Introducción a Visual Studio y la Seguridad en .NET Framework

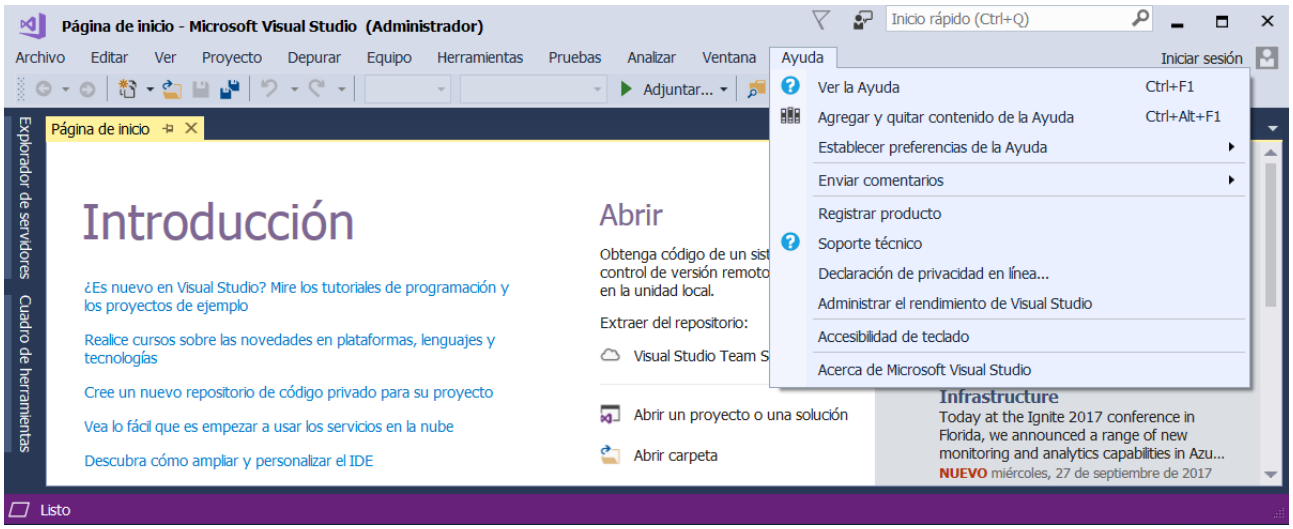
Práctica 2

1. Objetivo

En esta práctica el alumno debe familiarizarse con el entorno de trabajo de Visual Studio de Microsoft y en particular con las clases orientadas a la seguridad, como las proporcionadas por el entorno .NET Framework.

2. Analizar la ayuda

El alumno comenzará analizando la ayuda de VS. Al arrancar la ayuda se mostrará:



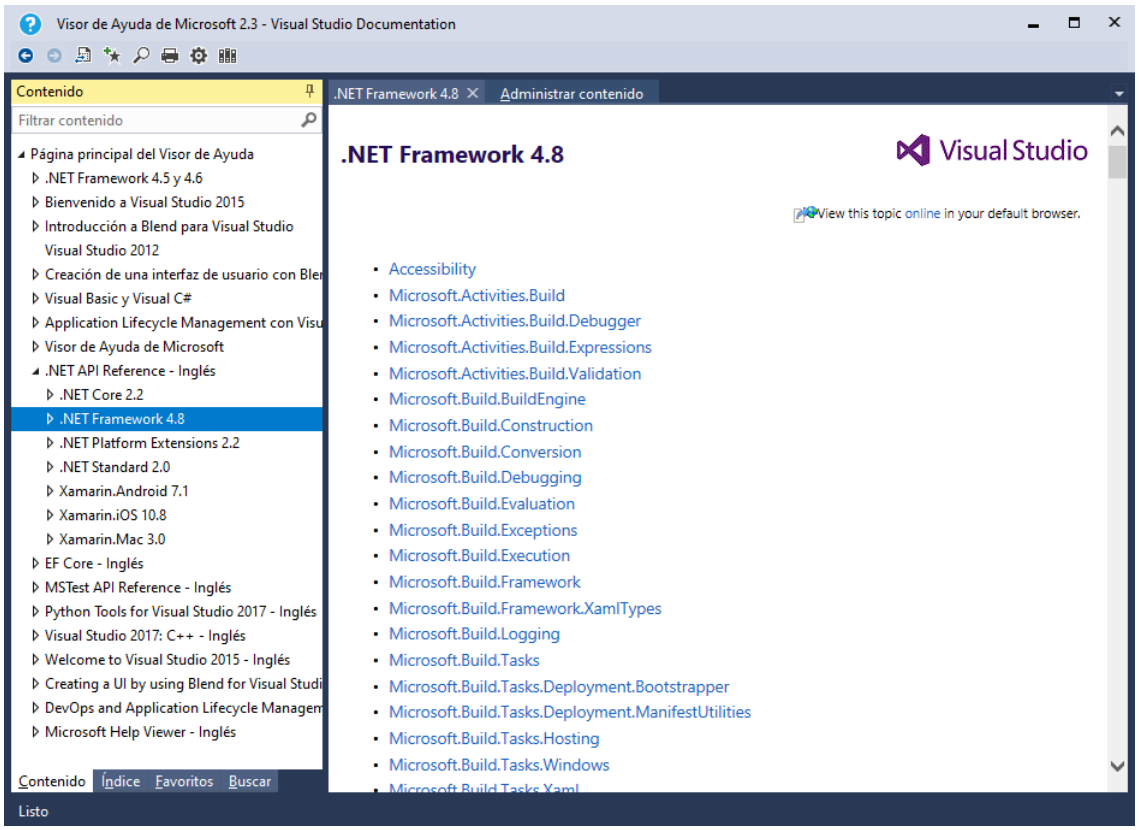
Si se selecciona "Ver la ayuda" se accederá a una página web introductoria. La pagina no es muy útil y es más recomendable acceder a la página web del la librería de clases de .NET Framework:

