Estándares de Desarrollo

Fecha de confección:	29/09/2008
Título:	Estándares de Desarrollo
Temática:	Aplicativos .NET
Confeccionado por:	Marcelo Oviedo
Revisado por:	
Aprobado por	
Clasificación:	Público
Auditorio:	
Fecha de Impresión:	13/04/2011 05:45:35 p.m.
Versión	1.5

Objetivos

El propósito del presente documento es describir la normativa de codificación a respetar para los desarrollos de aplicativos en .NET

Alcance

Esta normativa es aplicable a todo desarrollo de aplicaciones en .NET, ya sea internas de Grupo Action Line o Tercerizadas.

Elaboración / Revisión / Aprobación

El presente documento es elaborado por el **Arquitecto Líder de la División QA**, Revisado por el **Responsable de División QA** y aprobado por la **Gerencia de Desarrollo.**

Roles y Responsabilidades

Rol	Responsabilidad
Gerencia de desarrollo	Aprobar el estándar de Desarrollo.
Arquitecto Líder	Definir Estándares y auditar el estándar.
Responsable División QA	Revisión general del estándar.

Descripción del Estándar

1. Introducción

Uno de los instrumentos que facilitan el desarrollo de software y que influye en la mejora de calidad del mismo es la adopción de estándares de estilo y codificación.
Es por esto que la codificación de los módulos del Sistema a desarrollar debe cumplir ciertos requisitos, detallados en el presente documento.

El uso de estos estándares tiene innumerables ventajas, entre ellas:

- Asegurar la legibilidad del código entre distintos programadores, facilitando el debugging del mismo.

- Proveer una guía para el encargado de mantenimiento / actualización del sistema, con código claro y bien documentado.
- Facilitar la portabilidad entre plataformas y aplicaciones.

Toda aplicación .NET debe cumplir los estándares que se presentan en las siguientes secciones.

2. Boxing y unboxing

Conceptos y Ejemplos

Boxing: Conversión de un tipo determinado a un Object
Unboxing: Conversión del tipo Object a un tipo determinado

- Toda estructura deriva de System.Object
- Toda estructura puede declararse como Object.
- La conversión a su tipo original no es directa ya que existen diferencias semánticas y de almacenamiento.

El **Boxing** y el **Unboxing** son procesos lentos e implican la creación y destrucción de objetos envoltorios.

Definición 1: **Boxing y Unboxing**

Esta técnica se podrá utilizar sólo en casos consensuados con el responsable del producto y/o Jefe de Desarrollo, según corresponda.

3. Requerimientos Generales

Conceptos y Ejemplos

- El lenguaje de Programación C# (C Sharp) o Visual Basic .Net Se aconseja la utilización de C# para todo lo que es el BackEnd y VB Net para las aplicaciones FrontEnd.
- Deben evitarse nombres imprecisos que permitan interpretaciones subjetivas, como por ejemplo DefinirEsto(), o bien ytG8 para una variable. Tales nombres contribuyen más a la ambigüedad que a la abstracción.
- En POO es redundante incluir nombres de clases en el nombre de las propiedades de clases, como por ejemplo Rectangulo.RectanguloArea, en su lugar, utilice Rectangulo.Area, pues el nombre de la clase ya denota dicha información
- Utilice la técnica verbo-sustantivo para nombrar procedimientos que ejecuten alguna operación en un determinado objeto, como por ejemplo CalcularSaldo().
- Es recomendado que las variable boleanas contengan una palabra que describa su estado: puedeEliminarse, esGrande, tieneHijos, etc. Y siempre se debe referir al estado verdadero: tieneCredito en cambio de

noTieneCredito.

- Se aconseja el uso de comentarios en línea para facilitar la comprensión del código, sobre todo en procedimientos complejos. Los comentarios pueden ser con fin documental o bien como 'ayuda-memoria'.

