

Compresión y descompresión de flujos(streams) en .NET

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

Una capacidad muy útil en cualquier Framework es la de comprimir y descomprimir archivos o flujos de datos, ejemplos aplicados de esta capacidad los tenemos en las aplicaciones que intercambian datos a través de la red como el caso de comprimir el [viewstate](#) en las peticiones y respuestas del servidor al cliente en una aplicación ASP.NET o si necesitamos reducir el tamaño de los objetos JSON que viajan de una aplicación hacia un servicio web y viceversa.

.NET proporciona las clases [GzipStream](#) y [DeflateStream](#) que implementan el algoritmo de compresión estándar gzip que está libre de patente. De hecho la diferencia entre ambas clases es que la clase **GzipStream** utiliza la especificación *gzip* descrita en el [RFC](#) 1952 que reduce el tamaño de los datos utilizando un código Lempel-Ziv (LZ77) con un CRC de 32 bits, lo que significa que se puede incluir encabezados (headers) con información extra que puede ser utilizada por cualquier herramienta de descompresión o por el comando gunzip de Linux o UNIX.

En resumen si los datos comprimidos van a hacer descomprimidos por una aplicación .NET lo recomendable es usar la clase **DeflateStream**, de lo contrario si van a hacer descomprimidos por otra aplicación usar la clase **GzipStream**.

