1. OBJETIVO

Definir el nombramiento de variables, tablas, clases y funciones de la codificación en los proyectos de desarrollo de TI.

2. ALCANCE

Aplica para los proyectos del Ministerio del Interior, a menos que se acuerde en el proyecto el uso de otras reglas.

3. DESCRIPCIÓN DEL INSTRUCTIVO

3.1 Convenciones de nombramiento

3.1.1 Convención Pascal

La convención Pascal se refiere al nombramiento de elementos donde cada nueva palabra comienza con mayúscula y sigue con minúscula. Ejemplo: Color, Usuario Web, Crédito Cartera

3.1.2 Convención Camel

La convención Camel se refiere al nombramiento de elementos donde se especifica que el prefijo o primera palabra en el identificador se escribe toda en minúscula y luego se continúa de forma similar a la convención Pascal. Ejemplo: color, usuario Web, crédito Cartera.

3.2 Uso de las convenciones

La convención Pascal será utilizada para el nombramiento de:

- Proyectos (por ejemplo, ejecutables y componentes compilados)
- Espacios de Nombres (namespaces)
- Enumeraciones
- Interfaces
- Clases y Estructuras
- Funciones y métodos
- Propiedades
- Variables Públicas
- Tablas de SQL
- Campos en SQL

La convención Camel será utilizada para el nombramiento de:

- Argumentos de funciones
- Variables privadas y locales

Solo use mayúsculas sostenidas en aquellos casos que se refiera a acrónimos, como IO o ES para Entrada y Salida, XML, etc.

3.3 Reglas

3.3.1 Generales

- Evite el uso de abreviaturas, excepto en caso que:
  - Sea una abreviatura bien conocida del medio o nombre de una tecnología o producto. Ejemplo: usar IO para entrada / salida.
  - Sea una abreviatura o acrónimo usado comúnmente en el dominio del problema.
- Evite usar palabras que sean reservadas del lenguaje que se está usando.
- NO use la notación húngara para nombrar variables o clases.

3.3.2 Especificas

Elemento	Convención	Ejemplo	Observación
Espacios de Nombres (Namespaces)	Pascal	System.Drawing, Mininterior.Solicitudes	En lo posible, considere crear una jerarquía de namespaces representativa, y no agrupar en un solo namespace entidades no relacionada semánticamente. Se sugiere seguir el patrón: <Mininterior>.<Nombre de la compañía del cliente>.<Módulo de alto nivel>
Proyectos y Componentes	Pascal	Mininterior.Solicitudes.dll	Se deben nombrar siguiendo el mismo patrón que para los espacios de nombre (namespaces), cada assembly se debe formar a partir del nombre de su espacio de nombres principal.
Interfaces	Pascal	IDisposable	Identifique las interfaces con el prefijo I.
Clases, Módulos y Formularios	Pascal	AppDomain	Nombre las clases usando sustantivos. No use el carácter '_'. El nombre del archivo debe ser igual al nombre de la clase. Ejemplo la clase HelloWorld, el nombre del archivo debe ser HelloWorld.cs
Funciones y Métodos	Pascal	CrearUsuario(), CargarUsuarioDesdeDB()	Use el esquema de nombres <Verbo><Objeto>[<Predicado>]. Verbo debe ser un verbo en infinitivo que indique la acción que ejecuta el método. Objeto es el nombre de la entidad sobre la que se ejecuta la acción. Predicado representa las opciones o restricciones de la acción.
Enumeraciones	Pascal	ErrorLevel	No use ningún prefijo para el nombre de la enumeración, o el sufijo "Enum". Use sustantivos singulares para el nombre, excepto cuando la enumeración represente banderas, en cuyo caso use plurales.
Propiedades y Variables Públicas	Pascal	Color, Nombre,EstadoCivil	Use nombres descriptivos que tengan significado dentro del dominio del problema que está solucionando.
Variables Privadas y Argumentos	Camel	typeName	Siga recomendaciones similares a las establecidas para propiedades.
Constantes		NOMBRE_ARCHIVO, VALOR_DEFECTO	Nombre las constantes usando nombres en mayúsculas sostenidas, separando palabras mediante el caracter ´_´. Ej:

3.4 Formato del Código

Las siguientes son recomendaciones para una mejorar la apariencia del código escrito y facilitar su lectura y mantenimiento posteriores.