https://msdn.microsoft.com/es-es/library/mt472912

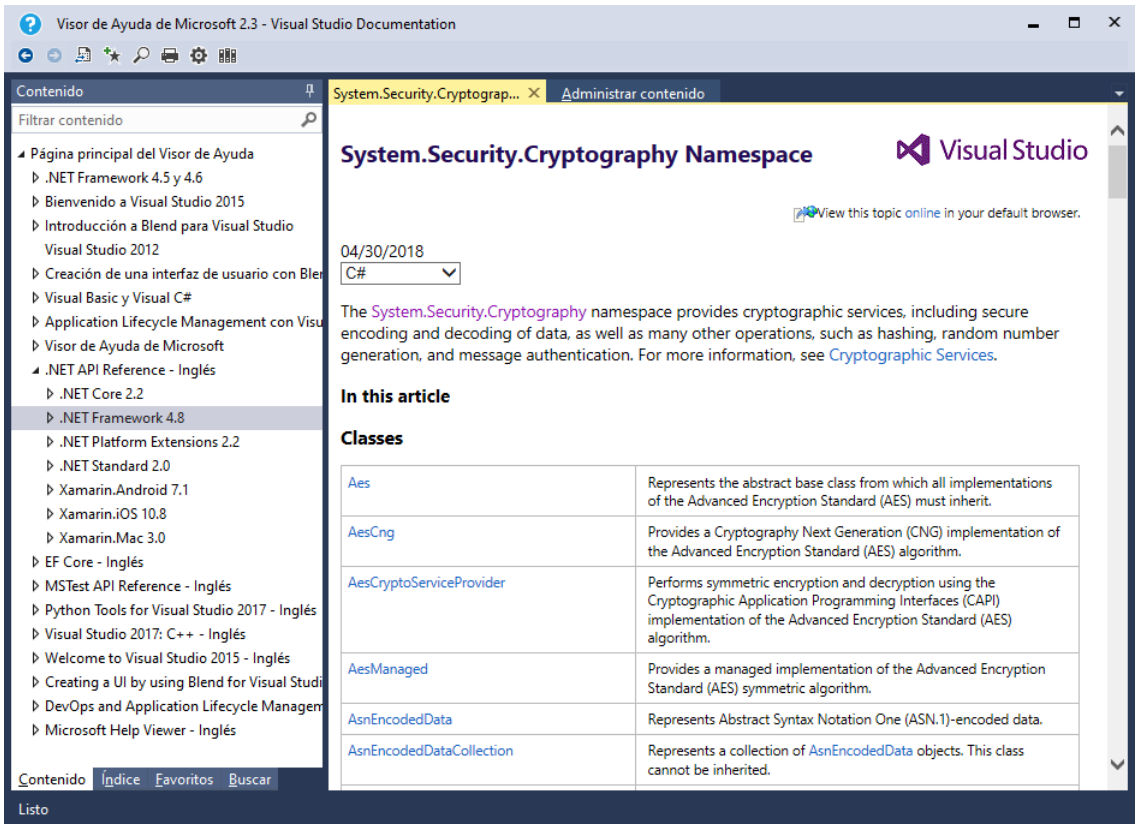
Para mostrar la ayuda disponible localmente selecciona la tercera opción del menú desplegado: Establecer preferencias de la ayuda > Iniciar en el visor de ayuda.

Para que el visor de ayuda sea útil debe descargarse y actualizarse periódicamente los contenidos, lo cual se hace seleccionando la opción "Agregar y quitar contenido de la Ayuda".

Al acceder al visor de ayuda se muestra en el panel derecho una información de bienvenida. En el panel izquierdo (debe estar seleccionada abajo la pestaña Contenido) seleccionar ".NET API Reference - Inglés" y después ".NET Framework 4.8". En el panel de la derecha aparecen los espacios de nombres que proporciona .NET en orden alfabético.