Para mostrar la utilización de la clase **DeflateStream** escribí el siguiente programa que ejecuta la acción de comprimir o descomprimir dependiendo de la extensión del archivo que le pasemos como parámetro. (Este código sirve para ambos flujos solo hay que reemplazar **DeflateStream** por **GzipStream**)

```
using System;
using System.IO;
using System.IO.Compression;

namespace GzipGunzip
{
    class MainClass
    {
        public static void Main (string[] args)
        {
            if (args.Length == 1)
            {
                try{
                    FileInfo fis = new FileInfo(args[0]);
                    Console.WriteLine("Input file {0}", fis.FullName);
                    if(string.Compare(fis.Extension, ".gz") == 0)
                        Uncompress(fis.FullName);
                    else
                        Compress(fis.FullName);
                }catch(Exception ex){
                    Console.WriteLine(ex.Message);
                }
            }
        }
    }
}
```

```

    }
}
else
    Console.WriteLine("\nUsage: GzipGunzip filename\n");
}

static void Uncompress (string filename)
{
    Console.WriteLine ("Uncompressing .... {0} ", filename);
    string destFile = filename.Substring (0, filename.Length - 3);
    FileStream inputStream = new FileStream (filename
                                           , FileMode.Open
                                           , FileAccess.Read);
    using (FileStream outputStream = new FileStream (destFile
                                                    , FileMode.Create
                                                    , FileAccess.Write)) {
        using(DeflateStream zipStream = new DeflateStream (inputStream
                                                         , CompressionMode.Decompress)){
            int inputByte = zipStream.ReadByte ();
            while (inputByte != -1) {
                outputStream.WriteByte ((byte)inputByte);
                inputByte = zipStream.ReadByte ();
            }
        }
    }

    Console.WriteLine("output: {0}",destFile);
}

static void Compress (string filename)
{
    Console.WriteLine ("Compressing .... {0} ", filename);
    string destFile = filename + ".gz";
    byte[] buffer = null;
    using (FileStream inputStream = new FileStream(filename
                                                    ,FileMode.Open
                                                    ,FileAccess.Read)) {
        buffer = new byte[inputStream.Length];
        inputStream.Read (buffer, 0, buffer.Length);
    }

    Console.WriteLine ("bytes read: {0}", buffer.Length);
    using (FileStream outputStream = new FileStream(destFile
                                                    ,FileMode.Create
                                                    ,FileAccess.Write)) {
        using(DeflateStream zipStream = new DeflateStream (outputStream
                                                         , CompressionMode.Compress)){
            zipStream.Write (buffer, 0, buffer.Length);
        }
    }

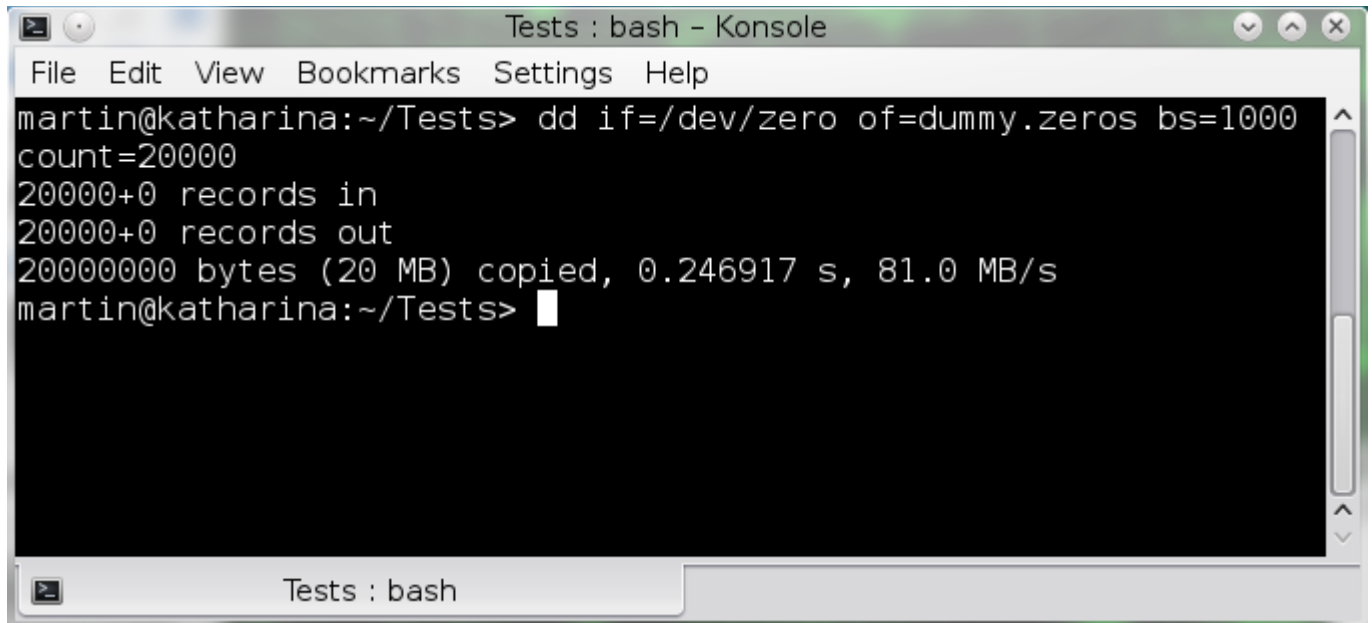
    Console.WriteLine("bytes written: {0}",buffer.Length);
}
}
}
}

```

Ahora para probar el código, voy a crear un archivo llamado *dummy.zeros* con un tamaño de 20 megas para pasárselo como parámetro al programa.