Definición 2: **Requerimientos Generales:**

- Se puede utilizar siempre C# (C Sharp) o VB (Visual Basic)
- No incluir nombres de clases en el nombre de las propiedades de las clases.
- Todo procedimiento debe ser nombrado utilizando la técnica verbo-sustantivo.
- Toda variable booleana debe nombrarse describiendo su estado.

4. Variables y Constantes

Conceptos y Ejemplos

Notación prefijos de ámbito + Húngara:

- Esta forma, permite la rápida y correcta legibilidad del código.
- Consiste en prefijos en minúsculas que se añaden a los nombres de las variables, y que indican su tipo. El resto del nombre indica, lo más claramente posible, la función que realiza la variable.
- El prefijo se compone de 1 o 2 partes, el tipo de variable (opcional) y el tipo de dato de variable, más el nombre de la variable.

Por ejemplo:

[ambito]~~[tipoDato]~~+[NombreVariable]
[w]~~[sz]~~+[ApellidoNombre] = w~~sz~~ApellidoNombre;

Otro ejemplo, usando el tipo de variable:

[ambito][tipoVariable]+[tipoDato]+[NombreVariable]
[m_]+~~[sz]~~+[ApellidoNombre] = m~~sz~~_ApellidoNombre;

A continuación, una tabla con los prefijos tipos de dato y variables más utilizados:

Tipos de Variables:

p_	La variable es un puntero a memoria.
m_	La variable es miembro. (atributos)
lp_	La variable es un puntero de 32bit o puntero largo.

~~*Tipos de datos:*~~

~~i~~	~~Entero (int, Int32)~~
~~n~~	~~Enumerador (enum).~~
~~t~~	~~Estructura o tipo (struct).~~
~~e~~	~~Evento (event). (para CallBacks)~~

~~o~~	~~Objeto (object).~~
~~d~~	~~Doble o decimal (double).~~
~~c~~	~~Carácter (char)~~
~~b~~	~~Boleano (bool)~~
~~l~~	~~Entero largo (long, Int64).~~
~~f~~	~~Flotante (float).~~
~~s~~	~~Simple o single (short, Int16).~~
~~p~~	~~Parámetro (utilizado en métodos y/o funciones).~~
~~sz~~	~~Cadena (string).~~

- Dentro de la lista de **IntelliSence** variables y propiedades de **VisualStudio**.**NET,** las variables declaradas de esta forma se encontraran ordenadas por su tipo y son más fáciles de localizar de esta forma.

Definición 3: **Definición de Variables:**

- Se debe utilizar la notación húngara para la declaración de variables.

a. Variables locales

Conceptos y Ejemplos

- Los nombres de algunas variables locales, como los iteradores o los contadores, pueden especificarse de forma abreviada, siempre que su contexto sea específicamente local y su lectura sea intuitiva.
- A las variables locales se les antepondrán un indicador de ámbito local "w".
- Patron:

[w] [NombreVariable]

Ejemplo:

```
/// <summary>
/// Obtiene la cantidad de clientes.
/// </summary>
private int ContarClientes() {
  //- Variable de tipo entero
  int wContador = 0;

  //- Variable de tipo objeto
  ClaseCliente wClaseCliente = new ClaseCliente();

  //- Comportamiento del método
  return wContador;
}
```

Definición 4: **Variables Locales:**

- Las variables locales se deben nombrar bajo el siguiente patrón **[w]+[NombreVariable]**

- Para el caso de variables locales iteradoras o contadores, pueden especificarse en forma abreviada siempre y cuando su contexto sea local y su lectura sea intuitiva.

b. Variables globales privadas (miembros de clase o datos miembros)

Conceptos y Ejemplos

- Las variables globales se nombrarán de igual forma que las locales solo que usando el prefijo "m", ejemplo: *m_CantidadLineas = 0.*
- Patron:

  [m][_]+[NombreVariable]
- Se deberá agregar una breve descripción de el objetivo o funcionalidad de dicha variable utilizando el documentador xml provisto por el IDE de Visual Studio.-