Desplazarse en el panel derecho hasta el espacio de nombres System.Security.Cryptography. Al seleccionarlo aparecen múltiples clases relacionadas con la criptografía.



Observar que la pestaña inferior del panel izquierdo es "Contenido". Una forma rápida de ir en la ayuda al espacio de nombres System.Security.Cryptography es seleccionar la pestaña "Indice" en la parte inferior del panel izquierdo y en el cuadro para introducir texto que se muestra en la parte superior del panel izquierdo introducir System.Security.Cryptography. El panel de la derecha muestra todas las clases disponibles en este espacio de nombres para realizar tareas criptográficas.

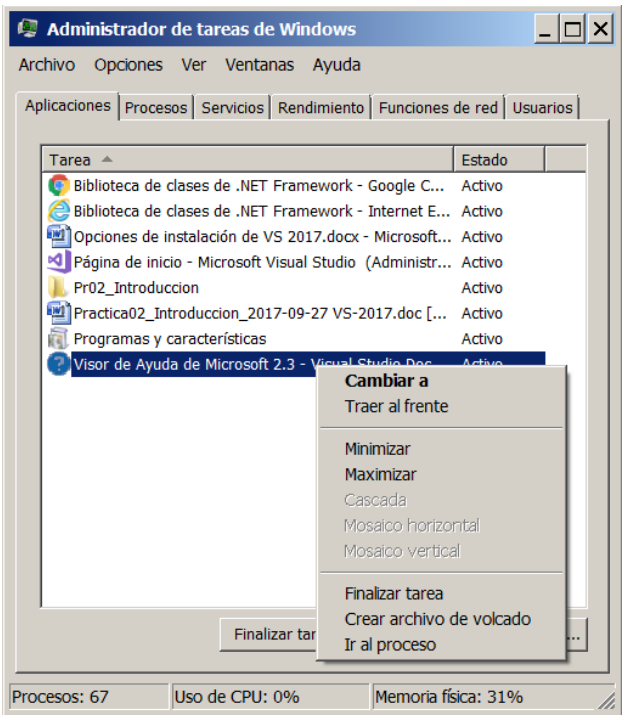
Para acceder a la ayuda disponible en Internet directamente:

http://msdn.microsoft.com/es-es/library/system.security.cryptography.aspx

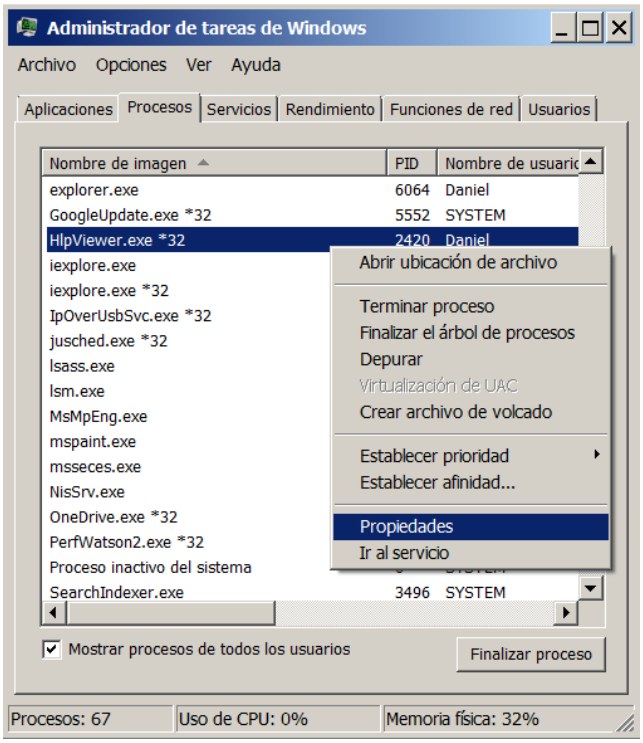
Para generar un icono de acceso directo a la ayuda en el escritorio hacer lo siguiente.

Localizar donde está el fichero ejecutable de la ayuda. Una forma posible de hacerlo es arrancar la ayuda desde el entorno de Visual Studio.