- Debe observarse correcta identificación del código. Se recomienda configurar el editor para usar una tabulación de 3 espacios y que los tabuladores sean reemplazados por espacios.
- En lo posible, haga el cuerpo de métodos y funciones corto, con el fin de facilitar su visualización en un solo "pantallazo" y hacer más fácil de entender y mantener. Es más fácil mantener un número grande de funciones cortas que hacen operaciones específicas y muy concretas, a un número pequeño de funciones que hacen muchas cosas a la vez.
- Si tiene expresiones que ocupen múltiples líneas de código, alinéelas correctamente. Siempre rompa la línea luego de un operador. Ejemplos:

```
if  ( (largo < LARGO_MINIMO) &&
      (largo > LARGO_MAXIMO) )
{
   //...
}
```

```
Dim str As String
str = “Una cadena muy larga que va aquí “ & _
     “otra cadena que aquí se junta”
```

- Evite líneas de código con más de 80 caracteres. Si la expresión es más larga, intente ajustarla sobre múltiples líneas.
- En lenguajes que usan llaves (braces) {}, se recomienda el uso del estilo BSD, donde las llaves van en líneas independientes y no indentadas:

```
class MiClase
{
    void MiFuncion()
    {
    }
};
```

Adicionalmente se recomienda el uso de este estilo en código en el lenguaje C# para la definición de propiedades, ejemplo:

```
public int Valor {
    get { return _valor; }
    set { _valor = value; }
}
```

- Si se tienen entidades de longitud "extensa" (ej: clases, métodos, namespaces, etc), se recomienda marcar claramente el fin del alcance de la entidad con un comentario. Ejemplo:

```
namespace Mininterior
{
  //
  // mucho código aquí
  //
} // namespace Mininterior
```

3.5 Convenciones de Documentación del Código

Esta sección discute las guías a seguir para la documentación de clases, módulos, métodos y variables. Es importante que siempre se mantenga la documentación del código en sincronía con el código mismo. Si realiza cambios en el código, estos se deben reflejar en su documentación asociada.

3.5.1 Documentación de Archivos

Todos los archivos de un proyecto que se editen a mano deben llevar un encabezado de documentación que contenga:

- Nombre del archivo o módulo
- Nombre del autor
- Mensaje de Copyright.

Si el archivo contiene la implementación o definición principal de un módulo, subsistema, componente, etc, el encabezado puede además contener una descripción completa del propósito de éste y una breve descripción de los contenidos del mismo.

Este es un ejemplo en C#:

```
//
// XmlDBAdmin.cs
```

```
//
// Autor:
//       Pepito Perez (pperez@Mininterior.com)
//
// COPYRIGHT(C), 2008, Mininterior
// Todos los derechos reservados.
//
```

Además de la documentación arriba mencionada, en el caso de las clases, utilice en C# los comentarios XML del lenguaje correctamente. Como mínimo, cada clase debe contener un elemento <summary>.

3.5.2 Documentación de Métodos

Cada método debe tener un encabezado que describa:

- Nombre de la rutina (opcional)
- Propósito o descripción de lo que hace
- Argumentos que toma la rutina y valores esperados por ésta.

El comentario debe denotar claramente la intención de la rutina y los supuestos que realiza acerca de sus parámetros o estado del módulo o clase a que pertenece.

Es importante notar que no todos los métodos requieren de demasiado detalle en su encabezado. En muchos casos, si la función y sus argumentos están correctamente nombrados, y la función es corta, probablemente baste simplemente con una breve descripción de su intención.

Para código en .NET y C#, se debe hacer uso de comentarios XML para clases, métodos, variables a nivel de clase y propiedades. Por ejemplo:

```
/// <summary>
/// Calcula el valor de la tasa
/// </summary>
/// <param name="numValor">Valor actual tasa</param>
/// <returns>El valor actualizado de la tasa</returns>
```

3.5.3 Documentación Interna

La documentación interna comprende diversos niveles de documentación del código: la declaración de variables globales o de clase, e interna a la implementación de los métodos.