  Ejemplo:

```
/// <summary>
/// Descripcion de variable privada.-
/// </summary>
private int m_LimiteCredito = 0;
```

  - Si la variable global también es una variable que será expuesta como atributo público de una clase se podrá utilizar el refactor del IDE de Visual Studio para generar tal propiedad con sus respectivos métodos get{} y set{}.-

***Nota**: el hecho de anteponer un "m_" al nombre de la variable permite al refactor reconocer dicha variable para generar y encapsular una propiedad de dicha variable con el nombre excluyendo "m[tipo]" del nombre de la variable*

Definición 5: **Variables Globales Privadas.**

- Las variables globales se deben nombrar bajo el siguiente patrón **[m][_]+[NombreVariable]**

- Toda declaración de variable debe tener una breve descripción del objetivo o funcionalidad de dicha variable, utilizando el documentador xml de Visual Studio.

c. Variables globales públicas o propiedades

Conceptos y Ejemplos

- Evite el uso de variables públicas para que ciertos estados o propiedades sean accesible desde el exterior de una clase. Siempre que se encuentre con esta necesidad cree una variable privada como se describe arriba y encapsúlela con refactor de Visual Studio definiendo de esta manera un

método que expone como propiedad de la clase dicha variable.

- Use nomenclatura prefijos + *PascalCase* para propiedades**.**

Ejemplo:

```
/// <summary>
/// Descripcion de variable privada.-
/// </summary>
private int m_LimiteCredito = 0;

/// <summary>
/// Descripcion del metodo que manipula los valores de la variable privada.-
/// </summary>
public int LimiteCredito {
  get { return m_LimiteCredito; }
  set { m_LimiteCredito = value; }
```

Definición 6: **Variables Globales Públicas o Propiedades**

- Se debe utilizar siempre nomenclatura prefijos + PascalCase.
- No se deben usar variables públicas, usar propiedades en caso de necesitar dicha funcionalidad.

d. Constantes

Conceptos y Ejemplos

- Se recomienda usar todo el nombre en mayúscula para constantes privadas, este estilo es apropiado cuando se invocan API's.

Ejemplo:

```
public const double MONTO_MAX = 10000;
public const double MONTO_MIN = 500;
private const string RUTA_ARCHIVO_LOGS = @"C:\Archivos\Logs";
```

Definición 7: **Constantes**

• Para constantes privadas se debe usar SIEMPRE todo el nombre en mayúsculas.

5. Nombres de controles

Conceptos y Ejemplos

A continuación se listan los controles mas comunes utilizados en las interfaces de usuario:

Prefijo	TipoPrefijo	Ejemplo
btn	Button	btnAceptar
chk	CheckBox	chkEliminar
ctl	Control	ctlControlPersonalizado
dgv	DataGridView	dgvListadoCliente
frm	Form	frmActualizarClientes
lbl	Label	lblTitulo
lst	ListBox	lstRubros
mnu	Menu	mnuActualizarClientes
img	Image	imgFotoCliente
dt	DataTable	dtPersona
ds	DataSet	dsFacturacion
tmr	Timer	tmrFecha
dtr	DataRow	dtrCliente
dtp	DateTimePicker	dtpFecha
cal	Calendar	calCalendario
cmb	ComboBox	cmbFormasDePago
opt	RadioButton	optActivo
txt	TextBox	txtApellidoNombre

- Para controles de formularios se utilizara Notación Húngara que Consiste en prefijos en minúsculas que se añaden a los nombres de las variables (controles en este caso), y que indican su tipo. El resto del nombre indica, lo más claramente posible, la función que realiza la variable, formateado en

PascalCase.

- Evite usar más de 25 caracteres ~~y "_" (guiones bajos)~~ en los nombres de controles.