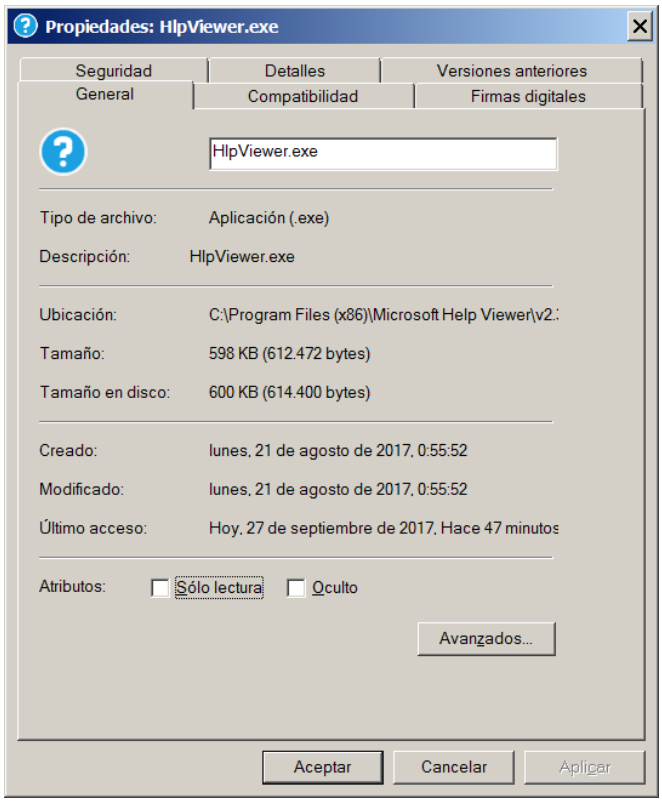
Arrancar el Administrador de tareas de Windows y localizar y seleccionar la aplicación "Visor de Ayuda de Microsoft 2.3". Hacer clic en el botón derecho del ratón y en el menú que se despliega seleccionar "Ir al proceso", tal como se indica en la figura siguiente:



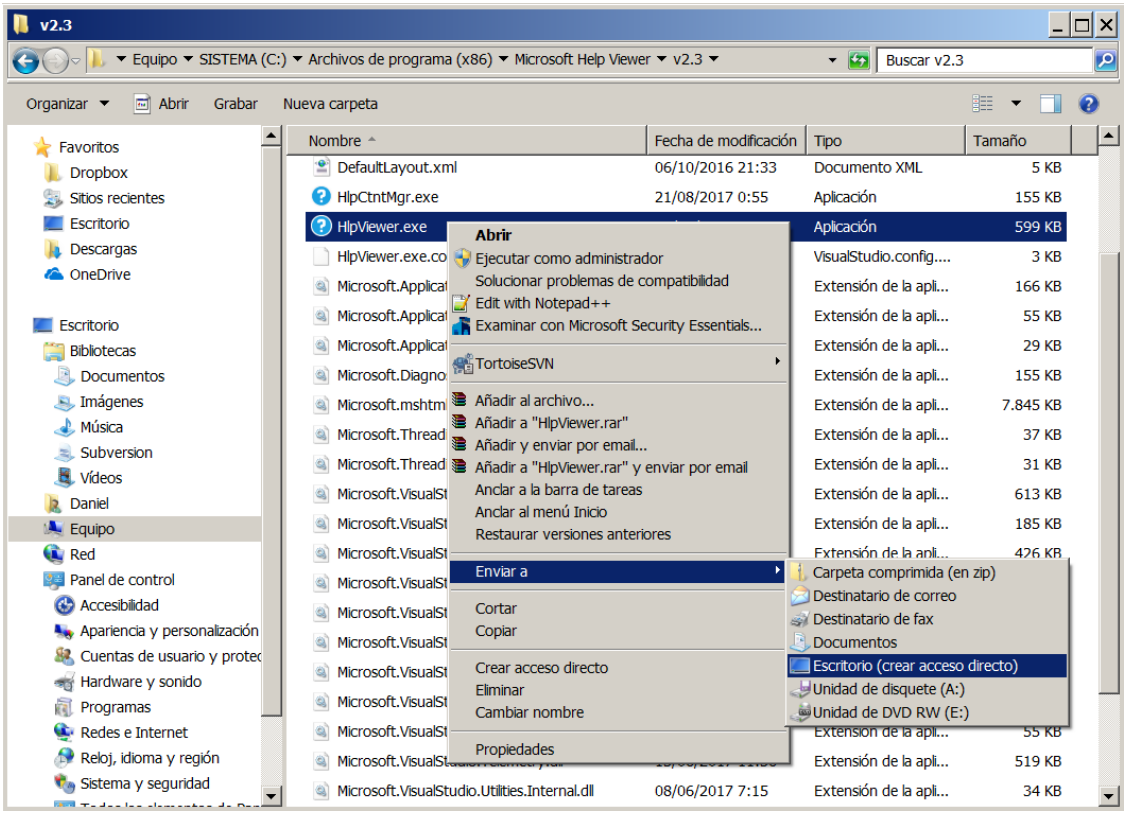
Se abre la ventana siguiente en la cual aparece seleccionado el proceso HlpViewer.exe. Hacer clic en el botón derecho del ratón y en el menú que se despliega seleccionar "Propiedades", tal como se indica en la figura siguiente:



Se abre la ventana siguiente que muestra la ubicación del fichero ejecutable, en este caso está en C:\Program Files (x86)\Microsoft Help Viewer\v2.3

