Universidad Abierta Interamericana

Campus Virtual UAIOnline

LENGUAJES DE ÚLTIMA GENERACIÓN

ENSAMBLADOS

UNIDAD 1

ENSAMBLADOS/CLASE 2

Autor de contenidos:
Mauricio Prinzo

PRESENTACIÓN

Le proponemos la rememoración y aplicación de algunos conceptos de los lenguajes de programación que son pilares fundamentales en el desarrollo de aplicaciones.

Esperamos que, a través del estudio de esta unidad, adquiera capacidad para:

- Comprender el concepto de código administrado.
- Utilizar herramientas adicionales facilitadas por el entorno de programación.
- Visualizar el Lenguaje intermedio.
- Identificar un ensamblado.

A continuación, le presentamos un detalle de los contenidos y actividades que integran esta unidad. Usted deberá ir avanzando en el estudio y profundización de los diferentes temas, realizando las lecturas requeridas y elaborando las actividades propuestas, algunas de desarrollo individual y otras para resolver en colaboración con otros estudiantes y con su profesor tutor.

01. INTRODUCCIÓN

Antes de comenzar le proponemos que articule el estudio de los contenidos de esta unidad con la siguiente actividad.

Actividades para la facilitación de los aprendizajes

Durante la lectura, aplique y desarrolle aplicaciones que usen los conceptos estudiados y acérquese al Foro para compartir experiencias, dudas y conocimientos.

02. DESARROLLO

Ahora sí, comencemos con el desarrollo de los contenidos de esta unidad.

Una de las características de las aplicaciones NET es su distribución en forma de ensamblados que incluyen ejecutables y componentes. De una forma simple, podríamos decir que los **ensamblados** se corresponden al concepto de programas EXE.

Al compilar una aplicación .NET, el resultado difiere un poco del conocimiento general. No obtendrá al final el típico archivo objeto, lo que obtendrá es un **módulo ensamblado**. Puede ser un EXE o una DLL dependiendo del tipo de proyecto que está desarrollando.

Una característica de estos módulos es que pueden ser autosuficientes o dependientes de otros módulos. Un ensamblado puede estar formado por uno o varios módulos.

Su composición se agrupa en encabezados de archivos Windows PE, encabezado de archivo .NET framework, metadatos y código de lenguaje intermedio MSIL. Veamos cada uno con algún detalle:

ENCABEZADO

Todos los archivos EXE al igual que los módulos tienen un encabezado PE. Una de las referencias más importantes que poseen es el código que se debe ejecutar cuando el SO los invoca.

Los módulos contiene código MSIL, este es un conjunto de instrucciones nativas comprendidas por la CPU destino. Por ello, la primera instrucción de los módulos administrados es JMP (salto) que apunta al código MSIL.

METADATOS

Generalmente se entiende un metadato como la descripción de los datos. En este caso, la definición no es ajena pues describe todas las clases, métodos e interfaces públicos que un ejecutable expone. Describen los tipos a los cuales hace referencia el modulo actual.

No existe posibilidad alguna de entregar un módulo sin sus metadatos o al revés. Puesto que los metadatos son indispensables en NET.

Aunque es un tema avanzado, desde NET se pueden leer los metadatos del módulo usando **Reflection.**

LENGUAJE INTERMEDIO

También conocido como MSIL, lenguaje intermedio de Microsoft. La ventaja es que todos los lenguajes de alto nivel usados por Microsoft, compilan en MSIL. Esto es una fortaleza y un debate social donde se plantea que la intensión a futuro es unificar todos los lenguajes.

MSIL presenta dos beneficios:

Proporciona un lenguaje intermedio de fácil adaptación a las CPU del mercado, razón por la cual se debe instalar el framework .NET en la PC donde se implemente el programa.

Refuerza a nuestra aplicación de mayor seguridad en su ejecución al hacer más robusto al código administrado.