Definición 8: **Nombres de Controles**

- Se debe utilizar Notación Húngara y la función de la variable formateado en PascalCase.
- Ningún control debe nombrarse con más de 25 caracteres ni guiones bajos.

6. Uso de Métodos

Conceptos y Ejemplos

En el uso de métodos se recomienda lo siguiente:

- Use nomenclatura **PascalCase** tanto para métodos públicos como privados
- Evite usar más de 25 caracteres ~~y "_" en los nombres de métodos~~.
- Use nombres consistentes, ejemplo: **CrearCliente**(), **ActualizarClientes**()
- Mantenga nombres consistentes para métodos con operaciones similares Ejemplo: Defina un método **SaveFile** si tiene un método **LoadFile**, defina un método **Close** si tiene un método **Open**, etc.
- No exceda la cantidad de 50 sentencias ejecutables en el método, si es necesario divídalo en procedimientos.
- El nombre del método debe ser lo más descriptivo posible.

Nota *El compilador JIT es más eficiente compilando métodos menores a 32bytes de código en IL.*

Definición 9: **Métodos**

- En todo método debe usar nomenclatura PascalCase.
- Ningún método debe nombrarse con más de 25 caracteres ni guiones bajos.
- Ningún método debe tener mas de 50 sentencias ejecutables.

7. Uso de parámetros o argumentos de los métodos

Conceptos y Ejemplos

Se recomienda lo siguiente en el uso de parámetros:

- Use la nomenclatura **Húngara** + **PascalCase** para el nombre de los

parámetros y se le antepondrá el prefijo "p" para distinguirlo de variables.

- No olvide comentar la funcionalidad del método esto facilitará el entendimiento de lo que hace el método cuando sus componentes sean utilizados por terceros al pasar el Mouse por encima del método.
- Es importante que se explique brevemente el significado de los parámetros de ingreso y los valores de retorno. No comente el valor de estos con "Copy & Paste" de el tipo que retorna o nombre de parámetro, esto no es de gran ayuda para quien tiene que usar estos métodos.-
- Utilice la notación húngara para definir los parámetros de un método o función.

Ejemplo **correcto**:

```
/// <summary>
/// Breve descripcion del metodo
ConsultarLineaDeCredito
/// </summary>
/// <param
name="pNroCliente">Significado del
parametro</param>
/// <returns>Significado del valor
retornado</returns>
/// <Author>Creador de la pieza de
codigo</ Author >
/// <CreationDate>Fecha de creacion</
CreationDate >
/// <LastModifAuthor>Autor última
modificación</ LastModifAuthor >
/// <LastModifDate>Fecha última
modificación LastModifDate >
///<LastModifSummary>Breve
descripción de la última modificación
realizada</ LastModifSummary >
private double
ConsultarLineaDeCredito(int
piNroCliente)
{
  ///Cuerpo del metodo .-
}
```

Ejemplo **incorrecto**:

```
/// <summary>
/// ConsultarLineaDeCredito
/// </summary>
/// <param
name="pNroCliente">pNroCliente
</param>
/// <returns>double</returns>
private double
ConsultarLineaDeCredito(int
piNroCliente)
{
  ///Cuerpo del metodo .-
}
```

- Para evitar el uso de muchos parámetros, defina una estructura o clase para más argumentos (para más detalle revisar documento de arquitectura).
- Se recomienda el uso de tipos diferentes en los parámetros para métodos sobrecargados.

Definición 10: **Parámetros o argumentos de los métodos.**

- Para nombrar parámetros debe usarse la nomenclatura **Húngara** + **PascalCase** anteponiendo el prefijo "p" para distinguirlo de variables.

- Todo método debe estar comentado incluyendo:
  a) Descripción del método,
  b) Significado del/los parámetro/s,

c) Significado del valor de retorno,
d) Autor,
e) Fecha de creación,
f) Fecha de última modificación,
g) Autor de la última modificación,
h) Descripción de la última modificación.