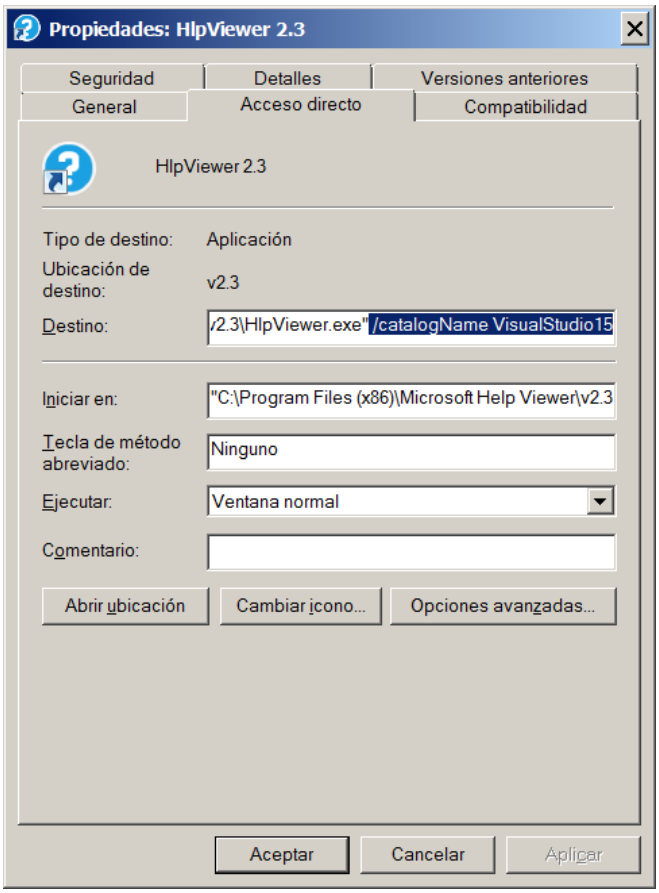


Con el explorador de ficheros se localiza el fichero HlpViewer.exe y selecciónalo. Hacer clic en el botón derecho del ratón y en el menú que se despliega seleccionar "Enviar a > Escritorio".



Una vez en que el icono de acceso directo está en el escritorio es necesario seleccionarlo y pulsar el botón derecho del ratón, En el menú que aparece seleccionar propiedades.

En el campo Destino de la ficha "Acceso directo" hay que añadir los parámetros con los que se debe invocar al ejecutable, que se muestran con fondo azul en la figura siguiente:



NOTA: hay un espacio entre .exe" y /catalogName.

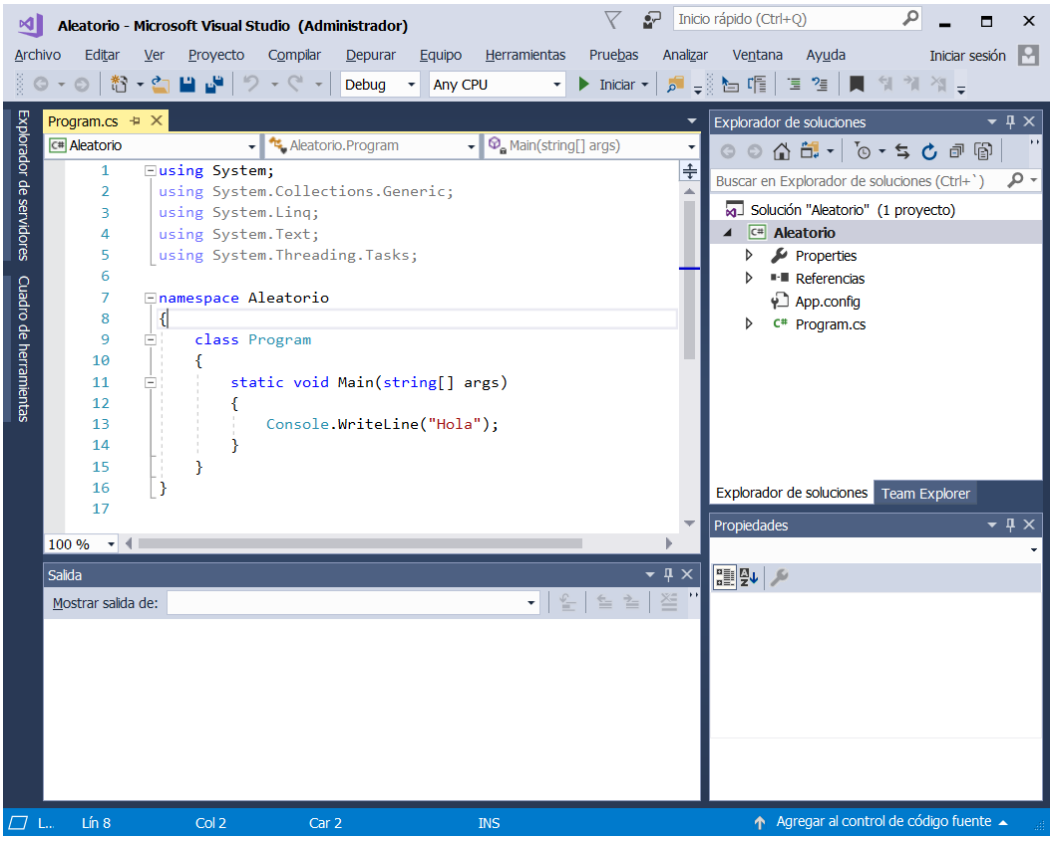
Al programar, es muy conveniente consultar la ayuda sobre una nueva clase a utilizar en el visor de ayuda y no basarse solo en la información online proporcionada por el editor integrado en Visual Studio.