3.5.4 Declaración de variables

Las variables globales o de clase pueden ser documentadas mediante cortos comentarios (preferiblemente de no más de una línea) inmediatamente encima de la declaración, o en la misma línea en aquellos lenguajes que lo soporten. Ejemplo:

```
int _numeroLineas; // numero de líneas visibles [0,maxLineas)
```

3.5.5 Documentación en el código mismo

Se pueden agregar comentarios lado a lado con el código mismo al interior de las rutinas. Sin embargo, debe buscarse que estos comentarios sean realmente útiles y evitar documentación excesiva, especialmente si no aporta nada al código.

Los mejores comentarios se enfocan en documentar la intención del código y preparar al lector para el código que sigue.

Deben evitarse comentarios que simplemente repitan lo que hace el código, como:

```
' cargar archivo
XmlDoc.Load("c:\\archivo.xml")
```

Deben evitarse comentarios que se enfoquen en el cómo del código que documenta. En cambio, documéntese el por qué o el qué del código. Por ejemplo, en vez de escribir:

```
// si el índice es cero
if ( index == 0 )
```

use algo como:

```
// si es el primer elemento
if ( index == 0 )
```

3.6 BUENAS PRÁCTICAS DE PROGRAMACION

3.6.1 Generales

- Evite tener archivos demasiado grandes. Si un archivo tiene más de 300-400 líneas de código, considere refactorizar en clases auxiliares.

- Evite escribir métodos muy largos. Un método debe tener típicamente entre 1 y 25 líneas de código. Si un método tiene más de 25 líneas de código, considere refactorizar en métodos separados.

- Los nombres de los métodos deben indicar lo que este hace. Si el nombre del método es obvio, no hay necesidad de largas explicaciones de lo que hace.

  Ideal:

```
void Save PhoneNumber ( string phoneNumber )
{
  // Save the phone number.
}
```

  No recomendado:

```
 // This method will save the phone number.
 void SaveData ( string phoneNumber )
 {
  // Save the phone number.
 }
```

- Un método debe hacer solamente “un trabajo”. No combine más de un trabajo en un solo método, igual si estos son muy pequeños.

  Ideal:

```
 // Save the address.
 SaveAddress (  address );

 // Send an email to the supervisor to inform that the address is updated.
 SendEmail ( address, email );

 void SaveAddress ( string address )
 {
  // Save the address.
  // ...
 }

 void SendEmail ( string address, string email )
 {
  // Send an email to inform the supervisor that the address is changed.
  // ...
 }
```

  No recomendado:

```
 // Save address and send an email to the supervisor
 // to inform that the address is updated.
 SaveAddress ( address, email );
```

```
void SaveAddress ( string address, string email )
{
 // Job 1.
 // Save the address.
 // ...
// Job 2.
 // Send an email to inform the supervisor that the address is changed.
 // ...
}
```

- No deje números quemados en código, use constantes en lugar de esto.
- No deje cadenas de texto quemadas en código, use archivos de recursos.
- Declare variables locales y páselas entre los métodos, evitando así compartir variables a nivel de la clase entre los métodos, lo cual hará más difícil llevar la traza de cual método cambio el valor y cuando.
- No haga las variables del miembro públicas o protegidas. Manténgalas privadas y exponga propiedades públicas o protegidas.
- Cuando la aplicación inicie, se recomienda realizar un auto-chequeo para asegurar que los archivos requeridos y las dependencias están disponibles en las localizaciones esperadas.
- Si las entradas en el archivo de configuración requeridas por la aplicación no están disponibles, la aplicación debe utilizar los valores por defecto o en caso de que tengan un valor errado informar al usuario.
- Los mensajes de error deben ayudar al usuario a resolver el problema. Nunca debe un mensaje al usuario como "Ocurrió un error", en lugar de esto presente un mensaje más especifico como: "No fue posible validar sus credenciales, verifique que el usuario y/o contraseña son correctos".

