    name="consoleLog"
    layout="${longdate} ${machinename} ${message}" />
</targets>

```

Para ver completa la lista de targets consultar el enlace:

<https://github.com/NLog/NLog/wiki/Targets>

## Layouts

Por cierta comodidad y porque así es la manera predeterminada de ver la información, puse los **layouts** de la siguiente manera:

```

<target xsi:type="File"
  name="fileLog"
  fileName="${basedir}/logs/${shortdate}.log"
  layout="${longdate} ${machinename} ${uppercase:${level}} ${message}" />

```

```

<target xsi:type="ColoredConsole"
        name="coloredConsoleLog"
        layout="{message}"
        useDefaultRowHighlightingRules="true"/>

<target xsi:type="Console"
        name="consoleLog"
        layout="{longdate} {machinename} {message}"/>

```

Se especifica un **layout** por cada uno de los **targets** Para más información de los **layouts** ver el siguiente enlace: <https://github.com/NLog/NLog/wiki/Layout-Renderers>

## Rules

Ahora la configuración para las rules, aquí con el '\*' le indico que ese nivel se use para todos los logs, únicamente en los niveles **Error** y **Fatal** escriban hacia el **target** llamado fileLog que es el **target** que escribe hacia un archivo de texto, en la segunda regla indico igual que para todos logs, los niveles Trace se escriban hacia el log llamado consoleLog que tiene la salida normal de consola y por último le indico que para los niveles **Warn** y **Info** escriban hacia el log de la consola con colores.

```

<rules>
  <logger name="*" levels="Error,Fatal" writeTo="fileLog"></logger>
  <logger name="*" level="Trace" writeTo="consoleLog"></logger>
  <logger name="*" levels="Error,Warn,Info"