En el examen de prácticas no se puede utilizar Internet, por lo que el uso del visor de ayuda es esencial y conviene acostumbrarse a utilizarlo durante todas las sesiones de prácticas.

3. Preparar un programa inicial

El alumno desarrollará un programa inicial que se utilizará posteriormente como base para las prácticas siguientes.

Para crear este programa y **todos** los de las prácticas el alumno arrancará VS y seleccionará las opciones: Archivo --> Nuevo --> Proyecto. En la ventana "Nuevo proyecto" seleccionar Visual C# y Aplicación de Consola (.NET Framework). ¡Ojo! NO (.NET Core). Proporcionar un Nombre en la parte inferior de la ventana y una ubicación (directorio) en la que el alumno tenga permisos de escritura. En la esquina inferior derecha NO marcar la casilla Crear directorio para la solución. Pulsar el botón finalizar y se llega a la pantalla siguiente:



En la cortina derecha aparece el Explorador de soluciones. Situar el puntero del ratón sobre el nombre del único proyecto que contiene la solución, en este caso denominado Aleatorio, y pulsar el botón izquierdo para seleccionarlo. Después, pulsando el botón derecho del ratón aparece un menú contextual. Seleccionar Agregar --> Nuevo elemento. En la ventana que aparece "Agregar nuevo elemento" se pueden seleccionar diversos tipos de Archivo y proporcionarles un nombre. No agregues nada ahora.

Copiar el archivo proporcionado con la práctica, "Ayuda.cs", en el directorio que contiene el archivo del proyecto Program.cs. Para agregarlo al proyecto situar el puntero del ratón sobre el nombre del proyecto (¡el proyecto, no la solución!), seleccionarlo y pulsar el botón derecho del ratón. Proceder a agregar un elemento existente en el menú contextual que aparece. Comprueba que el elemento agregado aparece en el Explorador de soluciones.

La tarea a realizar en ésta práctica consistirá en escribir un programa que genere un número aleatorio criptográfico (grande), almacenarlo en un fichero y recuperarlo del fichero. Estos números aleatorios grandes son necesarios para generar claves de cifrado o para mezclarlos con las contraseñas antes de cifrarlas.

Los objetivos de la práctica son: iniciarse en el uso de las clases que proporciona el espacio de nombres "**System.Security.Cryptography**", y visualizar, almacenar en fichero y recuperar desde fichero un buffer de memoria que puede contener texto plano, texto cifrado, claves, etc.

Hay que añadir al principio del programa los siguientes espacios de nombres:

```
using System.Security;
using System.Security.Cryptography;
using Apoyo;
```

Los dos primeros son necesarios para tener acceso a las clases criptográficas que proporciona .NET. El tercero es necesario para usar los métodos de la clase Ayuda que está contenida en el fichero Ayuda.cs.

Como **primera fase** de la práctica desarrollar un **programa para generar un número aleatorio criptográfico** realizando las tareas siguientes:

1) Crear un objeto de ayuda que permitirá invocar el método WriteHex(). Crear un array de bytes de longitud parametrizable para almacenar el número aleatorio a generar.

2) Crear un objeto de la clase **RNGCryptoServiceProvider** y utilizarlo para generar el número aleatorio en el array de bytes invocando el método **GetBytes()** del objeto creado. Volcar el número aleatorio a la consola utilizando el método WriteHex(). Liberar los recursos utilizados invocando al método **Dispose()**.

Como **segunda fase** de la práctica se agregarán tres métodos a la clase Ayuda (**antes de programar los métodos, lee la sección 5 de este guión de la práctica**):

```
public long BytesFichero(string NombreFich);
public void GuardaBufer(string NombreFich, byte[] Bufer);
public void CargaBufer(string NombreFich, byte[] Bufer);
```

El método BytesFichero() devolverá el número de bytes almacenados en el fichero.