3.6.2 Documentación

- Mantenga la documentación del código siempre actualizada, tanto la que se encuentra en comentarios en el código, como la documentación externa.
- Los comentarios deben decir cosas del código que éste no pueda decir por sí mismo.
- Tome la costumbre de documentar el código en el momento más apropiado: justo antes de escribir el código. Por ejemplo, cree el encabezado de un archivo justo después de crear el archivo. Escriba la documentación para una rutina justo antes de escribirla, o justo después de escribir la declaración, pero antes de realizar la implementación.
- La mejor documentación que se puede escribir para un fragmento de código es el código mismo. Siempre intente que el código sea lo más claro y fácil de entender posible. Si cree que necesita escribir un comentario para aclarar un fragmento de código, pregúntese "¿Qué puedo hacer para mejorar el código y hacer el comentario innecesario?"
- No escriba comentarios por cada línea de código o cada variable. Un código claro y fácil de leer, requerirá pocos comentarios para entenderlo.
- Si usted inicializa una variable numérica con un valor diferente de 0, documente la razón por la cual eligió es valor.
- Los comentarios deben decir cosas del código que éste no pueda decir por sí mismo.

3.6.3 Variables

- Si el lenguaje lo permite, acostumbre declarar una variable lo más cerca posible a su primer uso. Trate también de mantener cercanas las expresiones que referencian una misma variable.
- En general, si cree que se requiere de un comentario para aclarar o hacer explicito el significado del nombre de la variable, el nombre no ha sido correctamente definido
- Evite en la medida de lo posible las variables globales. En aquellos casos en que realmente se justifican, considere crear rutinas de acceso que encapsulen y protejan su uso para evitar efectos colaterales.
- Use cada variable para un solo propósito.
- Prefiera el uso de constantes sobre literales, bien sean éstas números, cadenas, etc.
- Para las variables booleanas, use un nombre tal que su significado sea evidente cuando el valor de la variable sea verdadero (true).

- Los nombres de variables buenos tienden a representar el que de la variable, más que el cómo. En la medida de lo posible, se debe buscar que el nombre de las variables corresponda directamente a entidades en el dominio del problema y no de la solución.

3.6.4 Formato

- Sea consistente.
- Tome provecho del espacio en blanco (whitespace) y los paréntesis para hacer las expresiones más legibles; no cuestan nada.
- Use el formato visual del código para reflejar la estructura lógica de este.
- Evite construcciones muy anidadas. En general, si requiere de más de tres niveles de anidamiento, considere usar rutinas auxiliares.
- Evite usar expresiones de condición (if, while, etc) complejas. Si realmente requiere de una expresión muy compleja, considere mover la evaluación de la condición a una rutina auxiliar ó hacer uso de tablas de decisión.

3.6.5 Manejo de excepciones

- Nunca atrape una excepción y no haga nada con ella. Si usted esconde una excepción, nunca sabrá que ocurrió.
- Si ocurre una excepción muestre un mensaje amigable al usuario y registre en el Log de la aplicación el mayor detalle posible, incluyendo fecha, hora, nombre de la clase, método e ideal los valores de los parámetros.
- Siempre atrape la excepción específica, no excepciones genéricas.
- No use try-catch en todos sus métodos. Hágalo solamente cuando puede agregar información de valor al error ocurrido, de lo contrario deje que estas fluyan y evidencien que la aplicación falló. Esto permitirá detectar los errores durante la etapa de desarrollo.
- No encierre grandes bloques de código en try-catch. Si es requerido, escriba un separado try-catch por cada tarea que usted ejecute y encierre solo esa pieza especifica dentro de un try-catch. Esto permitirá encontrar más fácilmente el código que génerо el error y dar información más específica al usuario.
- Si es requerido en su aplicación, escriba sus propias clases de excepciones.

4 CONTROL DE CAMBIOS

VERSIÓN	DESCRIPCIÓN DE CAMBIOS	FECHA
02	Se actualiza la plantilla a la nueva versión y se adiciona Campos en SQL en el numeral 3.2	11/12/2020

5 CONTROL DE FORMALIZACIÓN

ELABORÓ	REVISÓ	APROBÓ
ROSA RODRÍGUEZ MORENO ULIANOV ANTONIO ECHEVERRIA **PROFESIONALES GESTIÓN TECNOLÓGICA**	MARCO ALEXANDER MORALES RUEDA **COORDINADOR GESTIÓN TECNOLÓGICA**	**PAOLA VERA GÓMEZ** **JEFE OFICINA DE INFORMACIÓN PÚBLICA DEL INTERIOR** Validado a través del correo paola.vera@mininterior.gov.co con fecha del 11/12/2020