writeTo="coloredConsoleLog"></logger>
</rules>

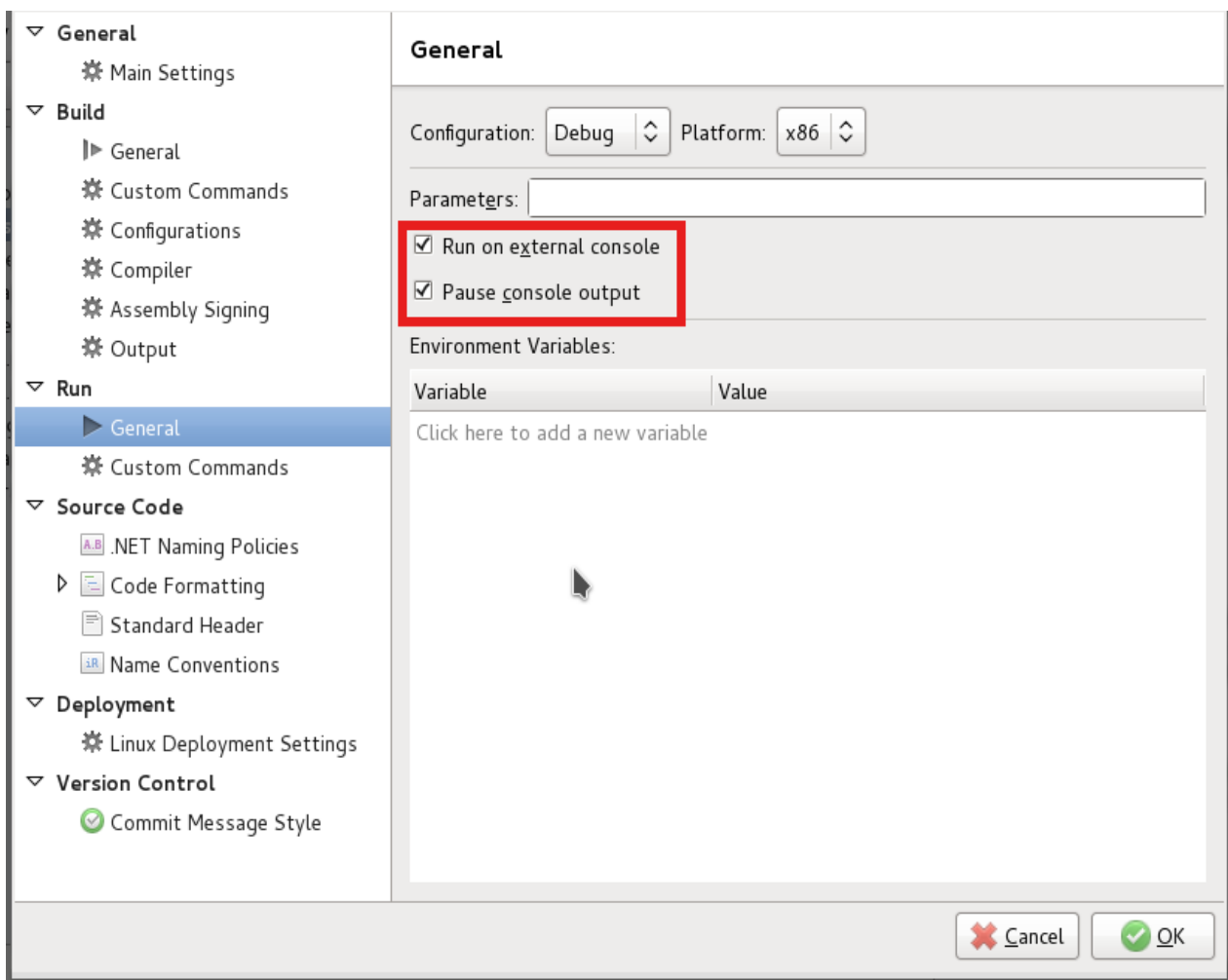
```

Para más información de las reglas ver el siguiente enlace:  
<https://github.com/nlog/NLog/wiki/Configuration-file#rules>

8-. Antes de ejecutar el programa, en el **Solution Explorer** haz click derecho sobre la solución después haz click en options, aparecerá la ventana **Project Options** ahí seleccionar las opciones **run on external console** y **pause console output**.

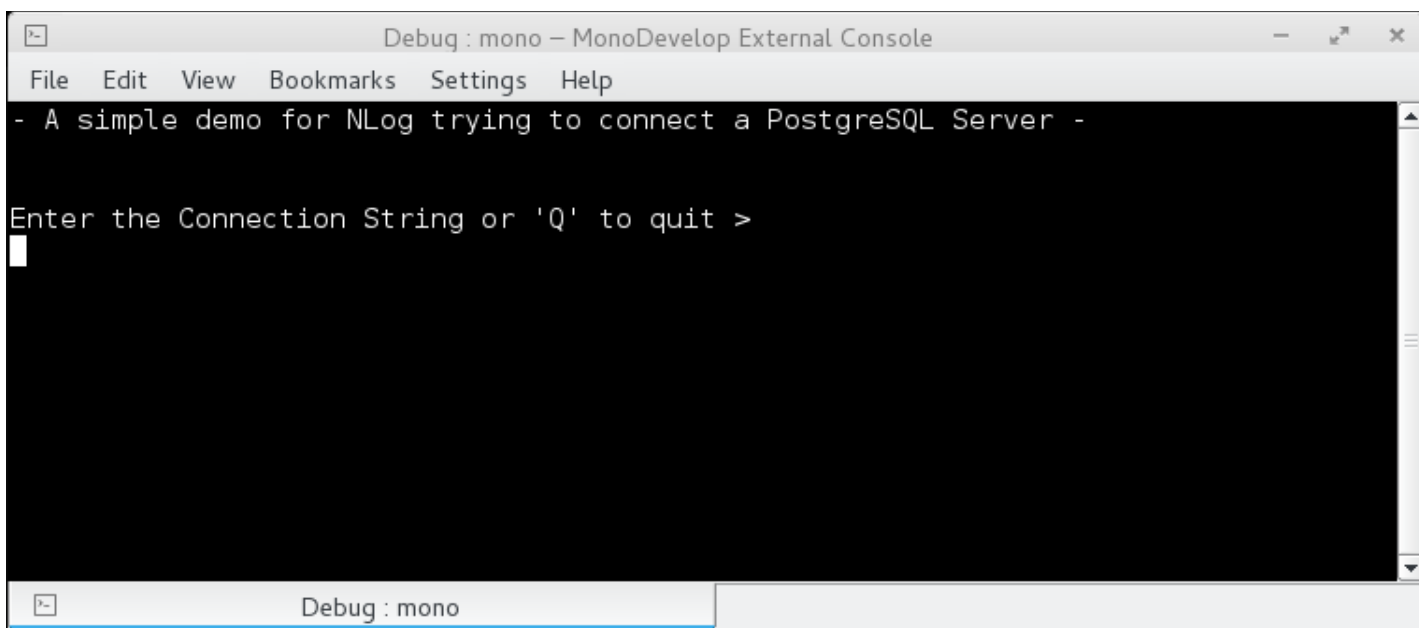


**Fig. 4 Opciones para ejecutar el proyecto.**



Bien al ejecutar el programa este solicita una cadena de conexión desde el inicio:

**Fig. 5 El programa solicita una cadena de conexión.**

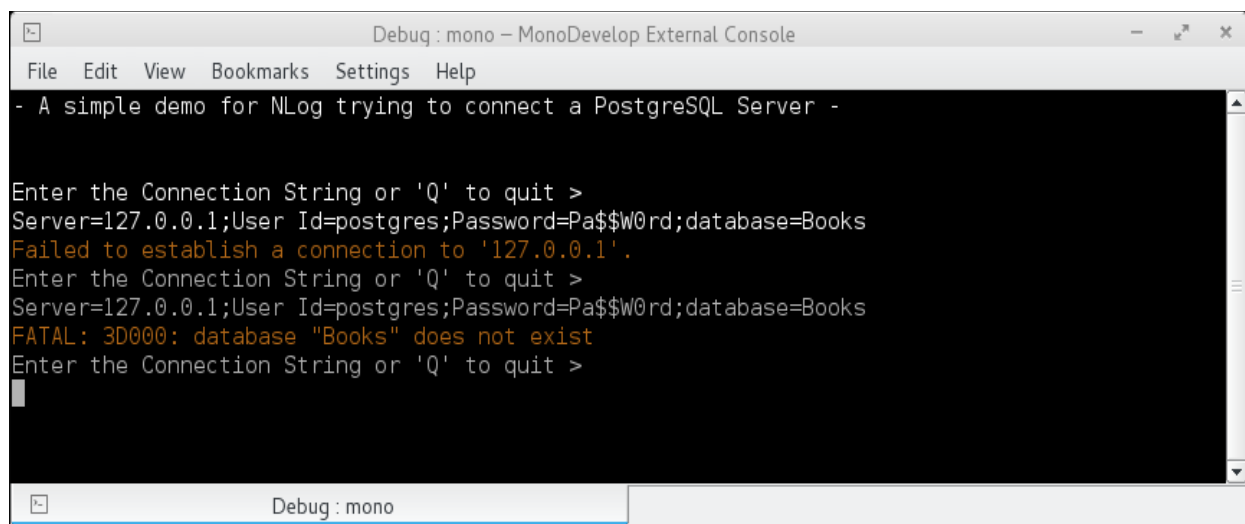