Este archivo lo creo con el comando **dd** en Linux

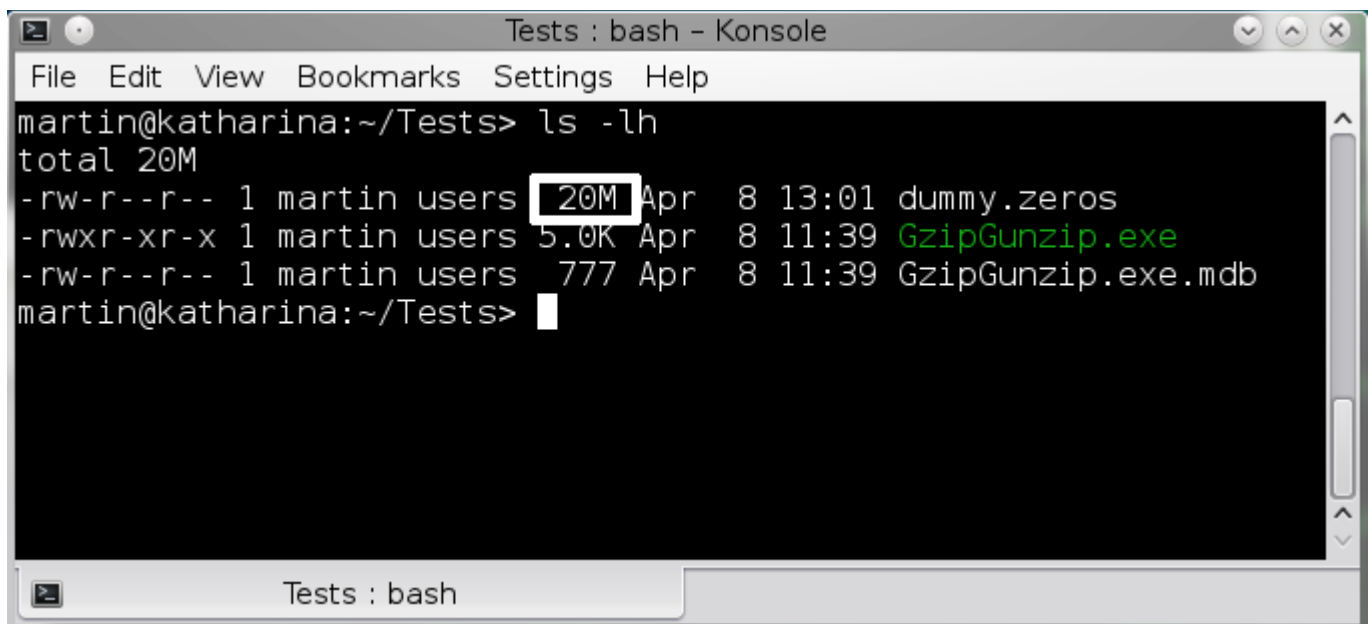
```
dd if=/dev/zero of=dummy.zeros bs=1000 count=20000
```



A terminal window titled "Tests : bash - Konsole" showing the execution of the `dd` command. The command is `dd if=/dev/zero of=dummy.zeros bs=1000 count=20000`. The output shows that 20,000 records were written, totaling 20,000,000 bytes (20 MB) at a speed of 81.0 MB/s. The prompt is `martin@katharina:~/Tests>`.

```
martin@katharina:~/Tests> dd if=/dev/zero of=dummy.zeros bs=1000
count=20000
20000+0 records in
20000+0 records out
20000000 bytes (20 MB) copied, 0.246917 s, 81.0 MB/s
martin@katharina:~/Tests>
```

Ahora muestro el tamaño del archivo *dummy.zeros*, siendo de 20Mb.



A terminal window titled "Tests : bash - Konsole" showing the output of the `ls -lh` command. The output lists the files in the current directory: `total 20M`, `-rw-r--r-- 1 martin users 20M Apr 8 13:01 dummy.zeros`, `-rwxr-xr-x 1 martin users 5.0K Apr 8 11:39 GzipGunzip.exe`, and `-rw-r--r-- 1 martin users 777 Apr 8 11:39 GzipGunzip.exe.mdb`. The file size `20M` for `dummy.zeros` is highlighted with a box. The prompt is `martin@katharina:~/Tests>`.

```
martin@katharina:~/Tests> ls -lh
total 20M
-rw-r--r-- 1 martin users 20M Apr 8 13:01 dummy.zeros
-rwxr-xr-x 1 martin users 5.0K Apr 8 11:39 GzipGunzip.exe
-rw-r--r-- 1 martin users 777 Apr 8 11:39 GzipGunzip.exe.mdb
martin@katharina:~/Tests>
```

Ahora comprimo el archivo *dummy.zeros* con el programa de ejemplo *GzipGunzip*, esto lo hago con el siguiente comando:

```
mono GzipGunzip.exe dummy.zeros
```

```
Tests : bash - Konsole
File Edit View Bookmarks Settings Help
martin@katharina:~/Tests> mono GzipGunzip.exe dummy.zeros
Input file /home/martin/Tests/dummy.zeros
Compressing .... /home/martin/Tests/dummy.zeros
bytes read: 20000000
bytes written: 20000000
martin@katharina:~/Tests> █
```