- Ningún método debe tener más de 6 parámetros.
- Para métodos sobrecargados se deben usar tipos diferentes en los parámetros.

8. Namespaces

Conceptos y Ejemplos

- Los *Namespaces* son utilizados para crear un sistema destinado organizar la forma de exponer la funcionalidad de un aplicativo. Dicha funcionalidad se expresa en clases .NET.

- El esquema de *Namespaces* es jerárquico, es decir permite definir un sistema de nombres comenzando por un mayor nivel de abstracción hasta llegar a niveles más específicos.

A continuación se visualiza la regla general para nombrar *Namespaces*:

<Compañía>.<[Tecnología]/[Sistema]>[.<Módulo>][.Diseño]

Dado que nuestros aplicativos se encuentran basados en un sistema de múltiples capas, inclusive capas físicamente bien diferenciadas, la regla que utilizaremos para definir *Namespaces* es:

<Compañía>.<[Tecnología][Sistema]>[.Capa Física][.< Módulo >][.Diseño]

A continuación citamos algunos ejemplos:

- **Action Line .Tools**
- **Action Line .Framework.Bases.BackEnd**
- **Action Line .SistemaFacturacion**
- **Action Line .SistemaFacturacion.Frontend**
- **Action Line .SistemaFacturacion.Frontend.FrontController**
- **Action Line .SistemaFacturacion.Backend**
- **Action Line SistemaFacturacion.Backend.Services**

Como se puede visualizar en los ejemplos, de acuerdo al área de competencia de la clase, se utiliza un *namespace* de mayor o menor nivel de abstracción o detalle. Es decir, si existen clases de tipo utilitarios que pueden ser utilizadas por más de un aplicativo, se utilizará el namespace Action Line .Tools o Action Line .Framework. Mientras que si el utilitario solo puede ser utilizado dentro del Sistema de Facturacion, el namespace a utilizar sería SistemaFacturacion.NombreSistema.Tools.

En los ejemplos también podemos visualizar la existencia de dos capas físicas bien diferenciadas: Frontend y Backend. Existe una nomenclatura que permite definir el namespaces para clases que son compartidas entre ambas capas:

"Common". Un ejemplo de cuando utilizar *Common* es cuando se definen los Request del Framework de Action Line .Framework, se utilizan tanto en el Frontend como en el Backend.

De Frontend se desprenden los namespaces utilizados para componentes como: Páginas ASPX, UIComponents, Piezas propias del Front Controller, Etc.

De Backend se desprenden los namespaces utilizados para representar a todos los servicios y las capas lógicas que se encuentran detrás: Componentes de Negocio, Componentes de Soporte, Entidades de Negocio, Entidades de Soporte, Etc.

Definición 11: **Parámetros o argumentos de los métodos.**

- Todo NamesPace debe ser definido de la siguiente forma:
  <Compañía>.<[Tecnología][Sistema]>[.Capa Física][.< Módulo >][.Diseño]

- Para clases que son compartidas entre ambas frontend y backend se debe utilizar "Common".

9. ORGANIZACIÓN VISUAL

Conceptos y Ejemplos

- No manejar en los programas más de una instrucción por línea. No incluya múltiples sentencias en una línea ya que no permite una buena lectura, además de no optimizar el compilador

- Declarar las variables en líneas separadas,
  Ejemplo:
  int a;
  int b;

- Añadir comentarios descriptivos junto a cada declaración de variables, si es necesario.

- Se recomienda declarar todas las variables en la cabecera del bloque general de ejecución y no en cualquier sector del código; esto permite que la variable utilizada pueda ser destruida en el bloque finally{}, y así facilitar la tarea del Garbage Collector.