Errores como si la cadena de conexión no tiene un formato correcto, el servidor **Postgresql** esta abajo o no existe la base de datos, etc. Son encerrados dentro por un bloque **try/catch** y enviados al método **LogException** para que **Nlog** utilice el level correspondiente y lo mande hacia el target. Aquí el código del método **LogException** dentro de la clase **NLoggerWrapper**

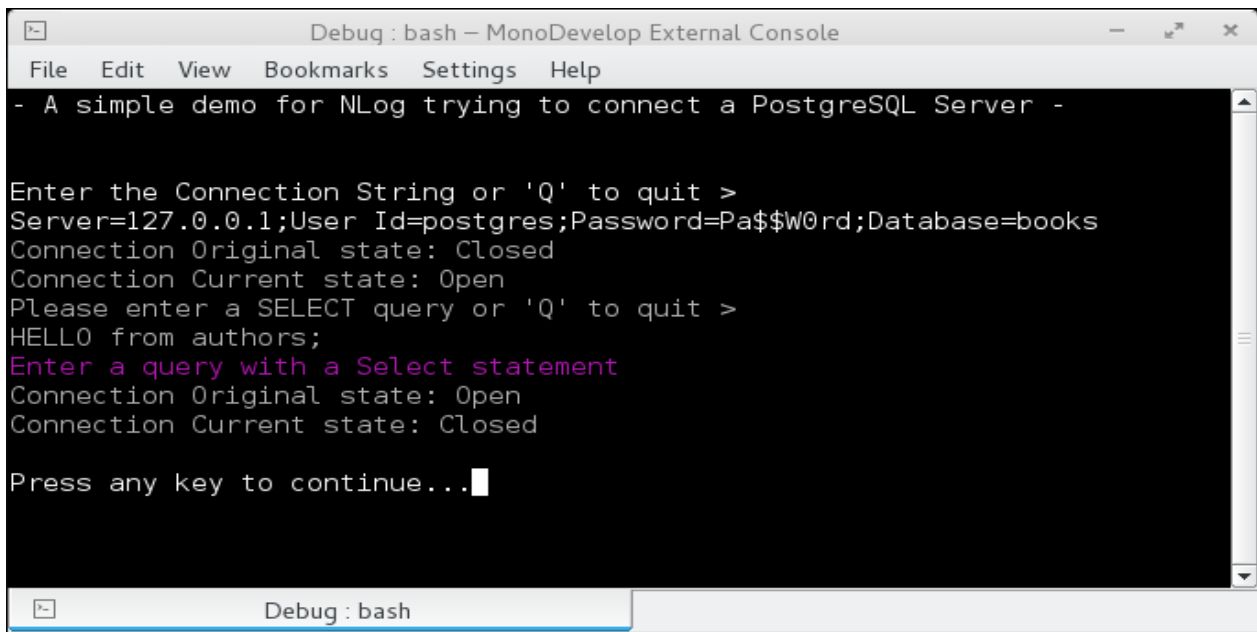
```
public static void LogException(Exception ex)
{
    if (ex is ArgumentException)
        logger.Warn (ex.Message);
    else
        if (ex is NpgsqlException)
            logger.Error (ex.Message);
        else
            logger.Fatal (ex.Message);
}
```

En este código dependiendo del tipo de excepción utiliza un nivel (level) de **Nlog** para escribir.