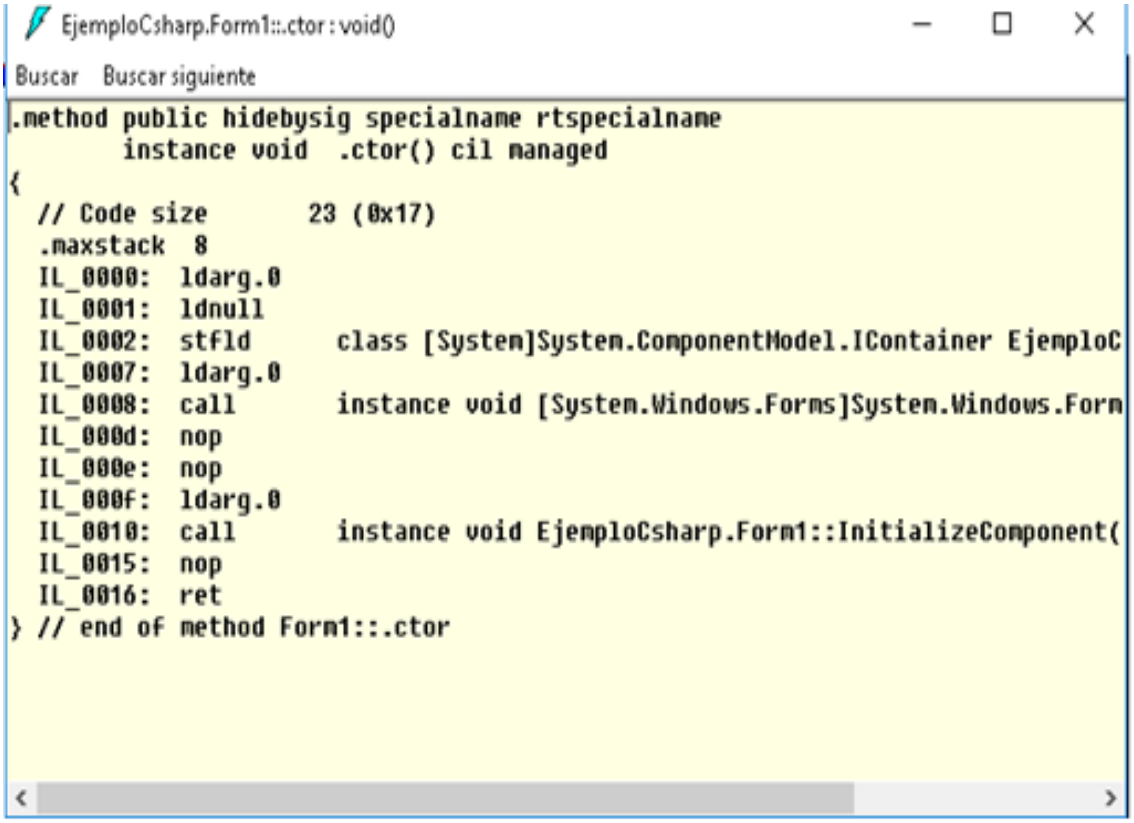
Cuando obtenga más experiencia podrá ver que el MSIL no solo unifica los lenguajes de alto nivel, sino que también es reversible y puede por ejemplo convertir un programa de vb.net en C# o viceversa.

Esto es un ejemplo de MSIL, consideremos el siguiente código de c#.net en una aplicación winforms.

```
private void Button1_Click(object sender, EventArgs e)
        {   int a = 0;
            int b = 0;
            int c = 0;

            a = Convert.ToInt32(TextBox1.Text);
            b = Convert.ToInt32(TextBox2.Text);
            c = a + b;
            TextBox3.Text = Convert.
```

El código MSIL resultante es el que se muestra a continuación:



2.1 ENSAMBLADOS

Como comentamos en párrafos anteriores, al compilar en NET generamos módulos, pero .NET trabaja con **ensamblados**. Para comprender mejor el concepto, un ensamblado puede ser el conjunto de uno o más módulos.