Si se ejecuta correctamente este programa debe crear un archivo llamado *dummy.zeros.gz*, el cual es el archivo comprimido de *dummy.zeros*. Comparamos el tamaño de ambos archivos.

```
Tests : bash - Konsole
File Edit View Bookmarks Settings Help
martin@katharina:~/Tests> ls -lh
total 20M
-rw-r--r-- 1 martin users 20M Apr  8 13:01 dummy.zeros
-rw-r--r-- 1 martin users 19K Apr  8 13:09 dummy.zeros.gz
-rwxr-xr-x 1 martin users 5.0K Apr  8 11:39 GzipGunzip.exe
-rw-r--r-- 1 martin users 777 Apr  8 11:39 GzipGunzip.exe.mdb
martin@katharina:~/Tests> █
```

Ahora voy a descomprimir el archivo *dummy.zeros.gz* , pero antes de hacerlo voy a renombrar el archivo original *dummy.zeros* como *dummy.zeros.bak* para evitar que el archivo original se sobrescriba y así poder comparar todos los archivos. Ejecutamos nuevamente el programa *GzipGunzip* pasándole como parámetro el nombre del archivo.

```
mono GzipGunzip.exe dummy.zeros.gz
```

```
Tests : bash - Konsole
File Edit View Bookmarks Settings Help
martin@katharina:~/Tests> mv dummy.zeros dummy.zeros.bak
martin@katharina:~/Tests> mono GzipGunzip.exe dummy.zeros.gz
Input file /home/martin/Tests/dummy.zeros.gz
Uncompressing .... /home/martin/Tests/dummy.zeros.gz
output: /home/martin/Tests/dummy.zeros
martin@katharina:~/Tests> 
```

Bien ahora con todos los archivos, podemos comparar sus tamaños y comprobar que las clases de compresión de .NET funcionaron.

```
Tests : bash - Konsole
File Edit View Bookmarks Settings Help
martin@katharina:~/Tests> ls -lh
total 39M
-rw-r--r-- 1 martin users 20M Apr  8 13:17 dummy.zeros
-rw-r--r-- 1 martin users 20M Apr  8 13:01 dummy.zeros.bak
-rw-r--r-- 1 martin users 19K Apr  8 13:09 dummy.zeros.gz
-rwxr-xr-x 1 martin users 5.0K Apr  8 11:39 GzipGunzip.exe
-rw-r--r-- 1 martin users 777 Apr  8 11:39 GzipGunzip.exe.mdb
martin@katharina:~/Tests> 
```

Para diseñar programas que utilicen las clases de compresión y descompresión de archivos en C#, el primer paso es crear los flujos de entrada y de salida.

```
FileStream inputStream = new FileStream (filename , FileMode.Open,
FileAccess.Read);
FileStream outputStream = new FileStream (destFile , FileMode.Create,
FileAccess.Write))
```

En el caso de la compresión el flujo de compresión (**DeflateStream**) debe de envolver al flujo de salida, porque la entrada es un archivo sin comprimir que tendrá como destino un archivo comprimido.

```
DeflateStream zipStream = new DeflateStream (outputStream,  
CompressionMode.Compress)
```

Para la descompresión el flujo de compresión (**DeflateStream**) deberá de envolver al flujo de entrada ya que es un archivo comprimido que tendrá como destino un archivo sin comprimir.

```
DeflateStream zipStream = new DeflateStream (inputStream,  
CompressionMode.Decompress)
```

Hay que recordar que uno de los factores que degradan el desempeño de las aplicaciones ASP.NET es cuando existe un cuello de botella en el ancho de banda y no en la utilización del procesador, en el caso de la compresión de datos usada junto con el protocolo HTTP se recomienda para texto: JSON, XML, HTML, etc. Esta compresión no es útil para archivos de imágenes, los cuales ya se encuentran comprimidos.



[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»