**Fig. 6 El programa con Nlog muestra las excepciones en la consola con color.**



**Fig. 7** Excepción atrapada por NLog que se muestra en color.



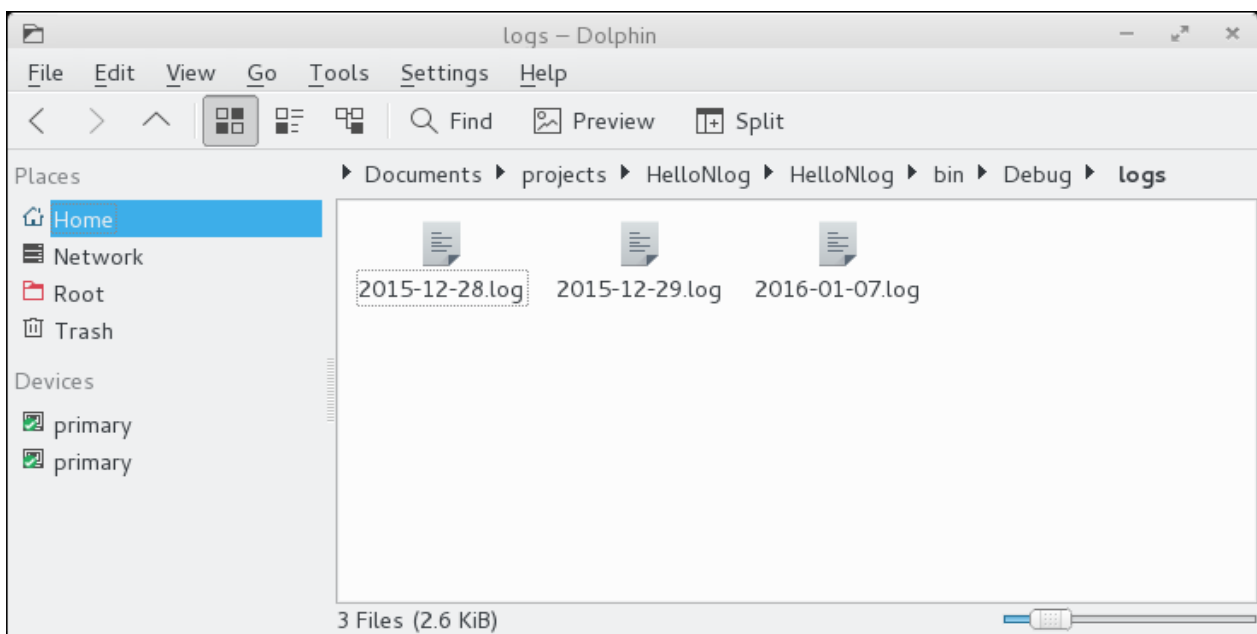
```
Debug : bash – MonoDevelop External Console
File Edit View Bookmarks Settings Help
- A simple demo for NLog trying to connect a PostgreSQL Server -

Enter the Connection String or 'Q' to quit >
Server=127.0.0.1;User Id=postgres;Password=Pa$$W0rd;Database=books
Connection Original state: Closed
Connection Current state: Open
Please enter a SELECT query or 'Q' to quit >
HELLO from authors;
Enter a query with a Select statement
Connection Original state: Open
Connection Current state: Closed

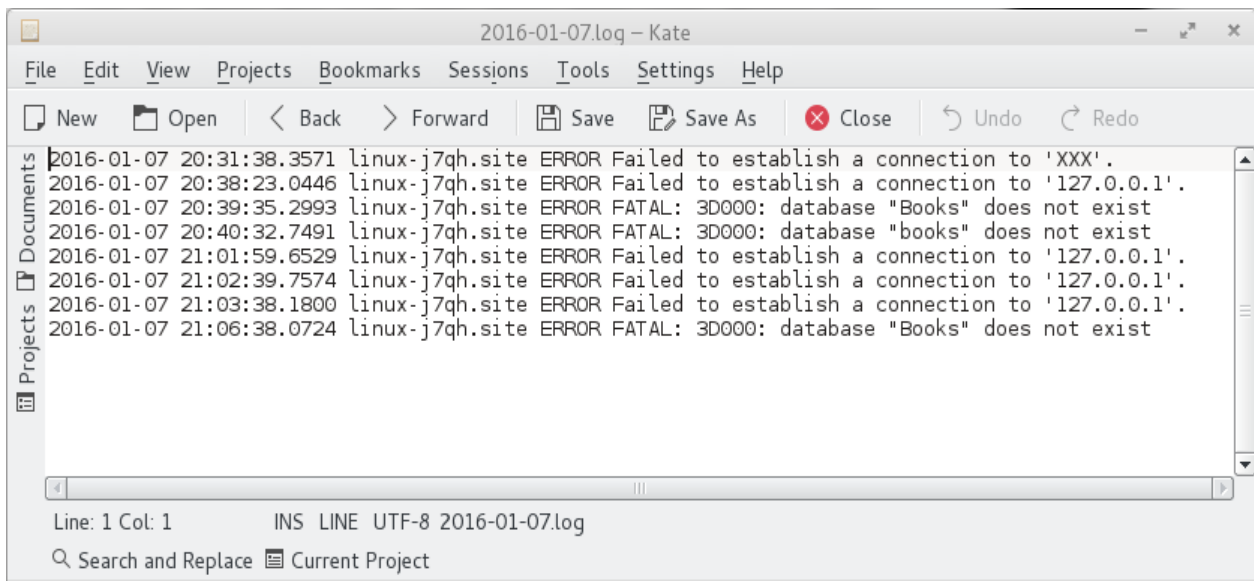
Press any key to continue...█
```

Cuando uno de los **target** como en este ejemplo esta dirigido a escribir en archivo se puede revisar la creación del archivo y posteriormente su contenido.