El método GuardaBufer() permitirá salvar en un fichero, cuyo nombre se indica en el primer parámetro, el buffer de memoria indicado en el segundo parámetro. El fichero deberá ser de tipo binario. Usa un programa visor/editor de ficheros binarios, como **HxD**, para comprobar que el fichero almacena los bytes deseados y el método desarrollado funciona bien.

El método CargaBufer() permitirá cargar en el buffer de memoria que se indica el segundo parámetro los bytes almacenados en el fichero cuyo nombre se indica en el primer parámetro. El fichero deberá ser de tipo binario.

Modificar el programa desarrollado para que, después de generar el número aleatorio, lo almacene en un fichero binario denominado "NumeroAleatorio.bin" usando el método GuardaBufer().

Después, crear un Nuevo Buffer y cargarle el número aleatorio, leyéndolo desde el fichero en el que está almacenado, usando el método CargaBufer(). Mostrar el contenido del Nuevo Buffer con el método WriteHex() para comprobar que todo ha funcionado correctamente.

4. Protección de datos

El sistema operativo Windows proporciona una "Data-Protection API" (DPAPI) que permite proteger datos críticos (claves de cifrado). .NET proporciona la clase **ProtectedData** para acceder a este servicio. Está integrada en el espacio de nombres System.Security.Cryptography.

Usa el método estático **Protect()** para proteger (≈cifrar) los datos. El primer parámetro, userData, recibe los datos a proteger. El segundo parámetro, optionalEntropy, contiene información para mezclarla con los datos a proteger y ocultar así su estructura. El tercer parámetro, scope, es uno de los valores de la enumeración **DataProtectionScope**. Se puede usar CurrentUser o LocalMachine. El cifrado y descifrado utiliza credenciales de CurrentUser o de LocalMachine para generar la clave utilizada por el algoritmo de cifrado/descifrado.

Crea un nuevo proyecto en Visual Studio e integra en este nuevo proyecto los archivos fuente del proyecto previo. En este nuevo proyecto hay que proteger el número aleatorio generado antes de guardarlo en un fichero usando el método estático Protect(). Muestra el contenido del array de bytes con el número cifrado antes de guardarlo en el fichero. Visual Studio no reconoce la clase ProtectedData ni la enumeración DataProtectionScope, a pesar de tener declarado al principio del programa: using System.Security y using System.Security.Cryptography;

Como la ayuda indica que la clase está en el ensamblado System.Security (en System.Security.dll) hay que añadir la referencia a este ensamblado en el proyecto. Para ello, en el Explorador de Soluciones de VS, desplegar la sección Referencias. Comprobar que no está System.Security. Seleccionar la sección Referencias clicando con el botón izquierdo del ratón sobre ella. Una vez seleccionada, clicar con el botón derecho y en el menú emergente seleccionar "Agregar referencia...". En la ventana Administrador de referencias seleccionar en el panel izquierdo Ensamblados > Framework Extensiones, y en el panel central seleccionar System.Security y pulsar el botón Aceptar. En el Explorador de soluciones desplegar nuevamente las referencias para comprobar que se ha seleccionado correctamente.

Finalmente, añade el código necesario para desproteger el número aleatorio cifrado, leído del fichero. Usa el método estático Unprotect() de la clase ProtectedData. La entropía utilizada en la desprotección debe ser la misma que la utilizada en la protección.

Comprueba si el número cifrado depende exclusivamente del número sin cifrar o no. Para ello sobrescribe el número generado con una secuencia de bytes fija, por ejemplo 0x00, 0x01, etc.

Comprueba el efecto de cambiar la entropía, usando un vector de 10 bytes, por ejemplo, y luego no usándola al asignar su valor a null.

Comprueba lo que ocurre al utilizar una entropía para la protección y otra para la desprotección.

El .NET Framework también proporciona la clase **ProtectedMemory** para proteger datos críticos en la propia memoria del computador. Usar el método estático **Protect()** para proteger (≈cifrar) los datos. El primer parámetro, userData, recibe los datos a proteger. El segundo parámetro, scope, es uno de los valores de la enumeración **MemoryProtectionScope**. Se puede usar CrossProcess, SameLogon o SameProcess.

Proteger el último número leído desde fichero usando el método Protect(). Volcar en pantalla su contenido para comprobar cómo ha sido cifrado en su misma posición en la memoria y luego desprotegerlo usando el método Unprotect().Volcar en pantalla su contenido para comprobar cómo se obtiene el número original.

5. Notas útiles sobre ficheros

Esta sección es útil para desarrollar métodos para leer de y escribir en ficheros. Por ejemplo para desarrollar los métodos GuardaBufer() y CargaBufer().

Para trabajar con ficheros .NET utiliza secuencias (streams). Hay que distinguir claramente los ficheros de las secuencias.