Un ensamblado es la mejor unidad de reuso en .NET. Es por ello que debe mantener juntos a los módulos que se complementen entre sí.

Otra característica es que posee **control de versiones** – aspecto muy importante para un correcto control de las aplicaciones. Los tipos contenidos en un ensamblado tienen los mismos permisos, si desea tener una configuración diferente deberá considerarlo al agrupar los módulos.

Hay dos tipos de ensamblados, **estáticos** y **dinámicos**. La mayoría de los compiladores de C# o vb.net usan ensamblados estáticos, estos se corresponden con uno o varios archivos físicos. Por su parte, los dinámicos, se crean en la RAM al momento de ser utilizados.

Dentro del ensamblado se encuentra un archivo importante, el **manifiesto**.

Este contiene los metadatos sobre el ensamblado. Datos como su versión, referencia cultural y la clave pública de la compañía que lo desarrolla y la lista de sistemas operativos y CPU donde se podrá ejecutar el ensamblado.

Posee también una colección de tablas que muestra la lista de archivos que forman un ensamblado.

Por defecto el runtime .NET busca los ensamblados que se encuentran en el mismo espacio de directorio de la aplicación que efectúa la llamada evitando con esto conflictos con los nombres de los archivos. Pero puede ocurrir, sobre todo cuando los ensamblados se comparten con otras aplicaciones, que los nombres entren en conflicto.

Para asegurar el nombre del ensamblado, puede generar una clave pública y privada aleatoria con la aplicación **SN**, con el siguiente comando.

```
Sn –k miclave.snk
```

Luego podrá firmar el ensamblado de la aplicación y así asegurar un nombre único para el ensamblado.

Visual basic: *Vbc modulo.vb /out:ensamblado.dll /keyfile:miclave.snk*

C#: *csc modulo.cs /out:ensamblado.dll /keyfile:miclave.snk*

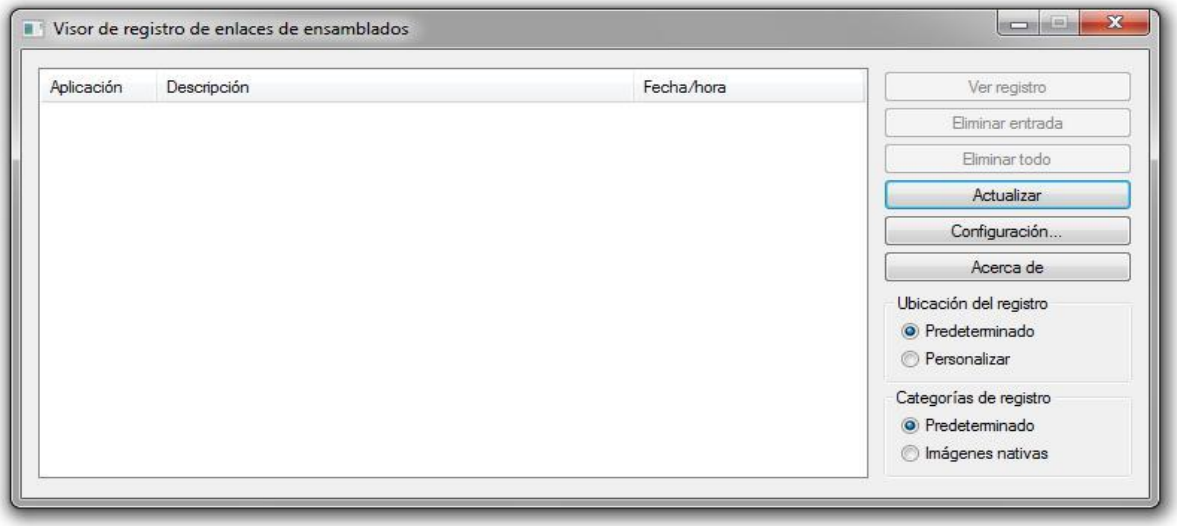
Le proponemos ampliar estos contenidos y acompañar la lectura de los siguientes con el texto que le sugerimos en la sección de bibliogtrafia de la clase.