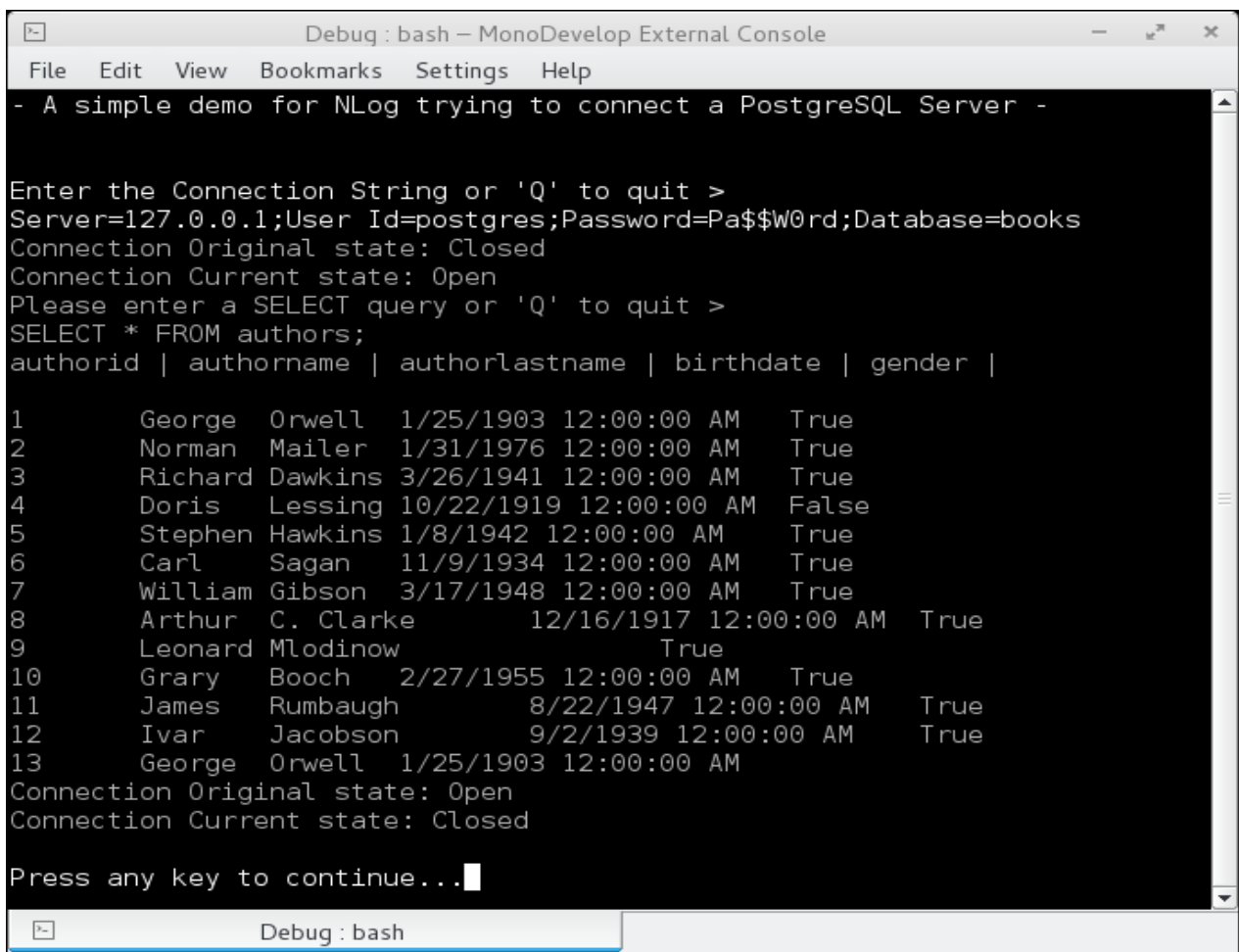
**Fig. 8** La creación de los archivos log con el target File.



**Fig. 9 El formato de los archivos log.**



**Fig. 10 La ejecución del programa sin errores**





## **Download el código fuente para Xamarin Studio o Visual Studio**

Este documento está protegido bajo la licencia de documentación libre *Free Documentacion License* del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»