Nota *la declaración no necesariamente implica la instanciación, así que de esta manera podemos declarar ciertas variables en la cabecera e instanciarlas tantas veces como sea necesario dentro del cuerpo del código.*

Ejemplo:

```
private void CrearCliente(ClienteDE poClienteDE) {
  //- Declaraciones
  int wNroCuenta = 0;
  double wSaldoMaximo = 0;
  ClienteBC wClienteBC = null;

  try {

    //- Comportamiento del metodo

    wClienteBC = new ClienteBC();

    dSaldoMaximo = wClienteBC.ObtenerSaldoMaximo();
    wiNroCuenta = wClienteBC.ObtenerNroCuenta();

    woClienteBC.Crear(poClienteDE, wdSaldoMaximo, wiNroCuenta);
  }
catch (Exception ex)
{
    throw ex;
}
finally
{
    //- Liberar variables.
    woClienteBC = null;
    //- Indico al GarbageCollector que puede iniciar su tarea de
recolección.
    GC.Collect();
  }
}
```

Invocando métodos en módulos

- Incluya el nombre del módulo cuando invoque a un método desde otro módulo; esto hace evidente que el método está definido en un tipo

diferente.

Ejemplo **Correcto**:

```
Helpers.ShowMessage("Hello World");
```

Ejemplo **Incorrecto**:

```
ShowMessage("Hello World");
```

- Use un único tipo de letra cuando publique versiones impresas del código fuente.
- Agregue los Namespaces en orden descendente empezando por los del sistema y terminado por los personalizados o de usuario.
- Determine una misma forma de definir una clase: ej.: Enumeraciones, Constantes, Variables Miembro, Constructores, Estructuras o Clases anidadas, propiedades y por ultimo los Métodos.
- Evite llamadas a métodos dentro de sentencias condicionales.
- No se debe pasar como parámetro la llamada a un método.

Ejemplo **Correcto**:

```
bool VerificarSaldoCliente(ClienteBE pClienteBE)
ClienteBE ObtenerCliente(int piIdCliente))
int wId = 123;
wClienteBE = ObtenerCliente(wId))
VerificarSaldoCliente(wClienteBE)
```

Ejemplo **Incorrecto**:

```
bool VerificarSaldoCliente(ClienteBE pClienteBE)
ClienteBE ObtenerCliente(int piIdCliente))
int wId = 123;
VerificarSaldoCliente(ObtenerCliente(wId))
```

Definición 12: **Organización Visual.**

- Siempre debe manejarse una única instrucción por línea.
- Todas las variables deben declararse en la cabecera del bloque.
- Al invocar un método desde otro módulo, debe especificarse el nombre del módulo.
- No ejecutar llamadas a métodos en sentencias condicionales.
- No pasar como parámetro una llamada a otro método.

10. Definición de clases

Conceptos y Ejemplos

- Evite albergar múltiples clases en un solo archivo.
- El nombre debe ser descriptivo, evitando abreviaturas, usando la

convención UpperCamelCase (para mas información visite el siguiente link: http://en.wikipedia.org/wiki/CamelCase).
- Todos los miembros de la clase deben ser privados, es decir, únicamente accesibles a través de métodos de la misma.
- Respetar convenciones de nombres de clases/métodos/propiedades del framework de desarrollo.

Definición 13: **Definición de clases.**

- En la definición de toda clase debe usarse UpperCamelCase. El nombre de cada clase y el archivo fuente deben ser iguales.

11. Definición de estructuras y enumeradores

Conceptos y Ejemplos

- Utilice carpetas para alojar los archivos que contengan definiciones de estructuras o enumeradores, por ejemplo "Structs" y "Enums".
- Evite albergar múltiples estructuras o enumeradores dentro de un solo archivo.
- El nombre debe ser descriptivo, evitando abreviaturas, usando la convención UpperCamelCase.
- Al tener estos archivos organizados en carpetas, VS utilizara la estructura jerárquica de estas para definir el Namespace de estos objetos.

Definición 14: **Definición de Estructuras y Enumeradores**

- Cada estructura o enumerador debe almacenarse en un único archivo, cuidando la consistencia del nombre del mismo.