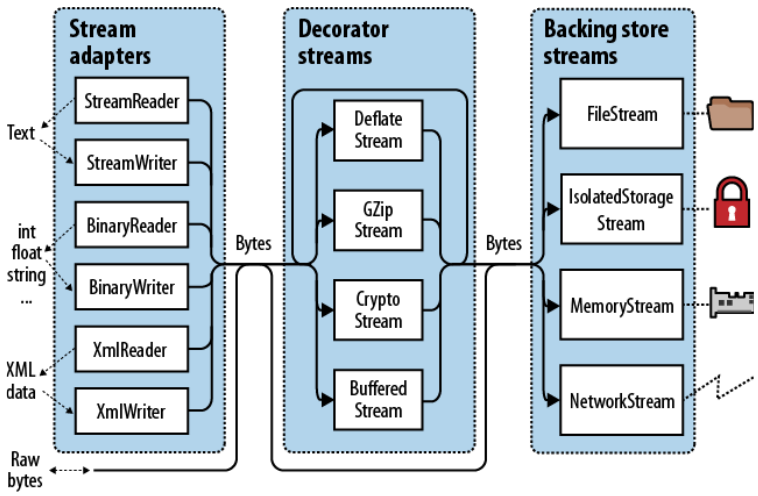
- Fichero = Colección ordenada de bytes concreta y con un nombre. Tiene un almacenamiento persistente.
- Secuencia = Mecanismo para leer y escribir bytes en un almacén de respaldo. Hay secuencias para cada medio de respaldo: ficheros, memoria, red, cinta.

Para manejar ficheros hay dos clases "equivalentes":

- La clase **File** proporciona **métodos estáticos** para hacer todo tipo de operaciones con ficheros, incluyendo métodos para leer/escribir de/en ficheros.
- La clase **FileInfo** proporciona **métodos de instancia** para hacer las mismas operaciones, pero no tiene métodos para leer/escribir de/en ficheros.

Ambas clases tienen métodos para crear objetos de tipo **stream** que sí tienen métodos para leer/escribir de/en ficheros.

La figura siguiente muestra cómo utilizar los diferentes objetos de tipo **stream** para acceder a los dispositivos finales que almacenan o transmiten la información.



Las clases para leer/escribir en los dispositivos que proporciona .NET están disponibles en el espacio de nombres **System.IO**.

Normalmente se utilizará una secuencia **FileStream**. Esta secuencia solo dispone de métodos para leer o escribir bytes directamente y sin ningún tipo de transformación. En la figura estas secuencias se denominan "*Backing store streams*".

Si se desea escribir texto en un fichero, es común codificarlo antes de escribirlo en el fichero. La clase **StreamWriter** proporciona métodos para escribir que codifican un string (típicamente representado en Unicode en C#) en una cadena de bytes con un determinado formato

(comúnmente UTF-8). La clase **StreamReader** proporciona métodos para leer bytes en una codificación y generar strings. Estas clases tienen en cuenta que los ficheros de texto se organizan en líneas, separadas con el carácter \n. También puede utilizar una cadena de formato que especifica cómo escribir variables enteras, flotantes, etc., en texto y viceversa. Cuando se utiliza una cadena de formato se usan de modo similar a las funciones fprintf() y fscanf() de C/C++.

Si se desea escribir/leer diversos tipos de variables en/de su formato binario se utilizan las clases **BinaryWriter** o **BinaryReader**. Son muy útiles para escribir en archivos números enteros o flotantes en su formato nativo y luego leerlos posteriormente.

Las secuencias StreamWriter/StreamReader y BinaryWriter/BinaryReader escriben bytes y leen bytes. Normalmente, cuando se trabaja con ficheros, los bytes que escriben los recibe una secuencia FileStream y los bytes que leen también los envía una secuencia FileStream. En la figura anterior estas secuencias se denominan "*Stream adapters*".

Por tanto, las secuencias trabajan de forma encadenada, tal como muestra la figura anterior.

Variable <--> Stream adapter <--> Backing store stream <--> Respaldo

No obstante y como excepción, las secuencias StreamWriter/StreamReader pueden trabajar directamente con un fichero de texto, sin necesidad de usar una secuencia FileStream intermedia.

La figura anterior muestra que entre un "*Stream adapter*" y un "*Backing store stream*" se puede colocar uno o varios "*Decorator streams*". Esto es muy importante porque el cifrado de información se realizará colocando un CryptoStream entre un StreamWriter y un FileStream. Similarmente, el descifrado se realizará usando otro CryptoStream entre un FileStream y un StreamReader.

CONCLUSION: Para los objetivos de la práctica (escribir y leer bytes) es suficiente utilizar los métodos estáticos proporcionados por la clase File.

Pero es conveniente utilizar la clase FileStream, ya que es muy interesante comprender el funcionamiento de las secuencias, pues se utilizan en las prácticas futuras.

La tercera opción es utilizar la combinación de las clases BinaryWriter y BinayReader con una clase FileStream pues es la forma estándar de hacer la escritura y lectura en ficheros.