03. HERRAMIENTAS

El frameWork .NET ofrece un conjunto de herramientas fuera del IDE de desarrollo que facilitan la lectura y la administración de los ensamblados. Ya vimos una en el punto anterior la llamada SN. En este momento seguiremos con otras de uso frecuente y sumamente útiles para el/la programdor/a.

FUSLOGVW

Es un visor del registro de enlaces de los ensamblados. Facilita la lectura de la localización de los ensamblados. Es una herramienta muy importante cuando es necesario seguir errores ocurridos en los ensamblados.



COMPILADOR DE LINEA DE COMANDO

Esto es una clara demostración de que NET es "open source"…

El Framewwork nos permite desarrollar aplicaciones NET desde un block de notas para luego compilarlas con **CSC.exe.** Claro que no tendremos todas las bondades del IDE de trabajo del Visual Studio.

Por línea de comando, podemos crear un **EXE** desde nuestro archivo. Por ejemplo, la siguiente línea convierte el archivo **modulo.vb** en **modulo.EXE**

```
Csc.modulo.cs
```

GACUTIL

Le permite administrar el **cache de ensamblados global (GAC)** desde la línea de comandos.

El cache global reemplaza el registro de las librerías en el modelo COM. Al registrarlo en la GAC una aplicación podrá usar un ensamblado que no se encuentra en su carpeta.

Para instalar un ensamblado en el GAC solo debe ingresar

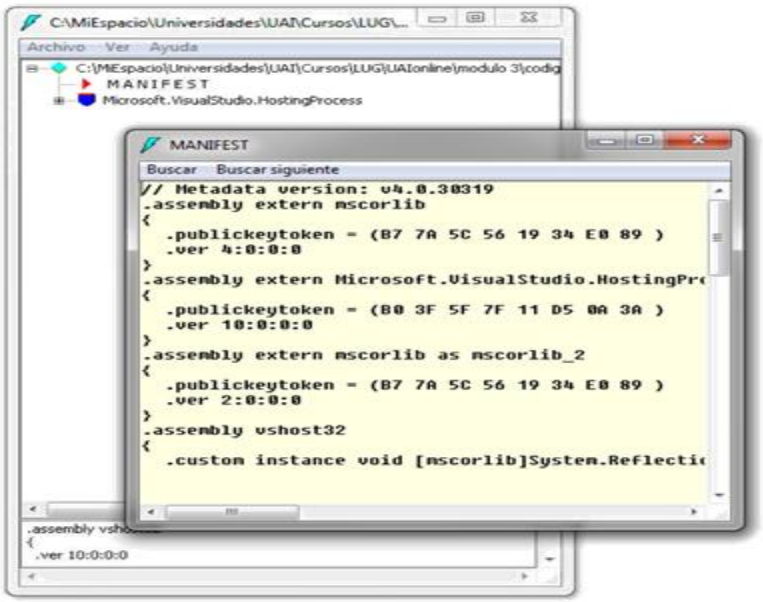
```
Gacutil /i modulo.dll
```

Para Borrarlo de la GAC

```
Gacutil –u modulo
```

ILDASM

Es una herramienta que nos permite ver el código administrado:

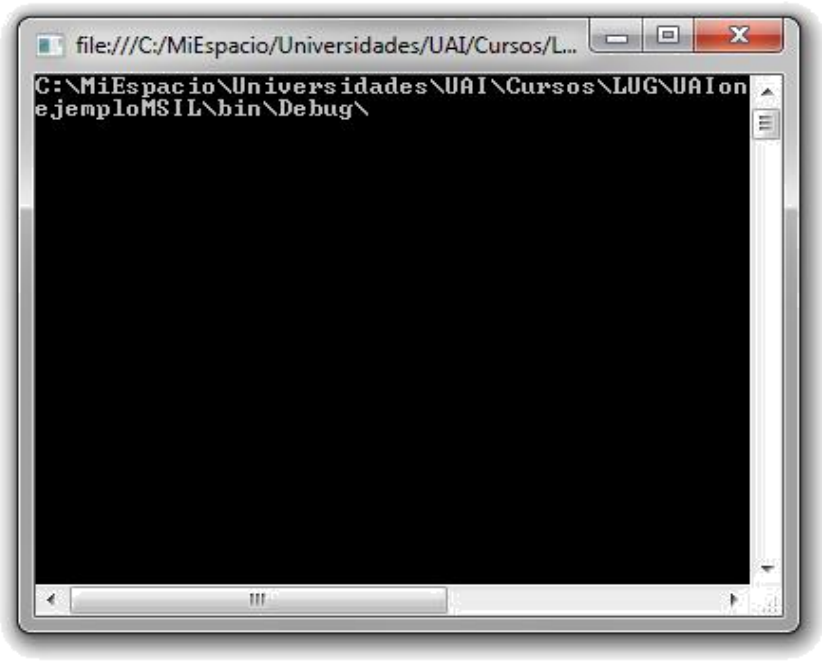


04. APPDOMAIN

Hemos visto anteriormente que todas las aplicaciones de NET corren en un **APPDOMAIN.** Intentaremos en este momento explicarle algunas características y especificidades acerca del uso de esta clase.

La **clase APPDOMAIN** nos permite conocer detalles e información acerca del dominio donde se encuentra la aplicación. Esta clase tiene un conjunto de propiedades, métodos y eventos muy útiles.

Por ejemplo, podemos conocer el directorio donde se encuentra la aplicación, datos muy importantes a la hora de manejar archivos locales.



El **código fuente** de este ejemplo es:

```
public void Main()
 {
   AppDomain D = AppDomain.CurrentDomain;
   Console.WriteLine(D.BaseDirectory);
   Console.ReadKey();
```

Note que creamos una variable de nombre **D** que asume el rol del dominio donde se ejecuta la aplicación. Luego usa la propiedad **BaseDirectory** para obtener información.

Otras propiedades que puede utilizar son:

- **FriendlyName**: Nombre descriptivo del dominio
- **ShadowCopyFile**: indica si todos los ensamblados fueron copiados en primer lugar en otro directorio.

4.1 CREACIÓN DE UN APPDOMAIN

Podrá crear un **APPDomain** de manera muy sencilla, pero recuerde que si no aplica un ensamblado para ejecutarlo dentro del dominio, el proceso de crear un **APPDomain** no servirá de nada.

Por eso, la creación de un **APPDomain**, estará acompañada de un **ExecuteAssembly**

```
AppDomainSetup d =  new AppDomainSetup();

d.ApplicationName=”NuevaAplicacion”;

d.AplicationBase=”c:\Windows\temp”;

d.PrivateBinPath=”bins;assemblies”;

AppDomain Dominio =AppDomain.CreatDomain(“nuevoAD”,nothing,
d);
```

En este módulo hemos abordado un conjunto de temas que no usará en lo cotidiano, pero que es importante conocer para poder resolver situaciones problemáticas en el momento adecuado.

Ha aprendido que no importa el lenguaje que use en sus desarrollo, al final todo termina siendo código **MSIL** o también conocido como **IL**. Esto que puede ser una ventaja, genera mucha *vulnerabilidad* puesto que el código MSIL no es compilado y se puede leer fácilmente con una herramienta.

Comprenderá que el señalamiento acerca de esta "debilidad" se debe a que al ser código abierto es accesible por todos, es decir, podemos no liberar el código fuente pero desde el IL se puede hacer una reingeniería y obtenerlo igual.

Por último, esperamos que haya aprendido a sacar provecho de la clase **APPDomain.**